**MARCH 2024**

**UNIVERSITY OF BUEA**

**FACULTY OF ENGINEERING AND TECHNOLOGY (FET)**

**DEPARTMENT OF COMPUTER ENGINEERING**

**COURSE: INTERNET PROGRAMMING AND MOBILE PROGRAMMING (CEF440)**

**REPUBLIQUE DU CAMEROUN**

***PAIX – TRAVAIL – PATRIE***

**MINISTERE DE L’ENSEIGNEMENT SUPERIEUR**

**REPUBLIC OF CAMEROON**

***PEACE – WORK – FATHERLAND***

**MINISTRY OF HIGHER EDUCATION**



**PRESENTED BY : GROUP 19**

|  |  |  |
| --- | --- | --- |
| **NAME** | **MATRICULE** | **SPECIALTY** |
| NKEMCHOU PIANKE OLIVIER | FE21A275 | SE |
| DJITUE TOGUE BRINDA SPACHELLE | FE21A171 | SE |
| TEGUE NONO MIKEL MODEIRO | FE21A321 | SE |
| REOUTADE ROLAND | FE21A301 | SE |
| AMARACHUKWU GODLOVE AHANONU | FE21A136 | SE |

TASK 1

**INSTRUCTOR : Dr. NKEMENI VALERY**

Table of content

[INTRODUCTION 2](#_Toc162862953)

[1.1 TYPES OF MOBILE APPs 2](#_Toc162862954)

[1.1.1 Progressive Web Applications (PWAs) 2](#_Toc162862955)

[1.1.2 Native Applications 3](#_Toc162862956)

[1.1.3 Hybrid Applications 5](#_Toc162862957)

[1.2 COMPARISON BETWEEN DIFFERENT TYPES OF MOBILE APPS 6](#_Toc162862958)

[2.0 MOBILE APPLICATION PROGRAMMING LANGUAGES 9](#_Toc162862959)

[3.0 MOBILE APP DEVELOPMENT FRAMEWORKS 10](#_Toc162862960)

[4.0 MOBILE ARCHITECTURE AND DESIGN PATTERNS 13](#_Toc162862961)

[4.1 MOBILE APPLICATION ARCHITECTUTRE 13](#_Toc162862962)

[4.1.1 Model View Controller (MVC) Architecture 13](#_Toc162862963)

[4.1.2 Model View Presenter (MVP) Architecture 15](#_Toc162862964)

[4.1.3 Mobile View ViewModel Architecture 15](#_Toc162862965)

[4.1.4 VIPER Architecture 16](#_Toc162862966)

[4.2 DESIGN PATTERNS 17](#_Toc162862967)

[4.2.1 Singleton Method Design Pattern 17](#_Toc162862968)

[4.2.2 Factory Method Design Pattern 17](#_Toc162862969)

[4.2.3 Observer Method Design Pattern 18](#_Toc162862970)

[4.2.4 Dependency Injection (DI) Method Design Pattern 18](#_Toc162862971)

[4.2.5 Adapter Method Design Pattern 18](#_Toc162862972)

[4.2.6 Strategy Method Design Pattern 18](#_Toc162862973)

[4.2.7 Composite Method Design Pattern 19](#_Toc162862974)

[5.0 REQUIREMENT ENGINEERING 19](#_Toc162862975)

[5.1 Mobile App Requirement Types 19](#_Toc162862976)

[6.0 HOW TO ESTIMATE AN APP DEVELOPMENT COST 25](#_Toc162862977)

[6.1 FACTORS AFFECTING MOBILE APP DEVELOPMENT COST 26](#_Toc162862978)

[6.2 ESTIMATION METHODS 26](#_Toc162862979)

[REFERENCE 27](#_Toc162862980)

# INTRODUCTION

Anyone who has a smartphone has already used mobile app (or mobile application). A mobile app is a software application developed specifically for use on small, wireless computing devices, such as [**smartphones**](https://www.techtarget.com/searchmobilecomputing/definition/smartphone)and[**tablets**](https://www.techtarget.com/searchmobilecomputing/definition/tablet-PC), rather than desktop or laptop computers. In today's digital age, mobile apps are an essential part of most people's daily lives. From [social networking](https://www.techtarget.com/whatis/definition/social-networking) and entertainment to productivity and business, mobile apps play a vital role in how we interact with technology.

**What is mobile application development?**

Mobile application development is the set of processes and procedures involved in writing software for small, wireless computing devices, such as smartphones and other hand-held devices. Hence, the mobile development process involves creating installable software bundles (code, binaries, assets, just to name a few.), implementing backend services such as data access with an API, and testing the application on target devices.

There are two dominant platforms in the modern smartphone market. One is the iOS platform from Apple Inc. The iOS platform is the operating system that powers Apple's popular line of iPhone smartphones. The second is Android from Google. The Android operating system is used not only by Google devices but also by many other OEMs to build their own smartphones and other smart devices.

Although there are some similarities between these two platforms when building applications, developing for iOS vs. developing for Android involves using different software development kits (SDKs) and different development toolchain. While Apple uses iOS exclusively for its own devices, Google makes Android available to other companies provided they meet specific requirements such as including certain Google applications on the devices they ship.

# TYPES OF MOBILE APPs

In the early years of mobile apps, the only way to ensure an app could perform optimally on any device was to develop the app natively. This meant that new code had to be written specifically for each device's specific systems. Today, the majority of mobile applications developed are device-agnostic meaning can adapt to multi systems.

Before developing an app, you need to determine which type you will be creating. Here is a breakdown of several types of mobile app with information about each.

# Progressive Web Applications (PWAs)

A progressive Web Application is one that uses modern web capabilities to deliver a native app-like user experience. PWAs are built using standard web technologies like HTML, CSS, and JavaScript. Unlike traditional web apps, PWAs take advantage of features like service workers and web app manifests to provide additional benefits like offline access, push notifications, and an icon on the user’s home screen. Example of PWA: Forbes, Tinder

**Key Architecture Enabler**

* **Manifest File:**

All manifest files are JSON files, supporting how PWA is treated as an installed application, including the look and feel, basic behavior within the operating system.

* **Service Worker:**

These are scripts that run separately from the website and are used to handle network request and caching. It enables PWA to deal with poor network connectivity and helps to increase the performance of the application. It also enables them to work offline and handle push notifications.

**Advantages of PWA**

* Reduced business cost. (This cuts down on development costs.)
* No installation needed. (Because it’s web-based, there is no need to customize to a platform or OS)
* Better reach as it can be accessed from anywhere.
* Always up-to-date. (Just push the update live over the web. Users don’t need to download the update at the app store/play store).

**Disadvantages of PWA**

* Web apps fail to work when you are offline. (Because they’re shells for websites, they won’t completely work offline)
* Limited number of functionalities as compared to Native apps. (There will be functionalities available within one browser and not available on another, possibly giving users varying experiences.)
* It takes a longer time to develop.
* Security risk. (Even if they have an offline mode, the device will still need an internet connection to back up the data on your device, offer up any new data, or refresh what’s on screen.)

# Native Applications

Native application, also known as a native app, is a software application constructed to operate on a specific platform or device, like iOS or Android. Native apps are coded using platform-specific programming languages. For example, Android apps are typically built using Java or Kotlin, while iOS apps are created with Swift or Objective-C. Applications built for Windows phones employ C# as their programming language.

Upon downloading, native apps reside on your device and can be accessed via icons displayed on the device’s home screen. These apps utilize the device’s hardware and functionality, like the camera, contact list, GPS, and Bluetooth, to their full potential, given their platform-specific nature.

The majority of the applications found on smartphone are native apps, downloadable from the respective app stores – Apple’s App Store for iOS apps and Google Play for Android apps.

Native apps are developed to operate on the device’s operating system, requiring complete access to the device’s hardware and features. The platforms provide app developers with standardized SDKs (software development kits) which are bundles of tools, libraries, guides, and code samples that facilitate the development of apps on a specific platform.

Powerful integrated development environments (IDEs), like XCode for Apple and Android Studio for Google, are used for creating native apps. These IDEs encompass a code editor, compiler, and debugger, streamlining the development process by handling the most challenging bugs and reducing the development time.

**Advantages of native applications**

* Fast performance due to simple code specific to device and OS. (Because of their singular focus, native apps have the advantage of being faster and more reliable in terms of performance).
* Better use of OS and device specific functionalities.
* Interactive UI/UX. (Native apps utilize the native device UI, giving users a more [optimized customer experience](https://clevertap.com/blog/customer-experience-optimization/)).
* Lesser compatibility issues and faster to configure.

**Disadvantages of native applications**

* Building OS specific apps can be time-consuming (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).
* OS specific programming languages like swift and java are hard to learn.
* Requires separate codebase to add new features. (Every time there’s an update to the app, the user has to download the new file and reinstall it.)
* It takes precious space in the device’s storage.

# Hybrid Applications

Hybrid apps are a combination of native and web technologies, which allows them to have the best of both worlds. Hybrid apps are coded in HTML5, CSS3, and JavaScript and run on a web view. They can be deployed to the App Store or Google Play.

Hybrid apps provide a wide range of benefits: they are easy to develop and maintain because they use web technologies; they don’t require compiling; they can be integrated with an existing website; they can be easily ported between platforms; they allow users to access offline functionality; they support multiple languages by using JSON files.

Example of hybrid app include: Gmail

**Advantages of hybrid mobile applications**

* Easy to build. (Building a hybrid app is much quicker and more economical than a native app)
* Shareable code makes it cheaper than a native app. (a hybrid app can be the minimum viable product)
* Easy to push new features since it uses a single code base. (there is much less code to maintain.)
* Can work offline. (They also load rapidly, are ideal for usage in countries with slower internet connections, and give users a consistent user experience)
* Shorter time to market, as the app can be deployed for multiple OS.

**Disadvantages of Hybrid apps**

* Complex apps with many functions will slow down the app.
* More expensive than web apps
* Less interactive than native apps
* Apps cannot perform OS specific tasks (Hybrid apps might lack in power and speed,).

# 1.2 COMPARISON BETWEEN DIFFERENT TYPES OF MOBILE APPS

* **Performance**

Performance is a crucial factor when choosing between PWAs, native apps, and hybrid apps. This includes both speed and responsiveness.

PWAs can offer near-native performance thanks to features like service workers for caching assets and background sync for network requests. Well-optimized PWAs load fast even on slow networks. They are also responsive since they rely on web technologies like JavaScript.

Native apps offer the best performance since they are built specifically for the target platform. They have direct access to native APIs and are compiled into machine code. However, the performance depends on the skills of the developer. A poorly coded native app can still be slow and clunky.

Hybrid apps fall somewhere in between native and PWA. They use web technologies but are wrapped in a native container, giving them access to some native APIs for performance boosts. However, they still cannot fully match the speed and fluidity of well-optimized native apps, especially for complex visuals and animations. The performance can also deteriorate over time as the app grows.

* **Development**

Developing each type of app comes with its own set of advantages and disadvantages from a developer’s perspective.

PWAs can be built using standard web development skills like HTML, CSS and JavaScript. The service worker and manifest file do add some additional configuration but overall PWAs are fast and easy for web developers to build. The main benefits of developing PWAs include:

* Leverage existing web development skills and avoid learning new languages
* Quick iterations since there is no compilation step
* Seamless integration with web analytics and other web-based tools
* Can be hosted and deployed just like a website

The downside is that PWAs do not have access to all native device capabilities out-of-the-box, so for advanced features, additional plugins or libraries may be needed.

Developing native apps requires learning platform-specific languages like Swift/Objective-C for iOS or Java/Kotlin for Android. However, native apps have full access to device capabilities and APIs for the richest user experience. Other benefits include:

* Maximize performance through direct access to native APIs
* Tighter integration with OS features like notifications or widgets
* Distribution through app stores for discoverability and updates

The cons are slower development speed due to compilation and more effort to build for multiple platforms.

Hybrid apps use web development skills combined with a native wrapper framework like React Native, Flutter, or Ionic. This allows access to some native features while reusing existing web skills. Benefits of the hybrid approach include:

* Faster cross-platform development using web skills
* Partial access to native APIs through the framework
* Ability to deploy as a website or native app package

The downsides are that performance may not be as fast as pure native and there are some device integration limitations. The framework also adds its own learning curve.

Overall, it depends on the specific app requirements and team skills when choosing the best development approach. PWAs offer the fastest web development while native maximizes performance and customization. Hybrid provides a middle ground for teams with web skills.

* **Accessibility**

Accessibility refers to how easily people with disabilities can use an app. This is an important consideration for any app development approach.

* PWAs have good accessibility support across modern browsers. They utilize web standards like WAI-ARIA for accessible components. However, PWAs may have limited accessibility on some older platforms.
* Native apps have the best accessibility support on their target platform. iOS and Android provide robust accessibility APIs to support screen readers, text-to-speech, captions, and more. However, a native app only targets one platform.
* Hybrid apps also have good accessibility support on mobile when built properly with web standards. The embedded web content can interact with native APIs for accessibility. But support will vary based on review implementation.

Overall, native apps tend to provide the most robust accessibility support, followed by hybrid apps and PWAs. However, PWAs have the advantage of a web codebase that can be accessible across multiple platforms. So there is no clear winner — it depends on the target platforms and user needs. Developers should aim for high accessibility regardless of approach.

* **Maintenance**

Maintenance needs vary greatly between PWAs, native apps, and hybrid apps. PWAs require very little maintenance since the app codebase lives on a server. As long as the server is maintained, the PWA will be updated automatically for all users when they load the app. Some additional maintenance may be required for the service worker and cache, but overall maintenance needs are minimal.

Native apps require releasing new versions of the app in the relevant app store(s) whenever new features or fixes need to be deployed. This makes native app maintenance more complicated than PWAs. Automated testing and CI/CD pipelines help streamline some of these processes.

Hybrid apps require app store deployments like native apps. But they also require maintenance of the web codebase that renders within the native wrapper. This makes maintenance more complicated than native or PWA. Any changes to UI, business logic, or data require repackaging and redeploying the hybrid app. Developers must also ensure the web and native layers remain in sync through ongoing maintenance.

So in summary, PWAs have the lowest maintenance overhead, followed by native apps. Hybrid apps require the most involved maintenance processes.

* **Security**

Native and hybrid apps have the advantage when it comes to security compared to PWAs. Native apps are written specifically for the platform they are built for, allowing access to built-in security features provided by the OS and hardware. For example, iOS apps can take advantage of Apple’s App Transport Security feature which enforces secure connections. Android apps can integrate with the Key store system to securely store cryptographic keys.

Hybrid apps also have access to native security features as they are essentially native apps running embedded web views. The native container handles permissions, certificates, and encryption while the web code accesses APIs through a bridge.

PWAs on the other hand rely on the security model of the browser, which sandboxes web resources to prevent access outside of it. Without native integration, PWAs cannot access native encryption, key stores, trusted hardware, just to name a few.

# MOBILE APPLICATION PROGRAMMING LANGUAGES

All mobile app development languages have different characteristics, strengths, and weaknesses that can significantly impact the development process and the app’s outcome. As such, there’s no single best language for mobile app development. A well-chosen programming language can facilitate efficient coding, [optimize app performance](https://www.taazaa.com/mobile-application-performance/), streamline maintenance, and enable seamless integration with existing systems.

* **JavaScript:** A versatile, interpreted language primarily used for web development. In mobile development, it is popular for cross-platform apps through frameworks like React Native, Ionic, and PhoneGap.
* **Java:** A mature, general-purpose language that has been the foundation of Android app development for a long time. It is known for its large ecosystem, libraries, and developer community.
* **Swift:** A modern, efficient language developed by Apple specifically for building high-performance native apps for iOS, iPadOS, macOS, tvOS, and watchOS. It prioritizes readability and safety features.
* **Kotlin:** A relatively new, concise, and readable language gaining significant traction in Android development. It offers features like null safety and seamless integration with existing Java codebases. It is officially supported by Google.
* **Objective-c**: Objective-C was the primary programming language used for developing applications for Apple's macOS, iOS, and other platforms before the rise of Swift. Many existing macOS and iOS applications are still written in Objective-C.
* **Dart:** The primary language used in the Flutter framework for building beautiful, fast cross-platform mobile apps. It features hot reload for quicker development cycles and can be compiled ahead-of-time for performance close to native apps.
* **C#:** A powerful language developed by Microsoft, primarily used for building Windows, Xbox, and Windows Phone apps (.NET development). It integrates well with Microsoft's development tools and is popular for game development using the Unity engine.
* **Python:** Not typically used for native mobile app development, but a valuable tool for building mobile app backend logic, APIs, or data science/machine learning components that power mobile apps.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Programming**  **language** | platform | Operating  system | Level of difficulty | Framework |
| JavasSript | Hybrid app  Web apps | Android and IOS | Easy to learn | React native |
| Kotlin | Native app | Android | Easy to learn | Native android SDK |
| Java | Native app | Android | Easy to learn |  |
| Swift | Native app | IOS | Easy to learn | Native IOs |
| Objective-C | Native app | IOS | Easy to learn | - |
| C# | Hybrid apps | Android and IOS | intermediate | Xamarin |
| Dart | Hybrid apps | Android and IOS | Easy to learn | Flutter |
| Python | Hybrid apps | Android and IOS | One of the easiest programming language | Kivy |

# MOBILE APP DEVELOPMENT FRAMEWORKS

A mobile development framework is a software design to facilitate the creation of mobile applications, by providing developers with a set of tools, libraries and reusable components that streamline the process of building testing and deploying mobile applications for various platforms such as IOS, Android and sometimes cross-platform solutions. Mobile app frameworks include:

**Native iOS (Swift)**

This is a programming language developed by Apple and it is used to build iOS apps build time is short, an extensive selection of debuffing tools in a dedicated XCTest testing framework as well as small size of apps and super-fast performance, both of which are common traits for all native development frameworks.

Apps build with swift are Uber, VSCO cam.

**Native Android SDK (Kotlin)**

It was first introduced by google in 2009 and was updated in 2017 and became the main language for building native Android apps. Android SDK allow developers to write lower-level-code, which make it easier to control the environment, build complex features and debug apps. Apps built with native frameworks do not require an interlayer of code to access device hardware such as camera and microphone.

**React Native:**

Developed and maintained by Facebook, react native is an open-source, cross-platform app development framework that has rapidly become the developer's top choice. It helps to develop Android and iOS mobile apps. Mobile apps of top brands such as Tesla, Airbnb, Skype, and Amazon Prime are the best examples of React Native apps

**Apache cordova**

Formally known as PhoneGap, AC is a developer-friendly mobile app development framework. It is a cross-platform app development framework that leverages CSS3, HTML5 and JavaScript to build mobile applications. Cordova plugins enables the developers to access device hardware capabilities such as GPS, Camera and accelerometer to deliver a native-like app experience features are fast development process with a single codebase and third-party app management tools.

**Flutter**

This is a Google’s open-source framework for developing native Android and iOS apps using a single codebase. It is revolutionary software Development Kit (SDK) for cross-platform app development that stands out as it uses a unique approach for delivering apps with a native-like look and feel.

Examples of applications built using Flutter development include Google, Alibaba, and Abbey Road Studios.

**Ionic**

This is an open-source framework built with Apache Cordova (Phone Gap) and Angular, allowing developers to build Android and iOS apps

The framework helps the developers build robust and feature-rich native applications.

An advantage of this framework is that it allows the developers to use several UI elements

Such as filters, forms, views; navigation menus and action sheets, in the application design.

This is a complete framework that enables developers to build Progressive Web Apps (PWAs) and hybrid and cross-platform mobile applications.

**Xamarin**

It is a Microsoft-owned open-source framework for developing native and high performance Android, iOS, macOS, tvOS and watchOS apps leveraging.

The framework empowers business to provide native performance and unique user experiences to end-users. Allows code sharing on multi-platforms in xamarin, an abstraction layer controls the communication between the underlying platform and shared code. The framework allows the developers to build an ecosystem with APIs, back-end, components.

The differences between these frameworks are summarized on the table below:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Features** | **Native IOS** | **Native Android** | **React Native** | **Cordova** | **Flutter** | **Ionic** | **Xamarin** |
| Language | Swift | kotlin | JavaScript | JavaScript | Dart | JavaScript | C# |
| Performance | Very high | Very high | high | low | high | low | average |
| Cost and time to market | More expensive and slower | More expensive and slower | Cheaper and faster | Cheaper and faster | Cheaper and faster | Cheaper and faster | Cheaper and faster |
| UI & UX | native | native | Adapts to platforms | Adapts to platforms | Adapts to platforms | Adapts to platforms | Adapts only partially |
| Complexity | high | high | moderate | moderate | moderate | low | moderate |
| Community support | Very popular | Very popular | Very popular | Not so popular | Popular | Not so popular | Not so popular |
| Where to use | Apple ecosystem only | Android only | iOS, Android, web | iOS, Android, web | iOS, Android, web | iOS, Android, web | iOS, Android, web |

# MOBILE ARCHITECTURE AND DESIGN PATTERNS

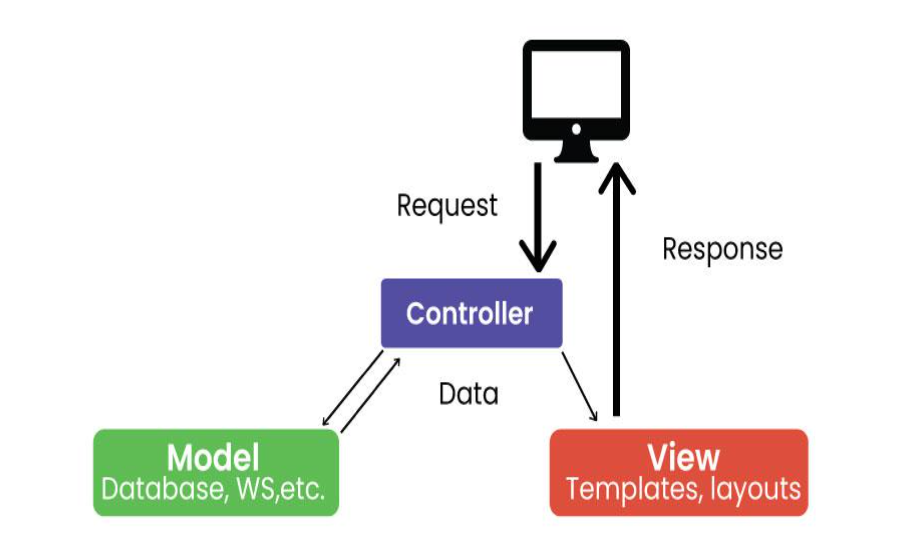
Mobile applications architecture and design patterns are fundamental principles that guides the organization and structure of mobile apps. They help developers build apps that are scalable, maintainable and efficient.

Design Patterns are an integral part of modern software development, playing a pivotal role in enhancing the user experience (UX). These patterns represent well-established solutions to common design and programming challenges, providing a blueprint for creating software that not only functions efficiently but also delivers a superior user interface.

# MOBILE APPLICATION ARCHITECTUTRE

# 4.1.1 Model View Controller (MVC) Architecture

MVC is a design model that separates an application into three interacting parts: Model, View, and Controller. This separation allows for better code design and modularization.



**Model**: represents application data and business logic, it is responsible for data handling. It contains the code for getting data from sources like a database, a microservice via API or a JSON object

**View**: displays data to the user

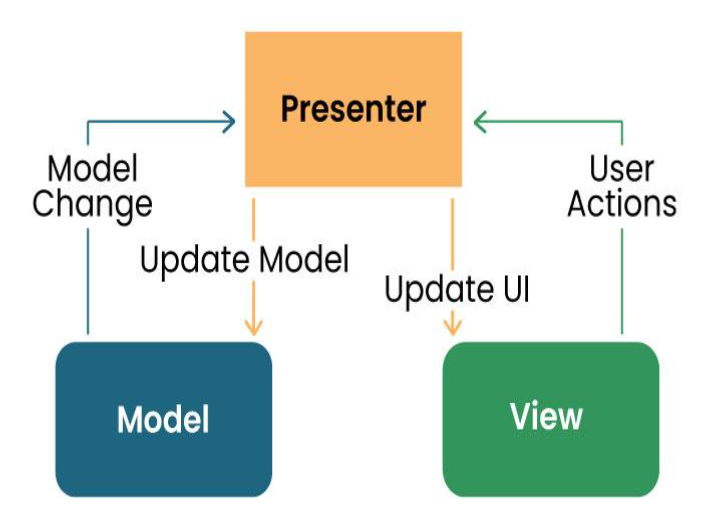
**Controller**: processes user inputs and controls data flow between model and view

For Example:

Imagine a mobile weather app. The model stores weather information, the View displays it to the user, and the controller handles user interactions such as updating the displayed location or converting units (e.g. from Celsius to Fahrenheit).

# Model View Presenter (MVP) Architecture

MVP is a new architecture that separates an application into three parts: Model, View, and Presenter. This is similar to MVC but puts more responsibility on the Teacher to manage the interaction between Model and View.



Model: Manages data and business logic.

Presenter: it contains the logic that processes data for the user.

View: Represents the user interface.

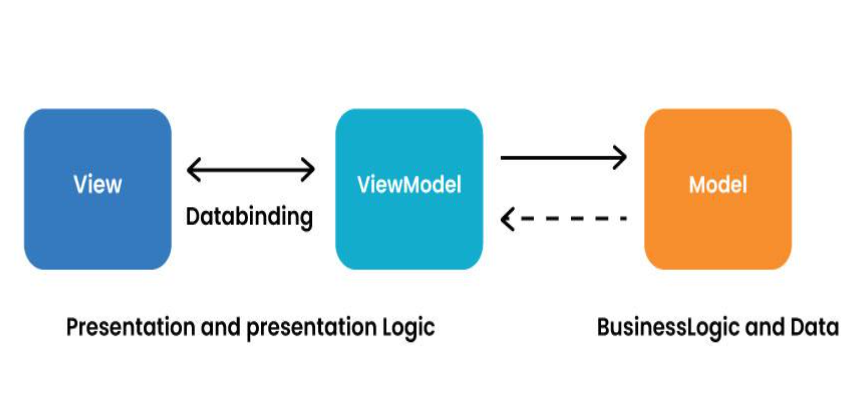
Designer: Acts as an intermediary processing user input and updating the View and Model.

For Example:

In a note-taking app, the Model would store the text, the View would display it, and the provider would handle user input such as typing, editing, or deleting the process.

# Mobile View ViewModel Architecture

MVVM is a design model widely used in mobile development, especially in frameworks like Android’s Jetpack. Its purpose is to separate the application into three parts: Model, View, and ViewModel.



**Model**: Represents data and business logic.

**View**: Represents the user interface.

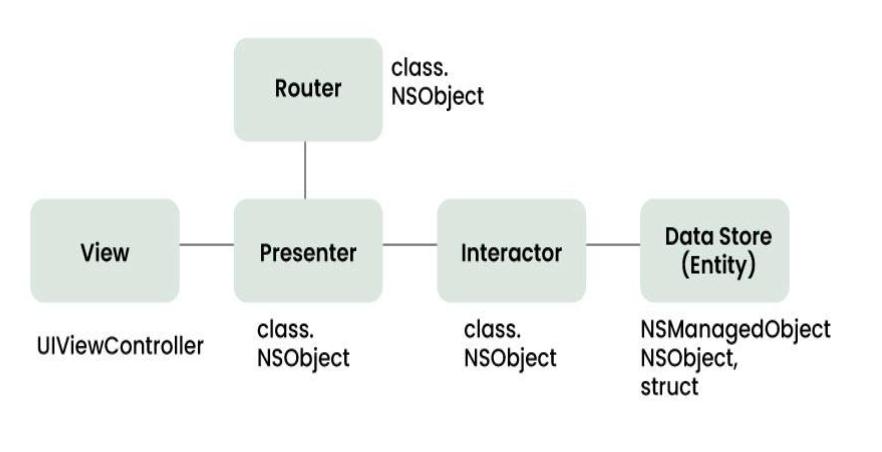
**ViewModel**: Acts as an interface between the Model and the View, which contains the reference logic.

For Example:

In an e-commerce application, the Model contains product data, the View displays product information, and the ViewModel manages interactions, such as adding items to a cart.

# VIPER Architecture

VIPER stands for View, Interactor, Presenter, Entity, and Router. VIPER is primarily based at the clean architecture ideas, which purpose to separate the concerns of different layers of the utility. Each layer has a single duty and communicates with different layers through properly-defined interfaces.



**View**: This is the consumer interface layer, wherein the perspectives and look at controllers are defined. The view is chargeable for showing the information provided by way of the presenter and forwarding the person moves to the presenter.

**Presenter**: This is the presentation layer, where the good judgment for formatting and imparting the records is defined. The presenter is liable for fetching the records from the interactor, reworking it right into a suitable layout for the view, and updating the view hence. The presenter additionally handles the consumer movements acquired from the view and calls the router to navigate to other screens.

**Interactor**: This is the enterprise good judgment layer, where the common sense for manipulating the data and interacting with external services is described. The interactor is accountable for gaining access to the facts from the service layer, acting any vital operations on it, and returning it to the presenter. The interactor additionally communicates with the entity layer to store and retrieve the information fashions.

**Entity**: This is the information layer, wherein the data models and systems are described. The entity is responsible for representing the data in a constant and coherent manner throughout the software. The entity layer also can encompass records get entry to gadgets (DAOs) or repositories that summary the information of records patience and retrieval.

**Router**: This is the navigation layer, where the logic for routing and transitioning among different monitors is defined. The router is chargeable for developing and providing the view controllers, passing any vital facts to them, and coping with any dependencies or configurations. The router also communicates with the presenter to get hold of the navigation requests and execute them.

# DESIGN PATTERNS

# Singleton Method Design Pattern

The singleton policy ensures that there is only one instance of a class and provides global access. This is especially useful when you want to manage a single instance of an object or control access to a delayed object.

For Example:

Singleton can be used to manage player’s score in a mobile game. There can only be one instance that is responsible for tracking scores and is updated throughout the game.

# 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.

For Example:

In mobile app that supports multiple payment gateways, payments can be made using the Factory Method. Each payment gateway (e.g., PayPal, Stripe) is a small business and provides its services.

# 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.

For Example: In the reports app, many features (Observers) such as the title widget, the report feed view, and the notification provider (Themes) can subscribe to updates when new information arrives. The observer model ensures that they are created all registered parts report, and accordingly it can be updated.

# Dependency Injection (DI) Method Design Pattern

Dependency Injection is a method of providing class dependencies from the outside, rather than creating them in the class. It improves code modularity and testability by making classes independent of their dependencies.

For Example:

In an Android app, instead of creating a single database connection object in the class, you can place the database object externally, allowing you to easily test and modify database operations.

# 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.

For Example: If you want to use a third-party library that provides data in a different way, you can create an adapter that will convert the library’s results to the format your app would expect and make sure that they are compatible meet without changing the library code.

# Strategy Method Design Pattern

The strategy model 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.

For Example:

In a weather application, you can use various methods to retrieve weather information, such as using a REST API, WebSocket, or local storage. The user can change these options, and the app adapts to his preferences.

# 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.

For Example:

You can use Composite pattern to create complex shapes from simple shapes in the mobile drawing app. Complex designs can contain individual designs, allowing users to manipulate and categorize resources as needed.

# REQUIREMENT ENGINEERING

Requirement engineering is the process of identifying and documenting the requirements for a software development project. It involves deriving requirements from goals and framework conditions, breaking them down into manageable units, and ensuring their transparency and verifiability. The ultimate goal is to satisfy customer demands by fulfilling a finite number of objectively verifiable criteria. Requirement engineering is crucial for the success of software engineering projects, as it helps in the selection of appropriate models and techniques, and ensures the generation of accurate and consistent requirements. It is a complex exercise that considers product demands from various viewpoints, roles, responsibilities, and objectives

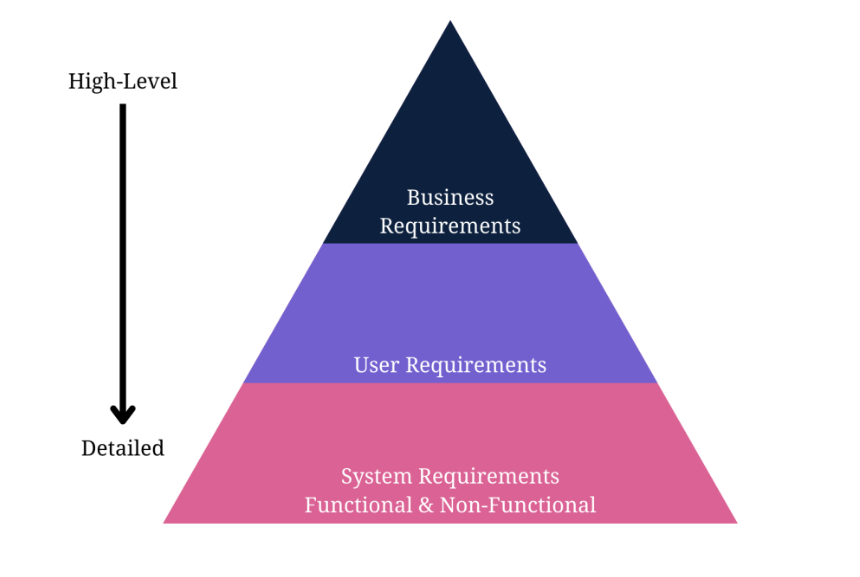
We often see that requirements-gathering for mobile or other applications has benefits, including:

* A better overall project development
* Improved project management throughout the software development process
* Enhanced efficiency to document the technical aspects required for an optimal product
* Higher chance of success in developing the desired features for the target user
* Cost optimization by removing unnecessary features

# 5.1 Mobile App Requirement Types

Mobile apps have different requirement types to collect, including:

* **Business requirements** are high-level requirements that ensure the app will align with business objectives, and the project’s scope, and identify the key stakeholders.
* **User requirements** are valuable insights into what your target audience needs and wants, how you can solve their problems, and what the audience experiences from your prototyping app.
* **Product or system requirements** are non-functional requirements and functional requirements that include technical requirements and technical specifications for the engineering team.



Steps to gather requirements for a mobile application to ensure we meet the business requirements and key features necessary to develop a profitable product are:

**Step 1:** Define Your App Idea and Purpose

Mobile development requirements-gathering starts with a business idea. The first information you need is the idea or purpose of the mobile app.

What purpose will it serve? Does it offer a solution to a potential problem?

You need to identify a problem the app will solve to recognize the idea or purpose behind it.

[Requirements gathering and management for mobile apps](https://www.requiment.com/requirements-gathering-and-management-for-mobile-apps/) require some effort with massive results.

**Step 2:** Gather and Align the App and Business Objectives or Goals

An app idea is fruitless without understanding business needs, business goals, and business rules.

Gathering business requirements to document involves:

* Identify the stakeholders for the right mobile application software development based on the business idea.
* Define clear and concise business goals and objectives to understand the project’s scope.
* Elicit stakeholder requirements and user requirements with elicitation techniques.
* Document the requirements in a business requirements document.
* Validate your requirements with stakeholders for a further transparent and opportunistic process.

**Step 3**: Run a Market Analysis and Competitor Analysis

Conduct a market or competitor analysis to truly understand the user’s perspective and design the appropriate user personas.

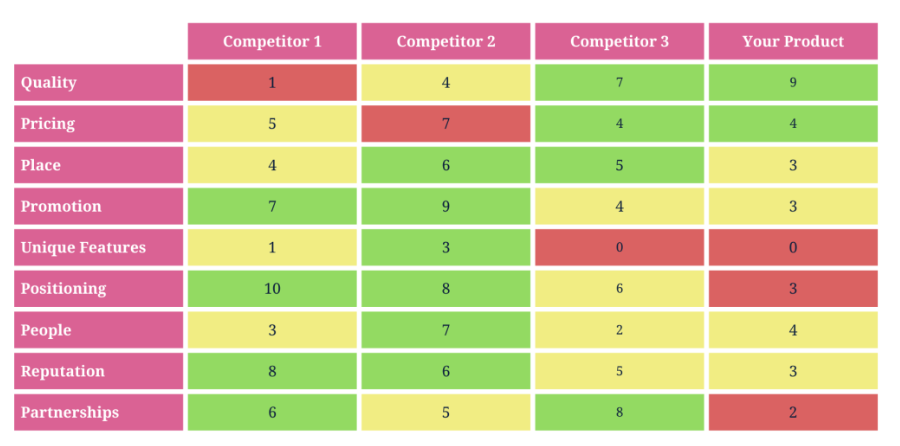
It also helps your team gather more user requirements for the development company.

The following steps explain the process of gathering user and competitor requirements:

* Identify the direct, indirect, secondary, and substitute competitors for the mobile app. Remember to recognize any businesses offering similar mobile app services or products and those offering different products in a broader niche umbrella.
* Gather competitor information, including products, descriptions, pricing structures, geographic reach, engaging promotions, target market positioning, business reputation, user profiles, and key partnerships to understand what your product needs to compete against.
* Use a SWOT analysis to determine your competitor’s strengths, weaknesses, opportunities, and threats. You could learn from another app’s mistakes to improve your requirements and identify possibly unique features other apps don’t provide.

The SWOT analysis in a table will help you see what the new product needs to do to compete better with the top market competitors.

Rank each competitor from 1-10 on each key element.



**Step 4**: Determine Scenarios and a User Persona

The next major step in mobile app requirements-gathering is to design user personas and scenarios to guide the requirements.

A user persona fictionalizes the target users for the mobile app.

It should describe the ideal person who uses the app, with some flexible aspects for alternate users.

The ultimate user person could include the following details about target users:

* Age (also, typical generational qualities)
* Behavioral considerations
* Gender (including non-binary if relevant to the product)
* Geographic location
* Goal or problem the app addresses or solves
* Goal quotes or principles
* Goal-related frustrations
* Motivation to use the app
* Range of hobbies and daily activities
* Typical occupation range

**Step** **5:** Gather and Prioritize Functional and Non-Functional Requirements

Here are some examples of functional mobile requirements:

* A complete description of a feature the app offers or software interfaces.
* How the app allows users to sign up, verify accounts, or subscribe to a newsletter.
* Buttons and dashboards users interact with to complete a specified task.
* External and internal interfaces users interact with on the app.
* The necessary administrative functions for different user classes.
* Transaction adjustment, correction, and cancellation functions.

Secondly, determine the non-functional requirements necessary to run your app.

Here are some examples of non-functional requirements in mobile development:

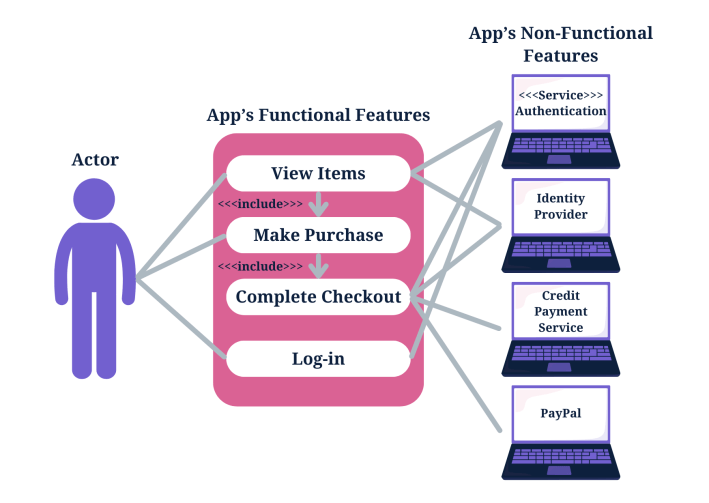
* How fast the app responds to user input.
* How the app protects user and business data.
* Whether the app can work on multiple platforms.
* How much data does the app store and is it scalable?
* How reliable and maintainable the app remains.
* Does the app comply with local laws and regulations?

**Step 6:** Design Use Cases

A mobile system requirements document won’t be complete unless you add use cases and stories.

**A use case** diagram lets everyone at the development company visualize an overview of how users will interact with the app. It’s an overview that includes actors, how actors interact with the app, and the sequence of interactions actors will deploy.

Here’s a simple example of a use-case diagram that shows an overview of how actors interact with functional features while non-functional features interact with the app from the back-end stack

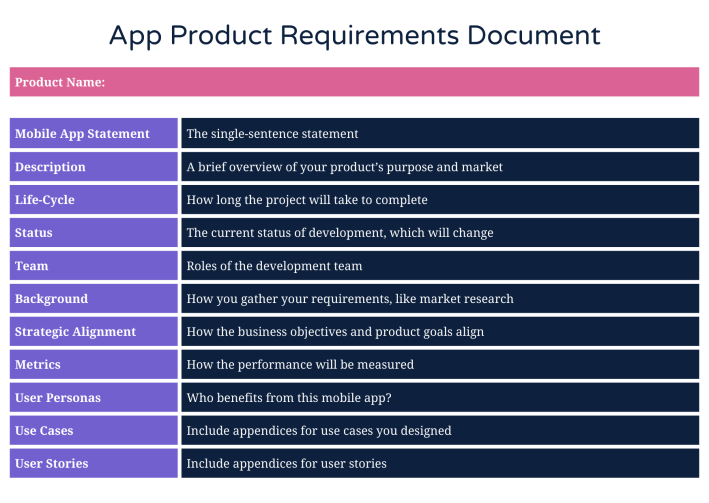


**Step 7:** Write an app requirement document

Delivering a proper app requirements document means you need to know how to write a mobile app requirements template.

The various steps for writing a mobile app requirement document can be listed below:

* Formulate the App’s Idea Statement
* Document All Relevant App Details
* Prepare a Navigation Sequence
* Add Requirements Formats for Visuals
* Add Cost Optimization Details
* Add Communication Protocols



**Step 8:** Deploy Prototyping and Wireframing

Prototyping and wireframing let you design the user flow of user interfaces and basic app functions.

It also lets you test and validate layouts and transitions between app pages.

Here are the steps to wireframe your requirements for an app, which you will then validate in the next step:

* Map the target user flow.
* Sketch the flow’s core part.
* Set a mobile wireframe.
* Determine the layout with boxes.
* Use design patterns.
* Add intended copy.
* Connect the app’s pages to design a flow.
* Design a prototype.
* Release the initial design to gather feedback.

**Step 9**: Validate the App Requirements

Validation is a quality control process you use before launching the final product based on your requirements.

The prototype app collects feedback from stakeholders, and you can invite stakeholders to verify that the app meets the documented requirements.

**Step 10**: Apply Agile Methodology

Agile methodology in requirements-gathering means you’ll always adapt the requirements document as per the feedback from stakeholders, testing, and initial product releases.

Agile methodology focuses on user experience and constant testing and validation to further improve your application.

**What Are Android Mobile App Requirements?**

An Android app requirements document will need the following additional technical details:

* APK file size
* App permissions
* App version
* Debug and test specifications
* Operating system and SDK version
* Restrictions

**What Are iOS Mobile App Requirements?**

An iOS app requirements document will need the following additional technical specifications:

* A business model
* Accurate meta descriptions for all app features
* Additional documents for UIKit, AppKit, WatchKit, iOS data storage, and app extensions
* An innovative and unique design sample
* Extensive safety features
* Legal and regulatory requirements list
* Performance metrics
* SDK versions and permissions
* User experience design

# HOW TO ESTIMATE AN APP DEVELOPMENT COST

Estimating the cost of mobile app development is a crucial step in the planning phase of any mobile app project. Several factors influence the overall cost, and understanding these factors can help businesses and developers make informed decisions. Below is a comprehensive study on how to estimate the cost of mobile app development.

# FACTORS AFFECTING MOBILE APP DEVELOPMENT COST

**Scope and Complexity of the App:** The number of features, functionalities, and complexity of the app significantly impact the development cost.

**Platform:** Developing for iOS, Android, or both (cross-platform) can influence the cost. Native apps generally cost more due to platform-specific development.

**Design:** Custom UI/UX design vs. using standard templates or themes can affect the cost. High-quality, custom designs usually increase the development cost.

**Development Approach:** Native, hybrid, or progressive web app. Each approach has its cost implications, with hybrid and progressive web apps generally being more cost-effective.

**Backend Development:** Setting up servers, integrating databases, developing APIs, and other backend services add to the development cost.

**Testing:** Quality assurance, bug fixing, and device testing are essential stages that contribute to the overall cost.

**Maintenance and Updates:** Ongoing costs for updates, bug fixes, and customer support should be considered for long-term budgeting**.**

**Geographical Location of the Development Team:** Rates vary based on the location of the development team, with developers in North America and Western Europe generally charging higher rates compared to developers in Asia or Eastern Europe.

# 6.2 ESTIMATION METHODS

* **Fixed Price**

**Description:** A fixed price is set based on the project scope, features, and requirements.

**Pros:** Predictable budget, clear scope of work, and timeline.

**Cons:** Less flexibility for changes and additions; may lead to scope creep.

* **Time and Material**

**Description:** Costs are calculated based on hourly rates multiplied by estimated development hours.

**Pros:** Flexibility to make changes during the development process; suitable for agile development**.**

**Cons:** Uncertain final cost, potential for exceeding the budget.

* **Value-Based Pricing**

**Description:** Pricing is based on the value provided to the client or business, considering factors like ROI, market demand, and competitive landscape.

**Pros:** Aligns with business goals and expected outcomes; potentially higher ROI.

**Cons:** Difficult to quantify value; requires a deep understanding of business objectives.

# REFERENCE

<https://typeset.io/questions/what-is-requirement-engineering-3zzlhq9j93>

<https://aws.amazon.com/mobile/mobile-application-development/>

<https://www.techtarget.com/whatis/definition/mobile-app>

<https://fireart.studio/blog/types-of-mobile-apps/>

https://www.requiment.com/requirements-gathering-and-management-for-mobile-apps/

https://www.youtube.com/watch?v=POih3P783a0

https://www.youtube.com/watch?v=1WWweyBaWZk

https://medium.com/@zorbisinc/pwa-vs-native-vs-hybrid-app-a-comprehensive-comparison-9b7fbe870d13

https://www.choicely.com/blog/how-to-estimate-your-total-app-development-costs-free-app-cost-calculator

https://www.geeksforgeeks.org/design-patterns-for-mobile-development/?ref=lbp