UNIVERSITY OF BUEA

Faculty Of Engineering and Technology

CEF440

Internet Programming and Mobile Programming

Group 14

TASK 1

Submitted to:

Dr. Nkemeni Valery

2023/2024

MEMBERS

LANGEH MOHAMMED YIENEH	FE21A223
JENNA EBOT AGBOR	FE21A205
ABO STEVE AKUM	FE21A125
NDEH TAMINANG	FE21A246
CHINEPOH DIVINE-FAVOUR	FE21A159

Contents

1. Mobile Apps

In the expansive world of software development, mobile apps are primarily categorized based on the technology employed in their development. The three fundamental types of mobile applications are **Native apps, Web apps,** and **Hybrid apps.**

1.1. Major Types of Mobile Apps

1.1.1. Native Apps

Native apps are specifically designed and developed for a particular platform or operating system.

Native mobile applications are built specifically for a particular operating system (OS), such as iOS for Apple devices or Android for Android devices. These apps are developed using platform-specific programming languages and software development kits (SDKs), which allow them to leverage the full capabilities of the operating system and the device's hardware.

One of the main advantages of native mobile applications is their performance and user experience. Since they are optimized for a specific OS, they can take full advantage of the device's features and offer seamless integration with the operating system. Native apps can also access device-specific functionalities like camera, GPS, and push notifications. However, developing native apps for multiple platforms requires separate development efforts and maintenance.

For instance, a native Android app is built using different tools and technologies from a native iOS app. These apps typically deliver high performance and optimal user experience due to their platform-specific nature.

Examples of native apps:

- Facebook (iOS and Android)
- Google Maps (iOS and Android)

Common technology used in native app development:

- iOS: Swift, Objective-C
- Android: Java, Kotlin

Advantages of native apps:

- Superior performance and responsiveness due to platform-centric development
- Enhanced user experience with platform-specific UI/UX

- Direct access to device features

Disadvantages of native apps:

- Higher development costs for multiple platforms
- Separate codebases for each platform required

1.1.2. Web apps

Web apps are essentially mobile-optimized websites accessible through mobile browsers. They are not installed on the device but are accessed via a web URL. These apps work across various platforms and devices, providing a cost-effective approach to reaching a broader audience.

Web-based mobile applications, also known as mobile web apps, are essentially websites that are designed to be accessed and used on mobile devices. They are built using web technologies such as HTML, CSS, and JavaScript and are accessed through a web browser on the mobile device. These apps are not installed on the device but are accessed through a URL like a regular website.

One of the main advantages of web-based mobile applications is their cross-platform compatibility. They can run on various devices and operating systems as long as they have a compatible web browser. Web apps also provide easier maintenance and updates since changes can be made on the server-side without requiring users to download updates. However, web-based apps may have limitations in terms of performance and access to device-specific features.

Examples of web apps:

- Twitter Lite
- Financial Times web app

Common technology used in web app development:

- HTML5, CSS, JavaScript
- Various web frameworks like Angular, React, Vue.js

Advantages of web apps:

- Cost-effective development with a single codebase for all platforms.
- Generally, require the least software testing effort among the three types.
- No installation is required, saving device memory.
- Easier maintenance and updates as changes are instantly reflected.

Disadvantages of web apps:

- Reliant on the device's browser, leading to varied user experiences.
- Limited offline functionality compared to native apps.

Progressive web apps (PWA) – A common subset of web apps

Progressive web apps are a modern approach to delivering app-like experiences on the web. PWAs are designed to address the weaknesses of standard web apps, particularly when it comes to mobile experiences, providing a more efficient, reliable, and engaging user experience on mobile devices.

KEY FEATURES OF PROGRESSIVE WEB APPS INCLUDE:

- Responsive design: PWAs are designed to work seamlessly on various screen sizes and devices, ensuring a consistent user experience.
- Offline functionality: PWAs can work with limited or no internet connectivity by caching essential resources.
- Fast loading: PWAs load quickly and respond instantly to user interactions, providing a smooth and responsive experience.
- Discoverability: PWAs can be discovered through search engines, making them easily accessible to users.
- Safe and secure: PWAs are served over HTTPS, ensuring data security and user privacy.
- Automatic updates: Changes and updates to PWAs are automatically reflected the next time a user accesses the app, ensuring users are always using the latest version.

1.1.3. Hybrid apps

Hybrid apps combine elements of both native and web apps. They are essentially web apps encapsulated within a native container, enabling users to download them from app stores. Hybrid apps provide a balance between performance and versatility.

Hybrid mobile applications combine elements of both native and web-based apps. They are developed using web technologies like HTML, CSS, and JavaScript and wrapped in a native container that enables them to be installed and run on a device like a native app. Hybrid apps are built using frameworks such as React Native, Ionic, or PhoneGap, which provide a bridge between web technologies and native functionality.

The main advantage of hybrid mobile applications is their ability to provide a balance between cross-platform compatibility and access to native features. Hybrid apps can be developed once and deployed on multiple platforms, reducing development time and costs. They can also access

device-specific functionalities through plugins or native API calls. However, hybrid apps may not offer the same level of performance and user experience as fully native apps.

Examples of hybrid apps:

- Instagram (uses React Native)
- UberEats (uses Ionic)

Common technology used in hybrid app development:

- HTML5, CSS, JavaScript
- Frameworks like React Native, Ionic, Xamarin

Advantages of hybrid apps:

- Efficient development with a single codebase, reducing costs.
- Relatively fast and able to function offline to some extent.
- Consistent user experience across platforms.

Disadvantages of hybrid apps:

- Slightly reduced performance compared to native app

1.2. Differences

The following table will help us draw the comparison between web app vs native app vs hybrid app more easily:

Parameters	Web	Native	Hybrid
Definition	Web apps are adaptive webpages that rely on browsers rather than operating systems	Native applications are created to run on a single operating system	Hybrid apps are designed to run on several platforms and operating systems

Parameters	Web	Native	Hybrid
Performance and Productivity	Although responsive, web applications are considerably slower than native applications and have less intuitive designs	Due to the outstanding performance of native applications, a wonderful user experience is guaranteed. Consequently, native apps are quick and have engaging user interfaces	Although hybrid apps are slower than their equivalents, they function according to the user interface
Cost and time	Web mobile app development is the quickest and least expensive to create. Therefore, organizations with limited resources and time opt for responsive web apps	Developing native apps is costly, especially if you want the same applications for different operating systems. The process of creating native apps takes time as well. Additionally, these applications demand a lot of maintenance	The cost of creating hybrid apps is lower than that of creating native apps. Maintenance is also significantly easier since a single code base develops different versions of the same application
Test Performed	Testing a web application includes UI testing, monitoring battery life, detecting network issues, and tracking blockages caused by advertisements	In addition to other tests, native apps are tested for network, screen, comparability, and gestures	Like native apps, hybrid applications undergo screen, comparability, gesture, and network testing
Advantages	Multiple platform compatibility, easier and more efficient deployment, and minimal end user requirements	Intuitive interface, out-of-the-box performance, and end-to-end feature set	Higher reach in the market, wonderful user experience, and less time in development

Parameters	Web	Native	Hybrid
Examples	Netflix, Google Docs, Basecamp, Microsoft Office, Trello, Starbucks	WhatsApp, Spotify, Pokémon Go, Waze, Duolingo, Hive	Instagram, Uber, Gmail, Evernote, Justwatch, Discord, Twitter

2. Mobile App Programing Languages

Imagine you've been tasked with developing a mobile application for your university. The choice of programming language is crucial, depending on who will use the app and what kind of functionality it needs to deliver. Let's explore the different options available:

2.1. Native Languages

2.1.1. Java

- Platform: Android
- Description: Java has been the backbone of Android development for years. Its robustness and performance make it a popular choice among developers. Java follows a class-based, object-oriented programming paradigm with syntax influenced by C++.
- Usage: Widely used for building native Android applications.

2.1.2. Kotlin

- Platform: Android
- Description: Kotlin is Google's favoured language for Android development, alongside Java. It offers a more concise syntax and enhanced safety features compared to Java, thanks to its type inference capabilities.
- Usage: Growing rapidly in popularity, especially for new Android projects.

2.1.3. Objective-C

- Platform: iOS

- Description: Although considered a legacy language, Objective-C is still used in some existing iOS projects as it was developed in the 1980's for Apple Computers. It combines high-level programming concepts with a Smalltalk-like syntax rooted in C.
- Usage: Commonly found in older iOS apps, gradually being replaced by Swift.

2.1.4. Swift

- Platform: iOS
- Description: Swift is Apple's official language for iOS development. Launched in 2014, it offers conciseness, safety, and high performance, making it a favourite among iOS developers.
- Usage: Preferred choice for new iOS app development.

2.2. Cross-Platform Languages:

2.2.1. JavaScript (with React Native)

- Description: With React Native, you can leverage your existing web development skills to build mobile apps for both iOS and Android using JavaScript. React Native offers rapid development and has a large community for support.
- Usage: Ideal for projects where code reuse and fast iteration are priorities.

2.2.2. Dart (with Flutter)

- Description: Flutter, powered by Dart, is gaining popularity for cross-platform development. It enables developers to create visually stunning and performant apps using a single codebase.
- Usage: Suitable for projects that require a high level of customization and smooth UI/UX.

2.3. Other Languages

2.3.1. C++

- Description: C++ is often used for performance-critical applications where speed is paramount. However, it comes with a steep learning curve and may require more development time compared to other languages.
- Usage: Commonly employed in gaming and graphics-intensive apps.

2.3.2. HTML5/CSS3/JavaScript

- Description: While not native mobile languages, this trio allows you to build hybrid apps that run within a web view. It's a good choice for simpler applications or those targeting both mobile and web audiences.
- Usage: Ideal for projects with limited resources or tight budgets.

3. Comparison of Mobile App Development Frameworks

Let’s compare some popular mobile app development frameworks based on some key features mentioned. The information is organized in a table format for easy reference

Framework	Language	Performance	Cost & Time to Market	UX & UI	Complexity	Community Support	Use Cases
Adobe Phone Gap	HTML, CSS, JavaScript	Moderate	Low cost, Quick development	Basic	Low	Active	Cross-Platform app
Ionic	HTML, CSS, JavaScript	God	Low cost, Rapid development	Good	Moderate	Strong	Cross-Platform app
Xamarin	C#	High	Moderate cost, Moderate development time	Excellent	Moderate	Robust	Cross-Platform app
React Native	JavaScript	High	Moderate cost, Efficient development	Excellent	Moderate	Large	Cross-Platform app
Flutter	Dart	High	Moderate cost, Fast development	Excellent	Moderate	Growing	Cross-Platform app
Corona SDK	Lua	Moderate	Low cost, Quick development	Basic	Low	Supportive	2D games, interactive apps

Mobile Angular UI	HTML, CSS, JavaScript	Moderate	Low cost, Quick development	Basic	Low	Active	Web apps, hybrid apps
Intel XDK	HTML, CSS, JavaScript	Moderate	Low cost, Rapid development	Basic	Low	Supportive	Cross-Platform app
Native (iOS/Android)	Swift/Objective-C(iOS), Java/Kotlin (Android)	Excellent	High cost, longer development time	Excellent	High	Established	Cross-Platform app
Web Apps	HTML, CSS, JavaScript	Varies (depends on browser)	Low cost, Quick development	Basic	Low	N/A (Web-	Cross-Platform app

4. Mobile Application Architecture and Design Patterns

An architecture pattern is a set of guidelines and best practices that describe how to structure and organize code, data, and components. It defines the roles and responsibilities of each element, the communication and interaction between them, and the overall design goals and principles. An architecture pattern can help you achieve consistency, modularity, scalability, and performance in your software system. It can also help you avoid common pitfalls and errors, such as tight coupling, code duplication, and poor testability.

4.1. Why use an architecture pattern for mobile development?

Mobile development poses some unique challenges that require careful design decisions. For example, you have to consider the limited resources and battery life of mobile devices, the varying screen sizes and orientations, the network connectivity and latency, the user expectations and preferences, and the security and privacy issues. Using an architecture pattern can help you address these challenges by providing a clear and proven way of organizing your code and data. It can also help you cope with the complexity and diversity of mobile platforms, such as Android, iOS, and Windows Phone, by abstracting away the platform-specific details and enabling cross-platform compatibility.

4.2. Types of Architectural Design Patterns

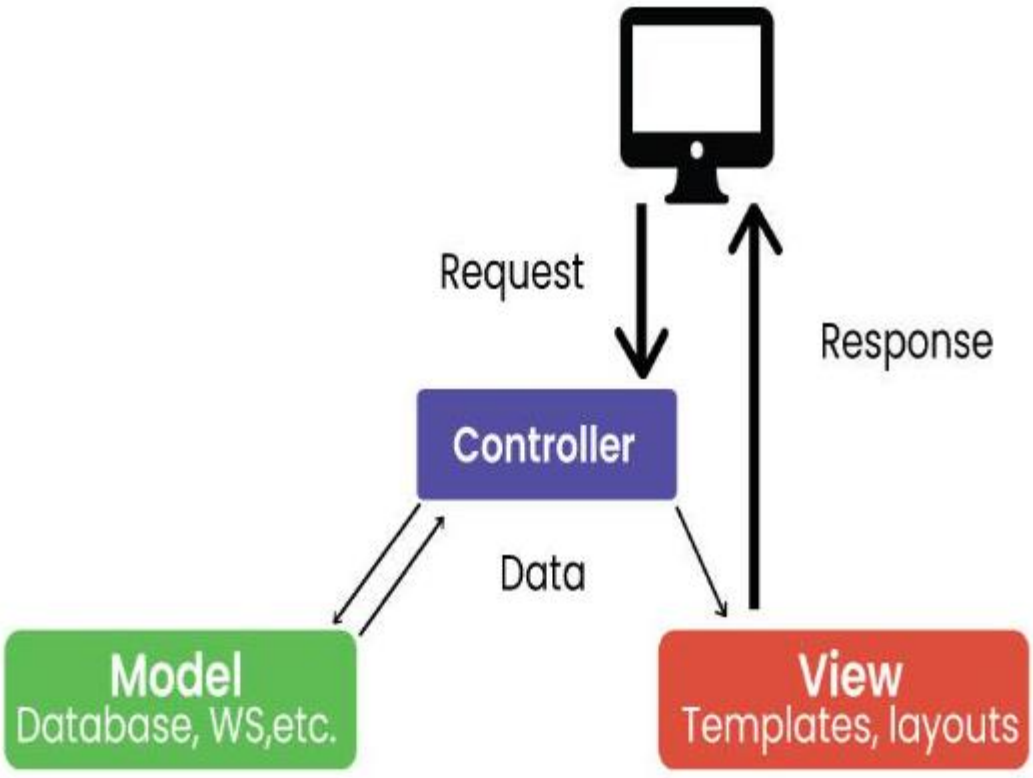
There are very few architectural design patterns available for mobile development.

4.2.1. Model View Controller (MVC) Architecture

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

- **Model:** Represents application data and business logic.
- **View:** Displays data to the user.
- **Controller:** Processes user input and controls data flow between Model and View.

The figure below gives us a view of what this architecture looks like



Let's take an example to better understand:

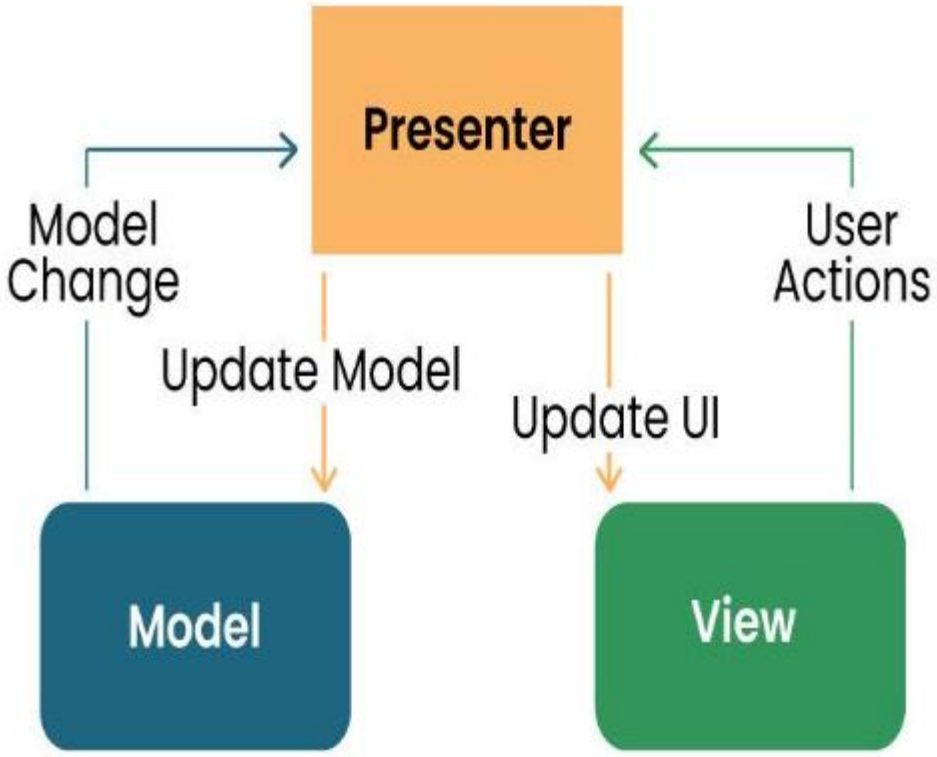
Imagine a mobile weather app. The model stores weather information, the View displays it to the user, and the controller handles user interactions such as updating the displayed location or converting units (e.g., from Celsius to Fahrenheit).

4.2.2. Model View Presenter (MVP) Architecture

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

- **Model:** Manages data and business logic.
- **View**: Represents the user interface.
- **Designer:** Acts as an intermediary processing user input and updating the View and Model.

The figure below gives us a view of what this architecture looks like



Let's take an example to better understand:

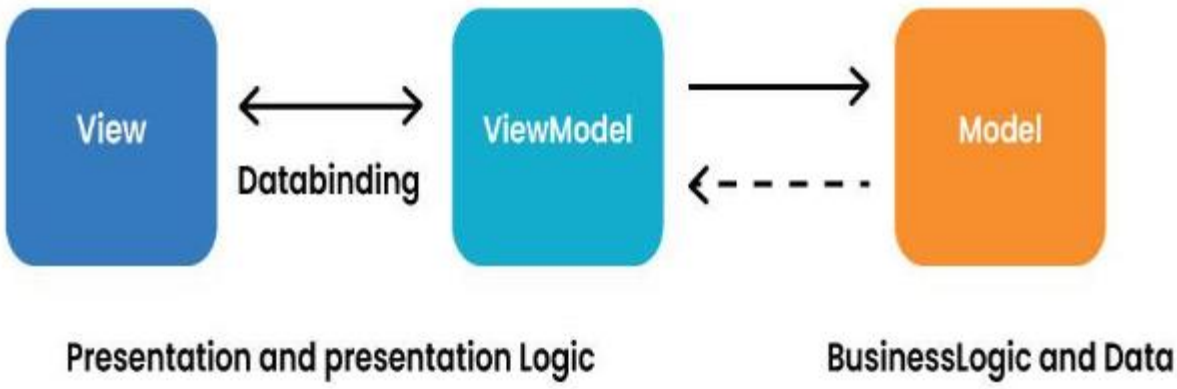
In a note-taking app, the Model would store the text, the View would display it, and the provider would handle user input such as typing, editing, or deletes the process.

4.2.3. Model ViewView Model (MVVM) Architecture

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

- **Model:** Represents data and business logic.
- **View:** Represents the user interface.
- **ViewModel:** Acts as an interface between the Model and the View, which contains the reference logic.

The figure below gives us a view of what this architecture looks like



Let's take an example to better understand:

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

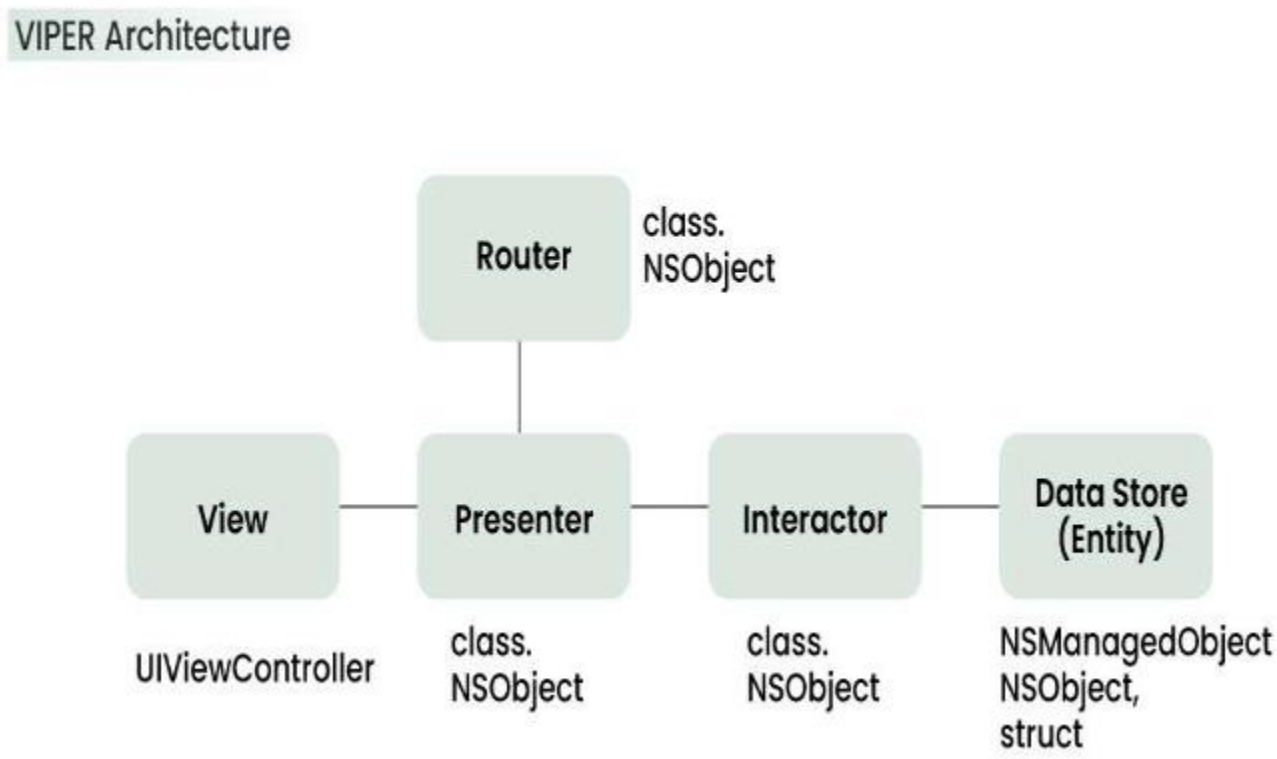
4.2.4. VIPER Architecture

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

Let's briefly explain the function of every element:

- **View:** This is the consumer interface layer, wherein the perspectives and look at controllers are defined. The view is chargeable for showing the information provided by way of the presenter and forwarding the person moves to the presenter.
- **Presenter:** This is the presentation layer, where the good judgment for formatting and imparting the records is defined. The presenter is liable for fetching the records from the interactor, reworking it right into a suitable layout for the view, and updating the view hence. The presenter additionally handles the consumer movements acquired from the view and calls the router to navigate to other screens.
- **Interactor:** This is the enterprise good judgment layer, where the common sense for manipulating the data and interacting with external services is described. The interactor is accountable for gaining access to the facts from the service layer, acting any vital operations on it, and returning it to the presenter. The interactor additionally communicates with the entity layer to store and retrieve the information fashions.

- **Entity:** This is the information layer, wherein the data models and systems are described. The entity is responsible for representing the data in a constant and coherent manner throughout the software. The entity layer also can encompass records get entry to gadgets (DAOs) or repositories that summary the information of records patience and retrieval.
- **Router:** This is the navigation layer, where the logic for routing and transitioning among different monitors is defined. The router is chargeable for developing and providing the view controllers, passing any vital facts to them, and coping with any dependencies or configurations. The router also communicates with the presenter to get hold of the navigation requests and execute them.



4.2.5. Singleton Method Design Pattern

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

Let's take an example to better understand:

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

4.2.6. Factory Method Design Pattern

The Factory Method model defines an interface for creating an object but allows subclasses to modify the type of the created object. Especially useful when you need to create objects with a common interface but different functionality.

Let's take an example to better understand:

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

4.2.7. Observer Method Design Pattern

The observer structure defines one to many dependencies between objects, so when one object changes its state, all its dependents are automatically notified and updated. This is useful for scheduling distributed events.

Let's take an example to better understand:

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

Design processes play an important role in mobile app development by providing proven solutions to common software design challenges. Using this framework allows developers to create maintainable, extensible, and efficient applications. Understanding when and how to apply these options can significantly improve the quality of your mobile app codebase. Whether you're working for Android, iOS, or any other mobile platform, a solid understanding of these design patterns will allow you to create robust, scalable mobile applications

5. Collection and Analyses of User Requirements for A Mobile Application

1. Identify Stakeholders: Start by identifying all stakeholders involved in the mobile application project. This includes end-users, business owners, developers, designers, marketing teams, and any other individuals or groups who have a vested interest in the app's success. Understanding the perspectives and needs of each stakeholder is crucial for gathering comprehensive requirements.

2. Gather Requirements: There are several techniques you can use to gather requirements:

- **Interviews:** Conduct one-on-one interviews with stakeholders to understand their goals, expectations, and pain points related to the mobile app.
- **Surveys:** Distribute surveys to a broader audience to gather feedback on features, usability, and preferences.
- **Workshops:** Host workshops with stakeholders to facilitate discussions, brainstorming sessions, and idea generation.
- **Observations:** Observe users in their natural environment to understand how they currently perform tasks that the mobile app aims to streamline or improve.

3. Analyse Requirements: Once requirements are gathered, it's essential to analyse them to ensure they meet certain criteria:

- **Clarity:** Requirements should be clearly articulated and free from ambiguity to avoid misunderstandings during development.

- **Completeness:** Ensure that all necessary features, functionalities, and constraints are included in the requirements.

- **Consistency:** Check for inconsistencies or conflicts between different requirements to prevent confusion or errors in implementation.

- **Feasibility:** Assess whether the proposed requirements are technically feasible within the constraints of the project, including time, budget, and technology limitations.

4. Document Requirements: Documenting requirements in a structured manner helps ensure that everyone involved in the project has a clear understanding of what needs to be built. Common techniques for documenting requirements include:

- **Use Cases:** Describe interactions between users and the system to accomplish specific tasks.

- **User Stories:** Capture user needs and requirements from the perspective of an end-user in a concise, non-technical format.

- **Requirement Specifications:** Provide detailed descriptions of each requirement, including functional and non-functional requirements, acceptance criteria, and any relevant constraints.

5. Validate Requirements: Once requirements are documented, it's crucial to validate them with stakeholders to ensure they accurately represent their needs and expectations. This may involve:

- **Reviews:** Conducting reviews of requirement documents with stakeholders to gather feedback and address any concerns or discrepancies.

- **Prototyping:** Creating prototypes or mock-ups of the mobile app to provide stakeholders with a visual representation of the proposed solution and gather feedback early in the development process.

6. Manage Requirements Changes: As the project progresses, requirements may change due to evolving user needs, market conditions, or technical constraints. It's essential to have a process in place for managing and tracking these changes, including:

- **Change Control:** Establishing a formal change control process to evaluate and prioritize proposed changes, ensuring they align with project objectives and constraints.

- **Documentation Updates:** Updating requirement documents and communicating changes to all relevant stakeholders to maintain alignment and transparency throughout the project.

7. Iterate: Requirement engineering is an iterative process that continues throughout the development lifecycle. It's essential to continuously gather feedback from users and stakeholders, incorporate changes as needed, and refine requirements based on new insights and discoveries.

By following these steps and incorporating best practices for requirement engineering, you can effectively collect and analyse user requirements for a mobile application, setting the stage for a successful development process and a product that meets users' needs and expectations.

6. Estimating mobile app development cost

Estimating mobile app development cost involves several factors such as platform (iOS, Android, or both), features, complexity, design, and developer rates. Here's a basic breakdown:

1. Define Requirements: Clearly outline the features and functionalities of your app.

2. Choose Platform: Decide whether you want to develop for iOS, Android, or both. Developing for both will cost more.

3. Complexity: Consider the complexity of features like user authentication, data storage, third-party integrations, etc.

4. Design: Design costs can vary based on complexity and whether you opt for custom designs or templates.

5. Development Time: Estimate the time it will take to develop each feature and multiply it by the developer's hourly rate.

6. Testing: Budget for testing across various devices and platforms to ensure compatibility and usability.

7. Maintenance: Factor in ongoing maintenance and updates post-launch.

8. Contingency: Include a buffer for unexpected changes or additional features during development.

9. Developer Rates: Rates vary based on location and expertise. Offshore developers may have lower rates but quality and communication can vary.

10. Documentation and Support: Consider costs for documentation, training, and ongoing support.

11. Marketing and Distribution: Budget for marketing and distribution costs if you plan to launch and promote the app widely.

By considering these factors, you can create a more accurate estimate of your mobile app development costs. It's also helpful to consult with experienced developers or firms to get more precise quotes based on your specific requirements.

7. References

- https://www.geeksforgeeks.org/design-patterns-for-mobile-development/
- https://www.linkedin.com/advice/0/how-do-you-use-architecture-pattern-mobile#:~:text=Three%20of%20the%20most%20common,hard%20to%20maintain%20and%20test.
- https://www.gurutechnolabs.com/mobile-app-technologies/
- https://medium.com/analytics-vidhya/top-4-programming-languages-and-frameworks-for-mobile-apps-in-2021-e181a78e8ac4
- http://chat.openai.com/