Republic of Cameroon
******
Peace - Work - Fatherland
*****
Minister of Higher Education
*****
University of Buea

Republique du Cameroun
******
Paix - Travail - Patrie
*****
Ministere de l'enseignement Superieur
*****
Universite de Buea

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER ENGINEERING

MOBILE APPLICATION DEVELOPMENT

CEF 440

TASK 1 REPORT:
INTERNET PROGRAMMING AND MOBILE PROGRAMMING

Presented by:

GROUP 8

N°	NAMES	MATRICULES
1	FOWEDLUNG ATSAFAC AGAFINA	FE21A196
2	KAMDEM KAMGAING GILLES CHRISTIAN	FE21A209
3	NEGUE KWAHAM MAEL GRACE	FE21A252
4	NGUEPI GNETEDEM PATERSON	FE21A264
5	NOUPOUWO DONGMO STEPHANE MERCI	FE21A283

Course Instructor: Dr Valery Nkemeni

April 2024

TABLE OF CONTENTS

I- INTRODUCTION

In today's tech-driven world, internet and mobile programming are like the building blocks of our digital experiences. Internet programming helps create the websites and apps we use every day, while mobile programming focuses on making apps specifically for smartphones and tablets. This report dives more into mobile programming, talking about the development life cycle

Definitions:

Internet Programming: Internet programming is all about creating websites and web applications that we use on our computers and browsers. It involves languages like HTML, CSS, and JavaScript to build web pages and make them interactive.

Mobile Programming: Mobile programming is about making apps for smartphones and tablets. It uses languages like Java or Kotlin for Android apps, and Swift or Objective-C for iOS apps. Mobile programmers make sure apps work smoothly on small screens and take advantage of features like touch controls and sensors.

Through this report, we'll break down these concepts into simpler terms, and see how mobile programming process looks like.

I- MOBILE APPs TYPES

I.1 Review of Mobile Apps

There exist different types of mobile apps categorized based on different features. Based on the technologies used in coding, there are basically three types; **Native** apps, **Progressive** web apps and **Hybrid** apps:

a) Native apps:

These are apps developed specifically for a particular platform (iOS or Android) and are written in the platform's native programming language. Native apps have the best performance and access to all the features of the device like camera, GPS, speaker, user contact lists etc., for an optimal user experience.

- ***Technologies Used*:** Native apps are coded using a variety of programming languages. Some examples include: Java, Kotlin, Python, Swift, Objective-C, C++, and React.

- ***Pros*:** Because of their singular focus, native apps have the advantage of being faster and more reliable in terms of performance. They're generally more efficient with the device's resources than other types of mobile apps. Native apps utilize the native device UI, giving users a more optimized customer experience.

- ***Cons**:* However, the problem with native apps lies in the fact that if you start developing them, you have to duplicate efforts for each of the different platforms. The code you create for one platform cannot be reused on another. This drives up costs. Not to mention the effort needed to maintain and update the codebase for each version.

- ***Examples:*** Examples of Native apps include WhatsApp, Facebook and Instagram by Meta Platforms, Inc and Google maps by Google LLC.

b) Progressive Web Apps(PWAs):

These are apps that use web technologies to provide users with an app-like-experience. They are designed to work across different devices and platforms and are associated with features of the native mobile applications such as offline access, push notifications and low storage requirements.

- ***Technologies Used:*** PWAs are designed using HTML5, CSS, JavaScript, Ruby, and similar programming languages used for web work.

- ***Pros:*** Because it is web-based, there is no need to customize to a platform or OS. This cuts down on development costs. Also, there is nothing to download and as such they won't take space on devices' memory. This makes maintenance easier as updates can be done automatically.

- ***Cons:*** PWAs are entirely dependent on the browser used on the device. Thus, functionalities might vary from one browser to another, possibly giving users varying experiences.

- ***Examples:*** Examples of PWAs include WhatsApp web, FreeCodeCamp ,etc..

c) Hybrid Apps:

These are mobile applications built using mobile framework or web technologies and wrapped into a native container that enables them to be installed and run like native apps. This approach enables developers to build a single codebase that can be deployed across different operating systems, such as iOS and Android.

- ***Technologies Used:*** Hybrid apps use a mixture of web technologies and native APIs. They're developed using: Ionic, Objective C, Swift, HTML5, and others.

- ***Pros:*** Building a hybrid app is much quicker and more economical than a native app. Also, they have a rapid load time and as such are ideal for usage in countries with slower internet connections and give users a consistent user experience. Finally, because they use a single code base, there is much less code to maintain.

- ***Cons:*** Hybrid apps might lack in power and speed.

- ***Examples:*** Examples of Hybrid Apps include Twitter, Instagram, Uber, etc..

I.2 Comparing Different Mobile App Types

a) Similarities

Though different, the above stated mobile apps share common features some of which are as follows:

- All can deliver app-like experiences: Users can install them on the home screen, receive push notifications, and (to some extent) work offline.
- Can be used across various devices: All can be accessed on different screen sizes and operating systems (though functionality might differ).
- Meet diverse needs: Each approach can be suitable for different app purposes, depending on complexity and desired features.

b) Differences

Aspect	Native apps	Progressive web apps	Hybrid apps
Development Language	Platform-specific languages (e.g., Swift, Objective-C for iOS; Java, Kotlin for Android)	Web technologies (HTML, CSS, JavaScript)	Web technologies (HTML, CSS, JavaScript) with native wrappers (e.g., React Native, Ionic)
Distribution	App Store (e.g., Apple App Store, Google Play Store)	Accessed through web browser	App Store (similar to native apps)
Development Cost	Higher due to separate codebases for each platform	Lower	Moderate (depending on chosen framework and complexity)
Performance	Best performance with direct hardware access.	Generally moderate performance as they run within a web browser, but advancements in browser technology are improving performance.	Performance varies depending on framework and complexity. Often falls between native and PWAs.
Offline Functionality	Works well offline with local data storage.	Can offer offline capabilities with data caching (level depends on implementation).	Limited offline functionality depending on framework. May require extra development effort.
Examples	Facebook (iOS and Android apps), Instagram (iOS and Android apps)	Gmail, Google Maps (mobile web versions)	Instagram, Twitter (using frameworks like React Native, Ionic)

II- MOBILE APP PROGRAMMING LANGUAGES

III.1 Reviewing Different Mobile App Programming Languages

Various programming languages do exist for mobile app development as there are many languages in the world to communicate. Depending on your objectives and the type of application intended to create, different programming languages are used.

Some of the top mobile app programming languages used in the world today are Java, Swift, Objective-C, JavaScript, HTML5, C++, Kotlin, Dart, etc…

1. **Java:**

   Java is widely used for Android app development, it offers performance, scalability, and robustness. It's backed by a mature ecosystem and extensive libraries.
   Java is probably the most used programming language in the world. Having about 7.1 million active users (Statistica). Its usage is not only limited to mobile apps but also desktop applications, web services, back-end frameworks. It uses the Java virtual machine (JVM).

2. **Swift:**

   Developed by Apple, Swift is the primary language for iOS app development. It provides modern syntax, safety features, and performance optimizations. Development and immersive application can be developed using this language especially for Augmented Reality and Virtual Reality. It has active usage of close to 2.3 million users.

3. **Objective-C:**

   It is an object-oriented programming language that supports general-purpose applications domains. Also used by Apple Inc. It was replaced by Swift in 2014 and still has an active population of 1.8 million users.

4. **JavaScript:**

   Used for hybrid mobile app development, JavaScript allows developers to build cross-platform apps using frameworks like React Native and Ionic. It offers code reusability and access to native device features with a large community and vast amount of libraries.

5. **Kotlin:**

   Emerging as the preferred language for Android app development, Kotlin combines readability with powerful features, concise syntax, aiming to improve upon Java.

6. **Dart (used with Flutter framework):**

   Google's take on cross-platform development. Offers a unique widget-based UI framework and good performance. Still under development compared to React Native.

7. **C#:**

With Microsoft's Xamarin framework, C# enables developers to create cross-platform mobile apps for iOS, Android, and Windows. It offers familiarity for .NET developers and native-like performance.

III.2 Comparing Different Mobile App Development Programming Languages

Language	Main Use	Key Features	Weaknesses
Java	Android app development	Performance, scalability, robustness	Verbose, can feel dated
Swift	iOS, macOS, watchOS app development	Modern syntax, safety features, performance optimizations	Limited to iOS
Kotlin	Android App development	Modern features, concise syntax, interoperability with Java	Still evolving
Javascript	Hybrid mobile app development	Cross-platform compatibility, access to native device features through frameworks	Performance limitations for complex apps
C#	Cross-platform mobile app development	Familiarity for .NET developers, native-like performance	Requires .NET knowledge
Dart	Cross-platform mobile app development	Unique UI framework, good performance	Under development compared to React Native
Objective-C	iOS app development	Offers dynamic runtime and object-oriented features for flexible, reusable code.	Verbose, and its declining popularity may limit future support and resources

III- MOBILE APP DEVELOPMENT FRAMEWORKS

Mobile app development frameworks provide developers with pre-built tools, libraries, and structures to expedite the creation of applications for various platforms like iOS and Android. These frameworks streamline the development process by offering features such as UI components, cross-platform compatibility, and simplified deployment. Frameworks empower developers to create robust, feature-rich mobile applications efficiently, saving time and effort in the development cycle.

Examples of mobile app development frameworks include ReactNative, Flutter, Ionic, Xamarin, Kotlin Native.

The table below elaborates and compares some common mobile app development frameworks;

Feature	React Native	Flutter	Ionic	Xamarin	Kotlin Native
Language	JavaScript	Dart	Html, Css, JavaScript	C#	Kotlin
Performance	Very good	Excellent	Moderate	Excellent	Excellent(Native performance)
Cost & time to market	Cost-effective and rapid	Cost-effective with slightly longer development	Cost-effective and quick	Initial costs may be higher due to Visual Studio license	Moderate (requires native development expertise for each platform)
UX & UI	Native-like experience with customizable Ui components	Highly customizable UI components with smooth animations	Web-like UI with customization options	Native UI components, with a familiar user experience	Excellent (native UI for each platform)
Complexity	Moderate	Moderate	Moderate	Variable	High
Community Support	Large and supported by Facebook	Growing and supported by Google	Supportive with extensive documentation	Supportive, .NET ecosystem	Moderate
Use Cases	Simple to complex apps (crossplatform)	Simple to complex apps (crossplatform)	Simple to moderate complexity apps (cross-platform)	Feature-rich apps (cross-platform, good for .NET developers)	Performance-critical native apps

IV- MOBILE APP ARCHITECTURES AND DESIGN PATTERNS

Mobile app architectures and design patterns provide a structured approach to developing robust, maintainable, and scalable mobile applications.

V.1 Mobile App Architectures

Architecture defines the overall structure of mobile app, including how different components interact and communicate with each other. Following are some common architectures used in mobile app development;

1. Model – View – Controller (MVC) Architecture:

MVC is commonly used when designing simple applications as it's more readily modified than others and makes implementing changes to the app simple. It consists of three components: Model, View, and Controller. "Model" is subject to the app's business logic and manages the state of an application and handles data changes and manipulations. "View" manages UI elements by presenting data to users and managing user interactions. "Controller" mediates between view and model by processing incoming requests. Depending on an app's requirements, there may be one or more controllers.

This pattern enables a faster development process and offers multiple views for the model. The figure below illustrates MVC architecture

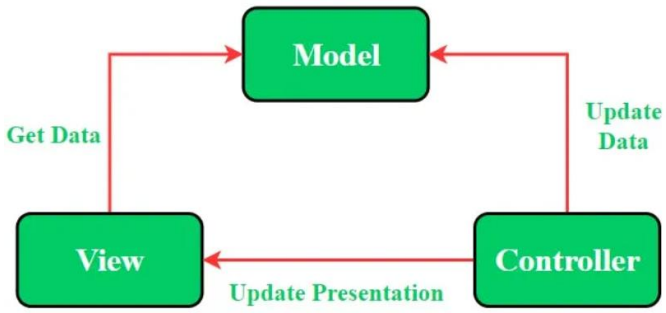

Figure 1: MVC Architecture illustration

2. Model – View – Presenter (MVP) Architecture:

This is derived from the MVC pattern, and here, the "controller" is replaced by "presenter". Performance-wise this pattern offers high reliability as there is less hindrance than with the other two models in terms of rendering frames. Similar to MVC, the model covers the app's business logic and how data is handled while the view is separated from the logic implemented in the process. The presenter's major function is to manipulate the model and update the view. When receiving input from a user via view, the presenter processes the data and sends the results back to view.

MVP offers easier debugging and allows code reusability.

The figure below illustrates MVP architecture functioning

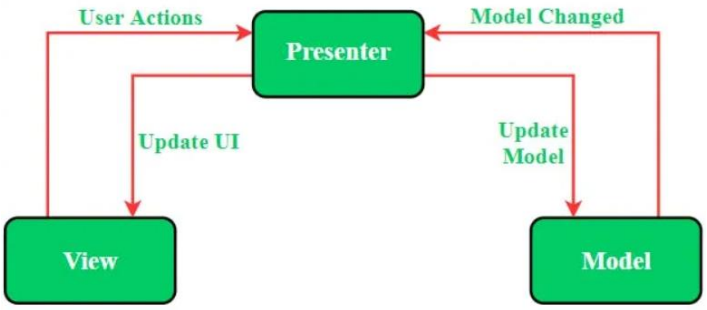

Figure 2: MVP Architecture Illustration

3. Model-View-ViewModel (MVVM) Architecture:

Designed for more explicit separation of UI development from business logic, MVVM is similar to MVC. The "model" here handles basic data and "view" displays the processed data. The View-Model element discloses methods and commands that help maintain the state of view and control the model. Two-way data binding synchronizes models and properties with the view. Due to data binding, this pattern has higher compatibility than others. MVVM's significant advantages include easier testing and maintenance as it allows developers to readily implement changes because the different kinds of code are separated.

The figure below illustrates MVVM architecture functioning

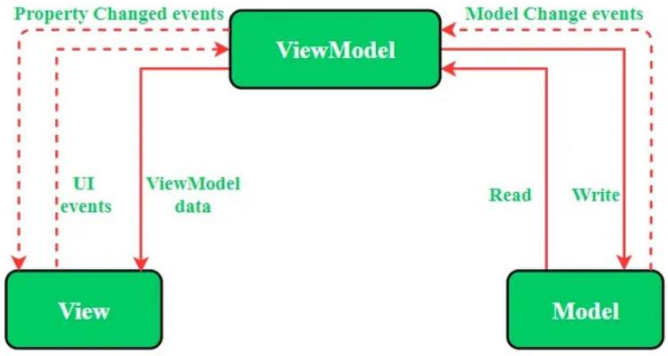

Figure 3: MVVM Architecture Illustration

4. **Microservice Architecture:**

Microservice architecture divides an application into smaller, loosely coupled services that communicate with each other over a network. While this is not unique to mobile application development, it is becoming increasingly relevant as mobile applications interact with backend services and APIs. Mobile applications can use different architectural patterns internally when interacting with server-side micro services.

The figure below illustrates Microservice architecture functioning.

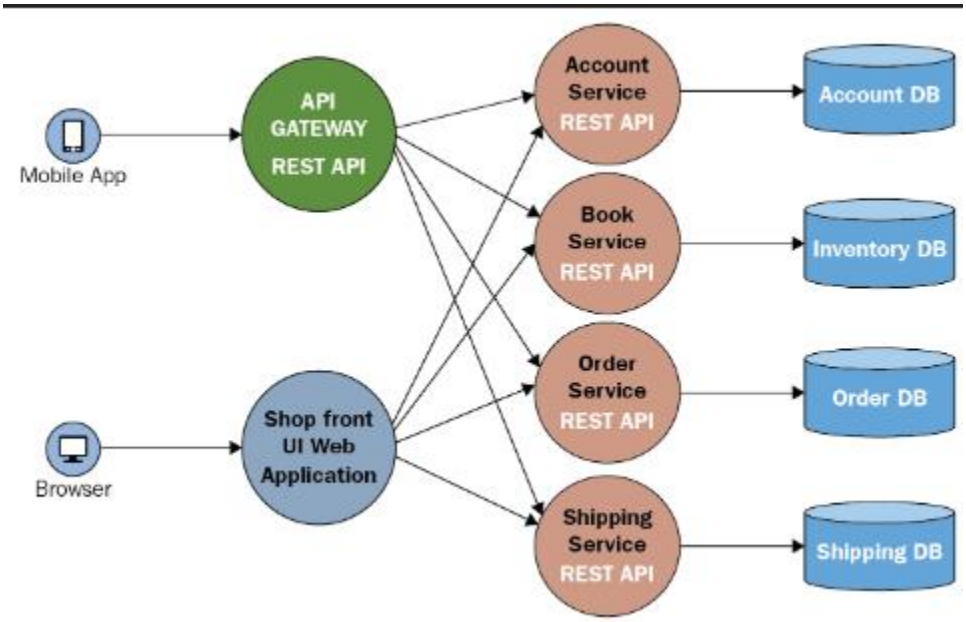

Figure 4: Microservice Architecture Illustration

Key Selection Principles for Good App architecture

A well-designed architecture should meet a set of specific conditions including:

- **Efficiency**: The application performs the tasks and functions under any condition and handles heavy loads effectively.

- **Portability**: Since frequent environmental (market or technical-specific) changes are natural in mobile applications, good architecture must allow adjustments without crashing the whole system.

- **Scalability:** Post-release updates and further development are a key part of the software development life cycle. Solid architecture results in faster changes and updates because it's developed in separate threads.

- **Testability:** If an app's architecture is easily tested it reduces the number of errors and increases reliability.

V.2 Mobile App Design Patterns

Design patterns are reusable solutions to common software development problems. They have had a significant impact on software development, including mobile app development. Following are some design patterns used in mobile app development;

1. **Strategy Method Design Pattern:** It defines a family of algorithms, contains each of them, and provides them with flexibility. It allows you to select the appropriate algorithm at runtime. This example is useful when you want to provide different options for a task.

2. **Composite Method Design Pattern:** A composite pattern allows you to arrange objects in a tree structure to represent a part-of-the-whole structure. This is helpful when you have to deal with individual objects and sets of objects accurately.

3. **Observer Method Design Pattern:** The observer structure defines one to many dependencies between objects, so when one object changes its state, all its dependents are automatically notified and updated. This is useful for scheduling distributed events.

4. **Adapter Method Design Pattern:** The adapter configuration allows you to use the interface of an existing class as a link to a new one. It is often used to work with others without modifying the source code of existing classes.

5. **Factory Method Design Pattern**: The Factory Method model defines an interface for creating an object but allows subclasses to modify the type of the created object. Especially useful when you need to create objects with a common interface but different functionality.

V- REQUIREMENTS ENGINEERING PROCESS

Requirements engineering refers to the process of defining, documenting and maintaining the requirements in the engineering process. To guarantee the effective creation of a software product, the requirements engineering process entails several tasks that help in understanding, recording, and managing the demands of stakeholder.

Following are steps involved in the requirement engineering process:

- Requirement Elicitation and Analysis
- Software Requirement Specification
- Software Requirement Validation
- Software Requirement Management

1. Requirement Elicitation and Analysis

Requirements elicitation is the process of gathering information about the needs and expectations of stakeholders for a software system. This is the first step in the requirements engineering process and it is critical to the success of the software development project. The goal of this step is to understand the problem that the software system is intended to solve and the needs and expectations of the stakeholders who will use the system.

Several techniques can be used to elicit requirements, including:

- ***Interviews:*** These are one-on-one conversations with stakeholders to gather information about their needs and expectations
- ***Surveys:*** These are questionnaires that are distributed to stakeholders to gather information about their needs and expectations.
- ***Focus Groups:*** These are small groups of stakeholders who are brought together to discuss their needs and expectations for the software system.
- ***Observation:*** This technique involves observing the stakeholders in their work environment to gather information about their needs and expectations.
- ***Prototyping:*** This technique involves creating a working model of the software system, which can be used to gather feedback from stakeholders and to validate requirements.

It's important to document, organize, and prioritize the requirements obtained from all these techniques to ensure that they are complete, consistent, and accurate.

2. Software Requirement Specification

Requirements specification is the process of documenting the requirements identified in the analysis step in a clear, consistent, and unambiguous manner. This activity is used to produce formal software requirement models. All the requirements including the **functional** as well as the **non-functional requirements** and the constraints are fully specified by these models. During specification, more knowledge about the problem may be required which can again trigger the

elicitation process. This step also involves *prioritizing* and *grouping* the requirements into manageable chunks.

The models used at this stage include *ER diagrams, data flow diagrams(DFDs), function decomposition diagrams(FDDs), data dictionaries*, etc.

Following are some of the types of requirements commonly specified in this step;

- ***Functional requirements:*** These describe what the software system should do. They specify the functionality that the system must provide, such as input validation, data storage, and user interface.

- ***Non-Functional requirements:*** These describe how well the software system should do it. They specify the quality attributes of the system, such as performance, reliability, usability, and security.

- ***Constraints:*** These describe any limitations or restrictions that must be considered when developing the software system.

- ***Acceptance Criteria:*** These describe the conditions that must be met for the software system to be considered complete and ready for release.

To make the requirements specification clear, the requirements should be written in a natural language and use simple terms, avoiding technical jargon, and using a consistent format throughout the document. It is also important to use diagrams, models, and other visual aids to help communicate the requirements effectively.

3. Requirements Verification and Validation

Verification refers to the set of tasks that ensures that the software correctly implements a specific function. It involves reviewing the requirements to ensure that they are clear, testable, and free of errors and inconsistencies. This can include reviewing the requirements document, models, and diagrams, and holding meetings and walkthroughs with stakeholders.

Validation refers to a different set of tasks that ensures that the software that has been built is traceable to customer requirements. It involves testing the requirements to ensure that they are valid and that the software system being developed will meet the needs of the stakeholders. This can include testing the software system through simulation, testing with prototypes, and testing with the final version of the software.

Requirements verification and validation is the process of checking that the requirements for a software system are complete, consistent, and accurate and that they meet the needs and expectations of the stakeholders. The goal here is to ensure that the software system being developed meets the requirements and that it is developed on time, within budget, and to the

required quality. It should be noted that verification and validation is an iterative process that occurs throughout the software development life cycle

4. Requirements Management:

Requirements management is the process of managing the requirements throughout the software development life cycle, including tracking and controlling changes, and ensuring that the requirements are still valid and relevant. The goal of requirements management is to ensure that the software system being developed meets the needs and expectations of the stakeholders and that it is developed on time, within budget, and to the required quality.

The key activities are involved in the requirements management include:

- ***Tracking and controlling changes:*** This involves monitoring and controlling changes to the requirements throughout the development process, including identifying the source of the change, assessing the impact of the change, and approving or rejecting the change.

- ***Version Control:*** This involves keeping track of different versions of the requirements document and other related artifacts.

- ***Traceability:*** This involves linking the requirements to other elements of the development process, such as design, testing, and validation.

- ***Communication:*** This involves ensuring that the requirements are communicated effectively to all stakeholders and that any changes or issues are addressed promptly.

- ***Monitoring and Reporting:*** This involves monitoring the progress of the development process and reporting on the status of the requirements.

VI- MOBILE APP COST EVALUATION

Accurately estimating the cost of developing your mobile app is crucial for budgeting and planning. Here's a roadmap to understand the key factors and pricing considerations for each development stage:

Factors Influencing Mobile App Development Cost

- ***App Complexity:*** Simpler apps with basic features will cost less than feature-rich apps with complex functionalities (e.g., real-time chat, AR/VR integration, AI integration).
- ***Platform (Android or iOS):*** Developing for both platforms typically doubles the cost compared to a single platform.
- ***Development Team Location:*** Hourly rates for developers vary depending on location.
- ***Development Approach:*** Native app development (using platform-specific languages) is often more expensive than cross-platform development (using frameworks like React Native or Flutter) due to the need for separate codebases.
- ***Features and Functionality:*** The number and complexity of features directly impacts development time and cost. Features like push notifications, in-app purchases, and social media integrations require additional development effort.
- ***UI/UX Design:*** A polished and user-friendly interface adds value but incurs design costs

Steps Involved in Mobile App Development (Estimated Ranges)

1. **Discovery & Planning (10-20% of total cost):**
   - ***Business Analysis & User Research****:* Understanding your target audience and defining app goals.
   - ***Feasibility Study & Competitive Analysis****:* Assessing technical feasibility and market competition.
   - ***Project Scope Definition & User Stories*****:** Defining app functionalities and user flows.
2. **Design (15-20% of total cost):**
   - ***Wireframing & Prototyping*****:** Creating low-fidelity mockups to visualize app flow and functionality.

- ***UI/UX Design:*** Designing the app's visual elements and user interface, optimizing the user experience for usability and intuitive interaction.
- ***Database Design:*** Designing the application database

3. **Development (40-50% of total cost):**
   - ***Native App Development:*** Building the app for a specific platform (Android or iOS). Costs vary depending on developer experience and location.
   - ***Cross-Platform Development:*** Developing the app using frameworks that work on both platforms. Can be faster and more cost-effective for simpler apps.
   - ***API Integration:*** Connecting the app to external data sources or services. Complexity of APIs affects cost.

4. **Testing & Quality Assurance (10-15% of total cost):**
   - ***Functional Testing:*** Ensuring all features work as intended.
   - ***Performance Testing:*** Evaluating app performance on different devices and network conditions.
   - ***Usability Testing:*** Observing users i**nteract with the app to identify** and fix usability issues.

5. **Deployment & Maintenance (5-10% of total cost):**
   - ***App Store Submission Fees:*** Fees for publishing the app on the App Store (iOS) or Google Play Store (Android).
   - ***Server Maintenance:*** Costs associated with ongoing server maintenance if the app relies on backend infrastructure.
   - ***App Maintenance & Updates:*** Budget for ongoing bug fixes, feature updates, and compatibility updates for new OS versions.

Important notes:

- These are estimated price ranges and can vary significantly depending on your specific project requirements.
- Getting quotes from multiple mobile app development companies is crucial for accurate cost comparisons.
- Consider ongoing maintenance costs when calculating your total budget.

REFERENCES

[1] https://www.outsystems.com/tech-hub/app-dev/what-is-mobile-app-development/#what-is-mobile-app-development (Visited on 28/03/2024)

[2] https://www.pcloudy.com/blogs/types-of-mobile-apps-native-hybrid-web-and-progressive-web-apps/ (Visited on 28/03/2024)

[3] https://madappgang.com/blog/mobile-app-architecture-everything-you-need-to-know/ (Visited on 28/03/2024)

[4] https://www.geeksforgeeks.org/design-patterns-for-mobile-development/ (Visited on 28/03/2024)

[5] https://clevertap.com/blog/types-of-mobile-apps/ (Visited on 30/03/2024)

[6] https://www.geeksforgeeks.org/software-engineering-requirements-engineering-process/ (Visited on 31/03/2024)