

**REPUBLIC OF CAMEROON**

**Peace-Work-Fatherland**

**MINISTER OF HIGHER EDUCATION**

**FACULTY OF ENGINEERING**

**AND TECHNOLOGY**

**REPUBLIQUE DU CAMEROON**

**PAIX-Travail-Patrie**

**MINISTRE DE L’ENSEIGNEMENT**

**SUPERIEUR**

**FACULTE D’INGINERIE**

**ET TECHGNOLOGIE**

**DEPARTMENT: Computer Engineering**

**SUB-DEPARTEMENT: Software Engineering**

**CEF 440: Internet Programming and Mobile programming**

**GROUP MEMBERS :**

|  |  |
| --- | --- |
| **Names** | **Matriculation** |
| **ALAIN NKEH MBUNKUH** | **FE21A133** |
| **ANANFACK ZANGO ANGELA CARINE** | **FE21A137** |
| **CHO WESLEY MUNGO** | **FE21A160** |
| **NKWETI MANGEM ANCELIA** | **FE21A280** |
| **NTUI RAOUL NTUI RAOUL** | **FE21A288** |

**COURSE INSTRUCTOR : Dr KAMENI Valery**

Table of Content

1. INTRODUCTION …………………………………………………………...3

* 1. Major types of Mobile Application………………………………….3
  2. Importance of Mobile Application…………………………………..4

2. Review and Compare mobile app Programming languages………………….6

3. Mobile Application Framework………………………………………………7

4. Study Mobile Application Architecture and Design Pattern………………….12

4.1. Mobile Application Architeccture …………………………………….13

4.1.1. Mobile App Architecture Principles…………………………..14

4.1.2. What to Consider When Developing Mobile App Architecture

Designs………………………………………………………..12

4.1.3. Why is App Architecture Essential?..........................................13

4.1.4. The Fundamental Layers of Mobile App Architecture……13

4.1.5. Type of Mobile Application Architecture…………………14

4.2. Design Pattern…………………………………………………. 18

4.2.1. Most Frequency Design Patterns in Mobile Development with

Examples and when to use it………………………………….18

**5**. Study how to collect and analyze user requirements for a mobile application ( Requirement Engineering)………………………………………………………….21

6. Study how to estimate mobile app development cost……………………………22

Table of Figures

Figure 1: Shows layer of mobile application architecture…………………………………14

Figure 2:Show Android Mobile Application Architecture flow and their layers………….16

Figure 3: Show iOS mobile application Mobile-View-Controller flow……………………17.

Figure 4:iOS mobile architecture application Model-View-View Model (MVVM)………17

Figure 5:Classic MVC flow diagram…………………………………………………….18

Figure 6: show MVVM flow diagram…………………………………………………….20

**INTRODUCTION**

A mobile app, short for mobile application, is a software application designed to run on mobile devices such as smartphones and tablets. It is typically downloaded and installed from an app store or accessed through a web browser. Mobile apps are developed to provide specific functionalities, services, or experiences to users on their mobile devices.

* 1. **Major Types Of Mobile Applications**

Mobile apps can be broadly classified into four major types: native apps, web apps, hybrid apps, and progressive web apps (PWAs). Each type has its own characteristics, development approach, and deployment method. Let's review and compare them:

1. **Native Apps:**
   * Native apps are built specifically for a particular mobile platform (e.g., iOS or Android) using platform-specific programming languages (e.g., Swift/Objective-C for iOS, Java/Kotlin for Android).
   * They have access to the device's native features and APIs, providing high performance and a rich user experience.
   * Native apps are typically distributed through app stores (e.g., Apple App Store, Google Play Store).
   * They can leverage platform-specific design guidelines and UI components, resulting in a consistent look and feel.
   * However, developing native apps requires separate codebases for each platform, which can increase development time and cost.
2. **Web Apps:**

* Web apps are essentially websites that are optimized for mobile devices and accessed through a web browser.
* They are built using web technologies such as HTML, CSS, and JavaScript and are not tied to a specific platform.
* Web apps have the advantage of being platform-independent, as they can run on any device with a compatible web browser.
* They do not require installation from an app store, making them easily accessible to users.
* However, web apps may not have access to all the device's features and APIs, resulting in limited functionality compared to native apps. They also rely on an internet connection to function.

1. **Hybrid Apps:**

* Hybrid apps combine elements of both native and web apps.
  + They are built using web technologies (HTML, CSS, and JavaScript) and then wrapped in a native shell, which allows them to be distributed through app stores.
  + Hybrid apps can access certain device features through plugins or frameworks, enabling them to have a native-like experience.
  + They offer a single codebase that can run on multiple platforms, reducing development time and effort.
  + However, hybrid apps may not provide the same level of performance and responsiveness as native apps, and the access to native features can be limited.

1. **Progressive Web Apps (PWAs):**
   * PWAs are web apps that incorporate progressive enhancements to offer a more app-like experience.
   * They are designed to work on any platform or device that uses a standards-compliant web browser.
   * PWAs can be installed on the user's home screen, allowing them to be accessed like native apps.
   * They can work offline or on low-quality networks by caching app resources.
   * PWAs have access to some device features and can send push notifications.
   * However, PWAs may not have access to all native features and APIs, and their functionality is still somewhat limited compared to native apps.
   1. **Importance Of Mobile Apps**

* Enhanced User Experience: Mobile apps offer a more optimized and tailored user experience compared to accessing services through a mobile web browser.
* Accessibility and Convenience: Mobile apps provide users with easy and instant access to information, services, and entertainment right at their fingertips.
* Business Opportunities: Mobile apps have become a significant avenue for businesses to reach and engage with their customers. Support, and gather valuable user data for analysis and improvement.
* Revenue Generation: Mobile apps have created opportunities for various revenue models, including paid apps, in-app purchases, subscriptions, and advertising.
* Brand Building and Customer Loyalty: Having a dedicated mobile app can contribute to brand recognition and customer loyalty. Apps provide a platform for businesses to reinforce their brand identity, showcase their products or services, and engage users through personalized experiences. Regular app usage and positive interactions can help strengthen the bond between the brand and its customers, leading to increased loyalty and advocacy.

1. **Review and Compare Mobile App Programming Languages**

Mobile app development can be done using a variety of programming languages, each with its own advantages and use cases. Some of the most commonly used programming languages for mobile app development include:

1. **Java**: Java is one of the most popular programming languages for Android app development. It's supported by the Android SDK (Software Development Kit) and offers a robust development environment. It offers the following advantages in mobile app development;

* Mature language with extensive documentation and community support.
* Offers strong performance and reliability.
* Backed by Google, making it well-supported for Android development.

It also has disadvantages such as; Memory management can be complex, leading to potential issues like memory leaks.

1. **Dart**: Dart is a programming language developed by Google, primarily known for its association with the Flutter framework for building cross-platform mobile, web, and desktop applications. It has the following advantages;

* Easy to Learn: Dart has a syntax that's relatively easy to pick up for developers familiar with languages like Java, JavaScript, or C#. Its clean and intuitive syntax makes it accessible for beginners and experienced developers alike.
* Dart is the primary language used for Flutter development, providing seamless integration with the Flutter framework. This combination enables developers to build cross-platform applications with a single codebase,
* Dart supports object-oriented programming paradigms, including classes, inheritance, interfaces, and mixins. This makes it suitable for building complex, object-oriented applications with clear code organization and reusability.

Dart also has disadvantages such as; Limited Adoption outside Flutter.

1. **Kotlin**: Kotlin is a modern programming language that is fully interoperable with Java

and is now the preferred language for Android app development by Google. It offers the following advantages

* Concise syntax and reduced boilerplate code compared to Java.
* Fully interoperable with Java, allowing developers to leverage existing codebases.
* Offers modern features like null safety and co-routines.

Disadvantages include; smaller community compared to Java, although rapidly growing

1. **Swift**: Swift is the primary programming language for iOS app development. It was introduced by Apple in 2014 and offers a more modern and safer alternative to Objective-C. it has the following advantages;

* Developed by Apple, Swift is highly optimized for iOS ecosystem.
* Offers modern syntax, improving readability and maintainability.
* Includes features like optional and generics to ensure safer code.

Disadvantages include; Rapid evolution of the language can lead to frequent updates, requiring developers to stay updated.

Smaller talent pool compared to more established languages like Objective-C.

1. **Objective-C**: Before Swift, Objective-C was the primary programming language used for iOS app development. Although its usage has declined with the adoption of Swift, there are still many existing iOS apps written in Objective-C. it has the following advantages;

* Provides access to extensive Apple frameworks and libraries.
* Well-established language with a large codebase and community support.
* Interoperable with C and C++, allowing integration with existing codebases.

Disadvantages include; Syntax can be verbose and less readable compared to Swift.

Manual memory management can lead to memory-related bugs if not handled properly.

These are just a few examples of programming languages commonly used in mobile app development. The choice of language often depends on factors such as platform requirements, developer preference, project complexity, and performance considerations.

1. **MOBILE APPLICATION FRAMEWORKS**

Mobile application frameworks can be grouped into **Native app frameworks, hybrid app frameworks** and **cross-platform app frameworks.** Below are some of the frameworks used in the development of mobile applications.

1. **Native app frameworks:**
2. **iOS (Swift, Objective-C):**

* **Language:** Swift provides modern syntax and performance improvements over Objective-C.
* **Performance:** Native performance optimized for iOS devices.
* **Cost and time to market:** Higher development costs but efficient for iOS-specific apps.
* **UX and UI:**  Allows for seamless integration with iOS design principles.
* **Complexity:** Swift simplifies codebase maintenance compared to Objective-C.
* **Community Support:** Active iOS developer community with Swift gaining popularity.
* **Usage:** used in developing iOS, macOS, watchOS and tvOS applications

1. **Android (Java, Kotlin):**

* **Language:** Kotlin offers concise code and eliminates common pitfalls of java.
* **Performance:** Efficient performance with compatibility for java code.
* **Cost and time to market:** Widely used languages with good community support.
* **UX and UI:**  Integrates well with material design principles.
* **Complexity:** Kotlin reduces boilerplate code and improves developer productivity.
* **Community Support:** Strong Android developer community embracing kotlin and has Google backing.
* **Usage:** suitable for building Android applications.

1. **Hybrid app frameworks:**

* **Ionic:**
* **Language:** HTML, CSS, JavaScript.
* **Performance:** Decent performance for most apps.
* **Cost and time to market:** Cost-effective and quick development.
* **UX and UI:**  Flexible but may require customization for complex UI.
* **Complexity:** Relatively easy to learn and use.
* **Community Support:** Large and active community.
* **Usage:** ideal for building cross-platform apps targeting iOS, Android and web.
* **React Native:**
* **Language:** JavaScript.
* **Performance:** Near-native performance with code reusability.
* **Cost and time to market:** Efficient development for iOS and Android.
* **UX and UI:** Provides a native-like experience across platforms.
* **Complexity:** Moderately complex but offers a powerful framework.
* **Community Support:** Strong community backed by Facebook.
* **Usage:** Suitable for building cross-platform apps for Android and iOS.
* **Apache Cordova:**
* **Language:** HTML, CSS, JavaScript.
* **Performance:** Moderate performance due to web view rendering.
* **Cost and time to market:** Cost-effective development for cross-platform apps.
* **UX and UI:**  Decent UI but may lack native performance.
* **Complexity:** Easy to start with for web developers.
* **Community Support:** Active community with plugins for extended functionality.
* **Usage:** Suitable for building cross-platform apps for iOS and Android.
* **Native Script:**
* **Language:** JavaScript, Typescript, Angular.
* **Performance:** Near-native performance with direct access to native APIs.
* **Cost and time to market:** Efficient development for native-like apps.
* **UX and UI:**  Provides near-native user experience.
* **Complexity:** Moderate complexity especially for web developers.
* **Community Support:** Growing community support.
* **Usage:** Ideal for building native mobile apps for iOS and Android.
* **Phone Gap:**
* **Language:** HTML, CSS, JavaScript.
* **Performance:** Moderate performance due to web view rendering.
* **Cost and time to market:** Cost effective and quick development.
* **UX and UI:**  Decent UI but may lack native performance.
* **Complexity:** Relatively easy to learn and use.
* **Community Support:** Active community support.
* **Usage**: Suitable for building cross-platform apps with simpler requirements.
* **jQuery Mobile:**
* **Language:** JavaScript.
* **Performance:** Decent performance for most apps due to native compilation.
* **Cost and time to market:** Efficient development, but may require platform-specific customization.
* **UX and UI:**  Allows for building native-like UI experience.
* **Complexity:** Moderate complexity, especially for web developers.
* **Community Support:** Growing community support.
* **Usage**: Suitable for building native cross-platform mobile apps for iOS, Android and Windows.

1. **Cross-platform app frameworks:**

* **Flutter:**
* **Language:** Dart.
* **Performance:** High performance due to compiled code.
* **Cost and time to market:** Quick development with hot reload features.
* **UX and UI:**  Highly customizable and design capabilities.
* **Complexity:** Moderate complexity but well-documented.
* **Community Support:** Growing community support.
* **Usage:** ideal for creating high quality cross-platform apps.
* **Xamarin:**
* **Language:** C#.
* **Performance:** Near-native performance due to direct access to native APIs.
* **Cost and time to market:** Efficient development with shared codebase.
* **UX and UI:**  Allows for creating native-like experiences on various platforms.
* **Complexity:** Moderate complexity especially for beginners.
* **Community Support:** Strong community support with Microsoft backing.
* **Usage:** Suitable for developing cross-platform apps for iOS, Android and Windows.
* **Unity:**
* **Language:** C#.
* **Performance:** High performance for game and AR/VR apps.
* **Cost and time to market:** Development time varies based on project complexity.
* **UX and UI:**  Excellent for creating interactive and immersive experiences.
* **Complexity:** High complexity especially in non-gaming apps.
* **Community Support:** Strong community focused on gaming and interactive applications.
* **Usage:** Best suited for game development and AR/VR experiences.

**Hybrid frameworks** are used to create mobile applications that run in web view within a native container. They typically use web technologies like HTML, CSS and JavaScript to build the user interface. Hybrid frameworks allow developers to write once and deploy across multiple platforms such as iOS and Android.

**Cross-platform frameworks** on the other hand, provide the ability to write code once and deploy it across multiple platforms while still maintaining a native look and feel. Cross-platform frameworks often use a single codebase to generate native code for different platforms.

Choosing the right framework is a key aspect to developing the most suitable application and it depends on factors such as project requirements, team expertise, budget and platform compatibility.

1. **Study Mobile Application Architectures And Design Patterns**

Mobile application architecture and design patterns play a crucial role in the development of successful and efficient mobile applications. We are going to outline the different types of mobile application architecture and design patterns and highlighting their impact on app performance, scalability, and maintainability.

* 1. **MOBILE APPLICATION ARCHITECTURE**

Mobile application architecture is the foundation upon which a successful app is built. It defines how the various components of the app interact with each other to deliver functionality and a seamless user experience. How smoothly and reliable an app runs depends significantly on the quality of its architecture.

* + 1. **Mobile App Architecture Principles**

When it comes to mobile app development, there are a few key principles that should be followed in order to create a successful and well-designed app.

1. **Sustainability**

As the world increasingly moves online and mobile devices become more ubiquitous, it’s crucial for developers to create applications that are sustainable. In the context of mobile app development, this means creating apps that are efficient in terms of both energy and resources. One way to make sure your app is sustainable is to use a [content delivery network](https://en.wikipedia.org/wiki/Content_delivery_network) (CDN). CDNs help to reduce latency and improve performance by caching content locally. This can help to reduce both data usage and energy consumption.

Additionally, using recycled materials for your app’s packaging can also help to reduce your app’s environmental impact. Think about how long the system will be able to continue operating without needing major overhauls. This can be achieved by using established design patterns, employing modularity and abstraction, and using standardized interfaces. By following these principles, architects can help ensure that their systems are sustainable and can stand the test of time.

1. **Maintainability & Manageability**

These characteristics specify how quickly and readily applications may be improved, monitored, and optimized. It includes tools and techniques for creating mobile apps that give developers the ability to manage app security logs, record system issues, and faults, maintain app improvement plans, guarantee top performance, and much more.

First, the code should be clean and well-organized. This will make it easier for other developers to understand it and work with. Second, an app should be designed with extensibility in mind. This means that adding new features or modifying existing ones should be easy without breaking an app. Finally, an app should be tested thoroughly before release. This will ensure that any bugs are found and fixed before users encounter them.

1. **Reusability**

In order to create a successful and sustainable app, it is essential to reuse as much code as possible. Not only does this help to cut down on development time, but it also reduces the chances of errors and can make maintenance easier. Any effective architecture incorporates the reusability component, which guarantees a shorter time-to-market for the introduction of new software versions and updates.

One common approach is creating a library of reusable components that can be used across different app parts. Another approach is to use a software framework that provides a set of core components that can be extended and customized as needed.

1. **Security**

When it comes to security, there are two primary considerations for mobile app developers: data security and user authentication. Data security is essential for protecting sensitive information such as customer credit card numbers and health records. There are a variety of ways to secure data, including encryption, password protection, and access control.

User authentication is another important consideration, as it helps to ensure that only authorized users can access confidential information. Various authentication methods are available, including biometrics, one-time passwords, and two-factor authentication. By incorporating these [app security measures](https://forbytes.com/blog/application-security/) into their mobile app architectures, developers can keep customer data safe and secure.

1. **Performance**

One of the most critical performance principles is to keep it simple. Complicated architectures can lead to performance issues such as excessive resource consumption and slow response times. Aim for simplicity and avoid unnecessary features or complex integrations when designing your app.

Another key principle is to optimize for the user’s context. Consider the user’s location, network conditions, and device type when designing your app. For example, if you’re developing a mapping app, you’ll need to take into account the user’s current location and whether they’re online or offline.

* + 1. **What to Consider When Developing Mobile App Architecture Design**

There are a number of factors that must be considered when developing the mobile app architecture design, including:

1. **UI/UX design:** A well-designed UI can improve the user experience by making an app more intuitive and easy to use. Conversely, a poorly designed UI can make an app more difficult to use, leading to frustration and abandoned users. Online success depends on having a good user experience (UX) design, but mobile UX is challenging due to the shifting user expectations and best practices of each OS and device type.
2. **Bandwidth:** Users encounter various bandwidth restrictions around the world, with some using 5G and others still having sporadic access. Depending on the size and complexity of an app, it may need to be designed to work with a variety of different network conditions, including low-bandwidth networks.
3. **Device type:** Different devices have different capabilities, which can impact the performance of an app. For example, a smartphone has a small screen and limited processing power, whereas a tablet has a larger screen and more processing power. This can impact the design of the user interface and the overall functionality of an app.
4. **Navigation method:** The way a mobile app is designed can have a big impact on how users interact with it. One crucial factor to consider is navigation. How will users move from one screen to another? Will they need to use menus? Or can they simply swipe between screens? The answers to these questions will help determine the overall architecture of an app.
5. **Real-time updates:** In the current mobile app landscape, it’s more important than ever to consider real-time updates when developing your architecture design. With the ubiquity of high-speed Internet and the widespread adoption of push notifications, users expect to be able to receive information in real-time. This has led to a shift in how apps are designed and developed, focusing on creating reactive, event-driven systems. While this can be a challenge for traditional architectures, it’s vital to consider real-time updates when designing your mobile app architecture.
   * 1. **Why is App Architecture Essential?**

Quality architecture helps with risk management and enables cost reductions. An application with robust, well-planned architecture is more likely to succeed in its target market. Any mobile app project starts with the planning and designing phase, and choosing the right architecture is a core priority. An insufficient approach to this step can slow down the development process and make it more extensive. It can also lead to various performance issues and system failures.

Poor mobile app architecture may also lead to:

* Difficulties with development and maintenance
* Lower code readability
* Source code testing complications
* Greater exposure to errors

For a more detailed understanding, let’s review three core layers on mobile app architecture.

* + 1. **The Fundamental Layers of Mobile App Architecture**

The multi-layer approach is widely used in mobile app development as it segregates the different, application-specific operational layers. Developing and executing each component separately allows developers to solve complicated matters quickly without changing the entire application.

The number of layers (or tiers) varies according to an app’s business and functional requirements, but the three-tier structure is the most common mobile app architecture pattern.

So what are these three layers? Let’s find out.

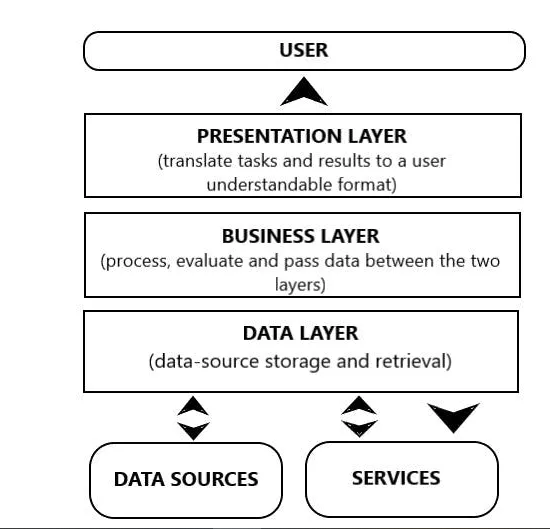
/

Figure 1: *Shows layer of mobile application architecture*

Layers of three-tier architecture and dependencies between them.

1. **Data Layer**

The data layer is responsible for storing and retrieving data and managing any data-related tasks such as synchronization and caching. A well- designed data layer can help improve performance by reducing the time spent fetching data from a remote server. Additionally, it can help improve the stability of an app by providing a consistent and reliable source of information.

When designing a data layer, it is important to consider the needs of an app and its users. For example, if an app requires real-time data updates, then it should be designed to support this. Alternatively, if an app only requires data that is updated regularly, then a more simplistic design may be sufficient. Ultimately the goal is to create an efficient and reliable data layer.

1. **Business Layer**

The business layer is the heart of any mobile app. It’s where all the critical data and logic lives and it’s what makes an app tick. In any mobile app architecture, the business layer is responsible for orchestrating the flow of data between the user interface and data access layer. It typically contains the bulk of the app’s functionality, and such it is often the most complex part of the app.

The business layer is typically divided into several sub players each of which handles a specific type of task. For example, there may be a sub player responsible for handling network requests, another for processing user input. By dividing the business layer into manageable chunks. It is easier to develop, test, and maintain an app

1. **Presentation Layer**

This layer is about how an app presents, itself to the user and includes user interface (UI) elements such as themes, fonts and colors and UI process components. The prioritization of features and functionalities also takes place on this level. Because end-users should be able to easily navigate in the app, the best practice is to keep the presentation level simple and avoid multi-level menus.

* + 1. **Type of Mobile Application Architecture**

## Android Mobile App Architecture

Android is the most widely used mobile operating system, so it’s no surprise that it provides the most support for app development. With it, app developers have access to a wealth of third-party libraries, which makes it easy to build high-quality apps. Android also provides a number of platform APIs that make it easy to implement common data layer needs, such as data storage and retrieval. It provides developers with a number of different options for data storage, including SQLite, which is a lightweight database option, and Realm, which provides a seamless API for data storage.

There isn’t a single Android architecture that is suggested. However, Clean Architecture is the one that is most frequently used for mobile apps.

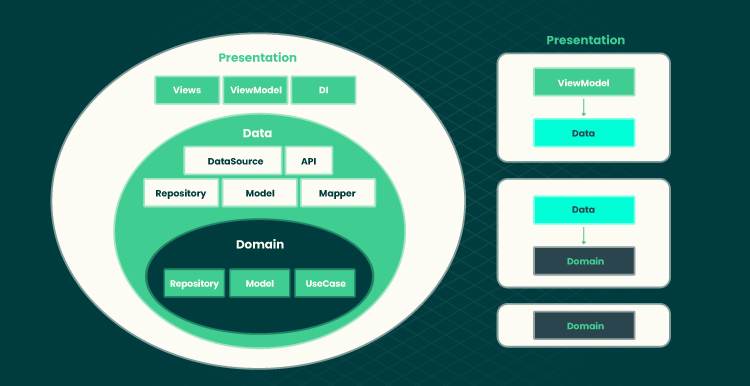


Figure 2: *Show Android Mobile Application Architecture flow and their layers*

Layers and inversion of control are the cornerstones of Clean’s architectural design. With the business layer occasionally referred to as the domain layer, clean concentrates on the same three-layer structure depicted above. The domain/business layer must use interfaces rather than rely on the other layers. Even while it can be challenging to understand, this makes it simple to expand and grow apps over time.

## iOS Mobile App Architecture

Objective-C and Swift are used to create native iOS apps, and Apple clearly outlines recommended practices for app development with the [Model-View-Controller paradigm](https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html) (MVC). This pattern separates the presentation of data from the business logic that powers an app. The Model is the data layer. The View layer is responsible for displaying data to the user, while the Model layer contains the data and the business logic. The Controller is a mediator level that uses a protocol to communicate with an abstraction.

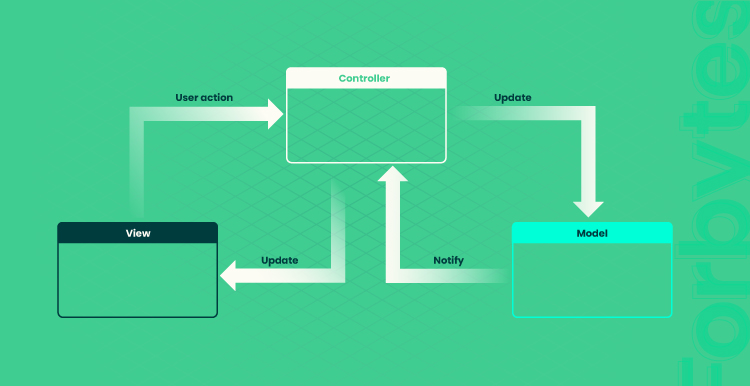


Figure 3: *Show iOS mobile application Mobile-View-Controller flow*

This separation of concerns makes it easier to understand and maintain the codebase. Furthermore, it enables developers to reuse components across different app parts.

Another popular architecture is [Model-View-View Model](https://www.techtarget.com/whatis/definition/Model-View-ViewModel) (MVVM). This pattern is similar to MVC but with a few key differences. First, the View layer is not responsible for handling user input; instead, this responsibility is delegated to the View Model layer. Second, the View Model exposes biddable properties that can be used by the View layer to automatically update itself when data changes. This separation of concerns results in a more testable and maintainable codebase. Finally, it makes it easier to create reactive user interfaces.

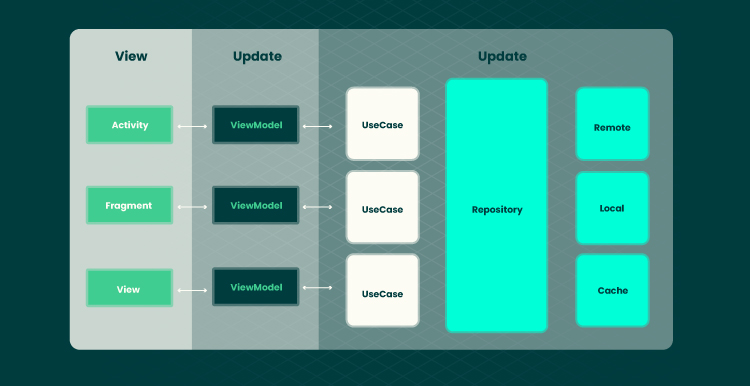


Figure 4: *iOS mobile architecture application Model-View-View Model (MVVM)*

## Cross Platform App Architecture

Cross-platform app architecture is a type of app development that allows for creating apps that can be used on multiple platforms. This can be particularly beneficial for businesses that want to reach a wider audience with their app, as it eliminates the need to develop separate versions for each platform. This can be achieved by using a shared codebase that can be compiled for different platforms or by using a platform-specific codebase that is run on a virtual machine. There are benefits and drawbacks to both approaches.

The main benefit of using a common codebase is that it can save development time and costs. In addition, cross-platform app architecture can also lead to cost savings, as it often requires less development time and effort. However, it is essential to note that cross-platform app architecture can also present some challenges, such as the need to account for different screen sizes and resolutions. As a result, businesses should weigh the pros and cons of cross-platform app architecture before deciding if it is the right solution for their needs.

* 1. **DESIGN PATTERNS**

Design pattern are like building blocks that developers can use to solve common problems and create well-structure, maintainable code. They have had a significant impact on software development, including mobile app development. Here are some of the most frequently used design patterns in mobile development

* + 1. **Most Frequency Design Patterns in Mobile Development with Examples and when to used it**

## Classic MVC or [MVC (Model-View-Controller) Architecture](https://www.geeksforgeeks.org/mvc-design-pattern/)

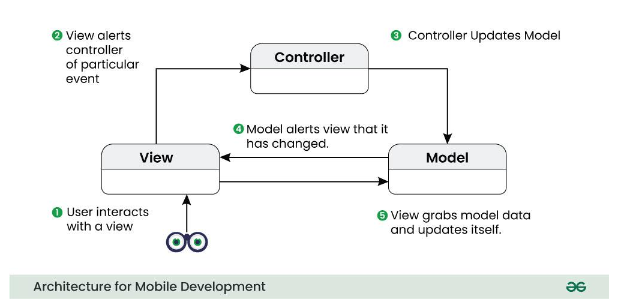


Figure 5: *Classic MVC flow diagram*

Classic MVC, or Model-View-Controller, is an architectural pattern used in software development to organize the code of an application in a structured way. Let’s break down each part of MVC:

* **Model:**
  + Think of the Model as the brain of your application. It manages the data and the rules for how the data can be changed. For example, if you’re building a to-do list app, the Model would handle tasks, deadlines, and any other information related to the to-do items.
* **View:**
  + The View is like the face of your app. It’s what the user sees and interacts with. In our to-do list example, the View would be the part of the app where you see your list of tasks, checkboxes, and maybe a button to add new tasks.
* **Controller:**
  + The Controller is the middleman between the Model and the View. It takes input from the user (like clicking a button to add a new task), updates the Model based on that input, and then tells the View to refresh and show the updated information. In our to-do list, the Controller would handle things like adding or deleting tasks.

### ****Example:****

Suppose you have a basic weather app.

* **Model:**This would handle the weather data, like the current temperature, forecast, and other related information.
* **View:**The View would be what the user sees on the screen – maybe a simple interface showing the current temperature and a forecast.
* **Controller:** The Controller would handle user interactions. If a user wants to switch from viewing the current temperature to the forecast, the Controller takes that request, updates the Model (gets the forecast data), and tells the View to display the new information.

## When to use Classic MVC Architecture

Use Classic MVC when your app is not too big or too small – it’s like the Goldilocks choice for moderate-sized projects. If you want a clear separation between how data is handled (Model), how it’s shown to the user (View), and how the user interacts with it (Controller), then Classic MVC is a good fit. It’s a balanced approach that works well for many different types of applications.

## APPLE’S MVC ARCHITECTURE

Apple’s MVC (Model-View-Controller) architecture is a specific implementation of the classic MVC pattern tailored for iOS app development. Let’s break down Apple’s MVC in simple terms:

* **Model:**
  + Just like in the classic MVC, the Model in Apple’s MVC is responsible for managing the data and business logic of your app. For example, in a note-taking app, the Model would handle tasks such as saving and retrieving notes.
* **View:**
  + The View is what the user interacts with on the screen. In an iOS app, this could be a button, a label, or any other interface element that users can see and touch. In the note-taking app, the View would be where you read and edit your notes.
* **Controller:**
  + The Controller acts as a coordinator between the Model and the View. It takes user input from the View, updates the Model accordingly, and manages the communication between the two. In the note-taking app, the Controller would handle actions like creating a new note or deleting an existing one.

### ****Example:****

Imagine you’re building a basic calculator app.

* **Model:** The Model would handle the numbers and operations. If you press the buttons ‘2’ and ‘3’, the Model keeps track of these numbers and knows that the next operation is to add or multiply, for example.
* **View:**The View is what the user sees – the buttons for numbers and operations, and a display area showing the current calculation.
* **Controller:** The Controller responds to user actions. If the user taps the ‘5’ button, the Controller takes that input, updates the Model, and tells the View to refresh and show the updated calculation.

## When to use Apple’s MVC Architecture

* Use Apple’s MVC when you’re developing an iOS app. It’s the default architecture recommended by Apple for building apps on their platform.
* If you’re comfortable working within the iOS development ecosystem and want a structure that aligns with Apple’s guidelines, Apple’s MVC is a straightforward and effective choice.
* It works well for a wide range of iOS applications, from simple utilities to more complex applications.

## MVMM (Model-View-View-Model) Architecture

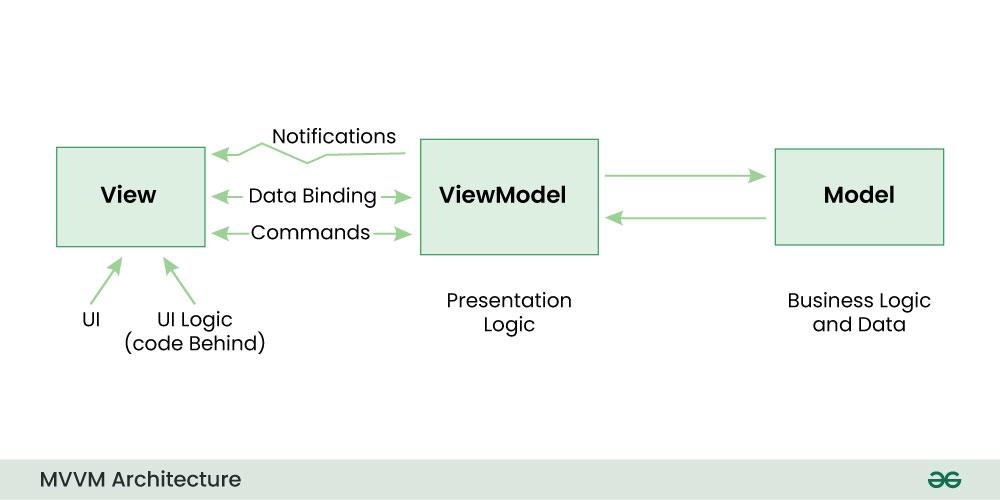


Figure 6: *show MVVM flow diagram*

MVVM, or Model-View-View Model, is an architectural pattern commonly used in software development, especially for building user interfaces. Let’s break it down:

* **Model:**
  + The Model is responsible for managing the data and the business logic of your application. If you’re building a weather app, for example, the Model would handle things like fetching weather data from the internet and storing it.
* **View:**
  + The View is what the user sees on the screen – buttons, text, images, etc. In a weather app, the View would be the part of the app where you see the current temperature, weather conditions, and a forecast.
* **View-Model:**
* The View-Model is like a bridge between the Model and the View. It takes data from
  + the Model and prepares it in a way that the View can easily display. It also handles user interactions and updates the Model accordingly. In the weather app, the View Model would take the raw weather data from the Model, format it, and then provide it to the View for display.

### ****Example:****

Imagine you are building a basic news app.

* Model: The Model would handle news articles – fetching them from an online source, storing them, and managing the data.
* View: The View is what the user interacts with – a screen displaying a list of news articles and maybe a button to read more details about a specific article.
* View-Model: The View-Model takes care of preparing the news data for display. It takes the raw data from the Model, formats it into a suitable form for the View, and handles interactions like tapping on an article to read more.

## When to use MVVM Architecture

Use MVVM when you want a structured way to organize your code and keep a clear separation between how your data is managed (Model), how it’s presented to the user (View), and the logic that connects the two (View-Model). MVVM is particularly effective when used with data-binding frameworks, as it simplifies the synchronization between the UI and the underlying data. It’s a good choice for projects with complex user interfaces or applications where data presentation and manipulation are crucial.

* **STUDY HOW TO COLLECT AND ANALYZE USER REQUIREMENTS FOR A MOBILE APPLICATION(REQUIREMENT ENGINEERING)**

To collect and analyze user requirements for a mobile application effectively, a structured approach is crucial. Here is a comprehensive step to follow:

* **Identify Stakeholders:** Determine the key stakeholders who will be involved in the mobile application project. This may include end-users, clients, business analysts, developers, and other relevant parties. Understand their roles, needs and expectations.
* **Conduct User Research:** Gather information about the target audience and their preferences. This can be done through surveys, interviews, focus groups, or market research. Identify their demographics, behaviors, and specific requirements for the mobile application.
* **Define Use Case and User Stories:** Use cases and user stories help in capturing functional requirements from the user’s perspective. Use cases describe specific interactions or scenarios, while user stories focus on the user’s goals and desired features. These help create a clear understanding of how the application will be used.
* **Create Requirement Documentation:** Document the collected requirements in a structured manner. This may include functional requirements (features, interactions, and workflows), non-functional requirements (performance, security, usability) and any constraints or dependencies.
* **Prioritize and Validate Requirements:** Prioritize the requirements based on their importance and feasibility. Validate them with stakeholders to ensure accuracy and alignment with their expectations. Use techniques such as prototyping or mockups to visualize and validate the requirements.
* **Review and Refine:** Conduct regular reviews and iterations of the requirements to refine and improve them. This may involve incorporating feedback from stakeholders, addressing conflicts or ambiguities, and ensuring the requirements are complete, consistent, and understandable.
* **Use Requirement Analysis Techniques:** Apply various techniques to analyze the requirements such as feasibility analysis. These techniques help assess the practicality and potential impact of the requirements on the mobile application development process
* **Document Assumptions and Constraints:** Identify any assumptions made during the requirement gathering process and document them. Also. Consider any constraints or limitations that may impact the development or implementation of the mobile application
* **Maintain Traceability:** Establish traceability between the requirement and other project artifacts, such as design, testing, and implementation. This helps ensure that the requirements are properly addressed throughout the development lifecycle.
* **Ensure Continuous Collaboration:** Maintain Open Channels of communication with stakeholders throughout the requirement engineering process. Collaborate closely with developers, designers, and testers to ensure a shared understanding of the requirements.
* **STUDY HOW TO ESTIMATE MOBILE APP DEVELOPMENT COST**

In a nutshell, the cost to build an app comes down to these basic elements:

* **Project Complexity:** What the app will do and what needs, it will fulfill for the customers. Project complexity influences the total hours needed for development that will affect the cost of development
* **Complexity of app features:** The complexity and number of features directly affect development time and cost. Simple apps with basic features like login, search, and notification cost less compared to mid-level or complex app (like food delivery app or cab booking app) with intricate business logic and multiple functions. The more features and integration an app has, the longer it takes to develop, impacting the overall cost.
* **Target devices and platforms:** Choosing between native and hybrid apps affects costs.
* Native apps, specific to either Android or iOS, require separate development teams for each platform, doubling the cost, longer development cycles, and maintenance efforts for each platform. However, native apps offer superior performance and access to platform-specific features, which can be crucial for certain types of application.
* Hybrid apps offer a more cost-effective solution as they allow developers to write code once and deploy it across multiple platforms. This approach reduces development time, effort, and costs associated with maintaining separate codebases for each platform. While hybrid apps may not match the performance of native apps, they provide a good balance between cost efficiency and cross-platform compatibility.
* **User Interface:** A more complex user interface can require more design and development time, as well as additional testing and debugging which will affect the cost of development cost. For example, a simpler app with standard UI elements like clock or calculator will cost less to build than mid-level app or complex app.
* **Third-party integration:** Do the app that you are building need to be integrate with third-party services (like location sharing or payment). If so, it will affect the cost. Implementing third-party APIs requires additional coding, testing, and debugging efforts to ensure seamless integration. This extra development work increase project complexity and time, leading to higher costs.
* **Testing and Debugging:** Once you finish building the app, you need to test it thoroughly to make sure it works as expected. This can involve testing on a variety of devices and platforms and can take a significant amount of time. Additionally, if there are any bugs or issues that need fixing, this can add to the average app development cost, it again depends on how complex and feature-rich your app is.
* **Hidden Costs of Developing an APP:** App development can be a complex process that can be a lot of time, effort, and resources. While the upfront costs of developing an app are often the most obvious, there are also hidden costs that can add up quickly, such as:
* **Design and User experience**

Creating an app that looks good and is easy to use requires a lot of design work. This can include everything from designing the user interface to creating icons and graphics. These design costs can quickly increase, especially if you need to hire a professional designer.

* **App Store Fee**

If you plan to distribute your app through app stores like the Apple App Store or Google Play Store, there are fees associated with each download. These fees can vary depending on the platform and the pricing model you choose.

* **Maintenance and Updates**

Apps require ongoing maintenance and updates to stay functional and keep pace with the latest technologies. This can include everything from fixing bugs to adding new features. The cost of application maintenance can be significant.