|  |  |  |
| --- | --- | --- |
| Republique du Cameroun |  | Republic of Cameroon |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| Paix-Travail-Patrie | Peace-Work-Fatherland |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| Ministere de enseigment superieure | Ministry of High Education |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| Universite de Buea | University of Buea |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  | Faculty of Engineering and Technology |
|  |  |
|  | Department of Computer Engineering |
|  |  |

**CEF 440 INTERNET AND MOBILE PROGRAMMING**

**PROJECT REPORT**

**PRESENT BY:**

|  |  |  |
| --- | --- | --- |
| **NAMES** | | **MATRICULES** |
| **GUEGUIM SONNA ZITHA UNELLE** | | **FE20A045** |
| **MAKONDI THIERRY JUNIOR** | | **FE20A062** |
| **NGANKEP FANDIO ORDY BENIDI** | | **FE20A075** |
| **HIEGA EMMANUEL JOEL** | | **FE20A050** |
| **OMYOM KILLENG ZACHARIE FRANKLIN** | **FE20A096** | |

***Abstract:***

This document is the report of our work done on the exercise that has been given to us during the internet and mobile programming course. This work has helped us study the various aspects involved in mobile development. We were able to do it by searching on the internet and asking questions to some of our seniors who are working in some startup and company in Buea. We have found out that there are three main types of mobile application (native application, web application, and hybrid application), some programming languages for developing mobile apps are Java, Dart, Kotlin, C# …, the more popular mobile apps development framework are React native, flutter, ionic, xamarin and phoneGap. We have also found out that the best way to estimate cost is by making good planning, this also helps to understand the requirement of an application.

***Table of content***

1. **Major types of mobile application and the different type that exist.**

* **Criteria us to compare them.**

1. **Review mobile application programming language.**

* **Give comparism between them.**

1. **What is the mobile application development framework?**

* **Review them and compare them to key features, that is language, performance, value etc.**

1. **Study how to collect and analyze the requirements for a mobile application to be developed.**

* **Give the different stages it takes to be developed**

1. **How to estimate the mobile application cost.**

1. ***DIFFERENT TYPES OF MOBILE APPS AND THEIR DIFFERENCES.***

*There are three different types of mobile app which include;*  
*Native apps*  
*Web apps*  
*Hybrid apps*

**1.**Native Application  
 Native application are created specifically for an operating system or a platform.

These apps keep mobile device operating systems in mind. This means that you can have apps native to Android or iOS.  
Native app builders use the following programming languages to create their apps:  
● JAVA  
● Python  
● C++  
● Objective-C  
● Kotlin  
● Swift  
● React.  
As these apps have a single focus point, they have the added advantage of being faster and more reliable. They are efficient when used with the correct operating system and utilize the native device UI to give users a more optimized experience. But on the other hand, one major problem that is understandable with native apps is that it is only available to a single operating system user, like people who only use iPhone or Android devices. So, the app needs to be duplicated for different platforms; otherwise, app publishers lose out on thousands of customers and also, the code once used for an app cannot be duplicated. This means that new codes need to be generated and updated. This can be quiet costly.  
Examples : Calculator, Notepad,  
 2. Web Apps  
Unlike native apps, Web apps are responsive versions of specific websites that work on any device as they are delivered using mobile browsers.  
 The technology used to build web applications include;  
● HTML5  
**●** JavaScript  
● CSS  
● Ruby   
Some advantages of the app is the fact that, not many development costs are involved as these apps are web-based and do not require customization. Also, since there is nothing to download, these apps do not take up much device storage memory like native apps. These apps are also easier to maintain, and no special downloads are required. Web apps are completely dependent on the browser used. Hence, these apps do not work offline. Even if they have an ‘offline mode,’ the divide needs to be connected to the internet to back up data or refresh previously stored information.   
example include; Google Apps and Microsoft 365  
 3. Hybrid Apps  
Just as the term hybrid means, Hybrid apps is the combination of both native and web apps but are warped in a native app style. It can have its icon or logo and be downloaded from an app store directly. These are web apps with the look and feel of native apps. They have a home screen icon, are built to be responsive, function offline, and even have faster performances than web apps. They are the perfect combination of both Native and Web apps.  
Hybrid apps are a mix of web technology and native APIs. Therefore they are built using:  
● Objective C  
● HTML5  
● Ionic  
● Swift  
The hybrid app is much faster and more responsive than a web app and is cheaper to build than native apps. These apps are considered minimum viable products – a way to prove the viability of moving forward with a native app.  
These apps load faster and are ideal for countries with slow internet connections. They also have a more consistent user experience across all platforms. Finally, these apps have a single code base which is easier to maintain and update.  
Just as it has its pros, its cons include;  
They are not as swift and influential as native applications.  
Examples: Facebook, Twitter, Yelp

**II.A REVIEW ON MOBILE APP PROGRAMMIG LANGUAGES**

There are several programming languages used for mobile app development, each with its own strengths and weaknesses. Here are six of the most popular ones and a comparison of their key features:

1. **Java:**

Java is one of the most widely used languages for developing Android apps. It has a large standard library and is highly versatile, making it suitable for developing complex applications. Java is an object-oriented language, which means it's easy to write and maintain code. However, Java can be verbose and require more code than some other languages.

1. **Swift:**

Swift is a modern language developed by Apple for building iOS, iPadOS, macOS, tvOS, and watchOS apps. It's a fast and efficient language that is easy to read and write. Swift has powerful features for handling optionals, generics, and closures, and it's a safer language than Objective-C, the predecessor of Swift. However, Swift is limited to Apple's platforms only.

1. **Kotlin:**

Kotlin is a relatively new language that has gained popularity in recent years. It's an officially supported language for Android development by Google, and it's interoperable with Java, making it easy to use alongside Java in Android projects. Kotlin has a concise syntax, powerful features such as coroutines and extensions, and strong support for functional programming. However, it's still less widely used than Java, so it may be harder to find support and resources compared to more established languages.

1. **C#:**

C# is a language developed by Microsoft that is used for developing Windows, Xbox, and Windows Phone apps. It's also a popular choice for game development using the Unity engine. C# is a strongly typed language that offers a high degree of performance and memory management, and it has features like garbage collection and LINQ. However, C# is limited to Microsoft's platforms only.

1. **JavaScript:**

JavaScript is a popular language that is used for developing cross-platform mobile apps using frameworks like React Native, Ionic, and PhoneGap. It's widely used for web development and offers a large number of libraries and frameworks. JavaScript is an interpreted language, which means it can be slower than compiled languages like Java or Swift.

1. **Python:**

Python is a language that is gaining popularity for mobile app development, thanks to frameworks like Kivy and BeeWare. Python has a simple and easy-to-learn syntax, making it a good choice for beginners. It's also highly portable and can be used for both Android and iOS platforms. However, like JavaScript, Python is an interpreted language, which can lead to slower performance compared to compiled languages.

1. **Dart:**

Dart is a language developed by Google for building mobile apps using the Flutter framework. Dart has a concise syntax and offers features like hot reloading, which allows developers to see changes to the code in real-time. Dart also has strong support for reactive programming, making it easier to build responsive and interactive user interfaces. However, since Flutter is a relatively new framework, Dart may not have as many resources or community support as more established languages.

In summary, the choice of programming language for mobile app development will depend on the specific needs of the project, the platform being targeted, and the developer's experience and preferences. Each language has its own strengths and weaknesses, so it's important to carefully evaluate these factors before choosing a language for a mobile app project.

**III. *Top Mobile App Development Frameworks in 2023***

Mobile app development frameworks are software platforms that provide developers with pre-built tools and libraries for creating mobile applications. They allow developers to write code in a specific programming language and use pre-built components to speed up development and simplify the process. Here are some of the popular mobile app development frameworks and a comparison of their key features:

1. React Native: React Native is a JavaScript-based mobile app development framework developed by Facebook. It allows developers to write code once and deploy it across both iOS and Android platforms. React Native is known for its performance and is popular among developers who want to create high-performance apps.

Key features:

* Language: JavaScript
* Performance: Fast performance, with most of the code running natively on the device
* UI/UX: Native look and feel on both iOS and Android platforms

1. Flutter: Flutter is a mobile app development framework developed by Google that uses the Dart programming language. Flutter allows developers to create high-performance, visually attractive mobile apps with a single codebase that can be deployed to both iOS and Android platforms.

Key features:

* Language: Dart
* Performance: Fast performance, with code running on a compiled machine code
* UI/UX: Customizable widgets that provide a visually appealing UI/UX

1. Xamarin: Xamarin is a mobile app development framework developed by Microsoft. It allows developers to create cross-platform mobile apps using C# and .NET programming languages. Here are some of its key features:

* Language: C# and .NET
* Performance: Good performance, with code running natively on the device
* UI/UX: Native look and feel on each platform with customizable UI components, with the ability to share code across different platforms

1. Ionic: Ionic is an open-source mobile app development framework that uses web technologies such as HTML, CSS, and JavaScript. It allows developers to create cross-platform mobile apps using a single codebase that can be deployed to multiple platforms. Here are some of its key features:

* Language: JavaScript
* Performance: Good performance, with most of the code running in a WebView
* UI/UX: Customizable UI components that provide a native look and feel on both iOS and Android platforms, with a wide range of pre-built UI components available

1. PhoneGap PhoneGap is a mobile app development framework that uses web technologies such as HTML, CSS, and JavaScript. It allows developers to create cross-platform mobile apps using a single codebase that can be deployed to multiple platforms. Here are some of its key features:

* Language: JavaScript
* Performance: Good performance, with most of the code running in a WebView
* UI/UX: Customizable UI components that provide a native look and feel on both iOS and Android platforms, with a large plugin library available for added functionalist

***IV. Study how to collect and analyze the requirement for a mobile application to be developed?***

A successful mobile app should provide an excellent user experience and help you reach your business goals. It should also be easy to use and give some added value to the user that they wouldn't get from a standard website, plus bring data together to help you improve customer experiences. There exist several steps to write a great mobile application. In the following line I will list some of them;

1. ***Describe the app idea***

A great mobile app requirements document should start with the core idea the main reason the app is worth building in the first place. Preferably, the description of an idea must be given in a single sentence. Hence, the reader should be able to understand the main purpose of the app and connect with its usability almost instantly.

1. ***Describe the navigation patterns***

Once your core idea has been established, the rest of the document should flesh out how you’ll achieve that in the app.

An important aspect to include is your chosen **navigation pattern**. This refers to how users navigate from one area of your app to another.

Different navigation patterns have their pros and cons. Your choice of one over the other will also depend on the nature of your app.

For example, one popular navigation pattern, especially for iOS apps, is the tab bar.

Menu bars are best for apps that have relatively few sub-pages. Their main advantage is that they are always visible, so it’s easy to switch between screens in your app.

However, it can’t be difficult to choose the right navigation pattern if you haven’t laid out your **user flow** first.

This process requires you to flesh out all the screens in your app and how they connect, starting from user registration and onboarding.

Here’s an example:

User flows are vital to understanding how users should go through your app.

This insight enables you to choose the best navigation pattern that reduces friction and improves user experience (UX).

1. ***Identify must-have features***

The next step is to define the features that should appear in your app.

But rather than include them in one long list, you should identify the core features, as opposed to mere nice-to-haves.

This is important so developers can prioritize what they need to work on first.

First, let’s define what core features actually mean. These are the *must-haves* that the app needs to fulfill its purpose.

For instance, searching for a ride and booking it are all core features in the Uber app.

But fare estimation—despite being useful—isn’t an essential function. It’s just a nice bonus, which is why Uber didn’t include it in their [MVP](https://word-edit.officeapps.live.com/we/wordeditorframe.aspx?ui=en-US&rs=en-US&hid=eNyUHA4edESd30xu8OxosQ.0&wopisrc=https%3A%2F%2Fwopi.onedrive.com%2Fwopi%2Ffiles%2FD1D2AE7E2EE0EAF1!392&wdpid=4a469ed1&wdo=2&wde=docx&sc=host%3D%26qt%3DDefault&mscc=1&wdp=0&uih=OneDrive&jsapi=1&jsapiver=v2&corrid=6fe111aa-1110-4faf-a891-8c69d2e10572&usid=6fe111aa-1110-4faf-a891-8c69d2e10572&newsession=1&sftc=1&wdorigin=Other&instantedit=1&wopicomplete=1&wdredirectionreason=Unified_SingleFlush#:~:text=anything%E2%80%99s%20fair%20game.-,uber,-Uber%20started%20life).

Admittedly, it can be tricky to isolate your core features, especially if everything *seems* useful to the user. But one tool that can help you is the MoSCoW priority matrix.

As the above diagram shows, the MoSCoW matrix forces you to segregate your features list into four quadrants based on three metrics—**risk, impact,** and **effort.**

Features in the **must-have** and **should-have** quadrants are your core features. These are the functionality that greatly impacts your app and the user experience.

The only difference is that should-have quadrants involve more effort and risk to implement.

Those in the **could-have** are features that are nice to have. They generally have a low to medium impact on the app, which means they’re not a priority.

But because they require little effort and risk, they’re easy to implement.

The last quadrant includes **will-not-have** features that are difficult and risky to implement yet have minimal impact. Avoid them at all costs.

1. ***Mention existing apps***

It’s a good idea to mention existing apps similar to yours as a reference point. As long as you don’t copy 100%, adopting what’s already out there is actually a good strategy. It can make your app familiar to users, which is great for building trust and enhancing usability.

Doing this helps the reader visualize the features and navigation patterns you describe.

That way, developers and designers know exactly how to implement them with minimal misinterpretations.

For example, no doubt Lyft used Uber as their inspiration when designing their interface, as these side-by-side screenshots show.

1. ***Focus on functional specifications***

Functional specifications are the aspects of your app that detail how it will solve the user’s problem.

Core features are essential to a functional specification, but you should go beyond that.

You must also consider the infrastructure behind the app, such as development tools and app-server interactions.

It tackles **what** the system should do to solve the user’s problem. In contrast, non-functional requirements are about **how** the app does it, mostly including performance metrics like scalability, speed, and security.

No doubt, non-functional specifications are vital for the user experience. After all, no one wants to use a slow app. But they’re not a priority when writing a requirements document.

The reason is focus.

1. ***Mind the app technology requirements***

You should specify the minimum operating system and hardware requirements for the app to function optimally.

For one, various devices have different amounts of RAM, as this infographic shows.

In addition, iOS and Android handle memory management differently.

Android runs on the Java runtime environment, making memory management easier for the developer.

However, the trade-off is that it also consumes more memory. iOS, in contrast, gives developers more freedom to optimize their app’s memory handling.

Pointing out these nuances is crucial in a requirements document. It allows your developers to prepare the necessary tools and approaches when coding.

If your app requires access to any device hardware, such as the microphone or camera, indicate that in this section.

You should also consider that various screen sizes are available in the market, even within the same operating system. The following infographic makes this evident:

Fortunately, iOS and Android include resizing features in their SDK, allowing your UI to be fluid and adaptable. But your developers can only prepare for it if you mention it in the document.

1. ***Provide wireframes***

A wireframe is a visual outline of your app’s UI, the equivalent of a blueprint in architecture. Including it in the requirement document is essential because it can give the reader more clarity.

Here’s what a wireframe looks like:

Wireframes are useful for illustrating to developers and designers exactly what you want the app to look like.

It’s like the reference point of your app project – helping avoid miscommunication and misinterpretation.

You can also use your wireframe to show your app’s user flow and interactive elements.

This wireframe by Risa Nkakajim is a great example of how you can do it effectively, even in a static medium.

It’s a good idea to annotate your wireframes as you go along. Aside from providing more context, you can also use it to provide additional instructions to your developers.

For instance, take a look at the annotation below.

Instead of simply saying “make this element accessible”, the annotation lays out the exact specifications. That way, there’s minimal guesswork on the part of the developer.

S

The bottom line is that wireframes can make your requirements document much clearer, readable, and even more engaging. Indeed, the best wireframes are those that can speak for themselves.

1. ***Choose the requirement format***

Once you’ve prepared the content of your requirement document, the last step is to decide on the format.

There are four types to choose from—**functional specification document (FSD)**, **user stories**, **sketches and wireframes**, and **mixed format**.

A **functional specification document (FSD)** is the standard format where the app’s features and functionality are listed in detail.

They’re similar to an instruction manual for building an app. An FSD is mostly written from a developer’s point of view.

In contrast, **user stories** are written from the user’s perspective. They list down the things a person can do while in the app.

This approach allows you to look at the business case of your app.

However, because user stories tend to be simple and non-technical, developers need further details to build an app from them.

**Wireframes and sketches** is a format where you mostly use wireframes, mockups, and other visual aids to describe your app.

They’re much more engaging and easier to understand than an FSD, but they require more time to create.

TShe **mixed format**. This one involves using different formats in your document to paint a clearer picture.

For instance, you can enhance an FSD by adding a wireframe as a visual aid.

***V.How to Estimate software development cost?***

**What affects software development cost?**

Aspect of software project that have larger impact on the cost of software development are the following:

- **App complexity:**

The complexity of a software depends on the number of features you want in the software. It increases the time spent creating, testing, debugging, and deploying.

The longer software development takes, the more expensive it will be.

The complexity of a software can be broken down into 3 categories:

-Feature set: More features require more communication between modules, custom functionality, and greater complexity.

-Technology: There are traditional, basic technologies that are often used in software development, and there are complex advanced technologies, such as AI, that get used when traditional options are not sufficient.

-Design: Personalized, custom design elements add an additional layer of complexity.

Complexity and customization go hand in hand. Customization is a time-consuming process tar requires new code, functionality, and addition testing. As a result, complexity arguably impacts cost most when developing software

**-Software size**

The size of the software will affect how long it takes to complete the project. The number of unique screens or pages defines size in software development.

A larger project will typically have 40 or more unique pages or screens.

Medium- sized project will generally have roughly 25 to 40 unique screens, and small project s will have 25 or fewer

**-Platforms**

The platform you choose to develop software for will affect software costs. Additionally, if you decide to develop software versions for multiple platforms, your organization will spend more money.

Native app development costs more than cross-platform or hybrid development. The software must be specialized and use specific native technologies and languages.

Organizations can control software development costs by developing a universal, cross-platform solution. However, cross-platform software cannot utilize native, device-specific hardware as effectively as native software.

Many small or medium-sized businesses will focus on a single platform that most aligns with their target audience if they choose native development over cross-platform.

**-Custom design**

A significant part of custom development in design. UX/UI design is vital to user engagement and satisfaction, as a result, design becomes a critical factor in software development pricing.

As we covered when discussing software complexity, more screens and page means more design work which means custom development costs more

The last thing your organization want to do is cut corners on the UX/UI

Design. User retention and engagement rely on good software design. Failing to deliver a pleasing User Experience will cost your business more in the long run

**-Development team size**

The size and location of the development team will play a significant role in the cost of the software development.

For example, offshore developers are cheaper to hire than nearshore and onshore teams. However, offshore developers will work in the time zones on the other side of the

**-Region:**

The region where the development is taking place can also affect costs. In general, app development costs tend to be higher in regions with higher labor costs, such as North America and Western Europe, compared to regions with lower labor costs, such as Asia and Eastern Europe.

North America and Western Europe: Development teams in [North America](https://www.payscale.com/research/US/Industry=Software_Development/Hourly_Rate) and Western Europe generally have the highest hourly rates.

Eastern Europe: Development teams located in Eastern Europe tend to have lower hourly rates compared to North America and Western Europe.

Asia: Development teams located in Asian countries, such as India, China, and the Philippines, generally have the lowest hourly rates.

Africa:

**Hidden App Development Costs:**

Hidden app development costs refer to expenses that may not be immediately apparent when developing an app. These costs can often be overlooked or underestimated, but they can significantly impact the overall cost of app development. Some examples of hidden app development costs include:

**Backend infrastructure**: Building an app requires more than just creating the user interface. A robust backend infrastructure is necessary to ensure the app works smoothly and efficiently. Maintaining this can involve server hosting, database management, and other related expenses.

**Integration with third-party services**: Many apps require integration with third-party services such as payment gateways, social media platforms, or APIs. Integrating these services can incur additional costs, such as licensing or API usage fees.

-**Security features**: Security should be a top priority for any app development project. However, implementing security features such as encryption, two-factor authentication, and Mutual TLS verification can incur additional costs.

**App store fees**: If you plan to distribute your app through app stores such as Apple’s App Store or Google Play, you’ll need to pay fees to these platforms. These fees can include charges for app submission, updates, and hosting.

**Maintenance and updates**: After launching an app, ongoing maintenance and updates are necessary to keep it running smoothly and to address any bugs or issues. Maintenance can include costs for engineering support time, server costs, and other related expenses.

**Marketing**: Promotions and ads help you to reach your audience and make a successful app.

It’s important to keep these hidden app development costs in mind when budgeting for your app development project to ensure that you have a comprehensive understanding of the total cost of your app.

**How to estimate cost?**

For us to better estimate the cost of a software development as a software engineer, we should do as follow:

### **Create a project brief and high-level modules map**

When it comes to estimating the cost of developing software, it is important to begin with a high-level understanding of the project at hand. If you do not already have one, write up a project brief that describes the purpose of the software product, its audience, and its basic functionality.

From there, describe the modules involved in the project as well as their dependencies. A high-level map of the modules and their features will help you to understand the scope of the project and give you a good basic understanding on which you can build a more detailed software development cost estimate.

As you put together a module map, make sure to take into account:

Databases

Algorithms

User roles

User interface

Infrastructure

Feature-specific modules

3rd party integrations

Next, consider the different dependencies between these modules, such as how each module relies on other modules or how the modules need to be tied into other systems.

### **Research and select technologies**

Once you have created a high-level map of the modules and their dependencies, the next step is to research the technologies that are most suitable for your project.

If you have one or more technologies that you know you’d like to use (or that are simply required), it’s important to work with your software development team to conduct additional validation work at this stage to ensure that they are suited for your project.

There may also be other technologies that are more flexible. For example, you may be able to utilize “building blocks” such as open-source or commercial components or services to speed up your software development project and keep costs down. This is a good time to research these and select the most appropriate and cost-effective options.

When researching technologies, consider the following:

What technologies are required for your project? Have you validated that there aren’t any better alternatives?

What open-source or commercial components are a good fit for your project?

What are the one-time costs for each of the technologies you are considering?

What are the ongoing subscription, maintenance, or software costs for each of the technologies you are considering?

What engineering or development costs would be required to customize the technologies you are considering?

How quickly will you be able to build your software solution with the technologies you’re considering?

How stable are the technologies that you are considering?

Will finding engineers to work with each technology be an issue?

Will the cost of hiring engineers be affected by each technology you’re considering?

What are the benefits of each technology?

What are the limitations of each technology?

### **Draft software requirements specification (SRS) document**

Before you can accurately estimate software costs, you need to create a [**Software Requirements Specification**](https://www.softkraft.co/software-requirements-specification/). This document should outline at a high level:

Product summary and scope

Intended audience

Assumptions and dependencies

Functional requirements (what the system is supposed to do)

Non-functional requirements (how the system should perform)

Acceptance criteria

By creating an SRS, you can ensure that all members of the project team and all other stakeholders have a clear understanding of the project requirements.

The SRS document is also a valuable tool for prospective vendors, as it will allow them to create more detailed estimates. By providing a complete and detailed SRS, vendors can identify the scope of the project and the potential costs involved.

### **Create a work breakdown structure (WBS)**

Another way to improve the accuracy of your estimate is to put together a work breakdown structure (WBS). Essentially, a WBS lists all of the tasks required to complete the project along with an estimate of the cost of each task.

The level of detail of your WBS is up to you. Teams can create anything from a high-level WBS (like the one shown below) to a low-level WBS that breaks tasks down into a detailed list of sub-tasks. When in doubt, aim for something in the middle that will provide you a reasonable level of accuracy without risking overestimation that can sometimes occur if you try to make it too detailed.

A work breakdown structure should:

Be put together either by or in tandem with your software development company. The people actually doing the work must be involved or it will not be accurate.

Help your project manager or development team to build a project timeline.

Provide a way for teams to monitor progress and keep an eye on tasks that may be at risk for running over budget.