Table of Contents

C1.1 Introduction to Mobile Application Development

Comparison of Apps

	Native	**Hybrid**	**Mobile Web**
Cost	High	Low	Low
Performance	Fast	Depending on speed of network	Depending on speed of network
Distribution	App stores	App stores	None
Device features	Wide	Limited	Very limited
Code maintenance	Multiple codebase	Single codebase	Single codebase

Key Mobile Challenges

Processing power	• **Limited CPU/GPU performance** compared to desktops/laptops. • **Thermal constraints** restrict high-speed processing to avoid overheating. • Energy efficiency is prioritized over raw power. • Impact: **Slower execution of complex tasks**; developers must optimize apps for performance and responsiveness.
Memory and storage	• **Limited RAM** affects multitasking and app stability. • **Restricted internal storage** limits app installations, media, and updates. • Impact: **Apps must be lightweight and manage memory/storage efficiently**. Cloud solutions help but depend on network quality.

Battery	• **Small battery size due to compact design**. • High power consumption from screens, processors, and radios. • User expectations for long battery life despite heavy usage. • Impact: **Developers must optimize for low power usage**; hardware innovation is needed for better battery tech and management.
Network	• **Reliance on wireless connectivity (Wi-Fi, 4G/5G) can be unstable or slow**. • Data costs may limit usage. • **Latency** issues affect real-time applications like gaming and video calls. • Impact: **Apps must handle poor connectivity gracefully**, use caching, and sync data efficiently.

Beyond Mobile App

- Wearable
- Smart Home Appliance
- Smart Car Dashboard
- Internet of Things (IoT)
- Augmented Reality (AR)
- Artificial Intelligence (AI)
- Blockchain

Mobile Application Software Architecture and Framework

Flutter

Description	• An **open-source mobile application development framework created by Google** • Used to **develop applications for Android and iOS, web and desktop** (Windows, Mac OS, Linux, etc) • It is an SDK

	• Language: **Dart**

Key Features	• **Cross-platform development**
	  ○ Write once, run on Android, iOS, Web, Windows, macOS, and Linux using a single codebase.
	• **Hot reload**
	  ○ Instantly see code changes in the app without restarting—great for fast development and debugging.
	• **Rich widgets**
	  ○ Flutter offers a wide range of pre-built and customizable widgets for building beautiful UIs easily.
	• **Customizable**
	  ○ You can fully customize widgets and UI elements to match any design, thanks to Flutter's flexible architecture.
	• **High performance**
	  ○ Flutter uses its own rendering engine (Skia), which enables smooth animations and fast UI rendering.
	• **Open-source**
	  ○ Flutter is free and backed by a strong community, with regular updates and lots of plugins and packages.

Framework	• A programming language used to build Flutter apps
	• Offers features:
	  ○ **Strong typing**
	  ■ Dart (Flutter's language) uses strong typing, which means variables must have a defined type. This helps catch errors early and improves code reliability.
	  ○ **Asynchronous programming**
	  ■ Flutter supports async and await, allowing smooth handling of tasks like network requests or file I/O without blocking the UI.
	  ○ **Just-In-Time (JIT) compilation** for rapid development
	  ■ Used during development, JIT compiles code on the fly, enabling hot reload and faster iteration.
	  ○ **Ahead-of-Time (AOT) compilation** for production builds to

	improve performance ■ Used for production builds, AOT compiles the app into native machine code, resulting in faster startup and better performance.
Engine	• Foundation of Flutter mainly written in **C++** • **Handles platform-specific tasks** such as rendering, input and network requests • Implemented the **Flutter framework** and **Dart runtime**
Embedder	• A platform-specific embedder **coordinates with the underlying operating system** • Written in a language that is appropriate for the platform: ○ Java and C++ for Android ○ Objective-C / Objective-C++ for iOS and macOS ○ C++ for Windows and Linux • With embedder, **Flutter code can be integrated into an existing application as a module**, or the code might be the entire content of the application
Reactive UI	• Focuses on **data flow** and **propagation of change** • Building **UI that react to changes in data in a declarative and efficient manner** • 3 key concepts: ○ **Data Flow** ○ **Declarative UI** ○ **State Management**

3 Key Concepts

Data Flow

- **Data flows unidirectionally from the source to the UI**
- **Changes to the data source trigger updates to the UI**
- Reactive UI - a programming paradigm that focuses on data flow and propagation

of change

- UI automatically update whenever the underlying data changes
- Leads to more **responsive, efficient and maintainable applications**

Declarative UI

- **Describe the desired UI state**, and **Flutter handles the rendering and updates**
- Makes it easier to reason about the UI and **write less code**
- **Widget** is the fundamental of UI

"Constraints go down. Sizes go up. Parent sets position."

- A **widget can decide its own size only within the constraints given to it by its parent**.
- **Widget's parent who decides the position of the widget**
- It is **impossible to precisely define the size and position of any widget without taking into consideration the parent-child tree** as a whole
- If a child wants a different size from its parent and the **parent does not have enough information to align it**, then the **child's size might be ignored**.

State Management

State

- The **condition or data of a system at a particular moment**
- A **snapshot** of the system's variables, properties and other data that define its behavior
- In Flutter, state determines the **dynamic behavior of your app**. Two main types of state:
  - **App State**
  - **Widget State**

App State	• **Global state that affects the entire app** • Managed using libraries like:

	○ Provider ○ Riverpod ○ BLoC
Widget State	• **Local state specific to a particular widget** • Managed using: ○ StatefulWidget ○ setState

React Framework

Feature	**Traditional UI**	**Reactive UI**
Programming Paradigm	Imperative	Declarative
State Management	Manual	Framework-assisted
UI Updates	Manual	Automatic
Performance	Can be less efficient	More efficient

C2: Mobile Application Models

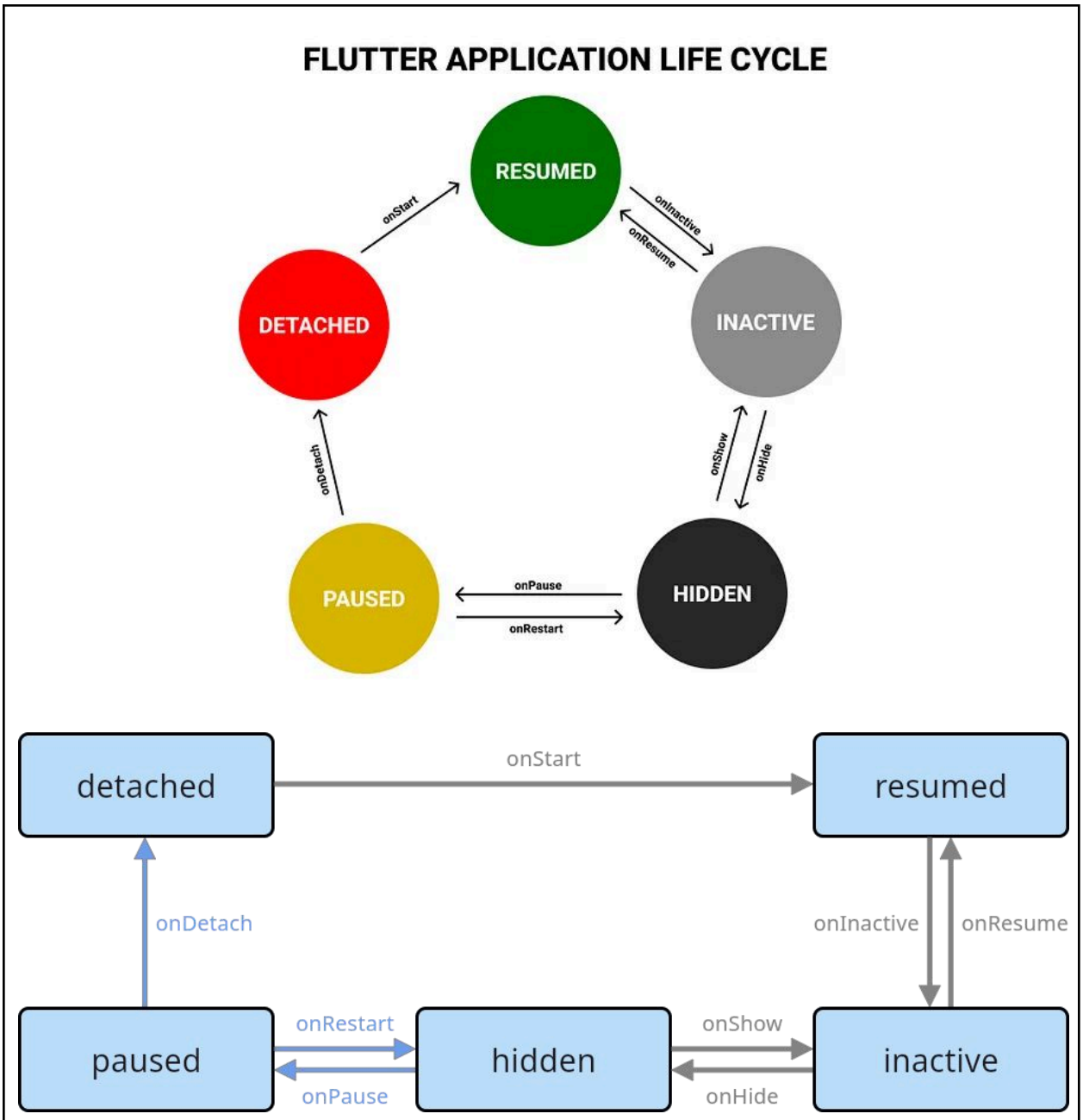
Route and Navigator

- In **Flutter**, screens and pages are called **routes**.
- In **Android**, a route is equivalent to an **Activity**.
- In **iOS**, a route is equivalent to a **ViewController**.
- In **Flutter**, a route is just a **widget**.

- When a user interacts with your app, they navigate between these routes
- For advanced navigation and routing requirements, use a routing package such as *go_router*.
- The Navigator widget displays screens as a **stack** using the correct transition animations for the target platform
- Navigator is suitable for small applications without complex deep linking
- To navigate to a new screen, access the Navigator through the route's BuildContext and call imperative methods such as *push()* and *pop()*

App Lifecycle

- Flutter app lifecycle: Different states a Flutter app can be in throughout its runtime
- Essential for building efficient and responsive apps
- Create more polished and user-friendly experience in Flutter applications



Resumed	• Ideal state where the app is **visible** on the screen, actively **responding to user interfaces**
	• Actions can be taken:
	  ○ Fetch and display data relevant to the user
	  ○ Process user interactions (taps, swipes, form submissions)
	  ○ Play audio or video

	○ Start animations ● Any functionality core to your app's active use happens here
Paused	● State where the app is **not visible** to the user and has **limited functionality** ● Usually in background ● Good time for: ○ Pausing tasks that consume resources (like animations or location updates) ○ Saving any unsaved user data ○ Releasing resources that aren't immediately needed ● **Avoid intensive tasks** here as the app might be suspended entirely soon 避免执行密集型任务，因为应用程序可能很快就会完全暂停
Inactive	● Brief transitional state between **resumed** and **paused**, where the app might **lose focus** but still resides 居住 in memory ● Might **not have time for complex actions** here, but you could: ○ Briefly pause ongoing tasks (like a network request) in case the app resumes quickly ○ Prepare for a potential state change (resumed or detached)
Detached	● State signifies the app is **no longer attached to any view** and **isn't actively running** ● Occurs before initialization and potentially after all views are detached (on Android and iOS) ● Good time for: ○ Cleaning up resources (closing databases, releasing memory) ○ Persisting any critical data that needs to be saved even after the app is closed ● **Avoid actions that require UI elements or user interaction**
Key Points: ● You can monitor app lifecycle changes using the *AppLifecycleState* enum and the *AppLifeCycleListener* class	

- Effectively handling lifecycle transitions allows for optimizing resource usage, performing actions when the app goes into the background (like pausing timers or saving data), and respond appropriately when it resumes

State	Resumed	Paused	Inactive	Detached
UI	Visible	Not visible	Not visible	Not visible
Interaction	Yes	No	No	No
Memory State	Maintain	Maintain	Maintain	Removed

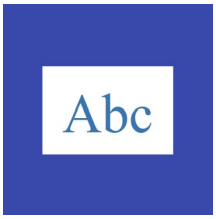
C3.1: User Interfaces

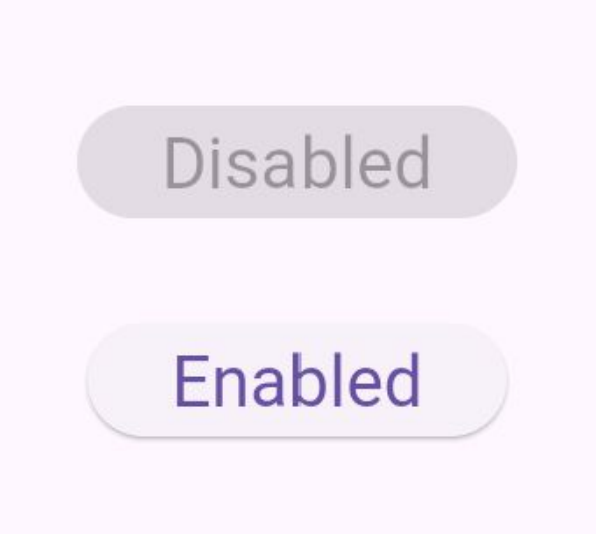
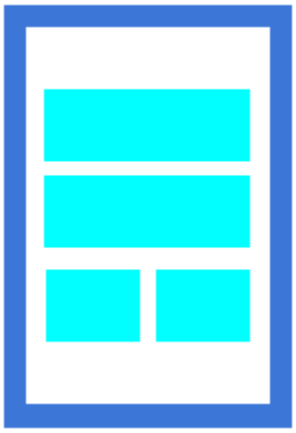
Widgets

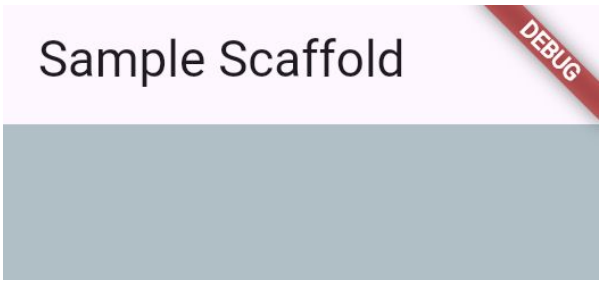
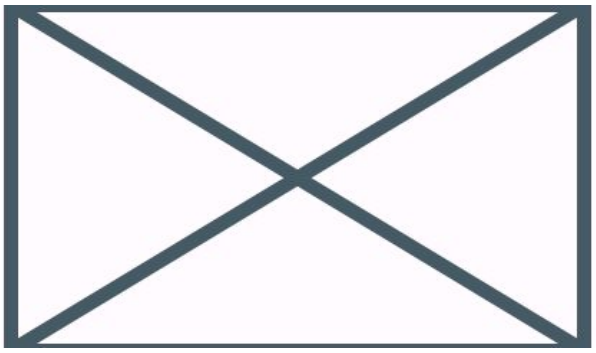
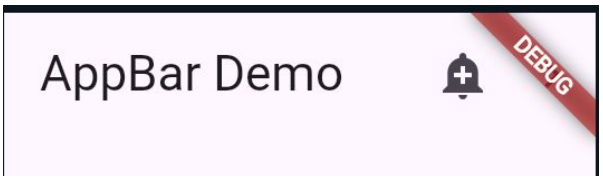
- Widgets are UI components of your app. They define the:
  - **appearance**
  - **layout**
  - **behavior of a specific part of your app's interface**
- A widget represents a **single element** (e.g. button, text field, image) or even a **complex layout container**
- Widgets are **nested hierarchically**, forming a tree-like structure that defines the **overall UI of your app**

Characteristics	Description
Customizable	• Most widgets offer various **properties that you can adjust** to **customize their appearance and behavior** • Allows for **creating unique and personalized UIs**
Reusable	• Widgets are designed to be **reused throughout your app**, promoting **code efficiency and consistency**

Basic Widgets

Text	
TextField	

ElevatedButton	
Image	
Row / Column	

Scaffold	
Placeholder	
Icon	
AppBar	

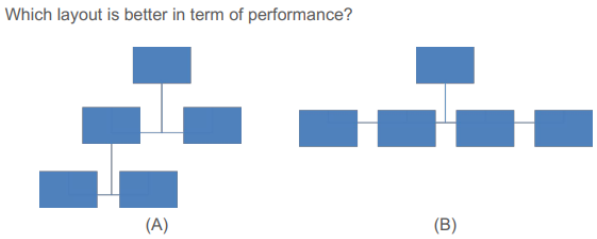
Stateless vs Stateful Widget

Stateless Widgets	**Stateful Widgets**
Simpler and have a **fixed appearance and behavior**	**Maintain their own state**, which **react to user interactions** or **changes in data**

Don't hold any internal state	**Manage state through a State object**

Performance Best Practices

Minimize Widget Rebuilds	• **Avoid repetitive and costly work in *build()* methods** since it can be invoked frequently when ancestor widgets rebuild
	• **Use *const* for immutable widgets**
	  ○ This tells Flutter that the widget's properties won't change, leading to optimizations
	• **Utilize *const* constructors**
	  ○ Use *const* constructors for widgets that don't require state

Optimize Layout	• **Avoid unnecessary nesting**
	  ○ Keep your widget tree as flat as possible to reduce rendering time
	• **Use *LayoutBuilder***
	  ○ For dynamic layouts that depend on available space, use *LayoutBuilder*
	 
	• Layout B is better in term of performance
	• It has **fewer levels of nesting** (2 levels)
	• Flatter tree means:
	  ○ **Faster rendering**
	  ○ **Less memory usage**
	  ○ **Reduced complexity in rebuilds**

Efficient Image	• **Use *Image.network***

Loading	○ For network images, use *Image.network* with appropriate caching and loading strategies ● **Optimize Image Size** ○ Resize images to the appropriate size for your app ● **Use efficient image formats like WebP**, which is raster graphics 光栅图形 file format to replace JPEG, PNG and GIF
State Management	● **Use *setState* judiciously** 审慎 ○ Only call *setState* when necessary to trigger a rebuild ● **Consider state management solutions** ○ For complex state management, explore options like Provider, Riverpod (builds on the Provider package), or Business Logic Component (BLoC)
Use Flutter's Build-in Widget	● **Use Flutter's optimized widgets** whenever possible

C3.2: Platforms, Themes and Design for Everyone

Colors

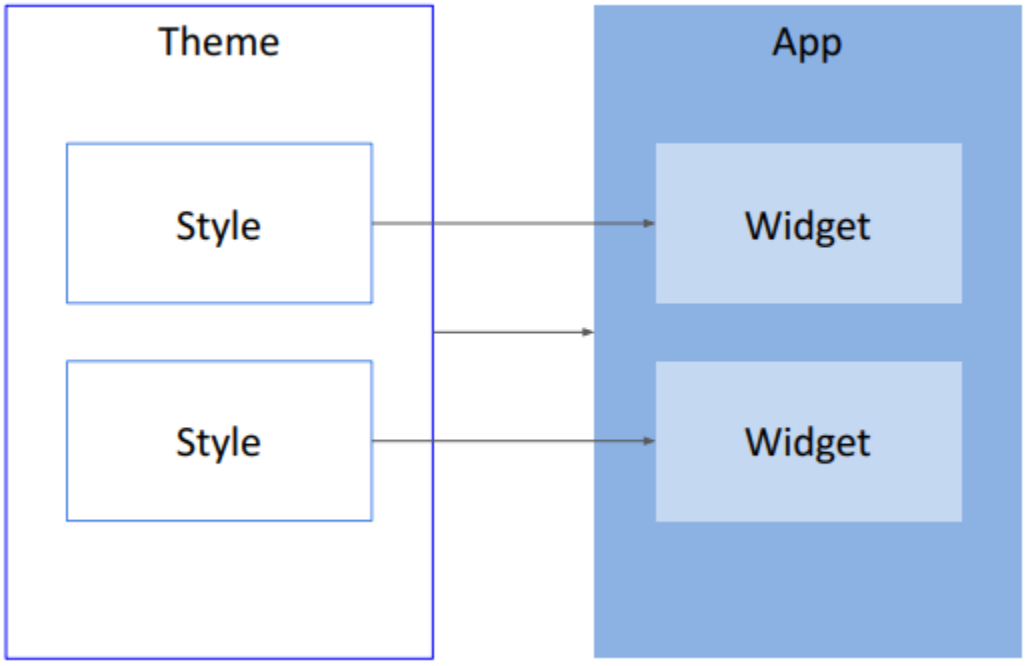
Colors

- Color and ColorSwatch constants which represent **Material design color palette**
- Most swatches have colors from 100 to 900 in increments of one hundred, plus the color 50
- The **smaller the number, the more pale the color**. The **greater the number, the darker the color**.
- The accent swatches only have the values 100, 200, 400 and 700

ColorSwatch

- A class that represents a **collection of colors related to a single color theme**
- Useful for **creating color palettes that vary in shades, tints and tones of a base color**

Styles and Themes



Themes

- To **share colors and font styles throughout an app**
- Allow to **define app-wide themes**
- Allow to **extend a theme to change a theme style for one component**
- Each theme defines the color, type style, and other parameters applicable for the type of Material component

Flutter applies styling in the following order:

1. **Styles applied to the specific widget**
2. **Themes that override the immediate parent theme**
3. **Main theme for the entire app**

Most instances of *ThemeData* set values for the following 2 properties:

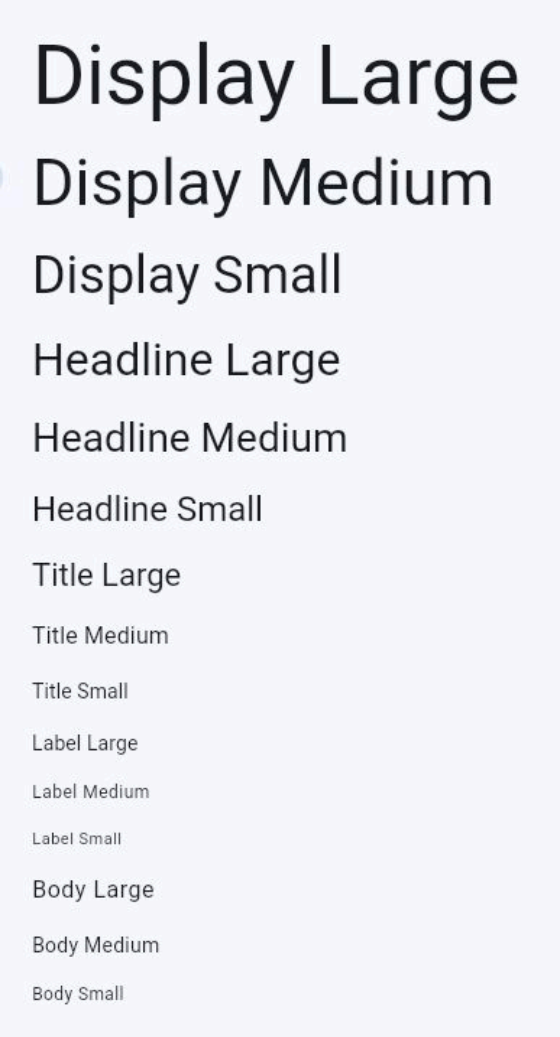
- ***colorScheme*** defines the **colors**
- ***textTheme*** defines **text styling**

Override a Theme

- To override the overall theme in part of an app, **wrap that section of the app in a *Theme* widget**
- Override a theme in 2 ways:
    - **Create a unique *ThemeData* instance**
    - **Extend the parent theme**

Font and Typography

- Typography covers the style and appearance of fonts
- Font Style:
  - Typeface: a set of common character rules (e.g. Roboto)
  - Size (e.g. Size 18)
  - Style (e.g. Regular, Italic)
  - Weight (e.g. Normal, Bold)
- TextTheme:
  - Display
  - Headline
  - Title
  - Label
  - Body
- Size Variation:
  - Small
  - Medium
  - Large



Design for Everyone

- Design philosophy that aims to create products and services that are **accessible** and **usable** by as many people as possible, regardless of their **age**, **ability** or **background**
- In context of **mobile app development**, it means making app accessible to as many as people as possible, regardless of their **ability**, **language** or **device**

Method	**Description**
Internationalizing	• Making app **adaptable to different languages, cultures and regions**

	• Example:
	  ○ Language Translation
	  ○ Generate Localization Delegates
	  ○ Date, Number and Currency Formatting
	  ○ Text Direction
	  ○ Cultural Nuances 文化差异

Supporting different screens	• **Adaptive Design**
	  ○ Utilize *Flexible* and *Expanded* widgets to help distributing space among child widgets
	  ○ Responsive images: Use *FittedBox* to scale images to fit the available space
	• **Platform-Specific Considerations**
	  ○ Provide alternative bitmap resources for good graphical quality and performance
	• **Consider screen orientation and slit screen**
	• **Support various display cutouts**
	• **Specify minimum and target API levels**
	  ○ **minSdkVersion**: the lowest API level with which your app is compatible
	  ○ **targetSdkVersion**: the highest API level against which you've designed and tested your app

Make apps more accessible	• Regardless of ability, users are **able to navigate, understand and use an app successfully**
	• Consideration:
	  ○ **Navigation**
	  ○ **Readability**
	  ○ **Guidance and Feedback**
	
	<u>Navigation</u>
	• **Support screen readers by *Semantics* 语义**
	• **Create easy-to-follow navigation**

	○ Support keyboards or gestures input ○ Avoid having UI elements fade out or disappear after a certain amount of time ○ Create flat navigation structure ● **Make touch targets large** ○ Min size 48 × 48 pixel ○ Space between elements min 8dp ● **Gesture Navigation** ○ Ensure your apps can extend content from edge to edge ○ Ensure your apps can handle conflicting gestures <u>Readability</u> ● **Provide adequate color contrast** ○ Use more than just color to convey information ● **Make media content more accessible** ○ Include controls for users to pause or stop video and audio files ○ Provide transcript / caption <u>Guidance and Feedback</u> ● **Make interactive controls clear and discoverable** ○ Interactive controls have text labels, tooltips or placeholder text to indicate their purpose ○ When naming elements, be consistent in terminology 术语 throughout your app

C3.3: State Management and Navigation Principles

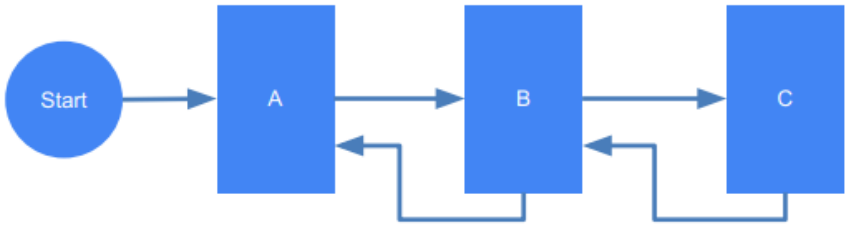
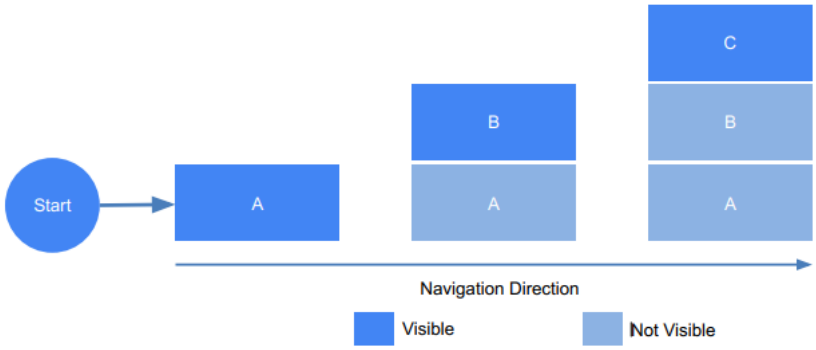
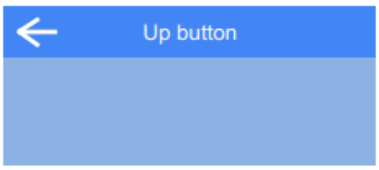
State Management

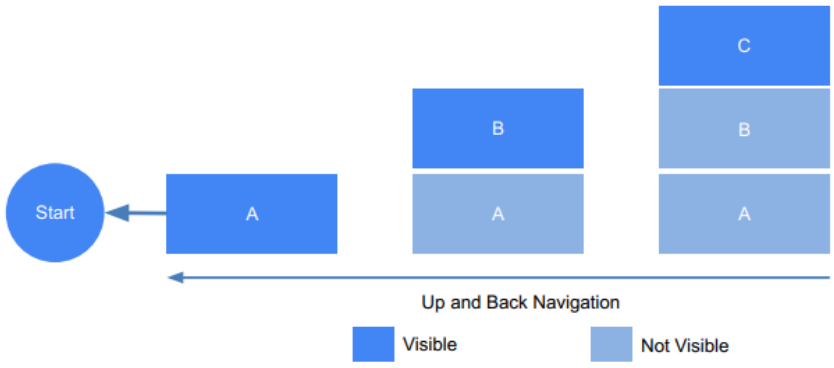
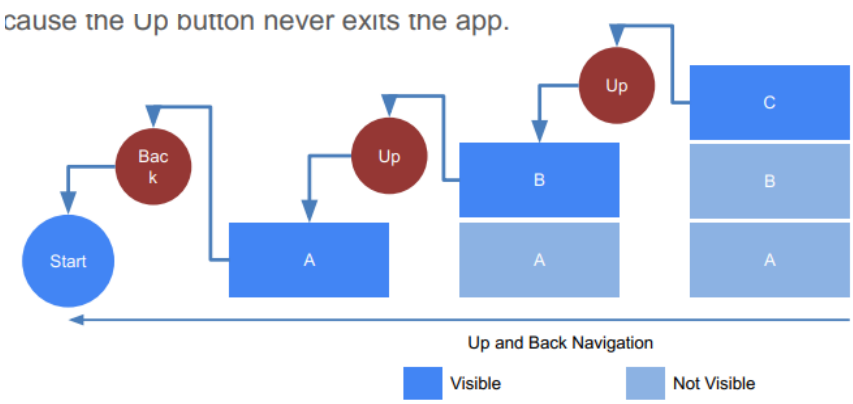
- State management involves **managing the data that changes over time** and **triggers UI updates**
- State of an app is **everything that exists in memory when the app is running**
- Includes the app's assets, all the variables that the Flutter framework keeps about the UI, animation state, textures, fonts and so on
- Provider
    - Provides a **simple way to share state across multiple widgets**
    - Uses *ChangeNotifier* to notify listeners when the state changes

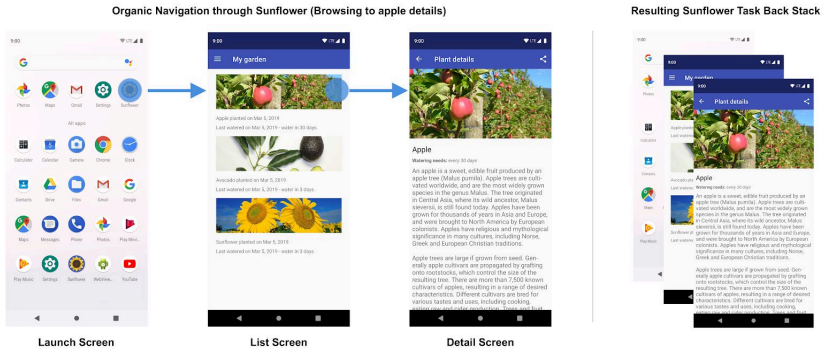
Ephemeral 短暫的 State	**App State**
A.k.a. **UI state or local state**	A.k.a. **shared state**
It is the state you can neatly **contain in a single widget**	UI state that you want to **share across many parts of your app**, and that you want to **keep between user sessions**
Examples: ● Current page in a PageView ● Current progress of a complex animation ● Current selected tab in a BottomNavigationBar	Examples: ● User preferences ● Login info ● Notifications in a social networking app ● Shopping cart in e-commerce app ● Read / unread state of articles in news app

Navigation Principles

Fixed start destination	● Every app you build has a fixed start destination

	• This destination is also the last screen the user sees when they return to the launcher after pressing the Back button 
Navigation state is represented as a stack of destinations	The top of the stack is the current screen, and the previous destinations in the stack represent the history of where you've been 
Up and Back are identical within your app's task	• The Up button appears in the app bar at the top of the screen • Within your app's task, the Up and Back buttons behave identically   When you press the Back button, the current destination is popped off the top of the back stack, and you then navigate to

	the previous destination 
The Up button never exits your app	If a user is at the app's start destination, then the Up button does not appear, because the Up button never exits the app 
Deep linking simulates manual navigation	• Navigation design should simulates manual navigation • e.g. From the app navigator to the main screen of an app, and then to a detail screen

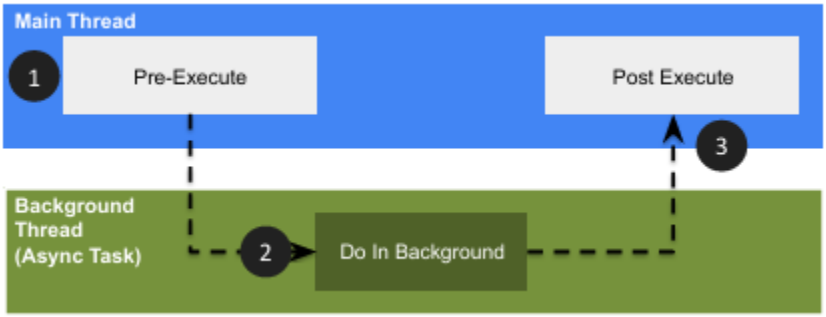
Snackbar

Component	Priority	User Action
Snackbar	Low	• **Optional** • It **disappears automatically**
Dialog	High	• **Required** • It **blocks app usage** until the user **takes a dialog action** or **exit the dialog**

C4.1: Resources and Data Storage

Main Thread (UI Thread) vs Background Thread

	Main Thread (UI Thread)	**Background Thread**
Purpose	**Rendering the UI** and **handling user interactions**	**Perform background tasks** that do not directly impact the UI
Explanation	All UI updates and user input handling occur on this thread	**Offloading tasks to background threads to ensure the main thread remains responsive** Performs background operations and public results on UI thread
Use Cases	UI updates and user input handling	Network Requests, database operations, complex calculations, file I/O operations (reading and writing files)
Asynchronous programming	-	*futures, async, await*



Saving Data

Methods	**Details**

Data Files	• **Larger amounts of data**, **structured data** (like JSON), **files** (like PDF or image) downloaded from the Internet, user-generated content
	• Private or Public
	• Data file:
	  ○ Text
	  ○ Sound
	  ○ Images

	● Use cases: ○ Persist data across app launches ○ Download data from the Internet and save it for later offline use ○ Share the data to others using instant messaging app
Shared Preferences	● **Suitable for storing small collection of key-values** ● Private or Public ● **Key-value pair** ● Limitations: ○ **Only primitive types** can be used: *int*, *double*, *String* and *List<String>* ○ It is **not designed to store large amounts of data** ○ There is **no guarantee that data will be persisted across app restarts**
SQLite	● **Lightweight, embedded SQL database engine that is perfect for mobile applications** ● Provides **faster inserts, updates and queries for large amount of data on the local storage** ● Private ● Repeating and structured data

Internal vs External Storage

Internal Storage

Definition	**Storage that is directly accessible to your app**
Use cases	Storing app data, preferences and cache
Permissions	**No permission is required** to perform read / write operations
Suitable for	● **Shared Preferences** ● **SQLite Database** * Due to security and sensitivity of data storage

External Storage

Definition	**Storage that is accessible to multiple apps on the device and outside of the device**
Use cases	Storing large files or media
Features	<table><tr><th>Feature</th><th>Android</th><th>iOS</th></tr></table>

<table>
<tr><td></td><td>
<table>
<tr><td>Expandable Storage</td><td>Yes</td><td>No (generally)</td></tr>
<tr><td>Removable</td><td>Possible (SD Card)</td><td>No</td></tr>
<tr><td>File System Access</td><td>More flexible</td><td>More restricted</td></tr>
<tr><td>Permissions</td><td>Requires explicit user permission</td><td>More controlled by the system</td></tr>
<tr><td>Data Sharing</td><td>Easier to share files between apps</td><td>More limited sharing options</td></tr>
</table>
</td></tr>
<tr><td>Considerations</td><td>

Permissions

Ensure that your app has the necessary permissions to access storage

Data security

Consider encrypting sensitive data before storing it

Performance

Optimize file I/O operations to avoid performance bottlenecks

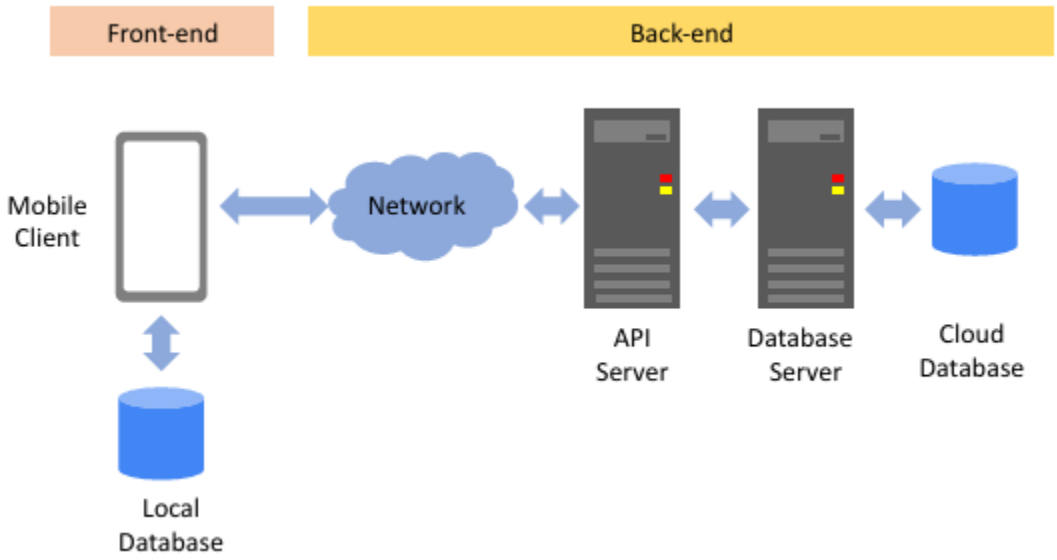
User experience

Provide clear feedback to the user during file operations, such as progress indicators

</td></tr>
</table>

C4.2: Network Operations

Mobile-to-Server Communication



Front-End (User Side)	
Mobile Client	• Actual mobile application installed on the user's device • Provides the **user interface (UI)** and handles **user interactions** (e.g. tapping buttons, filling forms) • Manages **local logic** like input validation, caching and sometimes offline features
Local Database	• A light weight database stored on the device such as SQLite and Realm • Used for **offline access**, caching frequently used data and reducing network calls • e.g. A messaging app stores recent chat history locally so you can read it without Internet
Network (Bridge)	
Network Layer	• Acts as the **communication channel** between the mobile client and the back-end • Uses protocols like HTTPS/REST, WebSockets, or gRPC to send requests and receive responses • Ensures **secure and reliable data transfer**
Back-End (Server Side)	
API Server	• The "middleman" that receives requests from the mobile app

	• Handles **business logic**: authentication, authorization, data processing and routing requests to the right services • e.g. When you log in, the API server checks your credentials and issues a token
Database Server	• Manages structured data storage and retrieval • Executes **queries** requested by the API server • Ensures **data consistency, indexing and transaction management** • e.g. When you search for a product, the API server queries the database server for matching records
Cloud Database	• A scalable, distributed database hosted in the cloud (e.g. AWS RDS, Firebase, MongoDB Atlas) • Provides **high availability, redundancy and scalability** for large amounts of data • Often used for **global apps** where millions of users need fast access

Server Options

Factors to Consider	
Project Scope	How complex is your app? How much data will it handle?
Budget	What is your budget for development and ongoing maintenance?
Team Expertise	Do you have in-house expertise in server-side development?
Scalability Requirements	How much growth do you anticipate for your app?
Time-to-Market	How quickly do you need to launch your app?
Server Options	

DIY (Do It Yourself)	• Advantages
	  ○ You have **complete control over the technology stack, data and infrastructure**
	  ○ It can become **cost-effective** in the long run if you manage it well
	  ○ You **earn valuable experience** in server-side development
	• Disadvantages
	  ○ **High initial investment** for setup, maintenance and scaling
	  ○ **Expertise required** in server administration, networking, security and database management
	  ○ Requires **ongoing maintenance and troubleshooting**

	● Best suited for: ○ **Large enterprises or startups with strong technical teams** ■ e.g. A fintech company 金融科技公司 that must comply with strict regulations such as banking apps. They need full control over data security, compliance and infrastructure ○ **Projects requiring maximum customization** ■ If your app has unique requirements that off-the-shelf solutions 现成解决方案 can't handle (e.g. custom algorithms, proprietary data pipelines 专有数据管道) ○ **Long-term cost optimization** ■ If you expect high, predictable traffic and want to avoid recurring subscription costs, owning your infrastructure may be cheaper in the long run ● **Not ideal for small teams or early-stage startups without backend expertise**, it can slow you down
Subscribe	● Advantages ○ **No need to manage servers, databases or infrastructure**. Gain access to pre-build features such as AI model, NLP, etc ○ **Providers handle scaling and performance automatically**. Allows you to focus on building your app's frontend and core functionality ○ **Fast and cost-effective deployment** - you can introduce your services to the public quickly ● Disadvantages ○ **Vendor Lock-in** 供应商锁定. It can be challenging to switch providers later. ○ May have **limitations on customization and flexibility** ○ **Costs can increase** as your app grows and usage increases ● Best suited for: ○ **Early-stage startups or solo developers** ■ You want to launch quickly without worrying about servers, scaling or security ○ **Apps with unpredictable growth** ■ e.g. A social media app that could suddenly go viral 突然病毒式传播. Providers handle auto-scaling so you don't crash ○ **Projects needing advanced features out-of-the-box** ■ Many providers offer AI, NLP, authentication, push notifications, analytics - things that would take months to build yourself

	○ **Proof-of-concept or MVPs** ■ If you need to validate an idea fast and cheaply before investing in custom infrastructure ● Watch out for vendor lock-in and rising costs as usage grows
Mix and Bang	● Advantages ○ It is **flexible**. Combines the benefits of DIY and BaaS ○ **Customizable**: Allows you to choose and manage specific components while relying on BaaS for others ○ **Improved Scalability**: Can scale specific components as needed ● Disadvantages ○ **Increased Complexity**: Requires careful planning and coordination ○ **Requires a deeper understanding** of both approaches ● Best suited for: ○ **Growing startups transitioning from MVP to scale** ■ e.g. You start with a BaaS for speed, but later move critical components (like payments or recommendation engines) to your own servers for control and optimization ○ **Apps with mixed requirements** ■ Some parts need tight control (e.g. sensitive user data), while others can rely on managed services (e.g. notifications, file storage) ○ **Teams with moderate expertise** ■ You have enough backend knowledge to manage certain components but want to offload the rest to providers ○ **Scalability-focused projects** ■ You can scale specific modules independently - DIY for performance-heavy parts, BaaS for commodity services 商品服务 ● Downside: More moving parts = more complexity in integration, monitoring and troubleshooting

C5: Location-Based Services

Source of Location Data

Type	**GPS**	**Network-Based**
Data Accuracy	High	Low
Speed	Slow	Fast
Power Consumption	High	Low
Environment	Outdoor	Indoor and outdoor

Battery Drain

Aspects	**Details**
Accuracy	● Refers to the **precision of the location data** ● **Higher accuracy** often requires **more power and resources**
Frequency	● Determines **how often the device's location is updated** ● A **higher frequency** can **drain the battery faster**
Latency	● Measures the **delay between the actual location change** and the **time it takes for the app to receive the updated location** ● The **lower latency** will **drain battery faster**

When to use High / Medium / Low of Accuracy / Frequency / Latency

Factors	**High**	**Medium**	**Low**
Accuracy	● GPS-level, ~1 - 5m ● Needed when exact positioning matters ● Examples: turn-by-turn navigation, ride-hailing pickup, fitness tracking (running / cycling routes) ● Trade-off: Drains battery faster,	● ~10 - 50m ● Good enough when you just need to know the general area ● Examples: food delivery tracking (customer view), local business search, "nearby friends" features ● Balance between precision and efficiency	● 100m - 1km ● Suitable when approximate location is fine ● Examples: weather apps, location-based ads, regional analytics ● Very battery-friendly

	slower to lock indoors		
Frequency	• Every second or continuous • Needed for real-time tracking • Examples: navigation apps, live sports tracking, emergency response • Trade-off: heaving battery and data usage	• Every few seconds / minutes • Works when updates don't need to be instant but should feel "live" • Examples: delivery driver updates, ride-hailing driver en route, fitness apps logging every 5 - 10 seconds	• Every few minutes / hours • Best for background or periodic updates • Examples: weather refresh, location check-ins, passive analytics • Very efficient, minimal battery impact
Latency	• Minutes delay • Fine for non-urgent, background tasks • Examples: weather updates, location-based reminders, background analytics)	• A few seconds delay • Acceptable when slight delay doesn't harm user experience • Examples: food delivery tracking, social apps showing "last seen nearby"	• Instant / near real-time • Needed when immediate response is critical • Examples: ride-hailing pickup, emergency services, AR gaming (like Pokémon Go)

Types of Accuracy

Type	**Precision**	**Hardware Use**	**Power**
High Accuracy	Most precise location possible	GPS	High
Balanced Power Priority	City block (100m)	Wi-Fi or cell tower	Less
Low Power	City-level (10km)	Wi-Fi or cell tower	Less
No Power	Receives locations from other apps	None	Very minimum

Location Strategies

Global Positioning System (GPS)	● **High Accuracy** ○ Provides **precise location data** ● **Power Consumption** ○ Can **drain the battery quickly,** especially when used continuously ● Use Cases ○ Navigation apps, outdoor activity tracking, geofencing
	Extra info: ● Best for: Outdoor, high-accuracy needs ● Examples: Navigation, fitness tracking, ride-hailing 打车服务 ● Strengths: Very precise (within a few meters) ● Weaknesses: High battery drain, slower to get a fix, poor performance indoors or in dense urban areas 密集城区
Network-Based Location	● **Lower Accuracy** ○ Relies on cell tower and Wi-Fi triangulation 三角定位 ● **Lower Power Consumption** ○ Less demanding on the device's battery ● Use Cases ○ General location-based services, weather apps, local search
	Extra info: ● Best for: Indoor or urban environments when approximate location is enough ● Examples: Weather apps, location-based ads, check-ins ● Strengths: Faster response, lower battery usage, works indoors ● Weaknesses: Less accurate (50 - 500m range depending on density of towers / Wi-Fi)
Hybrid Approach	● **Balanced Accuracy and Power Consumption** ○ **Combines GPS and network-based location** ○ Network-based for quick approximation ○ Refine with GPS if needed ● Use Cases: Most mobile apps that require location data

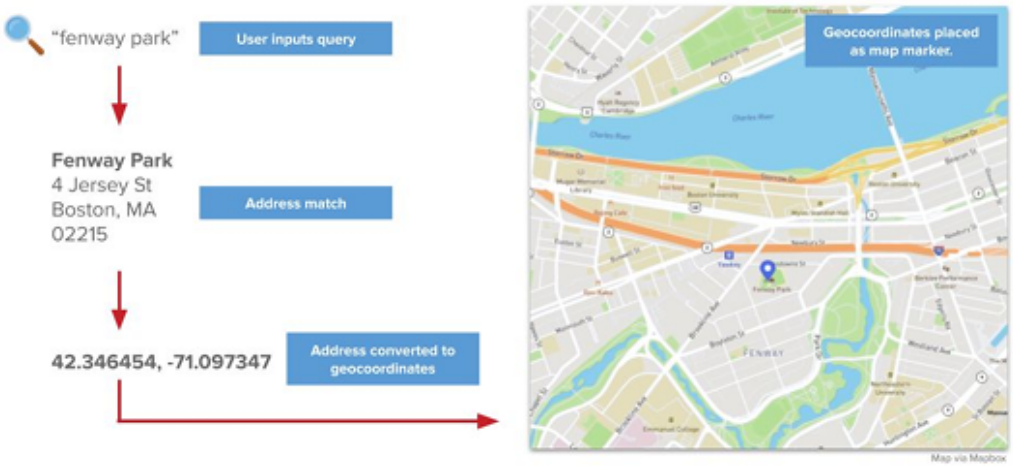
Types of Location Permission

Task / Aspect	**Foreground Location**	**Background Location**
Share Location	● Once ● Predefined period	Constant
Visibility of UI	Visible	Not visible

Show persistent notification	Yes	No
Examples	● Within a navigation app, a feature allows users to get turn-by-turn directions ● Within a messaging app, a feature allows users to share their current location with another user	● Within a family location sharing app, a feature allows users to continuously share location with family members ● Within an IoT app, a feature allows users to configure their home devices such that they turn off when the user leaves their home and turn back on when the user returns home
Insert Permission to Manifest File	● Foreground - Coarse 粗略 Location ○ Allows an app to access approximate location ○ Returns a location with an accuracy approximately equivalent to a city block 返回的位置精度大约相当于一个城市街区 ○ *<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />* ● Foreground - Fine Location ○ Allows an app to access precise location ○ *<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />*	● For Android 10 (API 29) and above ● *<uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />*

Location Services

Services	**Details**
Geocoding	● **Convert address into geographic coordinates (latitude and longitude) or the reverse** ● Geocoding APIs can **compensate for ambiguous address** 补偿模糊地址 **- ones that misspelled or inaccurate**

	• Examples:
	  ○ Street addresses can change
	  ○ Address coordinates won't
	  ○ When you type an address into a food delivery app and it finds the exact spot on the map, that is geocoding
	• Geocoding **integrates and aligns address information with geographic codes to verify the address** 地理编码将地址信息与地理代码整合和对齐，以验证地址
	• Possible errors:
	  ○ No location data provided
	  ○ Invalid latitude or longitude used
	  ○ No geocoder available
	  ○ No address found
	 

Geolocation	• **Use the device location detection hardware to determine current location**
	• Uses GPS, Wi-Fi, cell towers or IP address to estimate latitude and longitude
	• Example
	  ○ When your phone shows "You are here" on Google Maps, that is geolocation
	• Use Cases
	  ○ Navigation
	  ○ Ride-hailing pickup 乘车接送
	  ○ Location-based reminders
	  ○ Geo-fencing 地理围栏

Map	• **Display maps and markers** • Visual representation of geographic data, often layered with geolocation and geocoding results 地理数据的可视化显示，通常与地理定位和地理编码结果分层显示 • **Uses mapping libraries or providers to render roads, landmarks and overlays** • Examples of map service providers that can be used to integrate maps into Flutter app:

	○ Google Maps ○ Mapbox ○ Apple Maps ○ Here ○ Azure Maps ○ OpenStreetMap ○ TomTom ● Use Cases ○ Navigation ○ Route planning ○ Data visualization ○ Location-based analytics

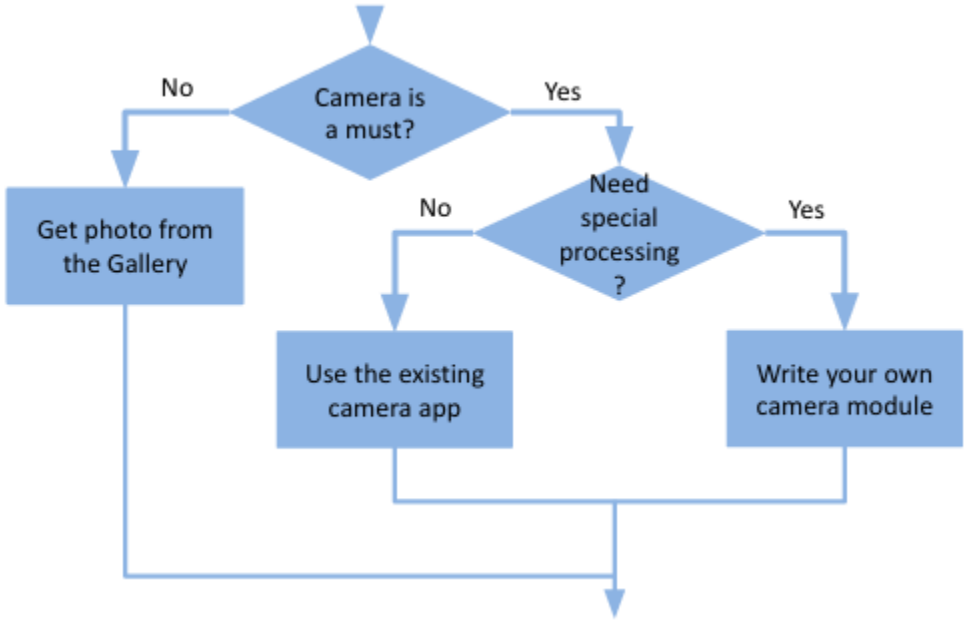
Location Best Practices

Aspects	Details
Remove location updates	● **Use *Geolocator.removeListener()*** ○ When you no longer need to track the user's location, remove the listener to conserve battery and resources ● **Check app state** ○ Only request location updates when the app is actively using them ● **User interaction** ○ Trigger location updates based on user actions, such as tapping a button or opening a specific screen
Set timeouts	● **Reasonable timeouts** ○ Set appropriate timeouts for location updates to prevent excessive battery drain ● **Dynamic timeouts** ○ Adjust timeouts based on the app's current state and user activity ● **Background mode timeouts** ○ If using background location updates, consider setting longer timeouts to balance accuracy and power consumption
Batch Requests	● **Combine multiple requests** ○ If your app needs multiple location updates, combine them into a single request to reduce network overhead and battery usage ● **Throttle Requests** ○ Limit the frequency of location updates, especially when the user is not actively using the location-based features
Passive location updates	● **Minimize battery consumption and improve user experience** ○ Rely on system-provided location data rather than continuously tracking the user's location

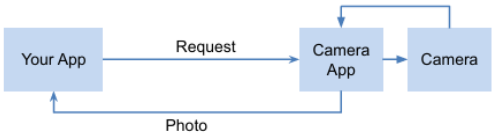
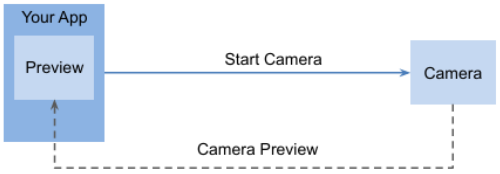
C6: Specialized Instrument and Devices

Camera

- Basic app features:
  - **Visual content creation** - take photos and videos
  - **Document scanning**
  - **QR code scanning**
- Advanced features:
  - **Augmented Reality (AR)**: Cameras can be used to overlay digital content onto the real world, creating immersive experiences
  - **Image Recognition**: Apps can recognize objects, text or faces in images to provide additional information or functionality
  - **Facial Recognition**: This technology can be used for secure authentication, unlocking devices or enabling personalized experiences
- Considerations:



	Use an existing camera app	**Build your own camera function**
Complexity	Simple implementation - using existing packages	Complex implementation - write your own code
How	Obtain a photo from the camera app or the gallery	Create a camera preview
Permission	No need permission	Request the Camera permission

Diagram		

Use Cases	• Profile picture upload (social apps, e-commerce, HR apps)
	  ○ User just needs to snap or pick a photo
	• Attach receipts or documents (banking expense tracking, insurance claims)
	  ○ No need for live edge detection or AR overlays
	• Basic QR code scanning (event tickets, Wi-Fi setup, payment links)
	  ○ Can rely on external scanner or OS-level intent
	• Simple photo / video sharing (chat apps, feedback forms)
	  ○ The native camera app already provides capture UI
		• Augmented Reality (AR) apps (IKEA Place, Pokémon GO, Snapchat filters)
		  ○ Need live camera feed to overlay 3D models or effects
		• Document scanning apps (CamScanner, Adobe Scan)
		  ○ Require edge detection, perspective correction and batch capture
		• Real-time QR / barcode scanning (GrabPay, delivery apps, inventory systems)
		  ○ Continuous scanning without leaving the app
		• Facial recognition / biometric login (banking apps, secure enterprise apps)
		  ○ Needs direct access to frames for authentication
		• Custom camera UI (Instagram, Tiktok, VSCO)
		  ○ Filters, stickers, timers, mutli-shot modes - all require control over preview and capture pipeline
		• Object or text recognition (Google Lens, translation apps)
		  ○ Must process frames in real time for OCR or classification

Media

- Plays local (assets) and external (streaming) files
- Supports any media codec that is provided by the platform and those that are device-specific
- Recommendation: use core media formats
- Use *video_player* plugin to play videos stored on the file system, as an asset or from

the internet (not available on Linux and Windows)	
Core media file formats:	
Audio	.3gp, .mp3, .mp4, .mid, .wab, .ogg
Picture	.jpg, .gif, .png, .bmp
Video	.3pg, .mp4

Sensors

Motion Sensors

Sensor	Details
Accelerometers 加速度计	● Provides data on the rate of change in **velocity** - the **speed in combination with the direction of motion** of an object, excluding gravity ● App features that use it: ○ Step counters / fitness trackers (e.g. Google Fit, Strava) ■ Detect walking, running, cycling by tracking movement patterns ○ Shake-to-trigger actions (e.g. undo typing, refresh content) ■ Apps like Gmail or note-taking apps use shake gestures ○ Fall detection (e.g. health monitoring apps for elderly users) ■ Sudden acceleration changes signal a potential fall ○ Driving behavior analysis (e.g. insurance apps) ■ Detect harsh braking 急刹 or rapid acceleration
Gravity Sensors	● It measures the **direction** and **intensity of gravity** ● Checks the **relative direction** of a device within a space ● App features that use it: ○ Screen orientation control (e.g. auto-rotate in video players or games) ■ Switch between portrait and landscape modes ○ Tilt-based controls 基于倾斜的控制 in games (e.g. racing games, ball maze games) ■ Use gravity vector to steer or balance 转向或平衡 ○ Leveling tools / spirit level apps 水平仪应用程序 (e.g. construction or DIY apps) ■ Check if a surface is flat or tilted 倾斜
Gyroscopes 陀螺仪	● It helps the accelerometer out with understanding which way your phone is **oriented** 它能帮助加速度计了解手机的方向 ● App features that use it:

	○ 360° panorama photography (e.g. Google Street View) ■ Track phone rotation to stitch images 跟踪手机旋转以拼接图像 ○ VR and AR experiences (e.g. Google Cardboard, IKEA Place) ■ Detect head movement and orientation ○ Motion-controlled games (e.g. flight simulators, shooting games) ■ Real-time rotation for immersive control ○ Gesture-based navigation (e.g. wrist flick 轻触手腕 to switch apps on wearables ■ Detect subtle rotational gestures 检测细微的旋转手势
Rotation Vector Sensors 旋转矢量传感器	● To **monitor and measure turning movements** ● **Combination of an angle and an axis** ● App features that use it: ○ AR object placement and tracking (e.g. Snapchat filters, AR measuring tools) ■ Maintain stable virtual object orientation ○ Star-gazing apps 观星应用程序 (e.g. SkyView, Star Walk) ■ Align phone with celestial coordinates 将手机与天体坐标对齐 ○ Indoor navigation / orientation (e.g. museum guides, shopping mall maps) ■ Determine user's facing direction ○ Immersive 3D experiences (e.g. educational apps, architectural walkthroughs) ■ Rotate and explore virtual environments

Environmental Sensors

Sensor	**Details**
Barometers 气压计	● It **measures atmospheric pressure** ● It assists the GPS chip inside the device to get a faster lock by instantly delivering altitude data (vertical location) ● App features that use it: ○ Altitude tracking 海拔跟踪 in fitness apps (e.g. hiking, climbing, cycling) ■ More accurate elevation gain / loss 海拔增高/降低 than GPS alone ○ Weather forecasting apps (e.g. AccuWeather, Windy) ■ Use pressure trends to predict storms or changes ○ GPS enhancement (e.g. navigation apps, location-based games) ■ Faster and more precise vertical positioning ○ Drone flight control apps ■ Use barometric pressure for altitude stabilization

Photometers 光度计	● It **sense the amount of ambient light present** ● App features that use it: ○ Auto-brightness adjustment (system-level or custom UI apps) ■ Saves battery and improves visibility ○ Camera exposure control (e.g. photography apps) ■ Adjust ISO and shutter speed based on lighting 根据光线调整 ISO 和快门速度 ○ Reading apps / eBook readers (e.g. Kindle Moon + Reader) ■ Switch between light / dark mode or adjust contrast ○ Smart home apps (e.g. lighting automation) ■ Trigger lights based on room brightness
Thermometers	● It **measures temperature within a mobile device** ● To **avoid overheating of the internal components** ● App features that use it: ○ Device health monitoring apps (e.g. CPU-Z, AIDA64) ■ Warn users about overheating or throttling 节流 ○ Gaming apps (high-performance games) ■ Adjust graphics or frame rate to prevent overheating ○ Battery management apps ■ Alert users when temperature affects battery health ○ Thermal throttling control 热节流控制 in productivity apps ■ Reduce background tasks or limit processing when hot

Position Sensors

Sensor	Details
Orientation Sensors	● **Measures the orientation of a device relative to an orthogonal coordinate frame** 测量设备相对于正交坐标框架的方向 ● App features that use it: ○ Tilt-based gaming controls (e.g. racing games, flight simulators) ■ Steer or navigate by tilting the device ○ Panorama or 460° photo capture (e.g. Google Street View) ■ Track device orientation to stitch images ○ AR experiences (e.g. Pokémon GO, IKEA Place) ■ Align virtual objects with real-world orientation ○ Virtual tours / 3D walkthroughs (e.g. real estate apps, museum guides) ■ Rotate the phone to explore environments ○ Educational apps (e.g. astronomy apps) ■ Point the device to the sky to identify stars or planets

Magnetometers	● **Determines your location with respect to Magnetic North (or South)** ● App features that use it: ○ Digital compass apps (e.g. hiking, navigation tools) ■ Show direction even without GPS ○ Navigation apps (e.g. Google Maps, Waze) ■ Improve heading accuracy, especially indoors or when GPS is weak ○ AR object placement (e.g. measuring tools, interior design apps) ■ Align virtual objects with real-world compass direction ○ Metal detector apps ■ Use magnetic field changes to detect nearby metal objects ○ Location-based games ■ Determine player orientation for directional gameplay

Other Sensors

Sensor	Details
Biometrics	● **Biometric authentication for multi-factor authentication (MFA)** to verify individual's identity that uses possession of a mobile device ● Verify unique biometric identifier ● App features that use it: ○ Secure login / authentication (e.g. banking apps, password managers) ■ Use fingerprint or Face ID for multi-factor authentication ○ App lock / sensitive action confirmation (e.g. WhatsApp, PayPal) ■ Require biometric verification to open app or approve transactions ○ Digital identity verification (e.g. eKYC in fintech apps) ■ Match biometric data with official documents ○ Personalized experiences (e.g. parental control apps) ■ Different profiles unlocked by different users
Bluetooth Low Energy (BLE)	● A **short-range wireless communication technology designed for low-power applications** ● **Low power consumption, high data rate** ● Applications: wearables, IoT, beacon technology, healthcare, indoor navigation ● App features that use it:

	○ Fitness and health tracking (e.g. Fitbit, Garmin Connect) ■ Sync data from wearables like heart rate, steps, sleep ○ Indoor navigation (e.g. airport or mall apps using BLE beacons 信标) ■ Guide users through complex indoor spaces ○ Smart home control (e.g. Philips Hue, smart locks) ■ Connect to IoT devices for lighting, security, automation ○ Contact tracing / proximity alerts 联系人追踪/近距离警报 (e.g. COVID-19 apps) ■ Detect nearby users for exposure notifications ○ Asset tracking / inventory management (e.g. warehouse apps) ■ Monitor tagged items via BLE beacons
Near-Field Communication (NFC)	● A **short-range wireless technology** ● **Allows two electronic devices to communicate with each other** ● Applications: mobile payments, data exchange, access control, product information ● App features that use it: ○ Mobile payments (e.g. Apple Pay, Google Pay, Samsung Pay) ■ Tap to pay at POS terminals ○ Access control (e.g. hotel key cards, office entry apps) ■ Unlock doors or authentication entry ○ Product information / smart posters ■ Tap to view details, promotions or URLs ○ Transit ticketing 公交票务 (e.g. Touch 'n Go, Octopus) ■ Tap to board buses or trains ○ Data exchange / pairing (e.g. Android Beam, smart device setup) ■ Share files or connect devices instantly
Nearlink	● A **short-range wireless communication technology developed by the NearLink Alliance**, led by Huawei ● **High speed, high reliability, high security, low latency, low energy** ● Applications: wearable, IoT, automotive, consumer products ● App features that use it: ○ Real-time wearable syncing (e.g. smartwatches, fitness bands) ■ Faster and more reliable than BLE ○ IoT device control (e.g. smart appliances, sensors) ■ Low-latency commands and feedback

	○ Automotive apps (e.g. car diagnostics, infotainment control) ■ Seamless interaction with vehicle systems ○ High-speed file transfer (e.g. Huawei Share) ■ Send large files quickly between devices ○ Gaming accessories (e.g. controllers, AR / VR gear) ■ Low-lag input for immersive experiences

Technology	**BLE**	**NFC**	**Nearlink**
Range	Up to 100 meters	Up to 4 cm	Up to 100 meters
Speed	Up to 2 Mbps	Up to 424 Kbps	Up to 900 Mbps
Power Consumption	Low	Very Low	Lowest (60% of BLE)
Latency	Moderate	Very Low	Ultra-low

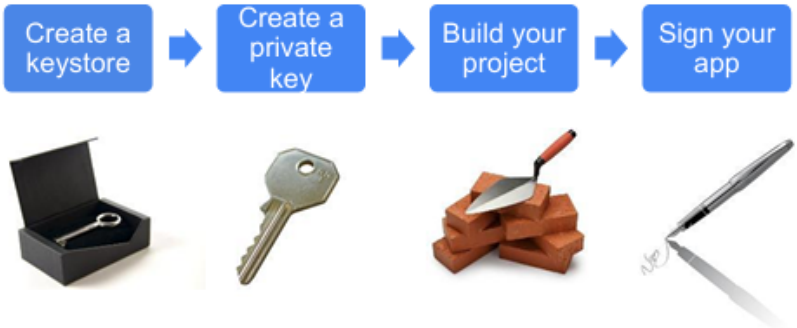
C7: Mobile Application Packaging and Publication

Checklist to Plan Your App Launch

Checklist	Detail
Developer Program Policies	● Mobile app stores have specific guidelines and policies that developers must adhere to ● Example of mobile app stores: ○ Google Play Store ○ Apple App Store ○ Huawei App Gallery ● Policies are designed to maintain a safe, secure and positive user experience ● **Restricted Content** ○ Have strict rules about the type of content allowed, including explicit content, malware and harmful apps ○ Explicit Content: pornography, sexually suggestive content and nudity ● **Intellectual Property** ○ App reviewers check for potential IP infringement 知识产权侵权 such as copyright infringement 版权侵权, trademark infringement 商标侵权 and patent infringement 专利侵权 ● **Privacy and Security** ○ Developers must have clear privacy policies that explain how user data is collected, used and shared ○ App stores may impose security requirements 提出安全要求 to protect user data from unauthorized access and breaches ● **Monetization and Ads** ○ App stores provide guidelines govern in-app purchases and subscriptions ○ Ad monetization and in-app purchases must adhere to guidelines, including fair pricing and transparent advertising ● **Store Listing and Promotion** ○ App stores control the listing and promotion (marketing materials) of apps through strict review processes, guidelines and algorithms ○ Developers must adhere to these rules to ensure their apps are visible, accessible and successful ○ e.g. setting of user's age groups, regions, etc
Developer Account	● A **publishing account** issued by **Platform Provider** to **developer** ● Enables developer to **post**, **display**, **offer for sale** and **distribute apps** through the Platform

	• Access developer tools, resources and support provided by the platform
	• Examples:
	  ○ Android: USD 25 (one time)
	  ○ Apple: USD 99 (per year)
	  ○ Huawei: FOC
	  ○ KaiOS: FOC
	  ○ Tizen: FOC

Localization	• Adaptation of an app to meet the needs of a particular **language**, **culture** or desired population's "look-and-feel" • A successfully localized app is one that appears to have been developed within the **local culture**
Device Compatibility	• Ensures that an app runs efficiently **across mobile devices** or **different configurations** • Ensures app run well on mobile devices of different **hardware**, **software** and **operating systems**
Quality Test: Alpha & Beta	• A process to ensure apps **meet specified regulations** and **standards** • It is a series of techniques that developers employ to **prevent issues from occurring** and **ensure they satisfy the customer** with their finished product

Store Listing	• Preparing for Release
	  ○ To prepare the app so that users can install and run the app on their Android-powered devices
	  ○ Release-ready .apk file is signed with your own certificate
	  ○ Minimum requirements:
	  ■ **Cryptographic keys (digitally signature that is owned by the application's developer)**
	  ■ **Application icon**
	  ■ **End-user license agreement**
	  ■ **Promotional and marketing materials**
	• Signing in Release Mode
	 
	• Signing Consideration
	  ○ Update

	○ Modularity ○ Code and data sharing ● Store Listing ○ **Distribution** ○ **App size < 150 MB + Expansion up to 2GB** ○ **Platforms** ○ **Fee or Free**

Distribution Channels

Distribution Channel	Detail
Marketplace	● **Centralized platform where users can discover, download and update apps or digital products** ● Examples: ○ Google Play Store (Android) ○ Apple App Store (iOS) ○ Huawei AppGallery ○ Microsoft Store ● Use Cases: ○ Distributing mobile apps to wide audience ○ Handling updates, reviews and ratings ○ Monetization via in-app purchases or subscriptions ● Benefits: ○ **Trusted by users** ○ **Built-in security and update mechanisms** ○ **Easy discoverability and global reach**
Email	● **Direct distribution of content, links, or app files via email to targeted users** ● Examples: ○ Sending beta APKs or TestFlight invites ○ Delivering promotional content, download links, or onboarding materials ○ Sharing personalized updates or exclusive access ● Use Cases: ○ Private beta testing ○ Targeted marketing campaigns ○ Re-engagement or retention strategies ● Benefits: ○ **Personalized and direct** ○ **No need for public testing** ○ **Useful for controlled or invite-only access**
Website or Server	● **Hosting the app or content on a web server, allowing users to download or access it directly** ● Examples: ○ Downloading APKs from a developer's site

	○ Accessing web apps or Progressive Web Apps (PWAs) ○ Hosting documentation, updates or support tools ● Use Cases: ○ Enterprise / internal app distribution ○ Bypassing app store restrictions (e.g. region locks) ○ Offering desktop or cross-platform versions ● Benefits: ○ **Full control over hosting and updates** ○ **No marketplace fees or approval delays** ○ **Ideal for custom or niche deployments**

Distribution Methods - Android

Distribution Method	**Detail**
Android Go	● OS designed for **entry-level smartphones** with **limited resources** ● An initiative that brings the benefits of Android to a wider range of users ● Optimized for devices with: ○ **Limited RAM**, typically 1GB or less ○ **Lower processing power** ○ **Less storage space** ● Minimum requirement: ○ Target Oreo (API 26) ○ App size less than 40 MB ○ RAM usage below 50 MB (apps) or 150 MB (games) ○ Start your app under 5 seconds ● Best for: Apps targeting entry-level smartphones with limited hardware ● Suitable circumstances: ○ Your app is designed for **emerging markets** 新兴市场 or **low-spec devices** ○ You need to support devices with **≤ 1GB RAM**, **low CPU** and **limited storage** ○ You are building **lightweight apps** (under 40MB) with **fast startup** and **low memory usage** ○ Ideal for: basic utilities, lightweight social apps, offline-first tools
App Bundles	● Efficient distribution technology for Android apps, especially on **devices with varying screen sizes and capabilities** ● Instead of building separate APKs for each device configuration (screen size, CPU architecture, etc), you create a single .aab (Android App Bundle) file ● Google Play **dynamically delivers only the necessary code and resources for a specific user's device during**

	installation  ● Benefits: ○ **Small app file** ■ Users download only the parts of your app they need, resulting in significantly smaller download sizes ○ **Faster installation** ■ Faster installation times due to smaller downloads ○ **Good performance** ■ Optimized for specific devices, leading to better performance and potentially reduced battery consumption ○ **Easy to manage** ■ Easier to manage app releases and updates ● Best for: Efficient, scalable distribution across **diverse Android devices** ● Suitable circumstances: ○ You want to **optimize app size** and **performance** for different screen sizes, architectures or locales ○ You are distributing via **Google Play** and want **automatic device-specific delivery** ○ You need **easy release management** and **smaller updates** ○ Ideal for: mainstream apps with broad device support - games, productivity tools, media apps
Google Play Instant (obsolete soon)	● **Allows users to try a portion of an app or game before downloading the full version** ● Users can access instant apps through links in search results, emails, ads or social media ● App has **limited functionality**, typically **focuses on core app functionality or a specific game level** ● Benefits: ○ **Try before you buy** ■ Users can experience the app before committing to a full download ○ **Reduced storage space** ■ No need to install the full app to try it out ○ **Faster access**

	■ Instant apps load quickly, allowing users to start using them immediately ● Limitations: ○ **Size limitations** ■ Instant apps have size restrictions to ensure fast loading times ○ **Functionality limitations** ■ Only a subset of the app's functionality may be available in the instant version ● Best for: **App previews** or **lightweight experiences** without full installation ● Suitable circumstances: ○ You want users to **try your app or game instantly** before downloading ○ You are promoting via **ads, search, social media or email** ○ You are offering a **core feature or demo level** with limited functionality ○ Ideal for: marketing campaigns, onboarding flows, game trials, lead generation
Support Chrome OS	● An operating system developed by Google that is designed to work primarily with web applications ● **Uses the Chrome web browser as its main user interface** ● **Supports Google Play Store and Android apps** ● Best for: Apps that run on **Chromebooks** or **web-first environments** ● Suitable circumstances: ○ Your app is compatible with **larger screens**, **keyboard / mouse input** or **web-based workflows** ○ You want to reach **education**, **enterprise** or **productivity users** on Chrome OS ○ You are building **cross-platform apps** that work on both Android and desktop-like environments ○ Ideal for: note-taking apps, IDEs, design tools, educational platforms

Monetize Your App

Costs Involved when Developing Mobile App

Cost	Detail
Development costs	● What it covers: ○ Hiring developers, designers and testers ○ Building core features, UI/UX and backend systems ● Why it matters:

	○ This is the largest upfront investment, especially for custom or complex apps ○ Costs vary based on app complexity, platform (iOS / Android) and team size
Tools and technologies	● What it covers: ○ SDKs, APIs, libraries and frameworks ○ Cloud services, analytics platforms and third-party integrations ● Why it matters: ○ Some tools are free but others (e.g. Firebase, Stripe, Mapbox) have usage-based pricing ○ Choosing the right stack affects scalability, performance and long-term cost
Marketing and promotion	● What it covers: ○ App store optimization (ASO), paid ads, influencer outreach 影响者推广 ○ Social media campaigns, email marketing, landing pages ● Why it matters: ○ Even the best app needs visibility - marketing drives download and user acquisition 即使是最好的应用程序也需要知名度--营销可推动下载和用户获取 ○ Budgeting for promotion is essential to avoid "launch and vanish" scenario 为避免出现 "推出后就消失 "的情况，编制推广预算至关重要
Ongoing costs - updates, server hosting, customer support, etc	● What it covers: ○ App updates, bug fixes, feature enhancements ○ Server hosting, database management, CDN ○ Customer support and user feedback channels ● Why it matters: ○ Apps are living products - maintaining performance and user satisfaction requires continuous investment ○ These costs scale with user base and app complexity
Legal and regulatory costs	● What it covers: ○ Privacy policies, terms of service, GDPR / PDPA compliance ○ Licensing, intellectual property protection ○ App store registration fees and business incorporation ● Why it matters: ○ Ensures your app is legally safe and compliant, especially if handling user data or payments ○ Neglecting 忽视 this can lead to fines, bans or

	reputational damage

Factors that Influence the Monetization Model

Factor	Detail
Who are the users?	● **Age, income level, tech savviness, geographic location** ● Why It Matters: ○ Younger users may prefer freemium models with optional upgrades ○ High-income users may tolerate premium pricing or subscriptions ○ Tech-savvy users expect value and transparency - they'll avoid intrusive ads or gimmicks 噱头 ○ Understanding your audience helps tailor pricing, ad formats, and payment flows
How valuable is your app?	● **Does it solve a real problem or offer unique value?** ● Why It Matters: ○ Apps that solve critical problems (e.g., health, finance, productivity) can justify subscriptions or upfront fees ○ Entertainment or novelty 新奇 apps may rely more on ads or in-app purchases ○ The more indispensable your app, the more users are willing to pay 应用程序越不可或缺，用户就越愿意付费
What are your competitors doing?	● **Pricing models, feature sets, user reviews** ● Why It Matters: ○ If competitors offer similar features for free, charging upfront may hurt adoption 如果竞争对手免费提供类似功能，预收费用可能会影响采用率 ○ Studying competitors helps you position your app - either as a better free alternative or a premium upgrade ○ Benchmarking helps avoid pricing missteps and identify gaps in the market 制定基准有助于避免定价失误并找出市场差距
How much does it cost to make and maintain the app?	● **Development, hosting, updates, support** ● Why It Matters: ○ High development and maintenance costs may require recurring revenue (e.g. subscriptions) ○ Low-cost apps can afford freemium or ad-supported models

	○ Your monetization must cover costs and leave room for growth
How will it impact user experience?	● **Intrusiveness 侵扰性 of ads, friction in payment, access to features** ● Why It Matters: ○ Too many ads or paywalls can drive users away ○ Seamless monetization (e.g. optional upgrades, non-intrusive ads) improves retention ○ Balancing revenue with usability is key to long-term success

Monetization Model

Monetization Model	**Detail**
Premium Apps	● Paid ● In-app billing ● Extensive features ● Narrow niche in the market
	Extra info from Copilot: ● Model: One-time payment to download ● Mechanism: In-app billing for purchase ● Key Traits: ○ Offers **full access upfront** — no ads or locked features ○ Targets **niche markets 瞄准利基市场** with high-value features (e.g. pro tools, educational apps) ○ Requires strong **value proposition** to justify the price 需要强有力的价值主张来证明价格合理 ● Examples: ○ Productivity tools (e.g., Tasker, Nova Launcher Prime) ○ Specialized calculators, offline dictionaries, or niche utilities
Freemium Apps	● Free ● Uses in-app billing ● Digital goods ○ Durable ○ Consumable
	Extra info from Copilot: ● Model: Free to download, with optional purchases ● Mechanism: In-app billing for digital goods ● Key Traits: ○ Core features are free; **premium content or tools** are

	paid ○ Digital goods can be: ■ **Durable**: permanent (e.g. unlocking a new feature or skin) ■ **Consumable**: temporary (e.g. coins, lives, boosts) ● Examples: ○ Games (e.g., Clash of Clans, Candy Crush) ○ Editing apps (e.g., VSCO, Canva)
Subscription	● Free trial ● In-app billing ● Subscription fee
	Extra info from Copilot: ● Model: Recurring payment (monthly/yearly) ● Mechanism: In-app billing with optional free trial ● Key Traits: ○ Offers **ongoing value** (e.g., content updates, cloud sync, premium support) ○ Ideal for apps with **dynamic content** or **service-based features** ● Examples: ○ Streaming apps (e.g., Netflix, Spotify) ○ Productivity suites (e.g., Notion, Evernote Premium)
Ads	● Free ● AdMob + Google Mobile Ads SDK ● Show income-generating ads
	Extra info from Copilot: ● Model: Free app supported by advertisements ● Mechanism: AdMob or similar SDKs ● Key Traits: ○ Generates revenue from **impressions, clicks, or video views** ○ Can include **banner ads, interstitials** 插播广告**, rewarded videos** ○ Must balance ad placement with user experience ● Examples: ○ News apps, quiz games, utility apps ○ Free versions of popular tools
E-Commerce	● Free ● B2C ● Technology, logistics and payment solutions ● Sales commissions + Setup fees
	Extra info from Copilot: ● Model: Selling physical or digital products

	● Mechanism: B2C platform with logistics and payment integration ● Key Traits: ○ App acts as a **storefront** 店面 or **marketplace** ○ Revenue from **sales commissions, setup fees, or product margins** 来自销售佣金、安装费或产品利润的收入 ● Examples: ○ Shopee, Lazada, Amazon ○ Brand-specific apps (e.g., Nike, Sephora)
Rewarded Products	● Free ● Share content on social media, perform simple tasks (e.g. scan QR code, answer survey and etc) ● Get free stuff, earn rewards
	Extra info from Copilot: ● Model: Users earn rewards by completing tasks ● Mechanism: Social sharing, surveys, QR scans, etc. ● Key Traits: ○ Drives **engagement and virality** 推动参与和病毒式传播 ○ Rewards can be **discounts, points, or free items** ○ Often used in **campaigns or loyalty programs** ● Examples: ○ BuzzBreak, GrabRewards, referral programs ○ Survey apps like Google Opinion Rewards
Service	● Free ● An extension of physical / online services
	Extra info from Copilot: ● Model: App extends a physical or online service ● Mechanism: **Free access**, **monetized through service delivery** ● Example: myTNB app, MAE by Maybank2u app ○ myTNB: bill payments, outage reports ○ MAE by Maybank2u: mobile banking, transfers, investments ● Use Case: Utilities, finance, healthcare, logistic
Data Collection	● Free ● Partnership / Affiliate 联营公司 ● Provide customer service, promotion, run a giveaway 举办赠品活动 and etc
	Extra info from Copilot: ● Model: Free app that monetizes user data via partnerships or ads ● Mechanism: Affiliate marketing 联属营销, behavioral analytics ● Example: Facebook, Chrome, X (Twitter) ○ Collect user behavior, interests, and engagement

	patterns ○ Monetize via targeted ads, sponsored content, or data partnerships ● Use Case: Social media, browsers, platforms with massive user bases