REPUBLIQUE DU CAMEROON
PAIX-Travail-Patrie
MINISTRE DE L'ENSEIGNEMENT SUPERIEUR

FACULTE DE L'ENGINERIE ET TECHGNOLOGIE

REPUBLIC OF CAMEROON
Peace-Work-Fatherland
MINISTER OF HIGHER EDUCATION

FACULTY OF ENGINEERING AND TECHNOLOGY

********** UNIVERSITY OF BUEA **********

DEPARTMENT OF COMPUTER ENGINEERING

COURSE: INTERNET PROGRAMMING AND MOBILE APPLICATIONS

COURSE CODE: CEF 440

TASK 1

Facilitator: Dr. Valery Nkemeni

APRIL 2024

GROUP MEMBERS

S/N	**Name**	Matricule Number
1	MUYANG ROSHELLA MBAMUZANG	FE21A243
2	TAMBONG KERSTEN MELENGFE	FE21A440
3	EBAI ENOWNKU JANE	FE21A176
4	AGBOR NKONGHO KELLY	FE21A126
5	NICCI NSE NCHAMI	FE21A268

1. **Major Types of Mobile Apps and their differences**

Mobile application software is an app that runs on a smartphone or tablet. Another defining characteristic that it has is portability. It is built for smaller devices hence they have limited capacity compared to desktop apps.

Mobile Apps can be grouped into four main types which are native, web, hybrid and progressive web apps.

i. Native Apps

These are apps that work only on one specific mobile platform or operating system (OS) using the platform's native programming language such as Swift/ Objective C for iOS, Java/Kotlin for Andriod. They have access to the full range of the device features and can deliver the best performance and user experience. They are typically downloaded from app stores like Google play store and Apple app store. Examples are Google maps, Spotify and iTunes.

Advantages:

- More efficient processing.
- Better performance speed.
- Provides a good user experience.
- Can access all hardware devices e.g microphone ,sensors etc.

Disadvantages:

- Can only run on a specific operating system.
- Maintenance cost is high.
- OS-exclusive apps take time build.

ii. Web Apps

These are mobile-optimized websites that can be accessed through a mobile browser. They are built using web technologies such as HTML, CSS and JavaScript. Web apps are platform independent and do not require installation from and app store. Examples include Amazon and Canva.

Advantages:

- Are cheaper and faster to build and release.

- No storage space problems.
- Easy to update and maintain.
- Accessible from anywhere via any mobile browser.

Disadvantages:

- Is web browser dependent.
- Needs internet connection.
- Has limited functionalities.

iii. Hybrid Apps

They combine the capabilities of a native app and a web app. They are built using the web technologies and are wrapped in an active container that allows them to be installed and run on mobile devices. Hybrid apps leverage frameworks like Apache Cordova /PhoneGap or Ionic to access native device features while using web-based UI components. Examples include Instagram, Facebook, LinkedIn and Uber.

Advantages:

- Have a quicker development lifecycle.
- Provides benefits of native and web app.
- Is cost-effective.
- Cross-platform functionality.

Disadvantages:

- Slower processing speed.
- Limited access to the hardware features.
- UI is less seamless.

iv. Progressive Web Apps

These are web apps with functionalities of native apps. They are designed to be responsive, adapting to different screen sizes and can be launched directly from the home screen without the need to go through a browser. These functionalities ensure a consistent user experience. Technologies used are AngularJS, Lighthouse and Polymer. Examples are Pinterest, Tinder and Trivago.

Advantages:

- More efficient in processing and loading data.
- Automatically updated whenever a user opens them.
- Perform well and run fast on all operating systems.

Disadvantages:

- There are potential problems with hardware integration.
- No full access to hardware features.
- UI may vary depending on the web browser.

2. Programming Languages of mobile applications.

There are several programming languages to choose from, each with its own strengths and weaknesses. Below is a review of some popular programming mobile app programing languages.

a) **Java**

Java is an object oriented and class-based programming language. Java programs are designed to "write once, run anywhere" and it is the official language of the Android operating system. Java applications instruct the syntax of Java keywords and packages and is widely used for numerous developments.

Strengths:

- Well suited for large, complex tasks.
- There is an extensive amount of resources and libraries available, so it's unusual for a developer to have to create new code in problem-solving.
- It is platform-independent and can be used for server-side development as well.
- Provides excellent performance.

Weaknesses:

- Has a slower development lifecycles compared to languages with modern tooling.
- Has some outstanding inefficiencies, which have been solved by other languages like Kotlin.
- Can be expansive and have a steeper learning curve compared to some modern languages.

b) **Swift**

Swift is a native language for iOS. It offers better performance to Objective-C, the older language used for iOS development. It is an object-oriented language that streamlines coding and is more high level towards readable English.

Strengths:

Strengths:

- Easy to read as a language.
- Due to its interactive nature, Swift makes it simpler to catch errors while coding, thus saving overall development and debugging time.
- Swift provides a great user experience and often requires less on-device memory.
- Has good support for concurrency and asynchronous programming.
- Better performance compared to Objective-C.

Weaknesses:

- Is not compatible with other operating systems.
- Has a relatively smaller community compared to other languages.

c) **Kotlin**

Kotlin is an object-oriented and statistically typed language that was unveiled by JetBrains. This codebase has become the new preferred language for Android app development. It was designed to run on the Java Virtual Machine, thus allowing Kotlin apps to run on any machine that supports the Java runtime environment.

Strengths:

- This language is interoperable with Java, making it easy for all Java-based programming to be expanded upon and improved with Kotlin.
- It provides native-level support while still running on multiple platforms.

- It offers concise syntax, null safety and enhanced developer productivity.
- It provides excellent tooling and integration with Android Studio.

Weaknesses:

- Since Kotlin is a newer language, there can be less information and limited resources on it.
- Is limited to android platform development.

d) **Objective-C**

Objective-C is an object-oriented programming language used in the development of iOS mobile applications. It was the primary language for iOS development before the introduction of Swift.

Strengths:

- Works incredibly well within the Apple ecosystem and supports older versions of iOS.
- Its maturity as a language has created a wealth of resources for it.
- It supports interoperability with Swift, C and C++.

Weaknesses:

- Some inadequacies remain that are fixed in newer languages like Swift.
- Apps built using Objective-C are more vulnerable to attacks.
- Has a cumbersome syntax compared to more modern languages.

e) **C#**

It is an object-oriented language with native support for Android app development. Initially, a drawback was that it could only run on Windows systems, but with the invention of the cross-platform framework Xamarin, C# can now run on any platform.

Strengths:

- It provides a near-native level of performance and speed on iOS and Android.
- C# Xamarin has simplified maintenance, which helps in minimizing continuing development hours.

Weaknesses:

- There can be a comparative lag in API calls.
- It's not a great choice for apps with a complex UI.

f) **JavaScript**

JavaScript is one of the older programming languages still commonly used, so it is time tested. React Native is a cross-platform codebase that works with it, which supports both frontend and backend development and has a large community of developers behind it.

Strengths:

- Its longtime use means there are a lot of frameworks, patches, and support available.
- It is the basis for some cross-network coding languages, like React Native, which is growing in popularity.
- JavaScript delivers very fast results.

Weaknesses:

- Code is viewable by client hence language is more prone to malicious attacks.
- Limited access to certain platform-specific features and APIs.

g) **Python**

Python is one of the most popular languages, as it's a general-purpose language that powers many versatile purposes. The cross-platform codebase uses the framework Kivy for app development.

Strengths:

- There are a ton of libraries and toolkits that support Python – especially for big data and researchers.
- Python is a very readable language and many developers know how to use it, so it's easy to hire someone who specializes in it.

Weaknesses:

- It doesn't perform well in some high-performance specialized tasks.
- Python is not native to either iOS or Android, so for mobile app deployment specifically, it can cause some cross-platform inconsistencies.

h) C++

It is an object oriented language that improved upon any inefficiencies in the older, yet popular language "C." It's very adaptable, in that a developer can write the code once, then run C++ on any platform that has a compiler for the language. It can become a cross-platform codebase by using Microsoft's tools in Visual Studio.

Advantages:

- 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.

Weaknesses:

- It is a very complicated language to learn.
- Memory is not being automatically managed by the application.

3. Mobile App Development Frameworks.

Frameworks are the backbone of mobile app development. One can either choose a native or cross platform app when choosing a framework, although they both have their individual pros over one another, cross platform applications work on multiple devices regardless of the OS.
There are four main frameworks for mobile app development namely: Flutter, Xamarin, Ionic and React Native.

i. Flutter

Flutter is an open-source UI toolkit developed by Google for building natively compiled apps. It uses the Dart programming language.

Key Features:

- Has best UI toolkit that comes with a rich set of customizable widgets.
- Supports iOS and android platforms.
- Comprehensible and precise.
- Allows for faster development.
- Visuals are appealing and engaging.
- Ability to develop high performance apps.
- Rich motion APIs.
- Relatively easy to learn.
- Offers excellent performance.

Use Cases:

- Flutter is suitable for building cross-platform apps that require a high level of customization and performance.

ii. Xamarin

Xamarin is a framework owned by Microsoft that allows you to build cross-platform mobile apps using C#. It provides access to native APIs and UI contols for each platform.

Key Features:

- Allows code sharing on multiple platforms and is known to offer a development ecosystem with a back-end, API and components.
- Faster development time.
- Lesser bugs.
- Best backend infrastructure.
- A component store which contains various cross-platform libraries, third party libraries and UI controls.
- Allows application indexing and deep linking.
- Has a steeper learning curve compared to other frameworks.
- Offers native performance as it compiles down to native code.

Use Cases:

- Xamarin is suitable for building apps that require access to platform-specific APIs and native performance
- Good choice for apps that need to tightly integrate with the underlying platform, have complex business logic and require a high level of code sharing.

iii. Ionic

Ionic is a popular open-source framework that for building hybrid mobile apps using web technologies such as HTML, CSS and JavaScript. It uses a WebView to render the app's UI in a native container.

Key Features:

- Used for building interactive hybrid mobile and progressive web apps with cross-platform apps.
- Fast and powerful.
- Allows complete control over app building.
- Allows development of both native and progressive web apps.
- Easy to learn and handle.
- Allows for a single code based app building.

- Poor performance compared to other frameworks.

Use Cases:

- Ionic is suitable for building cross-platform apps that primarily rely on web technologies and require a fast development cycle.
- Suitable for apps with relatively simple UI requirements.
- Suitable for apps that prioritize code reusability and rapid prototyping.

iv. React Native

This is a widely used java script library that helps to develop rich apps to give the best user experience. It allows developers to create platform-specific versions of various components.

Key Features:

- Single code base across multiple platforms.
- Low code requirement.
- Declarative API for productive UI.
- Compatible with third party plugins.
- Supports both iOS and Android.
- Provides a rich set of pre-built components and has a large developer community.
- Allows for faster development.
- Offers good performance.

Use Cases:

- Is suitable for building apps that need to run on both iOS and Android platforms.
- Good choice for apps that prioritize development speed, code reusability and have less complex UI requirements.

- Great for projects that need heavy interactivity.

4. Architectures and Design Patterns of Mobile Applications

A mobile application architecture is a combination of design and techniques used to build a mobile application ecosystem. When a mobile application is developed, the first step is to design its architecture. The architecture is like a foundation to the mobile application. Every successful application has a good architecture that forms its structure and features. It is a collection of UI/UX, dataflow, tech stack and everything that makes an application functional.

4.1. **Importance of a Good Mobile Application Architecture**

i. Speed and quality:

The speed and quality of your mobile application strongly rely on the underlying architecture. Undoubtedly, architecture defines the way different components interact and behave to make the whole app functional.
Architecture is the backbone of an application that provides shape and structure. Users like apps that are smooth-functioning and intuitive, and they hate laggy and unresponsive apps. An efficient architecture yields a superior user experience by providing quick app response.

ii. Compatibility:
If you want a mobile application that is compatible with different devices and mobile platforms, choosing the right architecture is vital. Different architectures for mobile apps provide different levels of compatibility with diverse systems. Compatibility refers to a mobile app's ability to share and receive data from other systems or perform intended functions without restrictions to interfaces. With a solid architecture, you can build a system that can interact with other systems and provide the requisite compatibility.

iii. Scalability:

Building an application without a defined architectural pattern makes it difficult to add new features without compromising the app's integrity. Mobile application architecture is crucial for scalability and adaptability. Well-defined architectures enable easy integration of additional features and modifications. They also ensure optimal performance and reliability as the user base grows. Scalable architecture is vital for accommodating an expanding user base.

4.2. The Key Elements in Mobile Application Architecture Designing

You need to create a robust architecture for your app that can stand the test of time. Creating it requires consideration of different factors as follows.

i. Determine the Device Requirements

In order to build a solid architecture for your mobile application, you should consider the device and hardware specifications you wish to target. It would require an understanding of the resolution, screen size, memory, CPU, storage, and other characteristics of the target devices, along with the development environment and tools. Since the application depends on the device features and hardware for its functioning, it’s crucial to get the details of the device it would run on.

ii. Handle Network Fluctuations

Through the lifecycle of your mobile application, it will encounter cases when the internet connection will be weak or fully unavailable. Hence, your app should be ready to handle the worst network conditions. It is crucial to create an application architecture that is capable of dealing with any state of network connections.

iii. User Interface

The importance of UI/UX cannot be overstated and it should be designed to keep users engaged with your application and provide a seamless experience. When it comes to creating an app architecture diagram, you should include the aspects of UI/UX to craft an app that resonates with your audience.

iv. Push Notifications

When formulizing your mobile app information architecture, determine if your users require real-time updates or notifications. Although push notifications are attractive, they are also expensive and drain phone data and battery.

4.3. The Different Layers of Mobile App Development Architecture

The architecture of a mobile application has multiple layers that communicate with each other and pass on data to trigger various functionalities. The following are the common layers for most mobile apps.

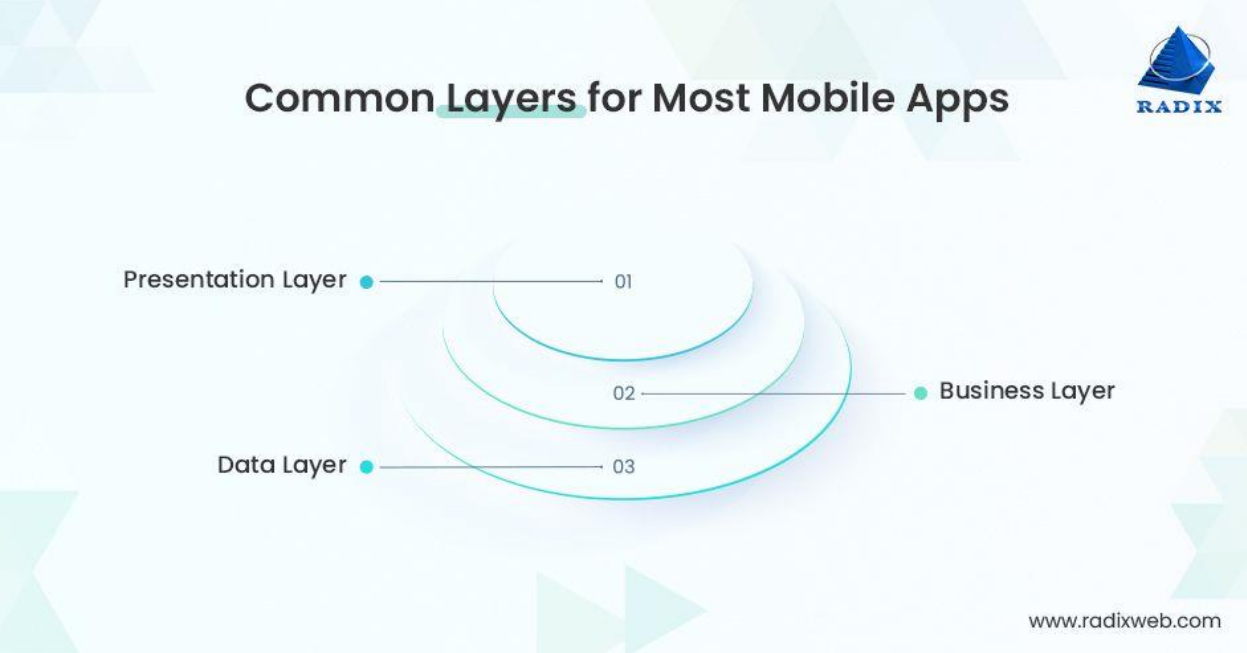

Fig 1: Common layers for most mobile apps.

I. Presentation Layer

The presentation layer sits on top of an app architecture stack and defines how an application will be presented to the end users. It's a user interface and communication layer where the users interact with the application. It offers presentation services collecting and displaying user data to the end users. The presentation layer is formed from the UI/UX of a mobile application.

Simply put, this is the layer that handles user interactions and represents the face of a mobile application where users can find all kinds of GUI elements.

II. Business Layer

This layer is related to the business logic for the application. It provides a set of rules and algorithms that manage the flow of data in the context of a business. Hence, the business layer defines how data will be generated, processed, stored, and used for a mobile application.
The primary function of this layer includes logging, data caching, security, data validation, and exception management. Depending on the operations of a mobile app, the business layer can exist on a device or on a server.

III. Data Layer

Mobile applications need a secure and efficient mechanism for data transactions. This responsibility is handled by the data layer which ensures the transfer and receipt of data seamlessly. This layer consists of various components like service agents, data access components, data utilities, etc, to enable data transactions within an app. When it comes to designing the data layer, the mobile app developer needs to consider the ease of modification as per the change in requirements and maintenance.

4.4. Some Real World Mobile Application Architecture

- Android mobile app architecture
- IOS mobile app architecture
- Hybrid mobile app architecture

a. Android Mobile App Architecture

Mobile apps that are specially developed for Android devices are native apps. Native apps are mobile apps developed specifically for a particular operating system like Android. There are a variety of manufacturers when it comes to Android devices. So, these apps have to meet different requirements and use Java and Kotlin languages. Thus, a single architecture will not work for Android app development. So the architecture used in android mobile development is clean

architecture. The clean architecture deals with layers and inversion of code principles. Each layer which was stated above is independent of each other and exchange data through interface.

Clean architecture offers benefits like:

- Easy testing and troubleshooting
- UI is separated
- Not dependent on external libraries, frameworks, and databases

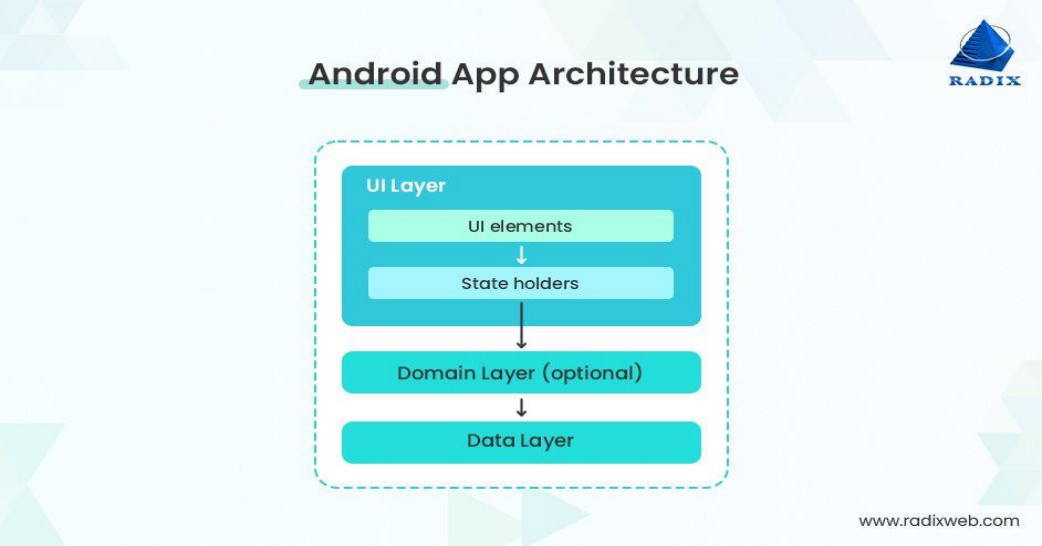

Fig 2: Android App Architecture.

b. iOS Mobile App Architecture

Objective-C and Swift are two languages to develop native iOS applications. While there are no specific guidelines for Android apps, Apple guides on developing apps using an architecture with an MVC pattern. Although iOS developers can choose any architecture, it's the most common pattern for IOS mobile app development. MVC is a three-tier architecture for mobile applications. Here are the different layers it has:

- **Model:** The model layer manages the data, rules, and logic of an application and consists of model objects, parsers, networking code, etc.
- **View:** It's the presentation layer for MVC architecture. There are elements for user interactions, and it doesn't have any business logic.
- **Controller:** This layer establishes communication between the two other layers that are the Model and the View layers.

MVC architecture provides benefits like:

- Faster app development process
- Clear communication between different layers
- Easy to maintain and simple app architecture

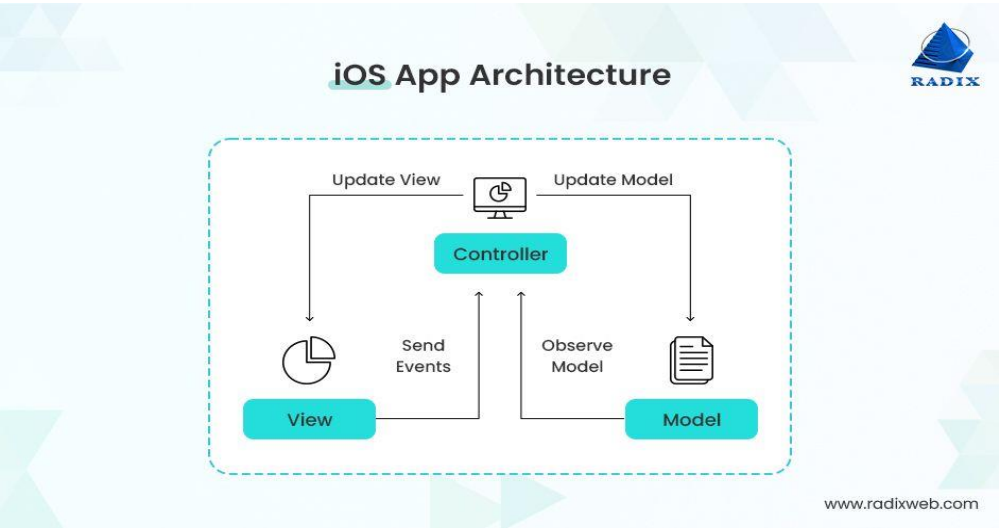

Fig 3: iOS app architecture

c. Hybrid Mobile App Architecture

These mobile apps are a hybrid of native and web applications. These apps have a native app shell in which web-based content is loaded. Web technologies are used to build Hybrid mobile app like HTML, CSS, and JavaScript. These apps can access the hardware features of a mobile app.

Hybrid mobile apps use a single codebase for different mobile platforms. So, a single hybrid app will function on different platforms whether Android or iOS.

Hybrid app architecture has benefits like:

- Faster development time
- Single codebase for different platforms
- Cost-effective and cross-functional
- Access hardware features

d. Cross-platform App Architecture

Another popular mobile app development architecture is cross-platform apps. Just like hybrid apps, there is a single codebase for various platforms, and they are compatible with Android and iOS. But they are built on frameworks.
There are several cross-platform app development frameworks technologies like Flutter, Ionic, Xamarin, React Native, etc. Compared to hybrid apps, these apps offer an experience that is much closer to native apps, and they are better in performance.
Cross-platform apps provide benefits like:

- Single codebase for multiple platforms
- Native app-like performance and quality
- Access to hardware features
- Faster loading speed
- Cost-effective than native apps

4.5. Key Attributes of a Result-Driven Mobile App Architecture

A mobile app architecture needs to be flexible, extensible, modifiable, and possess other qualities for a stable and robust mobile application. Here are some key characteristics of a well-designed and impeccable app architecture.

- Reusability

With code reusability, developers can develop and deploy application features at a faster speed. An efficient mobile app architecture is built with code reusability in mind and allows developers to quickly build and release application updates and versions.

- Sustainability

Changes in the environment where a mobile app resides are uncontrollable. For instance, technological changes or a shift in market demand. Hence, your app needs to be ready to adjust according to these changes. It calls for a mobile app architecture design that is resilient to these changes. A sustainable mobile app is capable of handling changes like improved technologies and servers.

- Performance

Mobile app developers prioritize user experience and functionality when it comes to app development. Users expect that applications, whether Android or iOS, should respond to their taps instantly and execute tasks. Hence, choose an architecture that offers excellent performance.

- Extensibility

Changes in user needs require you to integrate more functionality over time. If your app is built on a flexible architecture, it will be easy to implement new features, or it will be a complex process otherwise. Hence, dividing app components to form a loosely coupled architecture will simplify the integration of features.

- Scalability

An application architecture that supports easy scalability to accommodate new users and increased load on your app.

- Security

In the modern world, security has become so vital that big businesses spend millions to beef up security. Consider an app architecture that provides a high level of security and ensures compliance with various standards.

- Testability

If an app has high testibility, it will be more reliable as the chances of bugs in it are low. A testable architecture makes it easy to discover and fix errors or bugs early in the development phase, making the final application well-performing and versatile.

- Intuitiveness

Apart from making your app flexible and scalable, you should also focus on user experience. Build your app with a user-friendly interface that is easy to understand.

4.6. **What are Mobile App Architecture Best Practices**?

From performance to scalability, all vital features of your mobile app depend on its architecture. Therefore, it is important to choose the right architecture that meets

the requirements and offers excellent user experience. It will help to save money and precious time by eliminating reworks in the future.

A good mobile app architecture diagram can be started with these best practices.

Choose an architecture that separates the UI layer from the data layer. It will provide reusable code and simplify the process of change.

Work with good software engineering best practices and principles. Follow principles like KISS, YAGNI, SOLID, DRY, etc, when defining the architecture and developing the app.

For APIs, you can leverage simple and lightweight data formats such as JSON.

Stay updated with the current mobile app development trends to develop an application as per modern user needs.

4.7. **How to Choose a Good Mobile Application Architecture**?

Now that the importance of mobile application architecture is stated, the pros and cons of different architectures must be weighed to pick the best one. Some points stated below can help to choose a good architecture

Developing native applications should be considered if you don't have a shoestring budget. It offers many advantages including intuitive functionality and performance.

Does your target audience include Android and iOS users? Choose a cross-platform app or create native apps that will serve users on various platforms.

You can choose a hybrid solution to help access your brand from a more diverse set of devices.

There are different principles and architectural patterns when it comes to developing mobile apps, each with its own strengths and limitations. You cannot compromise on your app architecture because it's the foundation of your app and its performance and stability depend on the architecture.

The decision of choosing an architecture for your mobile app is influenced by several factors including business requirements, application type and functionality, tech stack, flexibility, development timeframe, etc.

You must evaluate different app architectures based on your needs and constraints to pick the right architecture for your mobile app.

5. Requirement Engineering

Requirement Engineering refers to the process of defining, documenting and maintaining the requirements in the engineering design process. In this section, we are going to discuss how user requirements for a mobile application are collected and analysed.

5.1. **Collection of user requirements**

a) Surveys:

- Design and distribute surveys to a representative sample of potential users.
- Include questions about user demographics, preferences, needs, and pain points related to the application. Use online survey tools like SurveyMonkey, Google .
- Forms or Typeform to reach a larger audience.

b) Interviews:

- Conduct one-on-one or group interviews with users to gain deeper insights into their behaviors, preferences, and challenges.
- Prepare a list of open-ended questions to encourage users to express their thoughts and ideas freely.

- Consider using video or voice calls for remote interviews if face-to-face meetings are not feasible.

c) Stakeholder Workshops:

- Organize workshops with key stakeholders, including clients, product managers, developers, and end-users, to collaboratively define and prioritize requirements.
- Use techniques like brainstorming, affinity mapping, and user story mapping to elicit and organize user needs and preferences.
- Facilitate discussions to reach a consensus on the most important features and functionalities for the application.

d) Prototyping and Usability Testing:

- Create prototypes or mockups of the mobile application to visualize proposed features and workflows.
- Conduct usability testing sessions with users to gather feedback on the prototype's design, usability, and functionality.
- Use tools like InVision, Adobe XD, or Figma to create interactive prototypes for testing.

e) Competitive Analysis:

- Analyze competing mobile applications to identify features, functionalities, and user experiences that resonate with users.
- Look for gaps or areas where competitors may be falling short, which can inform the development of unique selling points for your application.
- Consider conducting user surveys or interviews specifically focused on users' experiences with competing products.

f) Social Media Listening:

- Monitor social media platforms, online forums, and community groups related to the application's domain to gather feedback and insights from users.

- Look for mentions of the application, discussions about user needs and preferences, and common pain points or feature requests.
- Use social media monitoring tools like Hootsuite, Sprout Social, or Brandwatch to track relevant conversations and sentiment.

5.2. Analysing User Requirements

Analyzing user requirements is essential for understanding the needs, preferences, and constraints of the target users, as well as for ensuring that the resulting product meets their expectations. Here's how you can effectively analyze user requirements:

i. Organize and Prioritize Requirements:
- Start by organizing the collected requirements into categories, such as functional, non-functional, and technical requirements.
- Prioritize requirements based on their importance, feasibility, and impact on the overall application.
- Use techniques like MoSCoW (Must have, Should have, Could have, Won't have) prioritization to categorize and prioritize requirements.

ii. Validate Requirements:
- Verify that requirements are complete, consistent, feasible, and verifiable.
- Address any ambiguities or conflicting requirements through further discussions and clarification.

iii. Analyze Dependencies and Relationships:
- Identify dependencies and relationships between different requirements to understand their interdependencies.
- Determine how changes to one requirement may affect other requirements and the overall system.
- Use tools like dependency structure matrices (DSMs) or network diagrams to visualize and analyze dependencies.

iv. Assess Feasibility and Constraints:

- Evaluate the feasibility of implementing each requirement within the project constraints, including budget, time, and technical limitations.
- Consider factors such as available resources, technology stack, and regulatory requirements that may impact the implementation of certain requirements.
- Identify any constraints or limitations that may require adjustments to the requirements or project scope.

v. Identify Risks and Mitigation Strategies:

- Identify potential risks and uncertainties associated with the user requirements and their implementation.
- Assess the impact of these risks on the project timeline, budget, and overall success.
- Develop mitigation strategies to address identified risks and minimize their impact on the project.

vi. Define Acceptance Criteria:

- Define clear and measurable acceptance criteria for each requirement to ensure that it meets the desired quality standards.
- Specify the conditions under which a requirement will be considered satisfactorily implemented and accepted by stakeholders.
- Use acceptance criteria to validate and verify the implementation of requirements during testing and quality assurance activities.

vii. Prototype and Validate:

- Create prototypes or mockups of the application to visualize and validate the proposed features and workflows.

- Conduct usability testing sessions with users to gather feedback on the prototype's design, usability, and functionality.
- Use the insights gained from testing to refine and iterate on the requirements as needed.

viii. Document Analysis Results:

- Document the results of the requirement analysis process, including any findings, decisions, and recommendations.
- Ensure that the documentation is clear, comprehensive, and accessible to all stakeholders involved in the project.
- Use tools like requirement management software or collaboration platforms to document and track analysis results.

ix. Iterate and Refine:

- Requirement analysis is an iterative process, so be prepared to revisit and refine requirements throughout the project lifecycle.
- Continuously gather feedback from stakeholders and end-users to incorporate changes and improvements.
- Update and adjust requirements as needed based on evolving project needs and priorities.

6. **How To Estimate Mobile App Development Cost**

Estimating the cost of mobile app development can vary depending on several factors such as complexity, features, platform, development team rates, and location. Here are the general steps to estimate the cost:

a) **Define Requirements:** Clearly outline the features and functionalities you want your app to have. The complexity of these features will greatly impact the cost.

b) **Choose Platform:** Determine whether you want to develop for iOS, Android, or both. Developing for multiple platforms will increase the cost.

c) **Design:** Design includes UI/UX design, which affects the user experience. The complexity and quality of design can affect the cost significantly.

d) **Development:** This involves actual coding of the app. The complexity of the features, technology stack, and platform(s) chosen will impact the cost.

e) **Testing:** Testing is crucial, to ensure the app works smoothly across different devices and platforms. Testing can be iterative and may take time, affecting the cost.

f) **Deployment:** This involves submitting the app to the respective app stores (Apple App Store, Google Play Store). There might be associated costs with app store submissions.

g) **Post-launch Maintenance:** Consider ongoing maintenance, updates, and support after the app is launched. This may include bug fixes, feature updates, and support for new OS versions.

h) **Cost Breakdown:** Break down the cost based on different components such as design, development, testing, and deployment. Also, consider whether you're hiring freelancers, an agency, or an in-house team.

i) **Get Quotes:** Reach out to different app development companies or freelancers to get quotes based on your requirements.

j) **Consider Location:** Development costs can vary significantly based on the location of the development team. Rates in North America and Western Europe are generally higher compared to other regions.

k) **Contingency:** Keep a contingency budget for unforeseen circumstances or additional features that might be required during the development process.

l) **Use Cost Estimation Tools:** There are online tools available that can help estimate the cost based on your requirements and chosen platform. However, these are rough estimates and actual costs may vary.

REFERENCES

1. Bitstudios website (https://www.bitstudios.com/blog/types-of-mobile-applications/).
2. A review of Apps for programming PDF by Wei-Ling Wu.
3. Kelton Tech Blog (https://www.kellton.com/kellton-tech-blog/top-mobile-app-development-frameworks).
4. Radix.com (Mounil Shah, 2023) Mobile App architecture.