Republic Of Cameroon
Ministry Of Higher Education
Peace-Work-Fatherland

République Du Cameroon
Ministère De L'enseignement Supérieur
Paix-Travail-Patrie

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER ENGINEERING

COURSE CODE AND COURSE TITLE: CEF 440, INTERNET PROGRAMMING AND MOBILE PROGRAMMING

2nd SEMESTER 2023/2024

TASK 1: BASIC DEFINITIONS IN MOBILE AND INTERNET PROGRAMMING

NAME	MATRICULATION
KEDJU PRECIOUSE NGWE LEKUNZE	FE21A211
DJOUMESSI LEKANE WENDY FORTUNE	FE21A173
INDAH RISCOBELLE MBAH	FE21A204
NGUEDIA JEATSA JOYCE GRACE	FE21A263
EKUNDIME GLEAN MAKOGE	FE21A433

Supervisor: Dr NKEMENI VALERY

TABLE OF CONTENT

INTRODUCTION

An application is a software that lets you exchange information with customers and help them complete specific tasks. Different types of applications, or apps, are based on their development method and internal functionality. Web apps are delivered over an internet browser. Users don't need to install them on their devices. Native apps, on the other hand, are built for a specific platform or device type. The user must install the appropriate software version on their device of choice. Hybrid apps are native applications with a web browser embedded inside them.

A mobile app (or mobile application) is a software application developed specifically for use on a small, wireless computing devices, such as smartphones and tablets, rather than desktop or laptop computers.

Mobile applications frequently serve to provide users with similar services to those accessed on PCs. Apps are generally small, individual software units with limited function.

1. A review and comparison of major types of mobile apps and their differences (native, progressive web apps, hybrid apps).

1.1 Native apps

A native app, or native application, is a software application built in a specific programming language, for the specific device platform, such as iOS (iPhone) or Android. Example: Instagram, Duolingo
Native iOS apps are written in Swift or Objective-C and native Android apps are written in Java.
Because they are built in a native development language, native apps can only work on the device they've been coded for. That means a native iPhone app can't work on an Android phone, and vice versa.

1.1.1 Advantages of native applications include:

- Broad functionalities due to using the capabilities of the underlying device;
- Fast and responsive software performance;
- Push notifications;
- A UI that better matches with user experiences of the OS; and
- Quality assurance though ratings in application stores.

1.1.2 Disadvantages of native applications include:

- Multiple code bases because each device has its own version of the app;
- The cost for additional developers to build and manage a code base for each platform;
- Time spent on multiple builds for separate platforms in each feature update.

1.2 Progressive Web Apps

A Progressive Web App (PWA) is a type of web application that leverages modern web technologies to deliver an app-like experience to users. PWAs are designed to work on any platform that uses a standards-compliant browser, such as desktops, laptops, tablets, and smartphones. They are built using web technologies

like HTML, CSS, and JavaScript, and they are designed to be responsive, reliable, and engaging.

1.2.1 Advantages:

- Cross-platform compatibility
- Offline functionality
- Improved performance

1.2.2 Disadvantages:

- Limited native functionality
- Less discoverability
- Dependency on browser support

1.3 Hybrid Apps

A hybrid app is an app that combines web and mobile technologies, to be able to run in a variety of environments with largely the same code base.

Most of the time, a hybrid app is effectively a web app wrapped in the shell of a mobile app. Yet unlike web apps, hybrid apps can be downloaded and run locally, similar to a native mobile app.

1.3.1 Advantages:

- Cross-platform compatibility
- Access to native device features
- Faster development and deployment

1.3.2 Disadvantages:

- Performance may not match native apps
- Dependency on third-party frameworks
- Limited access to advanced device functionalities

1.4 Differences.

Aspects	Native Apps	Progressive Web Apps (PWA)	Hybrid Apps
Platform compatibility	Specific to each platform (iOS, Android)	Cross-platform (works on various devices and platforms with a single codebase)	Cross-platform (uses web technologies, but packaged as native apps)
Installation	Installed from app stores(e.g App Store, Google Play)	Can be installed directly from a browser without app stores	Installed from app stores or directly from a browser
Access to Device Features	Full access to native device features	Limited access to some device features via web APIs	Access to native device features through plugins
Offline Functionality	Works offline with local data storage	Limited online functionality (depends on caching)	Limited offline functionality (depends on caching and local storage)
Performance	Generally offers the best performance	Performance can be slightly less than native apps	Performance may vary based on the underlying web view
Development Time	Longer development	Faster development with a single codebase	Faster development with a single codebase
Updates	Updates require approval from app store	Updates are seamless and can be done server-side	Updates can be done through app stores or directly
Discoverability	Highly discoverable on app stores	Less discoverable	Discoverable on app stores or

		compared to native apps	through web links
Maintenance	Separate codebases for each platform	Single codebase for multiple platforms	Single codebase for multiple platforms
Examples	Facebook, Instagram, Twitter	Twitter Lite, Pinterest	Instagram, Uber, LinkedIn

2. A review and comparison between mobile app programming languages.

Over time, numerous programming languages have emerged as powerful tools for mobile app development, each contributing to the excellence of mobile programming. Below is a curated list of ten mobile programming languages that have been meticulously compared.

2.1 JavaScript

JavaScript is among the most popular mobile app development languages, leading the list that allows one to develop exceptional mobile applications. With highly flexibility and adaptability, it holds support with environments other than browsers.

With this impressive development language along the way, one can ease the process of managing and controlling every bit that involves development activities. Developing feature rich and interactive app is made easier with JavaScript mobile app development language.

With this development language, applications that hold cross platform support with compatibility on iOS, Android, and Windows can be developed.

2.1.1 Key features

- Javascript is an interpretive language with strong type safety, prototype-based, and multi-paradigm capability.
- A JavaScript application can be event-driven, functional, or imperative (including a prototype or object-oriented model).
- The Javascript language provides APIs for handling text, arrays, dates, and regular expressions but does not provide basic I/O functions.

2.1.2 Adavantages

- JavaScript is easy to learn and implements that makes it the first choice for developers.
- It allows you to develop applications that goes in support with the frontend and backend of your applications
- It allows you to create application with basic to top end features without any complications

2.1.3 Disadvantages

- JavaScript is way too slow with the speed when compared with other development languages like C++ or Java.

2.2 Kotlin

Kotlin is a language for mobile app development sweeping the Android app development industry. With its conciseness and expressiveness, it has gained great popularity among developers.

With Kotlin, one can seamlessly migrate to or integrate new projects with existing ones. Along with robust safety features and intuitive syntax, it assists developers in writing cleaner, easier-to-maintain code, which results in better app performance and user satisfaction.

2.2.1 Key features

- It allows you to create application with basic to top end features without any complications
- With Kotlin, you can use Android OS directly from the IDE's installation packages, as opposed to a Standard Java Compiler.

- By default, Kotlin determines a programme's value and expression by aggressive interference unless explicitly stated otherwise.

2.2.2 Adavantages

- Kotlin is compatible with Java, making it easy to extend and improve Java-based programs.
- Multiple platforms are supported, while native-level support is provided.

2.2.3 Disadvantages

- It is possible there is less information and resources available for Kotlin since it is a newer language. It makes it difficult to develop and solve problems.

2.3 Swift

Swift is convenient for iOS developers to create feature rich iOS applications. Swift was introduced by Apple with easy to use features overpowering Objective-C. With Swift, developers do not have to go through the development challenges and eliminate the flaws that become a talk of the day. Today Swift has turned out to be the first choice for iOS application development with rapid growth.

2.3.1 Key Features

- It’s easy to learn
- Quickly maintainable
- It is easy to code
- Programming language that requires less coding

2.3.2 Advantages

- With Swift’s interactive nature, catching errors and debugging are simplified, resulting in reduced development time.
- Reading ease of the language.
- With Swift, you will often have a better user experience and less memory demand on your device.

2.3.3 Disadvantages

- The Apple ecosystem works well with Apple products. It still doesn’t work with other operating systems. You can’t easily port a Swift app to Android or other platforms.

2.4 Dart

Dart is also one of those mobile programming languages that eases the cross-platform development journey. It is specifically designed keeping clients in mind. Using it, developers can create mobile apps that run across multiple platforms.

Code written in Dart can run on any platform without any substantial limitations. Open-source, garbage-collected Dart uses C-style syntax. Compiling it with native code or JavaScript is possible, and it includes Flutter, a framework for developing mobile apps.

2.4.1 Key features

- Support for garbage collection
- Object oriented programming language
- It offers quick and reliable compilers

2.4.2 Advantages

- There is the possibility of sharing the coding language across mobile apps and web applications.
- Apps can be launched quickly across multiple platforms.
- Flutter is often compared to native apps' ability to provide a smooth, near-native user experience.

2.4.3 Disadvantages

- Backend support is less extensive than in other established languages.
- Language packages are less common, which means some applications may need to be created from scratch.

2.5 Java

The most widely used programming language for developing high-end applications is Java. Known for its power and security in the mobile application industry.

With this language, applications for various platforms like Windows, Linux, Android, and Mac OS can be build.

2.5.1 Key Features

- The language of object-oriented programming
- Platform-independent
- API support makes integration a breeze
- Reads and learns easily
- Many open-source libraries are available

2.5.2 Advantages

- The documentation is extensive.
- An extensive pool of Java developers is available.
- Third-party libraries and tools are available.

2.5.3 Disadvantages

- The JIT compiler significantly slows down the program.
- Java requires a great deal of memory and computing power. The result is an increase in hardware costs.

2.6 HTML5

HTML5 is the most recent evolution of HTML and the most popular markup language. In this way, online content can be organized and presented more effectively. Besides being easier to learn, it is well-designed, has better accessibility, and has cleaner, improved code.

When developing location-based mobile applications, this might be a good choice. In addition to media elements, multi-platform capabilities, and agile app deployment, it facilitates efficient development management.

2.6.1 Key Features

- Support for multimedia
- Simple and short syntax
- Features that improve security

2.6.2 Advantages

- The HTML5 programming language is an affordable way to build cross-platform apps (apps that work on various operating systems).
- An HTML5 app is easier to manage due to its single code base. When adding a new feature or modifying an existing one, you only need to make one change and deploy it once.

2.6.3 Disadvantages

- Different browsers have issues with HTML5.
- The users of your application won't be able to access it unless they have a browser and access to the internet.

2.7 C#

The C# programming language is considered an object-oriented programming language developed by Microsoft. The language is widely used for creating command-line scripts and games for Android devices.

A low-code alternative like OutSystems or Kony has an SDK for multiple languages. C# is similar to system programming languages used to develop mobile apps. With an IDE for hybrid development, C# code on iOS and Android is cross-compiled for native performance.

2.7.1 Key features

- The C# programming language provides strong typing, declarative logic, generic, functional, component, and object-based programming.
- A C# implementation will be included in Common Language Infrastructure.
- This language is based on simplicity, modernity, general purpose, and object-oriented concepts.

2.7.2 Advantages

- A C# program is very similar to a C program.
- Programmers prefer C# over other languages.

2.7.3 Disadvantages

- The language is not suitable for beginners.

- Due to its case-sensitive nature, C# causes confusion if there is an error in matching alphabets.

2.8 C++

The C++ programming language is widely known among mobile app developers. The Objective-C language simplifies the development of iOS apps with C++. A C++ extension is a language extension for the C programming language.

With C++, you completely control the amount of system resources and memory you consume. Its versatility and power can be used for operating systems, web browsers, and games.

2.8.1 Key features

- It’s an incredibly powerful language which has built impressive programs like Google Chrome, Photoshop, and PayPal. It exists in sectors ranging from banking to VR.
- C++ can run the same program on different operating systems and interfaces, even if the original code isn’t supported.
- It optimizes memory storage and can deliver very fast results.

2.8.2 Advantages

- Programs written in C++ can often be executed on other platforms without any further ado
- C++ is considered a middle-level language because of its combination of low-level and high-level features.

2.8.3 Disadvantages

- Writing programs in C++ can be challenging, especially for developers more familiar with the niceties of modern programming languages.
- C++ still does not have any built-in support for multi-threaded applications.

3 MOBILE APP DEVELOPMENT FRAMEWORKS AND THEIR DIFFERENCES

3.1 React Native

- **Programming Language:** JavaScript
- **Performance:**
  Good. Utilizes native UI components for a performant experience, but might not be quite as fast as truly native apps.
- **Cost:** Moderate. The framework itself is free and open-source, but development costs can vary depending on developer expertise in JavaScript and React.
- **Time to Market:** Fast. Uses a single codebase for both iOS and Android, allowing for faster development compared to native development.
- **UX/UI:** Excellent. Can achieve a native look and feel with its access to native UI components.
- **Complexity:** Moderate. Requires knowledge of JavaScript and React, which might present a learning curve for developers unfamiliar with these technologies.
- **Community Support:** Large & Active. One of the most popular mobile app development frameworks, boasting a vast and active community for support and resources.
- **Platforms:** iOS & Android
- **Suited for:** Projects requiring good performance, a native look and feel, and faster development times. Ideal for developers familiar with JavaScript and React.

3.2 Flutter

- **Programming Language:** Dart
- **Performance:** Excellent. Uses its own rendering engine (Dart) for exceptional performance, often exceeding native app performance in some cases.

- **Cost:** Moderate. Similar to React Native, the framework is free and open-source, but development costs depend on developer expertise in Dart.
- **Time to Market:** Fast. Employs a single codebase for both iOS and Android, enabling faster development compared to native approaches.
- **UX/UI:** Excellent. Offers highly customizable UI components for achieving a native look and feel on both iOS and Android.
- **Complexity:** Moderate. Introduces a new language (Dart) to learn alongside the framework itself. While Dart is considered relatively easy to pick up, it adds another layer compared to JavaScript-based frameworks.
- **Community Support:** Growing & Active. The community surrounding Flutter is rapidly expanding, offering increasing support and resources as the framework gains popularity.
- **Platforms:** iOS, Android, Web, Desktop
- **Suited for:** Projects demanding top-notch performance, a beautiful and customizable UI, and faster development cycles. Ideal for developers open to learning a new language (Dart).

3.3 Xamarin

- **Programming Language:** C#
- **Performance:** Excellent (Native). Leverages native development for both iOS and Android, resulting in exceptional performance.
- **Cost:** High. Requires a commercial license, making it the costliest option on this list. Additionally, development often involves C# developers, which can further increase costs.
- **Time to Market:** Moderate. While faster than developing native apps from scratch, Xamarin requires separate codebases for iOS and Android, adding some development time compared to single-codebase frameworks.

- **UX/UI:** Excellent (Native). Provides full access to native UI components, ensuring a perfect native look and feel on both platforms.
- **Complexity:** High. C# development adds a significant learning curve for developers unfamiliar with the language.
- **Community Support:** Large & Active (Microsoft). Backed by Microsoft, Xamarin boasts a large and active community with extensive resources and support.
- **Platforms:** iOS & Android
- **Suited for:** Projects requiring the absolute best performance and a true native look and feel. Ideal for teams with existing C# expertise or those willing to invest in learning the language.

3.4 Ionic:

- **Programming Language:** Web technologies (HTML, CSS, JavaScript)
- **Performance:** Moderate. Relies on web technologies, resulting in potentially lower performance compared to native apps.
- **Cost:** Low. Open-source framework, making it a cost-effective choice.
- **Time to Market:** Fast. Utilizes a single codebase for both iOS and Android, enabling faster development.
- **UX/UI:** Good (Web-like). Can achieve a good user experience, but the UI might have a slight web-like feel compared to truly native apps.
- **Complexity:** Low. Leverages existing web development skills with HTML, CSS, and JavaScript.
- **Community Support:** Large & Active. Boasts a vast and active community for support and resources.
- **Platforms:** iOS, Android, Web
- **Suited for:** Projects requiring a fast development cycle, familiarity with web technologies, and a good user experience with a web-like

aesthetic. Ideal for developers comfortable with HTML, CSS, and JavaScript.

3.5 Apache Cordova (Formerly PhoneGap):

- **Programming Language:** Web technologies (HTML, CSS, JavaScript)
- **Performance:** Moderate. Similar to Ionic, relies on web technologies for development, potentially leading to lower performance compared to native apps.
- **Cost:** Low. Open-source framework, making it a cost-effective option.
- **Time to Market:** Fast. Employs a single codebase for both iOS and Android, enabling faster development cycles.
- **UX/UI:** Good (Web-like). Offers a good user experience, but the UI might have a web-like feel compared to native apps.
- **Complexity:** Low. Leverages existing web development skills with HTML, CSS, and JavaScript.
- **Community Support:** Large & Active. Inherited a large and active community from PhoneGap.
- **Platforms:** iOS, Android, Windows
- **Suited for:** Projects requiring a fast development cycle, familiarity with web technologies, and a good user experience with a web-like aesthetic. Similar to Ionic, it caters to developers comfortable with HTML, CSS, and JavaScript.

3.6 jQuery Mobile:

- **Programming Language:** JavaScript
- **Performance:** Moderate. Relies on JavaScript libraries for mobile development, potentially leading to lower performance compared to native apps.
- **Cost:** Low. Open-source framework, making it a cost-effective choice.

- **Time to Market:** Moderate. While faster than native development, separate codebases might be needed for iOS and Android, adding some development time compared to single-codebase frameworks.
- **UX/UI:** Good (Web-like). Offers a good user experience but might have a web-like look and feel compared to native apps.
- **Complexity:** Moderate. Requires knowledge of JavaScript and familiarity with the jQuery Mobile library.
- **Community Support:** Large & Active. Boasts a large and active community for support and resources.
- **Platforms:** Primarily web-based, with limited native app capabilities
- **Suited for:** Projects requiring a web-based mobile experience with a good user interface. Ideal for developers comfortable with JavaScript and the jQuery library.

3.7 NativeScript:

- **Programming Language:** JavaScript, Angular, TypeScript
- **Performance:** Good. Offers good performance with native-looking UIs using the chosen language.
- **Cost:** Moderate. Open-source framework, but development costs can vary depending on the chosen language (JavaScript, Angular, or TypeScript) and developer expertise.
- **Time to Market:** Moderate. Utilizes a single codebase but requires learning NativeScript in addition to the chosen language, potentially impacting development speed.
- **UX/UI:** Good (Native-like). Can achieve a native look and feel on both iOS and Android.
- **Complexity:** Moderate. Requires learning NativeScript along with your chosen language (JavaScript, Angular, or TypeScript), adding complexity compared to pure web-based frameworks.
- **Community Support:** Active. The community is growing and offers support and resources for NativeScript development.
- **Platforms:** iOS & Android

- **Suited for:** Projects requiring good performance, a native look and feel, and the ability to leverage existing JavaScript, Angular, or TypeScript skills. Ideal for developers comfortable with these languages and willing to learn NativeScript.

3.8 Sencha Ext JS (Commercial):

- **Programming Language:** JavaScript
- **Performance:** Good. Offers good performance with its own UI component library.
- **Cost:** High. Requires a commercial license, making it the most expensive option on this list.
- **Time to Market:** Moderate. Separate codebases for iOS and Android
- **UI/UX:** Good (can achieve native look and feel)
- **Complexity:** High (Commercial product). Sencha Ext JS is a commercial product with a rich feature set and extensive UI component library. This can add complexity compared to some open-source frameworks as it requires learning the framework itself in addition to JavaScript.
- **Suited for:** Projects requiring a fast development cycle, familiarity with web technologies, and a good user experience with a web-like aesthetic. Ideal for developers comfortable with HTML, CSS, and JavaScript.
- **Community Support:** Large (Commercial). While not as vast as some open-source communities, Sencha Ext JS boasts a large and active community supported by the commercial product and its user base. Support resources and forums are typically available through Sencha subscriptions.
- **Platforms:** iOS & Android (Web).

Summary :

Feature	React Native	Flutter	Xomarin	Icon	Apache Cordova	jQuery Mobile	NativeScript	Sencha Ext JS
Programming Language	JavaScript	Dart	C#	Web technologies (HTML, CSS, JS)	Web technologies (HTML, CSS, JS)	JavaScript	JavaScript, Angular, TypeScript	JavaScript
Performance	Good (uses native UI components)	Excellent (own rendering engine)	Excellent (native code)	Moderate (web-based rendering)	Moderate (web-based rendering)	Moderate (web-based rendering)	Good (uses native APIs)	Good (uses native APIs)
Cost	Moderate (developer expertise can impact cost)	Moderate (developer expertise can impact cost)	High (licensing fees)	Low (open-source)	Low (open-source)	Low (open-source)	Moderate (developer expertise can impact cost)	High (commercial license)
Time to market	Moderate (developer experti	Fast (single codebase	Moderate (separate codeb	Fast (single codebase for multipl	Fast (single codebase for multipl	Fast (single codebase for multipl	Moderate (separate codeb	Moderate (separate codeba

	se can impact cost)	for multiple platforms)	ases for each platform)	e platforms)	e platforms)	e platforms)	ases for each platform, potential for code reuse)	ses for each platform)
UX/UI	Excellent (can achieve native look and feel)	Excellent (custom widgets for native look and feel)	Excellent (native UI components)	Good (web-like UI, can be customized)	Good (web-like UI, can be customized)	Good (web-like UI, can be customized)	Good (can achieve native look and feel)	Good (can achieve native look and feel)
Complexity		Moderate (learning Dart adds another layer)	High (C# development experience needed)	Low (uses web technologies)	Low (uses web technologies)	Low (uses web technologies)	Moderate (learning curve for NativeScript and chosen	High (steeper learning curve, commercial product)

							language)	
Community support	Large & Active	Growing & Active	Large & Active (Microsoft)	Large & Active	Large & Active	Large & Active	Active	Large & Active (commercial)
Platforms	iOS, Android	iOS, Andriod, Web, Desktop	iOS, Android	iOS, Android, Web	iOS, Android, Windows	Web	iOS, Android	iOS, Android (Web)

4 Mobile application architecture and design patterns

4.1 Mobile Application Architecture:

Mobile application architectures and design patterns are fundamental concepts for building well-structured, maintainable, and scalable mobile apps. Here's a breakdown of both:

- An application architecture defines the overall structure of your mobile app, including how components interact and data flows.

- Choosing the right architecture helps you organize your codebase, improve maintainability, and ensure a smooth user experience.

4.1.1 **Monolithic Architecture**:

- Traditional architecture where the entire application is built as a single unit.
- Simple to develop and deploy but can become difficult to maintain and scale as the app grows.
- Commonly used for small or prototype projects.

4.1.2 **Layered Architecture**:

- Divides the application into layers such as presentation, business logic, and data access.
- Promotes separation of concerns, making the application easier to understand and maintain.
- Common layers include UI layer (presentation), business logic layer (application logic), and data access layer.

4.1.3 **Microservices Architecture**:

- Breaks down the application into smaller, independent services that can be developed, deployed, and scaled independently.
- Promotes modularity, flexibility, and scalability but adds complexity in terms of communication between services.
- Commonly used for large-scale, distributed systems.

4.1.4 MVC (Model-View-Controller):

- Separates an application into three interconnected components: Model (data), View (UI), and Controller (logic).
- Promotes modular design, allowing for easier maintenance and testing.
- Commonly used in web and mobile development.

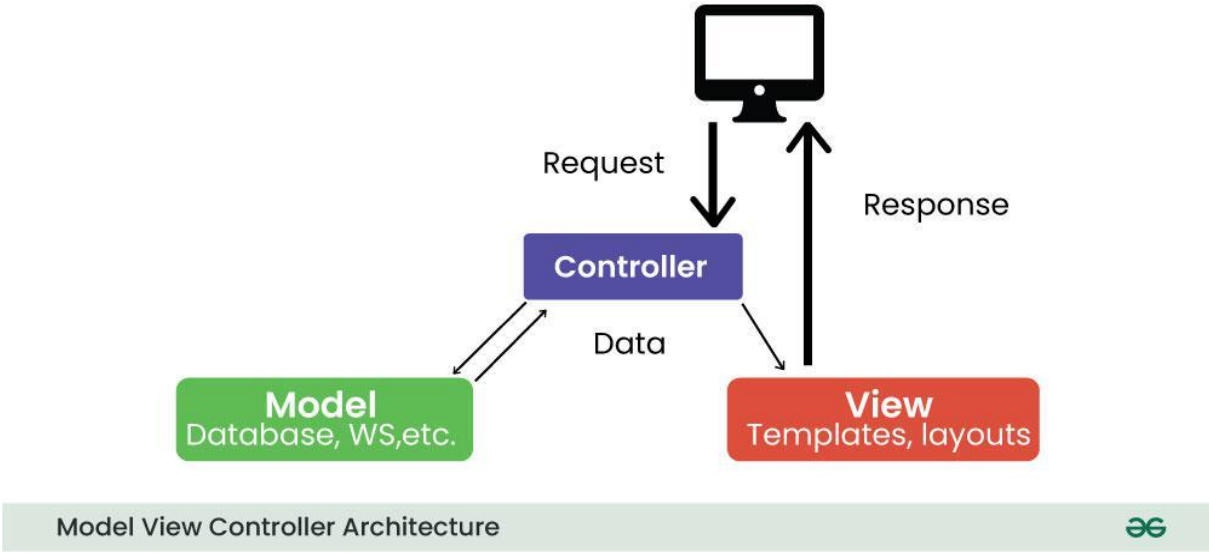

Figure 1

4.1.5 MVVM (Model-View-ViewModel):

- Similar to MVC but introduces a ViewModel layer that abstracts the View's state and behavior from the Model.
- Facilitates data binding between the View and ViewModel, enabling a more reactive UI.
- Commonly used in modern mobile app development, especially with frameworks like Android Jetpack and SwiftUI.

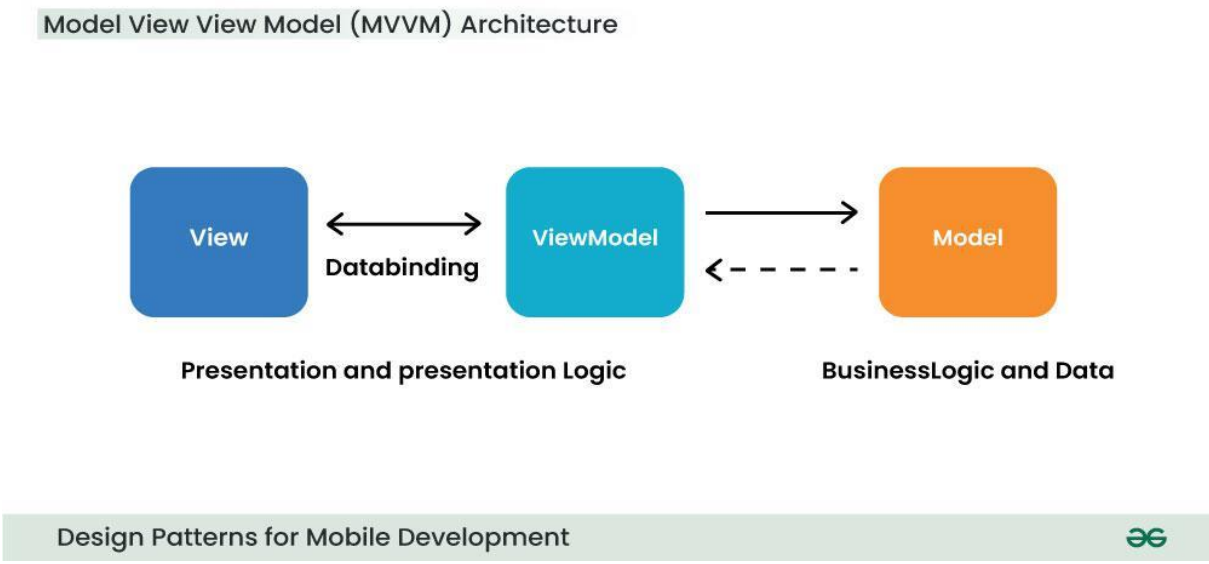

Figure 2

4.2 Mobile Application Design Patterns:

1. **Singleton**:
    - Ensures that a class has only one instance and provides a global point of access to it.
    - Commonly used for managing resources such as database connections, caches, and preferences.

2. **Factory**:
    - Defines an interface for creating objects but allows subclasses to alter the type of objects that will be created.
    - Commonly used for object instantiation, providing a way to decouple the creation of objects from their implementation.

3. **Observer**:
    - Defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.
    - Commonly used for implementing event handling and UI updates in response to data changes.

4. **Adapter**:

    - Allows incompatible interfaces to work together by converting the interface of a class into another interface that a client expects.
    - Commonly used for integrating third-party libraries or components with different interfaces.

5. **Decorator**:

    - Allows behavior to be added to individual objects dynamically, without affecting the behavior of other objects from the same class.
    - Commonly used for extending the functionality of objects at runtime, such as adding features to UI components.

6. **Repository**:

    - Abstracts the data layer and provides a clean API for accessing data from multiple sources (e.g., database, network).
    - Commonly used for separating data access logic from the rest of the application, promoting testability and maintainability.

7. **Dependency Injection**:

    - Allows objects to be injected with their dependencies rather than creating them internally.
    - Commonly used for managing object dependencies, improving code reusability, and facilitating unit testing.

5 How to collect and analyse user requirements for a mobile application(Requirement Engineering).

Various types of requirements for mobile apps:

- Business requirements: there 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. There 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. There are non-functional requirements and functional requirements that include technical requirements and technical specifications for the engineering team
- System requirements (functional and non-functional requirements)

5.1 Steps to gather requirements for a mobile application.

Step 1: Define your app idea and purpose:

Requirement gathering starts with a business idea where you need to clearly define the purpose of the mobile app.
To clearly define this, we use such questions as;
What purpose will the mobile app serve?
Does it offer a solution to a potential problem?

Step 2: Gather and align the app and business objective goals

This is a step that encourages you to gather business requirements to understand how the enterprise aligns with the idea from the first step.

Gathering business requirements to document involves these steps:

- 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.

5.1.1 Techniques used to gather stakeholder requirements for the business requirement

- Analyse similar documents
- Analyse similar external and internal interfaces
- Brainstorm use cases and user stories
- Create user stories and use cases
- Hold stakeholder focus groups
- Host requirements workshops
- Interview all the relevant stakeholders
- Observe documents and case studies
- Prototype visual examples for feedback
- Reverse engineer the processes
- Use online surveys/questionnaires
- Validate ideas with stakeholders

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 recognise 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.

- Next, determine what competitive advantage you hold over other apps. For example, the sample analysis shows that your product quality is far superior to others. The pricing is also better than competitors. Choose at least three elements in which you wish to compete with other apps.

Step 4: Determine scenarios and a user Personal

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

A user persona fictionalises 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)
- Behavioural 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 priorities functional and non-functional requirements.

Meanwhile, start prioritising the functional and non-functional requirements for the technical details, which also design use cases.

Next, you'll prioritise the non-functional requirements (NFRs) and functional requirements for an app.

Priorities determine the tech stack and importance of each function.

The MoSCow prioritisation technique helps with any requirements prioritisation before documenting the requirements.

The technique requires you to put every technical requirement into one of four categories:

- **Must Have** – The highest-level requirements are critical to the requirements document to ensure the project's success.
- **Should Have** – The second highest-level specifications are necessary for the project but won't delay the progress of development or success.
- **Could Have** – The medium-level specifications could enhance user experience but aren't dealbreakers if you don't develop them right away.
- **Won't Have** – The low-level requirements aren't important to stakeholders at the time of requirements documentation and won't affect the development process.

Step 6: Design Use Cases And User stories

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

Design them before documenting the specifications for the development company or team.

Use stories and cases to add visual representation to your documents for mobile app development documentation.

How to Design a Use Case for App Requirements Documents

A use case diagram lets everyone at the development company visualise 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.

How to Design a User Story for App Requirements Documents

A story evolves the user persona you wrote earlier. Creating user stories lets the development company design an app that meets acceptance criteria.

Transforming a user persona into a story means you must follow the INVEST acronym to guide the criteria.

In other words, every user story must meet the following factors:

- **Independent** – stories must be independent from each other.
- **Negotiable** – The *what* and *why* of stories should be concrete, while the *how* shouldn't be.
- **Valuable** – Stories should add value for all the stakeholders.
- **Estimable** – Each story should be capable of estimation.
- **Small** – Stories must be small enough for sprint completion.
- **Testable** – Stories must be testable with written acceptance criteria possible immediately.

Step 7: Write an App Requirements Document

Mobile application development relies on the requirements document to design proper flow or the best app features and hit the right target audience.

Step 7a: Formulate the app's idea statement

Every app requirements document should include an idea statement that lets every stakeholder and software developer understand the document before diving into the details.

Start your app requirements document with a simple single-sentence statement that aligns with the app's idea.

Step 7b: Document all the relevant app details

A detailed description of development plans in the requirements document is instrumental to completing the documentation.

A successful mobile app requirements document includes more than the details of an application and its functions for the development team.

Step 7c: Prepare a navigation sequence

The development team requires a simple navigation sequence they can follow during software development.

The mobile app requirements document outlines the sequence in which the software development process.

Add the details from step 7b in a sequence that every developer can easily grasp.

Step 7: Add requirements format for visuals

Add your user stories from a user's perspective and the use case overviews you designed to the app requirements document to help stakeholders and the development team understand every aspect of the app requirements document.

Successful software development means knowing the intended users.

Step 7e: Add cost optimisation details

The development team, stakeholders, and the client will appreciate a cost-benefit analysis to ensure cost optimisation throughout the software development process.

You'll find guidelines for cost optimisation in your business objectives or budget.

Business analysts also insist on adding a cost-benefit analysis to an app requirements document to meet the business needs and have a greater chance of success against competing apps.

Step 7f: Add communication protocols

Ensure another key element is in your document before delivering the mobile app requirements document to a project manager, stakeholder, developer, or operating environment.

Add communication methods for a collaborative process. Collaboration relies on dependable communication.

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. Use the feedback for the final step.

6. Estimating mobile app development cost

Estimating the cost of mobile app development can be tricky as it depends on several factors. Here's a breakdown of the key elements to consider:

6.1 Features and Complexity:

The number, complexity, and custom nature of features significantly impact cost. Simple apps with basic functionalities will be less expensive than those with intricate features, integrations, or custom animations.

6.2 App Platform:

Building for multiple platforms (iOS and Android) typically costs more than developing for a single platform. Cross-platform development frameworks can help reduce costs, but might have limitations in achieving a fully native feel.

6.3 Development Team:

The experience, location, and size of your development team affect the cost. Hourly rates for developers vary depending on their expertise and geographic location. Inhouse developers might seem more expensive upfront, but can offer better control and communication. Freelance or outsourced developers can be a cost-effective option, but require clear project scoping and communication.

6.4 UI/UX Design:

A well-designed user interface (UI) and user experience (UX) are crucial for a successful app. The complexity of the design and number of screens will influence the cost.

6.5 Backend Development:

If your app requires a robust back-end for data storage, user authentication, or complex logic, factor in the cost of back-end development and server maintenance.

6.6 Other Costs:

Consider additional expenses like app store fees, third-party API integrations, and ongoing maintenance costs.

6.6.1 Here's a general approach to estimating mobile app development cost:

- **Define App Requirements:** Clearly outline the app's features, functionalities, and target platforms.
- **Break Down Features:** Divide the app into smaller features and estimate the development time for each.
- **Choose a Development Model:** Decide between in-house, freelance, or outsourced development, considering their hourly rates.
- **Factor in Additional Costs:** Estimate costs for UI/UX design, back-end development, and ongoing maintenance.

6.6.2 Here are some resources that can help you estimate mobile app development costs:

- **Mobile App Development Cost Calculators:** Many online calculators provide a rough estimate based on features and platforms. (These are estimates and may not account for all factors)
- **App Development Agencies:** Consult with mobile app development agencies. They can provide a more accurate estimate based on your specific requirements.

Conclusion

The choice of the programming language and approach depends on factors like project requirements, performance considerations, developer expertise and target audience. Each programming language/ framework and design patterns has its strength and weaknesses, so it is essential to evaluate them based on the specific needs of the project.

References:

https://www.techtarget.com/whatis/definition/mobile-app
https://www.techopedia.com/definition/2953/mobile-application-mobile-app
https://aws.amazon.com/compare/the-difference-between-web-apps-native-apps-and-hybrid-apps/#:~:text=The%20term%20web%20app%20indicates,specifically%20for%20a%20mobile%20device.
https://www.mobiloud.com/blog/native-web-or-hybrid-apps