THE UNIVERSITY OF BUEA

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER ENGINEERING

CEF 440: INTERNET PROGRAMMING AND MOBILE PROGRAMMING

Group 22

Task 1

Name	Matricule	Option
Enow Myke-Austine Eta	FE21A183	Software
Mokfembam Fabrice Kongnyuy	FE21A240	Software
Ndong Henry Ndang	FE21A248	Software
Niba Verine Kajock	FE21A267	Network
Takem Jim-Rawlings E.	FE21A309	Software

COURSE INSTRUCTOR: **Dr. Nkemeni Valery**

Table of content

1. Major types of mobile apps and their differences

1.1 What is an application?

An application is software that helps you exchange information with customers and help them complete specific tasks.

1.2 What is a mobile application?

A mobile application is an application that is designed to run on mobile devices.

1.3 Types of mobile apps

Mobile apps can be categorized into three major types: native apps, progressive web apps (PWAs), and hybrid apps. Each type has its own characteristics and trade-offs.

1.3.1 Native Applications

These are apps are designed and optimized for a particular operating system or platform.

A. Characteristics

- They have access to all the device features.
- They are built using native languages such as Java or Kotlin for android and swift or objective-c for ios.
- They can be accessed by downloading either from play store or App Store.

B. **Advantages**

- Highest performance and responsiveness as apps are optimized for the specific platform.
- Full access to native device features and APIs.

- Enhanced user experience and platform-specific design.

C. **Disadvantages**

- Higher development and maintenance costs as separate codebases need to be maintained for each platform.
- Longer development timeframes due to platform-specific development.
- Less code reusability between platforms.

1.3.2 Hybrid Applications

These are mobile apps built using web technologies (HTML, CSS and JavaScript) and then wrapped in a native container.

A. Characteristics

- They can be accessed by downloading from either Google play store or app store.
- They access native device features via plugins.

B. Advantages

- Code is written once and can be deployed across multiple platforms/operating systems.
- Faster development time as a single codebase is used.
- Less costly compared to native apps
- Easier maintenance and updates as changes can be applied universally.

C. Disadvantages

- Slightly lower performance compared to native apps due to the WebView layer.
- Limited access to native device features, requiring the use of plugins or additional libraries.

- UI may not look and feel entirely native, potentially impacting user experience. Also there are usually issued achieving the same level of UI/UX across both platforms. E.g. for react native, when you give a text a border radius, it applies in android but not in ios as it doesn't support border radius for text elements.

1.3.3 Progressive Web App (PWA)

A PWA is a type of web app that can operate as a website and a mobile application on any device.

A. Characteristics

- It's aimed at delivering a high performance website with an app-like user experience. A pwa can hide all of the browser controls like the url toolbar, to give it the exact feel of a native app.
- They are built using technologies such as HTML, CSS and JavaScript and are built using a single codebase.
- They are accessed via the browser, but can be saved into your device for use.
- They have limited access to device features compared to native apps.
- They make use of service workers which helps them run offline.
- They enable push notifications (i.e. they can send the device notifications even when the app is not open) and even make use of some device features such as camera and geolocation.
- They are responsive, and can work on any device e.g. mobile, laptop, and desktop.

B. Advantages

- Cross-Platform Compatibility: PWAs are built using web technologies (HTML, CSS, and JavaScript), making them compatible with multiple platforms and devices. They can run on various web browsers, including desktop and mobile platforms, providing a broader reach to your audience.

- PWA are written in a single codebase, so it has a lower costs and development time.
- Easy Deployment and Updates: PWAs can be accessed directly through a URL without the need for app store distribution. This makes deployment easier and allows for instant updates. Users can always access the latest version of the app without needing to download and install updates from an app store.
- Offline Functionality: PWAs can work offline or in low connectivity environments. Service Workers, a key component of PWAs, enable caching of app assets and content, allowing users to access the app even without an internet connection.
- Discoverability: PWAs can be indexed by search engines, making them discoverable through web searches. This can help improve the visibility and reach of your app.
- App-Like Experience: PWAs can provide an app-like experience by utilizing the Web App Manifest. They can be installed on the user's home screen and launched like a native app, giving users a more immersive and familiar experience.

C. Disadvantages of PWAs

- Limited Access to Native Capabilities: PWAs have limited access to native device features and APIs compared to native apps. While PWAs can leverage some device capabilities like geolocation and push notifications, they may not have access to all the features provided by the underlying operating system.

- Performance: PWAs may not always match the performance of native apps, especially for computationally intensive tasks or graphics-intensive applications. The use of a WebView and the reliance on browser capabilities can introduce some performance overhead.

- Limited App Store Distribution: PWAs cannot be distributed through traditional app stores like the Apple App Store or Google Play Store. While they can be discovered through search engines, they may not benefit from the app store's built-in discoverability and monetization features.

- Limited iOS Support: While PWAs are compatible with most modern web browsers, iOS has historically had limited support for certain PWA features. However, iOS support for PWAs has been improving over time.
- User Awareness and Adoption: PWAs are still relatively new compared to traditional native apps. User awareness and adoption of PWAs might be lower, and users may have varying expectations and familiarity with PWAs compared to native apps.

1.4 PWA vs Web App

1.4.1 Web App

- A web app is a software application that is accessed and used through a web browser and are typically built using web technologies such as HTML, CSS, and JavaScript.
- They require an active internet connection to function as they rely on server requests to retrieve data and update the user interface.
- Web apps can be accessed across different devices and platforms with a compatible web browser.
- They are not installed on the user's device and do not have access to native device features unless specifically supported by the browser.

1.4.2 Progressive Web App (PWA)

- A PWA is a type of web app that leverages modern web technologies to provide an app-like experience.
- PWAs are built using the same web technologies as web apps (HTML, CSS, JavaScript), but they incorporate additional features and capabilities.
- PWAs can be installed on the user's device and appear as icons on the home screen, providing a more app-like experience.
- They can work offline or with limited connectivity by utilizing service workers, which enable caching of app resources and offline functionality.
- PWAs can send push notifications to users, even when the app is not actively open.

- They can access certain native device features, such as camera, geolocation, and device sensors, using browser APIs like the Web API.
- PWAs are designed to be responsive and work across different devices and screen sizes.

1.5 Making a choice (Native or Hybrid or PWA)

What should you consider before choosing between Progressive Web Apps (PWAs), hybrid apps, or native apps? Here are some considerations to help you decide which approach to choose:

A. **App Complexity:** If your app requires complex and resource-intensive features native app development may be a better choice. Native apps offer the highest performance and full access to platform-specific capabilities.

   For simpler apps with basic functionalities, PWAs or hybrid apps can be suitable options as they leverage web technologies and provide a cost-effective and faster development process.

B. Target Platforms: If your target audience is predominantly using a specific platform (e.g., iOS or Android), developing a native app for that platform may provide a better user experience and take advantage of the platform's specific design guidelines and features.

   PWAs and hybrid apps offer cross-platform compatibility, allowing you to reach a wider audience across different platforms with a single codebase.

C. **Development Resources and Time Constraints:** If you have limited development resources or tight time and cost constraints, PWAs or hybrid apps can be more efficient choices as they allow code sharing and faster development cycles.

D. **Offline Capabilities:** If your app needs to work offline or in low connectivity environments, PWAs with Service Workers are a good option. They can cache app assets and content, enabling offline access and a seamless user experience.
   Hybrid apps can also provide offline capabilities through the use of plugins or frameworks like Ionic or React Native, but their offline support may not be as robust as PWAs.

E. **App Distribution:** If you want to distribute your app through app stores (e.g., Apple App Store, Google Play Store) and leverage their discoverability and monetization features, native or hybrid apps are the way to go.
   - PWAs can be accessed directly through web browsers, eliminating the need for app store distribution.

F. **Update and Maintenance:** PWAs offer the advantage of easy and instant updates as changes can be applied universally without requiring users to download and install app updates. This can be beneficial for fast iterations and bug fixes.
   Native and hybrid apps require users to download updates from the app stores, which can introduce some delays in delivering updates to users.

2. Comparing mobile app programming languages

Mobile app development involves using different programming languages depending on the platform and development approach. Below are a list of programming languages for mobile development and their frameworks, categorized into native, hybrid, PWA's and cross platform.

2.1 Native apps

- **Swift (ios):** it is supported by apple and is the preferred programming language for ios devices.
- **Objective-c (ios)**: it is an object oriented programming language, and was the preferred programming language for ios before swift..

- **Kotlin (android)**: it is a statically-typed programming language supported by Google as a first-class language for Android development.
- **Java (android)**: ava is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible.
- Example AirBnB (Swift and Kotlin)

2.2 Hybrid apps

The core hybrid languages are HTML, CSS and JavaScript / Typescript, but it includes the frameworks below

- **React native**: it is an open-source UI software framework which uses the React library along with native platform capabilities. It interacts with platform specific API's through a JavaScript bridge. It does not compile to native code but rather interprets it at runtime.
- **Ionic**: Ionic an open-source framework for building mobile applications using web and Ionic's UI components and styling. Developers can utilize various Cordova plugins (or Capacitor plugins) to access native device functionalities like camera, geolocation, or device sensors. Ionic apps do not compile to native code like traditional native apps.The app's web code (HTML, CSS, JavaScript) is bundled and packaged into the app's binary file alongside the WebView and other required resources.
- **Apache Cordova**: It is an open-source framework that allows developers to build hybrid mobile applications using web technologies. Cordova provides a set of JavaScript APIs, known as Apache Cordova Plugins, which act as a bridge between the web code and native device functionalities. Developers can use these plugins to access native features like camera, geolocation, contacts, sensors, and more. Cordova apps do not compile to native code either; they package the app's web code along with the WebView and other resources into the app's binary file. The WebView interprets and executes the bundled web code at runtime, providing a hybrid app experience.

- Examples of popular hybrid apps include Facebook (react native), Microsoft Teams (ionic), AirBnB(React Native), Salesforce, Trello, and Slack.

2.3 PWA apps

They are built using web technologies such HTML CSS and JavaScript. Also it uses service workers to continue functioning and providing a seamless offline experience. Some examples of popular PWAs include Twitter Lite, LinkedIn Lite, and Google Maps Go.

2.4 Cross Platform apps: **(apps that are built using a single codebase, but can run on both android and ios)**

- All hybrid languages/frameworks are cross platform.
- **Flutter**: Flutter is a Dart framework used for both mobile and web development. Flutter compiles Dart code directly to native machine code for each platform. This allows flutter to interact directly with the underlying system which improves performance and response time.
- **Xamarin** (C#, C++)
- **Unity** (C#)

3. Mobile app development frameworks comparison using key features (language, performance, cost and time to market, UI and UX, complexity, community support)

3.1 What is a Framework?

A programming framework is a combination of a set of tools, libraries, rules and conventions that provide a structured way for developing an application

or software in an efficient way, written in a particular programming language. It often contains pre-written code to perform generic functions like input/output, database manipulation, authentication etc. such that developers can focus on the specific and unique logic of their application.

3.2 What is a Mobile Framework?

This is a framework for development of application meant to run on mobile devices and operating systems. There exist many different mobile frameworks with each excelling in different features but the most popular will follow below:

A. React Native

Developed by Facebook, this framework allows developers to use both JavaScript and React library to write code once and deploy for both android and iOS platforms without a change in the codebase.

B. Flutter

Created by Google, Flutter is an open-source UI software development kit that allows developers to build locally compiled applications for mobile, web and desktop applications from a single codebase. It uses DART programming language and makes use of prebuilt widgets for creating interactive applications.

C. Ionic

Ionic is an open-source mobile framework for building crossplatform mobile applications using web technologies like HTML, CSS, and JavaScript. It provides a library optimized UI components and tools for developing high-performance applications that can run on various
platforms.

D. **Xamarin**:

Xamarin is a free and open-source mobile app platform owned by Microsoft for building native and high-performance iOS, Android, tvOS, watchOS, macOS, and Windows apps in C# with .NET. It allows you to develop a mobile app using C# and reuse most of the codebase across multiple platforms. Xamarin is now part of .NET and is integrated directly into the .NET developer platform with tools and libraries specifically for building apps.

E. **Apache Cordova (formally known as PhoneGap)**:

Apache Cordova is an open-source mobile development framework that allows developers to build mobile applications using web technologies such as HTML, CSS, and JavaScript. It enables you to use standard web technologies for cross-platform development, avoiding each mobile platform's native development language. Applications execute within wrappers targeted to each platform, and rely on standards-compliant API bindings to access each device's capabilities.

F. **NativeScript**:

This is an open-source framework for building truly native mobile apps. It is a cross-platform mobile development framework that allows developers to use modern web technologies such as Angular and Vue.js or just HTML, CSS, and JavaScript/TypeScript to build mobile apps. NativeScript provides platform APIs directly to the JavaScript runtime for a rich TypeScript development experience

G. **JQuery Mobile**:

JQuery Mobile is a user interface framework, which is built on jQuery Core and used for developing responsive websites or applications that are accessible on mobile, tablet, and desktop devices. It uses the features of both jQuery and jQuery UI to provide API features for mobile web applications. The jQuery Mobile framework takes the "write less, do more" mantra to the next level: Instead of writing unique applications for each mobile

device or OS, the jQuery mobile framework allows you to design a single highly-branded responsive web site or application that will work on all popular smartphone, tablet, and desktop platforms.

3.3 Features necessary to note before choosing a mobile framework

A. Language: This refers to the programming language used by the framework for development of mobile applications.

B. Performance: In the context of mobile frameworks, performance refers to how efficiently a mobile app runs, which is determined by the speed and smoothness of its operations. It includes factors like load times, responsiveness, resource usage (like battery and memory), and the ability to handle high volumes of data or users.

C. Cost and Time to Market: Cost and time to market refer to the resources (both financial and temporal) required to develop a mobile app and launch it to the public. Using a mobile framework can reduce both cost and time to market by allowing developers to use a single codebase for multiple platforms, thereby eliminating the need to develop separate codebases for each platform

D. UI/UX: UI (User Interface) and UX (User Experience) in mobile frameworks refer to the design and interaction elements of a mobile app. UI involves the visual elements that users interact with, such as buttons, menus, and images. UX, on the other hand, involves the overall experience of using the app, including the ease of use, the flow from one screen to another, and the intuitiveness of the interface.

E. Complexity: In the context of mobile frameworks, complexity refers to the difficulty involved in developing a mobile app. This can include the learning curve of the framework's language and tools, the intricacy of the app's features and functionality,

and the challenges in ensuring compatibility across different devices and operating systems.

F. **Community Support:** Community support refers to the resources and assistance available from the community of developers who use the framework. This can include documentation, tutorials, forums, and libraries of reusable code. A strong community can provide valuable help and resources, making it easier to solve problems and develop applications with more ease.

3.4 Comparison of mobile frameworks

3.4.1 React Native

- **Performance**: Provides near-native performance and a core set of components that map directly to their native counterparts.
- **UI/UX**: Uses native components for rendering, providing a native-like look and feel.
- **Cost and Time to Market**: Shared codebase between React (web) and React Native (mobile) projects can reduce both cost and time to market.
- **Complexity**: Has a steep learning curve, especially for developers unfamiliar with JavaScript and React.
- **Community Support**: Has a large and mature ecosystem with a vast number of libraries and plugins.
- **Programming Language**: Uses JavaScript.

3.4.2 Flutter

- **Performance**: Provides high-performance apps with native-like performance due to an ahead-of time (AOT) compilation.
- **UI/UX**: Has its own rendering engines, allowing for precise control over UI elements.
- **Cost and Time to Market**: The hot reload functionality accelerates the development process, making it easier to experiment with different UI designs and fix issues in real-time.

- **Complexity**: Dart is less popular than JavaScript, which may require developers to learn a new language.
- **Community Support**: Has strong support from Google and a growing community.
- **Programming Language**: Uses Dart.

3.4.3 Ionic

- **Performance**: Offers performance close to native apps, especially when used with Angular.
- **UI/UX**: Provides a library of pre-built UI components that mimic native UI elements.
- **Cost and Time to Market**: Allows developers to use a single codebase for multiple platforms, reducing both cost and time to market.
- **Complexity**: Requires knowledge of web technologies such as HTML, CSS, and JavaScript.
- **Community Support**: Has a large and active community, providing many resources and assistance.
- **Programming Language**: Uses HTML, CSS, and JavaScript.

3.4.4 Xamarin

- **Performance**: Xamarin apps have native performance, as they use native UI controls and APIs.
- **UI/UX**: Provides access to native APIs and performance optimizations through Xamarin.Forms and Xamarin.Native.
- **Cost and Time to Market**: Shared codebase across platforms, with up to 90% code reuse.
- **Complexity**: Has a steep learning curve, as developers need to have a solid understanding of C# and native development.
- **Community Support**: Has strong support from Microsoft and integration with Visual Studio.
- **Programming Language**: Uses C#.

3.4.5 Apache Cordova

- **Performance**: Cordova apps may have lower performance compared to native apps.
- **UI/UX**: Allows developers to build mobile applications using web technologies such as HTML, CSS, and JavaScript.
- **Cost and Time to Market**: Enables you to use standard web technologies for cross-platform development, avoiding each mobile platform's native development language.
- **Complexity**: Requires knowledge of web technologies.
- **Community Support**: Its popularity is decreasing, and the number of software developers who used Apache Cordova in 2019–2021 fell from 29% to 16%.
- **Programming Language**: Uses HTML, CSS, and JavaScript.

3.4.6 NativeScript

- **Performance**: Provides native-like performance and a rich TypeScript development experience.
- **UI/UX**: Provides platform APIs directly to the JavaScript runtime.
- **Cost and Time to Market**: Allows developers to use a single codebase for multiple platforms, reducing both cost and time to market.
- **Complexity**: Requires knowledge of JavaScript or any other JavaScriptrelated language.
- **Community Support**: Has a growing community.
- Programming Language: Uses XML, CSS, and JavaScrip

3.4.7 JQuery mobile

- **Performance**: Offers performance close to native apps.
- **UI/UX**: Uses web UI elements that are shared across any platform, conforming to the native look & feel of wherever they are deployed.
- **Cost and Time to Market**: Allows developers to use a single codebase for multiple platforms, reducing both cost and time to market.
- **Complexity**: Requires knowledge of jQuery and jQuery UI.
- **Community Support**: Has a large and active community.

- **Programming Language**: Uses HTML, CSS, and JavaScript

4. Mobile application architecture and design patterns

4.1 Architectures

A. Model-View-Controller (MVC) Architecture

This design model divides an application into three interacting (interconnected components) parts: model (data logic), view (presentation), and controller (user input). This separation allows for better design and modularization, easy maintainability and promotes code reusability. The model represents application data and business logic, the view displays data to the user and the controller processes user input and controls data flow between model and view.

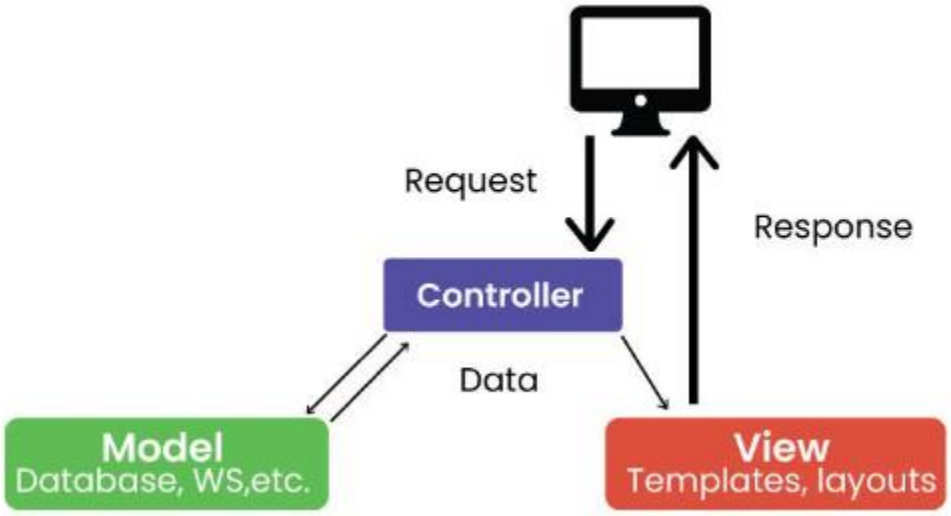

Fig 1: Model View Controller Architecture

B. Model View Presenter (MVP) Architecture

In this architecture, an application is separated into three parts: model, view and presenter. This is similar to the MVC but puts more responsibility on the presenter to manage the interaction between model and view. Hence in this architecture, the model manages data and business logic, the view represents the user interface and the presenter acts as an intermediary processing user input and updating the view and model.

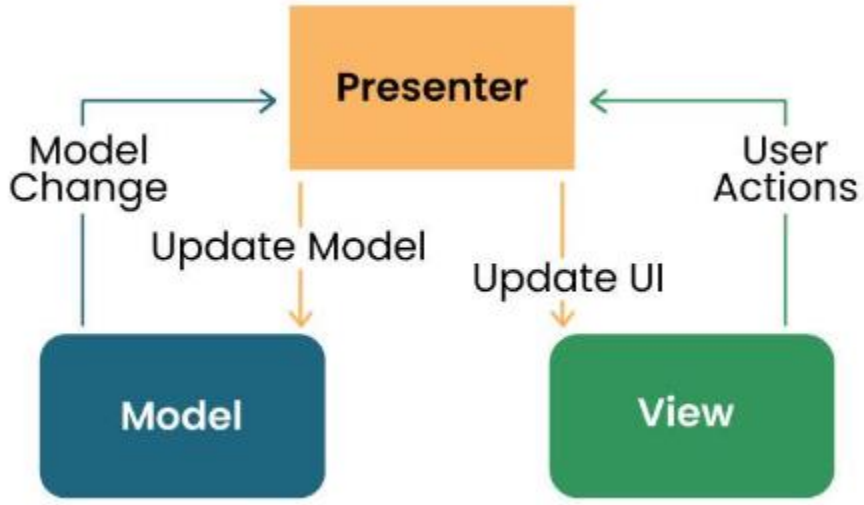

Fig 2: Model View presenter architecture

C. Model View View-Model (MVVM)

It enhances the MVC by introducing View-Model to manage presentation logic. It separates an application in to three parts: the model which represents the data and business logic, the view which represents the user interface, and the view-model which acts as an interface between the model and the view.

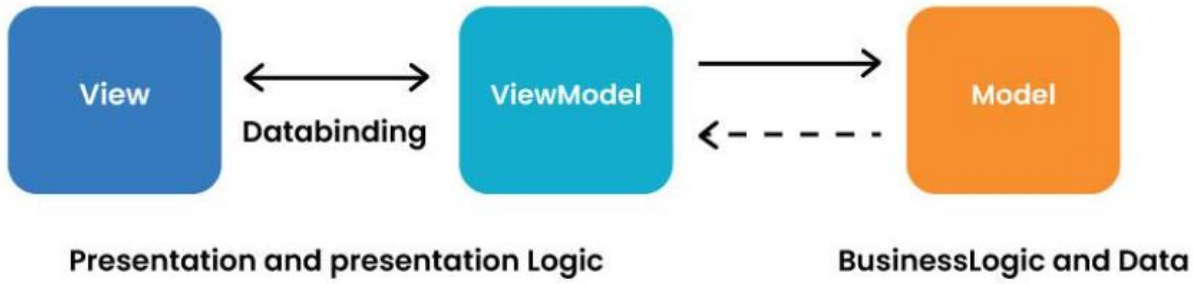

Fig 3: Model View View-Model architecture

D. Clean architecture

The purpose of this architecture is to separate an application into different layers for clean dependency management. Each layer has a single duty and communicates with different layers through properly-defined interfaces.

4.2 Design patterns

A. Singleton pattern

The singleton policy ensures a class has only one instance and provides a global point of access to it. This is important when a single instance of an object needs to be managed.

B. Factory pattern

The factory method model defines an interface for creating an object but allows subclasses to modify the type of objects that will be created. This is especially useful when you need to create objects with a common interface but different functionality.

C. Observer pattern

It defines a one to many dependencies between objects, ensuring that when one object changes state, all its dependents are notified and updated automatically.

D. Adapter pattern

The adapter configuration allows objects with incompatible interfaces to collaborate. It is often used to work with others without modifying the source code of the existing classes.

E. Strategy pattern

This model defines a set of algorithms, encapsulates each one, and makes them interchangeable (flexible) such that the appropriate algorithm is selected at runtime. This is useful when different options for a task need to be implemented.

F. Decorator pattern

It allows behavior (responsibilities) to be added to individual objects dynamically, at runtime. This is useful for adding features to existing classes without altering their structure.

5. Requirement Engineering

5.1 Introduction

Requirements engineering is a crucial phase in the development of mobile applications. It involves gathering, documenting, analyzing, and validating user requirements to ensure that the final product meets user expectations. In this document, we will provide a step-by-step guide to effectively collect and analyze user requirements for mobile applications.

5.2 Scope and Objectives:

Clearly define the scope and objectives of the mobile application. Identify its purpose, target audience, the problem it aims to solve, and desired outcomes. This information will guide the requirements gathering process and help align the development efforts with the overall vision.

5.3 Stakeholder Identification:

Identify all stakeholders involved in the mobile application project. This includes end-users, clients, developers, designers, and business analysts. Each stakeholder group will have unique perspectives and requirements. Engage with them to understand their expectations and gather valuable insights

5.4 User Research:

Conduct thorough user research to gain a deep understanding of the target audience. Employ various research methods such as surveys, interviews, focus groups, and observation. Explore their daily routines, challenges, and how the app can provide value in their lives. Collect data on user preferences, pain points, and behavior patterns. Analyze the data to identify common patterns and prioritize the most critical user needs.

5.5 User Personas:

Develop user personas based on the collected user research. User personas are fictional characters that represent different types of users. They embody the characteristics, goals, and behaviors of your target audience. User personas serve as a reference throughout the development process, ensuring user-centric decision-making.

5.6 Requirement Elicitation:

Elicit requirements from stakeholders and users using appropriate techniques. Conduct interviews, workshops, questionnaires, and brainstorming sessions to gather comprehensive and accurate requirements. Encourage open communication and active listening to ensure all perspectives are considered.

5.7 Categorization and Prioritization:

Categorize the collected requirements into functional and non-functional requirements. Functional requirements define what the application should do, while non-functional requirements focus on qualities such as performance, security, and usability. Prioritize the requirements based on their importance and impact on the user experience.

5.8 Requirement Documentation:

Document the requirements in a clear and concise manner. Use requirement specification documents or visual models such as use-case diagrams or user stories to capture the essence of each requirement. Include necessary details like functional descriptions, acceptance criteria, and dependencies.

5.9 Requirement Validation:

Validate the documented requirements with stakeholders to ensure accuracy and mutual understanding. Conduct reviews and walkthroughs to gather feedback and address any concerns or suggestions. Validate the requirements against the defined scope and objectives of the mobile application.

5.10 Requirement Analysis and Refinement:

Analyze the requirements for potential conflicts, gaps, or inconsistencies. Evaluate their feasibility considering technical constraints. Collaborate with stakeholders and subject matter experts to refine and clarify the requirements. This iterative process helps uncover hidden requirements and improve the overall quality of the application.

5.11 Review and Approval:

Regularly review the requirements with stakeholders to gather feedback and address any changes or updates. Once the requirements are finalized, obtain formal approval from all relevant stakeholders. This approval signifies a common understanding and agreement on the project's scope and establishes a foundation for development.

5.12 Conclusion:

By following a systematic approach to requirements engineering, you can ensure that your mobile application meets user expectations and achieves its intended goals. Effective requirements engineering lays the groundwork for a user-centric design and development process, ultimately resulting in a high-quality mobile application that fulfills user needs and delivers a great user experience.

6. Estimating the Cost of a Mobile Application Development Project

6.1 Introduction:

Estimating the cost of a mobile application development project is a critical step in planning and budgeting. Accurate cost estimation helps stakeholders allocate resources effectively and make informed decisions. In this guide, we will outline the key factors to consider and steps to follow when estimating the cost of a mobile application.

6.2 Define Project Requirements:

Start by defining the project requirements in detail. Gather information about the desired features, functionality, platforms (iOS, Android, or both), and any specific integrations or third-party services required. The complexity and scope of the project will have a significant impact on the overall cost.

6.3 Break Down the Project:

Break down the project into smaller components or modules. This allows for a more granular estimation process and helps identify potential cost drivers. Consider aspects such as user interface design, backend development, database integration, API development, testing, and deployment.

6.4 Evaluate Development Approach:

Choose the development approach that best suits your project: native, hybrid, or cross-platform. Native development involves building separate applications for each platform using platform-specific technologies (e.g., Swift for iOS, Java/Kotlin for Android). Cross-platform development uses frameworks like React Native or Flutter to build a single codebase that can run on multiple platforms. Hybrid development utilizes web technologies (HTML, CSS, and JavaScript) wrapped in a native container. The development approach can impact the cost, as native development tends to be more time-consuming and costly than hybrid or cross-platform approaches.

6.5 Determine Resource Requirements:

Identify the resources needed for the project, including developers, designers, testers, project managers, and any specialized expertise required. Consider the duration of the project and the skill level of the resources. The cost of resources will depend on factors such as location, experience, and demand in the market.

6.6 Estimate Development Effort:

Break down each module or component into tasks and estimate the effort required for development. Consider factors such as complexity, functionality, integration points, and any dependencies. Use historical data from previous projects or industry benchmarks to estimate the development effort required for each task.

6.7 Consider External Costs:

In addition to internal development costs, consider external costs that may arise during the project. These costs may include expenses related to third-party services, cloud hosting, licensing fees, and any necessary hardware or software purchases.

6.8 Account for Testing and Quality Assurance:

Testing and quality assurance are crucial to ensure a high-quality mobile application. Allocate resources and budget for testing activities, including functional testing, usability testing, performance testing, security testing, and device compatibility testing. Estimating testing efforts and resources will help prevent surprises and ensure a robust final product.

6.9 Factor in Maintenance and Support:

Consider the ongoing maintenance and support needs of the application once it is launched. Budget for bug fixes, updates, feature enhancements, and technical support. Maintenance and support costs are typically calculated as a percentage of the initial development cost (e.g., 20% per year).

6.10 Consider Project Management:

Include project management costs in your estimation. Project managers oversee the development process, coordinate resources, manage timelines, and ensure effective communication between stakeholders. The complexity and duration of the project will influence the level of project management effort required.

6.11 Contingency Planning:

Account for contingencies and unforeseen events in your cost estimation. It is recommended to allocate a contingency budget (e.g., 10-15% of the total estimated cost) to handle any unexpected challenges or changes during the project.

6.12 Review and Refine:

Review the estimated costs with stakeholders and subject matter experts. Seek their feedback and input to refine the estimation further. Consider incorporating industry standards, benchmarks, and lessons learned from previous projects to ensure accuracy.

6.13 Document and Present the Estimate:

Document the estimated costs in a clear and comprehensive manner. Present the estimate to stakeholders, highlighting the breakdown of costs, assumptions made, and any risks identified. Clearly communicate the basis for the estimate and provide a range or confidence level to account for uncertainties.

6.14 Conclusion:

Estimating the cost of a mobile application development project requires a systematic and comprehensive approach. By considering key factors such as project requirements, development approach, resource requirements, testing, maintenance, and support, you can create a realistic and accurate cost estimate. Regularly review and update the estimate as the project progresses to ensure effective budget management and successful project delivery.

References

https://www.imaginarycloud.com/blog/native-vs-hybrid-vs-pwa/

https://www.geeksforgeeks.org/design-patterns-for-mobile-development/