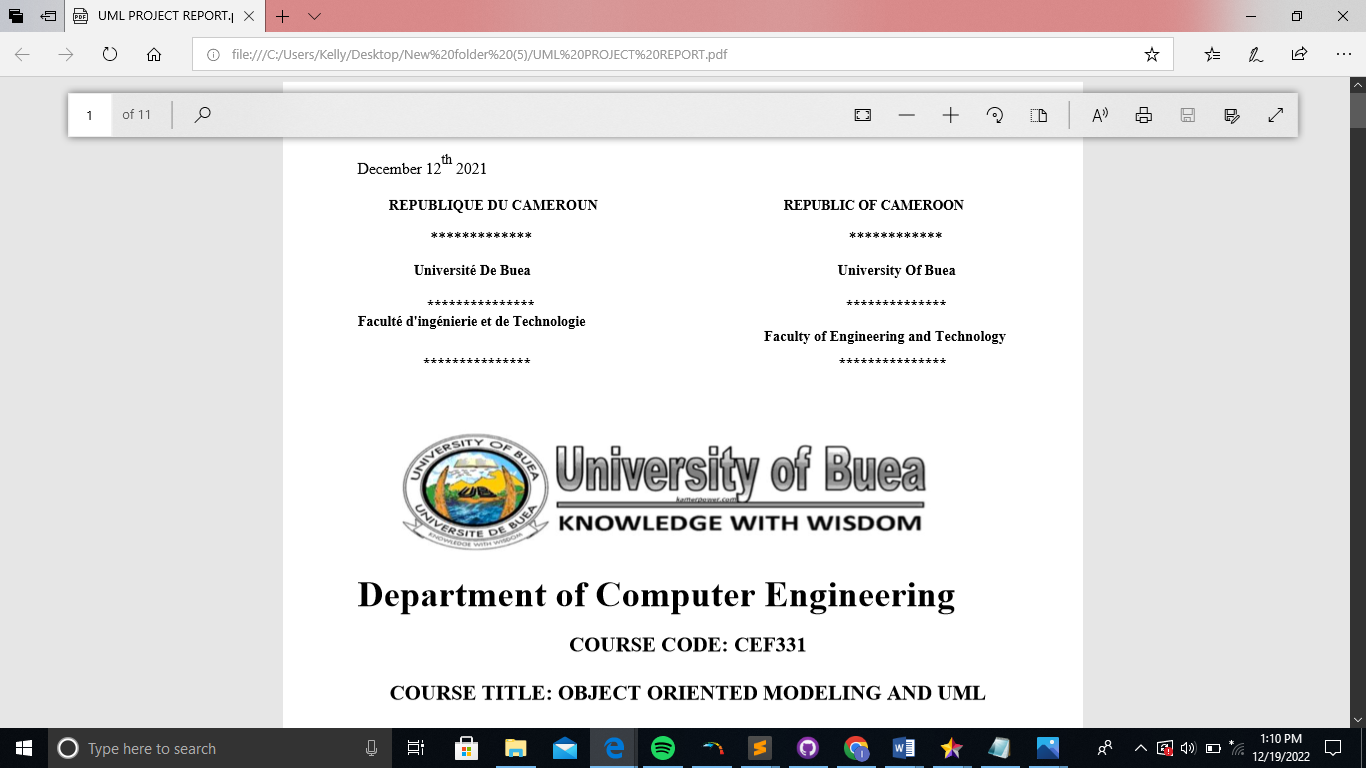
**REPUBLIC OF CAMEROON**

**UNIVERSITY OF BUEA**



**FACULTY OF ENGINEERING AND TECHNOLOGY**

**DEPARTMENT OF COMPUTER ENGINEERING**

**COURSE TITLE: INTERNET PROGRAMMING(J2EE) AND MOBILE PROGRAMMING**

**COURSE CODE: CF440**

**TASK 1**

**COURSE INSTUCTOR:**

**Dr. NKEMENI VALERY**

***PRESENTED BY:*  GROUP 11**

|  |  |  |
| --- | --- | --- |
| SN | **NAMES** | **MATRICULE** |
| 1 | NGULEFAC JERRY MBUOH | FE21A265 |
| 2 | NEBA PRINCEWILL AMBE | FE21A251 |
| 3 | NKENGBEZA DERICK | FE21A277 |
| 4 | KAH JOSPEN NGUM | FE21A264 |
| 5 | NYOCHENBENG ENZO NKENGAFACK | FE21A293 |
| 6 | NFOR WILLY LINWE | FE21A254 |

Table of Contents

[1. Major Types of Mobile Apps: 3](#_Toc162717031)

[**1.1.** Native Apps 3](#_Toc162717032)

[**1.2.** Progressive Web Apps (PWAs) 3](#_Toc162717033)

[**1.3.** Hybrid Apps 3](#_Toc162717034)

[2. Mobile app programming languages 4](#_Toc162717035)

[2.1. C#: 4](#_Toc162717036)

[2.2. Dart: 4](#_Toc162717037)

[2.3. Java: 4](#_Toc162717038)

[2.4. Kotlin: 4](#_Toc162717039)

[2.5. Swift: 5](#_Toc162717040)

[2.6. Python: 5](#_Toc162717041)

[3. Mobile app development framework 5](#_Toc162717042)

[**3.1.** Flutter 5](#_Toc162717043)

[**3.2.** SwiftUI 5](#_Toc162717044)

[**3.3.** React Native 5](#_Toc162717045)

[**3.4.** NativeScript 6](#_Toc162717046)

[**3.5.** Xamarin 6](#_Toc162717047)

[**4.** Mobile Application Architectures and Design Patterns 6](#_Toc162717048)

[4.1. Architectures 6](#_Toc162717049)

[**4.1.1.** MVC (Model-View-Controller) 6](#_Toc162717050)

[**4.1.2.** Model-View-Presenter (MVP) 6](#_Toc162717051)

[**4.1.3.** MVVM (Model-View-ViewModel) 6](#_Toc162717052)

[**4.1.4.** Clean Architecture 7](#_Toc162717053)

[4.2. Design Patterns 7](#_Toc162717054)

[**4.2.1.** Singleton 7](#_Toc162717055)

[**4.2.2.** Observer 7](#_Toc162717056)

[**4.2.3.** Factory 7](#_Toc162717057)

[**4.2.4.** Adapter 8](#_Toc162717058)

[**4.2.5.** Builder 8](#_Toc162717059)

[**4.2.6.** Factory Method 8](#_Toc162717060)

[**5.** Collection and Analysis of user requirement 8](#_Toc162717061)

[o Personnel Identification 8](#_Toc162717062)

[o Gathering process 8](#_Toc162717063)

[o Requirements Documentation 8](#_Toc162717064)

[o Analysis and Prioritization 8](#_Toc162717065)

[o Validation and Verification 9](#_Toc162717066)

[**6.** How to estimating mobile app development cost 9](#_Toc162717067)

[ Key Factors 9](#_Toc162717068)

[ Estimation Methods 9](#_Toc162717069)

[Examples of Estimated Development Costs 9](#_Toc162717070)

[References. 9](#_Toc162717071)

# Major Types of Mobile Apps:

* 1. Native Apps**:**
* Native apps are specifically developed for a particular mobile platform (e.g., iOS or Android) using platform-specific programming languages (Swift for iOS, Kotlin for Android ).
* Provides the best performance and user experience as they can leverage the full capabilities of the device.
* Native apps provide a seamless and consistent user experience.
* They have direct access to device features (camera, GPS, sensors, etc).
* Would require a separate development for each platform, which can be time-consuming and expensive.
* **Examples**: Instagram (iOS and Android), WhatsApp (iOS and Android)
  1. Progressive Web Apps (PWAs)**:**
* Web applications that use modern web capabilities to provide a user experience similar to that of mobile apps.
* Built using web technologies like HTML, CSS, and JavaScript.
* Offer cross-platform compatibility and can be accessed through web browsers, eliminating the need for installation.
* May lack some device-specific features and performance compared to native apps.
* Web apps adapt to various screen sizes.
* **Examples**: Twitter Lite, Pinterest.
  1. Hybrid Apps**:**
* Developed using web technologies (HTML, CSS, JavaScript) and wrapped within a native container.
* Can access device features through plugins or APIs.
* Offer cross-platform compatibility and easier maintenance compared to native apps.
* Offer a balance between development speed and functionality.
* Performance may not match that of native apps, and there may be limitations in accessing certain device capabilities.
* **Examples**: Ionic, Apache Cordova.

# 2. Mobile app programming languages

2.1. C#:

* **Platform**: Xamarin (cross-platform) app development.
* **Advantages**:
* **Cross-Platform**: Works for iOS, Android, and Windows.
* **Unity Integration**: C# is also used for game development.
* **Disadvantages**:
* **Complexity**: Learning curve for Xamarin.

2.2. Dart:

* **Platform**: Flutter (cross-platform) app development.
* **Advantages**:
* **Cross-Platform**: Works for iOS, Android, and Windows.
* **Fast development cycle**.
* **Disadvantages**:
* **New language for many developers**

2.3. Java:

* **Platform**: Android Studio.
* **Advantages**:
* **Ecosystem**: Rich libraries and tools.
* **Widely used**: Established language for Android.
* **Disadvantages**:
* **Verbose Syntax**: More code is often required.

2.4. Kotlin:

* **Platform:** Android Studio.
* **Advantages**:
* **Interoperability**: Works seamlessly with Java.
* **Concise:** Reduces the boilerplate code.
* **Disadvantages**:
* **Android Only**: Limited to Android development.

2.5. Swift:

* **Platform:** iOS app development.
* **Advantages**:
* **Safety Features:** Strongly typed and memory-safe.
* **Modern**: Developed by Apple, more modern and concise.
* **Disadvantages**:
* **Limited to Apple Ecosystem**: Only for iOS, macOS

**Objective-C(legacy)**

2.6. Python:

* **Platform:** Kivy app development.
* **Advantages**:
* **Readable Syntax**: Easy to learn and read.
* **Versatile:** Used beyond mobile apps.
* **Disadvantages**:
* **Performance**: Slower than native languages.

# 3. Mobile app development framework

* 1. Flutter**:**

**Language**: Dart

**Performance**: Very good due to Ahead-of-Time (AOT) compilation.

**Cost & Time to Market**: Faster development with hot reload.

**UX & UI**: Provides a rich set of widgets for custom UI.

**Complexity**: Moderate.

**Community Support**: Growing community.

**Where to Use**: Cross-platform (iOS, Android, web)

* 1. SwiftUI**:**

**Language**: Swift

**Performance**: Native performance on Apple devices.

**Cost & Time to Market**: Efficient for Apple ecosystem.

**UX & UI**: Declarative UI design.

**Complexity**: Moderate.

**Community Support**: Growing community.

**Where to Use**: Native iOS/macOS apps.

* 1. React Native**:**

**Language**: JavaScript (React)

**Performance**: Good, but may require native modules for complex tasks.

**Cost & Time to Market**: Fast development with live reload.

**UX & UI**: Utilizes native components for a native feel.

**Complexity**: Moderate, especially for web integration.

**Community Support**: Large and active community.

**Where to Use**: Cross-platform (Android, iOS).

* 1. NativeScript**:**

**Language**: JavaScript (TypeScript)

**Performance**: Near-native performance.

**Cost & Time to Market**: Moderate setup, but efficient.

**UX & UI**: Access to native UI components.

**Complexity**: Moderate.

**Community Support**: Growing community.

**Where to Use**: Cross-platform (iOS, Android).

* 1. Xamarin**:**

**Language**: C# (.NET)

**Performance**: Native performance with Xamarin.Forms.

**Cost & Time to Market**: Longer setup but efficient once familiar.

**UX & UI**: Access to native APIs and UI components.

**Complexity**: Moderate to high.

**Community Support**: Established community.

**Where to Use**: Cross-platform (iOS, Android, Windows).

1. Mobile Application Architectures and Design Patterns**:**
   1. Architectures:
      1. MVC (Model-View-Controller)

Separates app components into three layers: Model (data), View (UI), and Controller (logic).

* **Model:** Data and business logic
* **View:** UI components that display data
* **Controller:** Intermediary between view and model.Handles user inputs and updates model and view accordingly
* **Usage:** Common in traditional native apps, especially with frameworks like UIKit and Android SDK.
* Promotes modularity and maintainability.
  + 1. Model-View-Presenter (MVP):

Having the Model and View similar to MVC but with a Presenter acting as an intermediary between View and Model.

* + - **Presenter:** Acts as an intermediary between model and view.

It fetches data from the model, formats it and update the view.

It handles user interactions by capturing input from the view. Processing it and updating the model accordingly.

* + 1. MVVM (Model-View-ViewModel)**:**

Having the Model and View similar to MVC but with a ViewModel acting as an intermediary between View and Model.

* **ViewModel:** Intermediary between model and view. Exposes data from model to view through observable properties and handles the presentation logic.

It doesn’t directly reference (unlike with the presenter in MVP) the view but communicates with the view through data bindings provided by the framework

* **Usage:** Popular in modern native apps, especially with SwiftUI and Jetpack Compose.
* Popular in Android development using **Jetpack** libraries.
  + 1. Clean Architecture**:**
* **Entities:** Business objects or entities.
* **Use Cases:** Contains application-specific business rules.
* **Interface Adapters:** Converts data from the Use Cases into a format suitable for presentation in the UI.
* **Frameworks & Drivers:** Contains implementation details like UI, databases, etc.
* **Usage:** Promotes separation of concerns, testability, and maintainability, commonly used in modern app development across platforms.
  1. Design Patterns:
     1. Singleton**:**
* Ensures that a class has only one instance and provides a global point of access to it.
* Useful for scenarios where only one instance of a class is required throughout the application (e.g., database connection, logger).
  + 1. Observer**:**
* Defines a one-to-many dependency between objects, where if one object changes state, all its dependents are notified and updated automatically.
* Commonly used in scenarios where multiple objects need to be updated when the state of one object changes (e.g., UI components observing changes in data).
  + 1. Factory**:**
* Creates objects without exposing the instantiation logic to the client and refers to the newly created object through a common interface.
* Useful for scenarios where the client code needs to create objects without knowing the specific class of the object.
  + 1. Adapter**:**
* Allows incompatible interfaces to work together by converting the interface of one class into another interface that a client expects.
* Useful for integrating third-party libraries or components with existing codebases.
  + 1. Builder**:**
* Separates the construction of a complex object from its representation, allowing the same construction process to create different representations.
* Useful for creating objects with many optional parameters or complex initialization logic.
  + 1. Factory Method**:**
* Defines an interface for creating an object but allows subclasses to alter the type of objects that will be instantiated.
* Useful for scenarios where a class cannot anticipate the class of objects it must create.

1. Collection and Analysis of user requirement.

Requirement engineering is a critical stage in software development, ensuring that the app meets user needs effectively. below are the steps:

* + Personnel Identification:
    - Identify the relevant stakeholders such as users, business owners, clients and developers.
    - Understand their roles, expectations, and perspectives.
  + Gathering process:

Gather requirements through various techniques:

* + - * **Interviews**: Conduct one-on-one discussions with stakeholders.
      * **Workshops**: Collaborate with groups to brainstorm and define needs.
      * **Surveys**: Collect feedback from a broader audience.
      * **Observations**: Observe users in their natural context.
  + Requirements Documentation:

Document the requirements:

* + - **Functional Requirements**: Specify what the app should do (features, interactions).
    - **Non-Functional Requirements**: Define quality attributes (performance, security, usability).
  + Analysis and Prioritization:

Analyse collected requirements:

* + - * **Feasibility**: Verify if requirements are achievable within constraints.
      * **Consistency**: Resolve conflicting requirements.
      * **Completeness**: Verify that all essential aspects are covered.
    - Prioritize requirements based on business value and urgency.
  + Validation and Verification:
    - **Validation**: Confirm that requirements align with stakeholders' needs.
    - **Verification**: Confirm that requirements are correctly documented and understood.

1. How to estimating mobile app development cost**:**

* Key Factors**:**
  + **App Complexity:** Simple apps cost less than Mid-level apps which cost less than complex ones (e-commerce platforms) with extensive features.
  + **Platform:** Native development (separate code for iOS & Android) can be more expensive than cross-platform options.
  + **Development Team:** Location, experience, and size of the development team affect hourly rates.
  + **Features & Functionality:** Integrations with third-party services, complex animations, and backend infrastructure increase cost.
  + **Design Complexity:** High-fidelity UI/UX design iterations add to development time and cost
* Estimation Methods**:**
  + **Hourly Rate x Development Time:** Multiply the estimated development hours by the developer's hourly rate.
  + **App Development Cost Calculators:** Online tools provide a rough estimate based on features and complexity.
  + **Consult with Development Agencies:** Discuss your project with experienced agencies for a more accurate quote

Examples of Estimated Development Costs:

* **Netflix (Entertainment)**: Varies based on features and complexity.
* **YouTube (Video Sharing)**: Custom video streaming features increase costs.
* **PayPal (Finance)**: Complex security and transaction features.
* **Facebook (Social Media)**: High development costs due to extensive features.

# References.

1. [Web Apps vs. Native Apps vs. Hybrid Apps - Difference Between](https://aws.amazon.com/compare/the-difference-between-web-apps-native-apps-and-hybrid-apps/)

2. [The 10 Best Mobile App Development Languages in 2023 - Whiz pool](https://whizpool.com/mobile-app-development-languages/)3

3. [Top 10 Mobile App Development Frameworks in 2024 | Technostacks](https://technostacks.com/blog/mobile-app-development-frameworks/)

4. [Mobile App Analytics: A Comprehensive Guide for Success](https://www.techaheadcorp.com/blog/mobile-app-analytics-comprehensive-guide/)

5. [Design Patterns for Mobile Development - GeeksforGeeks](https://www.geeksforgeeks.org/design-patterns-for-mobile-development/)

6. . [Mobile App Development Cost in 2023: A Complete Guide](https://www.topdevelopers.co/blog/app-development-cost/)