Sub task 4: **Mobile application architecture and design patterns.**

**What is mobile application architecture?**

Mobile app architecture is the blueprint for building applications. The rules, processes, and structure that determines how components like the front-end UI, backend database, APIs, etc, interact to process inputs and give outputs. Think of architecture as the internal skeleton that supports the application’s outward function and form.

**Importance of a good mobile app architecture.**

A well-designed mobile app architecture is crucial to create robust, scalable, and user-friendly applications. It offers several benefits that contribute to app’s success like:

* Firstly, **it enhances modularity**, allowing different app components to be developed and maintained independently, leading to easier updates and modifications.
* For instance, in an e-commerce app, a modular architecture enables seamless integration of new payment gateways without affecting other functionalities.
* Secondly, **robust security measures** in the architecture ensure data protection and prevent unauthorized access.
* A banking app, for instance, can be fortified with encryption protocols and secure authentication methods, instilling trust among users and safeguarding their sensitive information
* Moreover, **reliability is heightened** as a well-structured architecture minimizes bugs and errors, providing users a seamless experience.
* Think about a messaging app that rarely crashes or experiences glitches due to its meticulously designed architecture.
* Furthermore, **performance and scalability are optimized**, allowing the app to handle increasing user loads and adapt to growing demands over time.
* An example is a social media platform capable of accommodating millions of users concurrently without compromising speed or functionality.
* Lastly, an excellent mobile app architecture has different layers that structure the application into logical components with distinct roles for **robust and scalable mobile apps**.

**Layers of mobile application architecture.**

Most mobile app architectures have three layers: presentation, business, and data.

1. Presentation layer

The presentation layer, or front end, is the user interface (UI) you see when opening an app. It includes the screens, navigation, controls, and visual elements.

For example, in a messaging app like WhatsApp, the presentation layer comprises chat screens, contact lists, settings menus, etc. Its primary function is to enable user interactions by taking input from users and displaying output from the lower layers.

2. Business layer

The business layer contains the core application logic that handles tasks like computations, validations, analytics, notifications, background jobs, etc.

In a messaging app, this layer handles functions like sending and receiving messages, encrypting data, detecting spam, managing notifications, etc. It takes input from the presentation and data layers, processes it, and prepares the responses displayed in the UI.

3. Data layer

The data layer handles connections to databases and storage systems, allowing the app to save and retrieve data. For example, in WhatsApp, message data is stored in various databases.

The data layer abstracts the physical storage, so other layers don’t need to worry about the specifics of databases. It handles queries, connections, caching, concurrency, and other data access mechanics. The business layer interacts with the data layer by calling methods it exposes.

**Types of mobile application architecture.**

There are three major types of mobile application architecture including: layered, monolithic, and microservice.

1. Layered architecture

Layered architecture organizes the application into layers, each responsible for a specific aspect of the application’s functionality. Typically, these layers include presentation, business logic, and data access layers.

This architecture promotes modularity and separation of concerns, which makes it easier to maintain and scale the application. However, if not implemented carefully, it may lead to tight coupling between layers.

Layered architecture works well for large, complex applications that require frequent updates. By isolating frontend, business, and data layers, you can focus on specific components and accelerate development and testing cycles for iterative delivery.

2. Monolithic architecture

Monolithic architecture structures the entire application as a single, tightly integrated unit. All application components, including the user interface, business logic, and data access, are packaged together and deployed as a single entity.

While this architecture can simplify development and deployment, it can also lead to scalability and maintainability issues as the application grows in size and complexity.

Monolithic architecture gives you a lightweight, all-in-one bundle for simple apps with well-defined, stable requirements. Since everything is tightly coupled, development and deployment can be fast, especially for apps with limited scope and low chances of change.

3. Microservice architecture

The microservice architecture breaks the application into smaller, independent services, each responsible for specific functionalities. These services communicate through APIs, offering flexibility and scalability.

You can develop, deploy, and scale microservices independently, making updating and maintaining the application easier. However, managing many services can introduce complexity, and additional overhead may be associated with coordinating communication between services.

Microservices architecture excels when you need to update complex apps and scale them across multiple teams frequently. By decomposing into discrete services, you can independently develop, deploy, and scale components to accelerate iteration for large, evolving applications.

**Key principles of mobile application architecture**

Like pillars supporting a building, principles in mobile architecture provide foundational guidance so you can construct an app that is stable, scalable, and prepared for future growth. Let’s explore vital architectural principles to be considered, when designing mobile apps:

|  |  |
| --- | --- |
| **Principles** | **Summary** |
| Flexibility | The architecture can adapt to changing requirements and new technologies |
| Maintainability | The app is easy to maintain via modularity, loose coupling, and encapsulation |
| Reusability | Components and modules can be reused across applications |
| Security | Data and identity are protected through access controls and encryption |
| Performance | The app delivers speed, reliability, and resource efficiency |
| Sustainability | The architecture supports continuity over changes and future growth |
| Extensibility | New capabilities can be added via plugins, extensions, and integrations |
| Testability | Components can be easily tested in isolation |
| Intuitiveness | The architecture follows established and familiar patterns |
| Portability | The app can be deployed on different mobile platforms |

**Factors to consider while designing a mobile app architecture**

Device type, user interface, push notifications, navigation method, etc., are a few of the several factors that help build a robust mobile app architecture.

Let’s discuss them briefly:

a. Device type

The device type determines the hardware capabilities and screen sizes your utility needs to support. For example, designing an app for smartphones and tablets requires accommodating different screen resolutions and aspect ratios.

Imagine you’re developing a fitness-tracking application. In that case, you should ensure that these small smartphone screens are optimized for large tablet displays for an immersive user experience regardless of the device used.

When evaluating device types for your mobile app architecture, you must consider features like screen size, resolution, memory, processors, battery life, sensors, camera capabilities, and OS versions.

b. Development framework

Selecting the right development framework is vital for efficient application development and maintenance. For example, if you aim to build cross-platform applications, frameworks like React Native or Flutter allow you to write code once and deploy it across multiple platforms.

On the other hand, if you prioritize performance and native experience, platform-specific frameworks like Swift for iOS or Kotlin for Android might be preferable.

c. Bandwidth

The available bandwidth directly impacts how your app communicates with servers and fetches data. Designing your app to minimize data usage becomes paramount in regions with limited internet connectivity. For instance, if creating a video streaming app, you might implement adaptive bit streaming to adjust video quality based on the user’s network speed, ensuring smooth playback even in low bandwidth conditions.

d. Network fluctuations

Mobile networks are prone to varying signal strength and stability fluctuations, which can affect app performance. Designing your app to handle network fluctuations gracefully is essential for providing a seamless user experience. Implementing caching mechanisms to store frequently accessed data locally can reduce reliance on real-time network requests and mitigate the impact of temporary network outages.

e. User interface

The user interface (UI) plays a crucial role in shaping the overall user experience of your app. It should be intuitive, visually appealing, and consistent across different devices and screen sizes. For instance, if you are designing a shopping app, you should prioritize easy navigation, intuitive search functionality, and visually appealing product displays to enhance the shopping experience for users.

f. Navigation method

The navigation method defines how users navigate your app and access its features and content. Whether you opt for tab-based navigation, drawer menus, or bottom navigation bars depends on the complexity of your app and user preferences. For example, a news app might use tab-based navigation to allow users to quickly switch between different sections like top stories, sports, and entertainment.

g. Real-time updates vs. push notifications

Deciding between real-time updates and push notifications depends on the nature of your app and the importance of timely information delivery. For example, a messaging app requires real-time updates to ensure instant user communication. In contrast, a news app might use push notifications to notify users of breaking news stories or personalized updates based on their interests.

**Mobile app architecture patterns**

Mobile app architecture patterns serve as blueprints that define the structure, behavior, and interaction of various components within an app.

Here are the four major patterns:

1. Model-View-Controller (MVC)

MVC separates an app into three interconnected components: Model, View, and Controller. The Model manages your data and app logic; the View displays the UI, and the Controller handles user input and interactions.

This separation of concerns allows for better code organization, reusability, and testability. However, it can lead to massive View Controllers and tight coupling between components, which can make maintenance challenging as the app grows.

Using MVC is better when you need to manage complex user interfaces and want to organize your codebase in a structured and maintainable manner.

2. Model-View-Presenter (MVP)

MVP addresses some of the limitations of MVC, particularly concerning testability and separation of concerns. In MVP, the View only shows data and captures user input. The Presenter acts as an intermediary between the View and the Model, handling all the logic related to user interactions and data manipulation. This separation makes it easier to test the Presenter without involving the Android framework or UI components.

MVP promotes cleaner code and improved maintainability by reducing the dependencies between different app layers.

MVP architecture is better for faster development and easier maintenance because it promotes a clear separation of concerns and facilitates iterative development cycles.

3. Model-View-ViewModel (MVVM)

MVVM, originating from Microsoft and widely used in Android app development, introduces architectural components like LiveData and ViewModel. In MVVM, the ViewModel acts as a link between the View and the underlying data sources.

Unlike MVP, MVVM enables data binding, where changes in the ViewModel are automatically reflected in the View without manual updates. This enhances the separation of concerns, improves code readability, and supports reactive UIs.

MVVM is a good choice for apps that require a structured and modular design, especially those with complex UIs and interactions.

4. View-Interactor-Presenter-Entity-Router (VIPER)

VIPER stresses modularity, scalability, and testability by dividing an app into layers: View, Interactor, Presenter, Entity, and Router. Each layer has a specific responsibility, such as handling user interactions, business logic, data manipulation, and navigation.

VIPER enforces firm boundaries between components, reducing coupling and making it easier to replace or modify individual modules without affecting the rest of the app. While VIPER offers excellent separation of concerns and scalability, it introduces more complexity upfront, requiring careful planning and implementation.

You should prefer VIPER architecture when you want a clear separation of concerns in your iOS apps, especially in complex projects where scalability and maintainability are crucial.

**Example of modern mobile application architectures**

A few examples of modern mobile app architecture include Android, iOS, hybrid, and enterprise.

1. Android mobile app architecture

Android mobile app architecture typically follows the MVVM pattern, leveraging frameworks like Jetpack for robust mobile app development. Components like LiveData and ViewModel help manage lifecycle awareness and data persistence, ensuring a responsive and scalable app.

2. iOS mobile app architecture

iOS mobile app architecture often revolves around the MVC or MVVM design patterns.

Apple’s SwiftUI framework has gained popularity for its declarative syntax and real-time previews for rapid development. With SwiftUI, you can create responsive and intuitive user interfaces while adhering to modern design principles. Additionally, the Combine framework facilitates reactive programming, enabling seamless data flow and event handling in iOS apps.

3. Hybrid mobile app architecture

Hybrid mobile app architecture combines elements of web and native app development, allowing you to build cross-platform apps using web technologies like HTML, CSS, and JavaScript. While hybrid apps offer broad platform compatibility and faster time-to-market than web apps, they may face performance limitations and restricted access to native APIs.

4. Enterprise mobile app architecture

Enterprise mobile app architecture addresses the unique requirements of large-scale business applications, emphasizing security, scalability, and integration with existing systems.

Enterprise mobility platforms like SAP Mobile Platform, IBM MobileFirst, and Oracle Mobile Cloud provide tools and services for building, deploying, and managing enterprise-grade mobile apps with security features such as data encryption, authentication, and role-based access control at the forefront.

**Choosing the right mobile application architecture**

Selecting the right mobile architecture is crucial because it impacts user experience, performance, scalability, security, and the overall success of the app. To level up your app to enterprise-grade, you need to opt for the right architecture, which comes with challenges such as balancing client-server communication and offline support, determining the optimal granularity for different components, and managing the complexity of transitioning to microservices