Mobile Application Development

Week 1 - Lecture Number 1

Introduction to Mobile App Development in 2025

Agenda

- The Mobile App Ecosystem (Environment) in 2025
- Categories of Mobile Apps
- What is Native App Development?
- What is Hybrid App Development?
- What is Cross-Platform Development?
- Cross-Platform in 2025
- Why Flutter Dominates
- Flutter in Numbers (2025)
- Flutter vs React Native
- Flutter vs Kotlin Multiplatform
- Hybrid vs. Native vs. Cross-Platform

The Mobile App Ecosystem (Environment) in 2025

- Over 6.8 billion smartphone users worldwide
- More than 255 billion app downloads annually
- Mobile apps = An economy exceeding $600 billion

Categories of Mobile Apps

- Social Media (TikTok, Instagram)
- FinTech (Revolut, PayPal)
- Health & Fitness (MyFitnessPal, Fitbit)
- Gaming (PUBG, Genshin Impact)
- Productivity (Notion, Slack)

FinTech mobile app

In simple terms, **FinTech in a mobile app refers to any financial service or technology you can access, manage, and control directly from your smartphone.**

It's the merger of **Fin**ancial services with **Tech**nology, delivered through the convenience of a mobile application. It removes the need for physical bank branches, paper forms, and face-to-face meetings for many financial tasks.

Think of it as your entire financial life—banking, investing, borrowing, and planning—living in your pocket.

Key Categories of FinTech Mobile Apps

FinTech apps aren't just one thing; they cover a wide range of services. Here are the most common types:

1. Digital Banking & Neobanks

These are banks that exist entirely on your phone, with no physical branches.

- **Examples:** Chime, Revolut, N26, Varo.
- **They** offer checking/savings accounts, debit cards, direct deposit, bill pay, and money transfers, all through an app. They often have lower fees and more user-friendly interfaces than traditional banks.

2. Digital Payments & Wallets

Apps that facilitate the transfer of money between people or to merchants.

- **Examples:** PayPal, Venmo, Cash App, Zelle, Apple Pay, Google Pay.
- **They** allow you to send money to a friend, split a bill, pay for goods online or in store using your phone, and sometimes even buy cryptocurrency.

3. Investment & Trading Apps

These apps democratize investing, making it accessible to everyone, not just the wealthy.

- **Examples:** Robinhood, Acorns, Stash, E*TRADE, Webull.

- **They** let you buy and sell stocks, ETFs, options, and cryptocurrencies, often with low or no commission fees. Some use "micro-investing" to automatically invest your spare change.

4. Budgeting & Personal Finance Management (PFM)

Apps that help you understand and manage your money better.

- **Examples:** Mint, YNAB (You Need A Budget), PocketGuard, Copilot.
- **They** connect to your bank and credit card accounts, categorize your spending, create budgets, track your bills, and give you a full picture of your net worth.

5. Lending & Borrowing Apps (Financing)

Platforms that streamline the process of getting a loan.

- **Examples:** SoFi, Affirm, Afterpay, Klarna.
- **They** offer personal loans, student loan refinancing, or "Buy Now, Pay Later" (BNPL) options directly at the point of sale, with instant approval decisions.

6. Cryptocurrency & Blockchain Wallets

Apps dedicated to managing digital assets and interacting with blockchain technology.

- **Examples:** Coinbase, Binance, Crypto.com, MetaMask.

- **They** allow you to buy, sell, store, and trade cryptocurrencies like Bitcoin and Ethereum. Some also let you stake coins to earn interest or connect to decentralized applications (dApps).

7. Insurance Technology (InsurTech)

Apps that use technology to simplify buying and managing insurance.

- **Examples:** Lemonade (for renters/home insurance), Oscar (for health insurance).
- **What they do:** They offer instant quotes, automated claims processing (sometimes using AI), and policy management entirely through the app.

Core Technologies Powering These Apps

- **APIs (Application Programming Interfaces):** The secret glue. They allow the app to securely connect to your bank, credit card company, or credit score provider to pull in your data.
- **Artificial Intelligence (AI) & Machine Learning:** Used for fraud detection (noticing unusual spending patterns), providing chatbot customer support, and offering personalized financial advice (robo-advisors).
- **Blockchain:** Provides the foundation for cryptocurrencies and enables secure, transparent transactions.

- **Biometric Authentication:** Uses fingerprint scanners (Touch ID) and facial recognition (Face ID) for secure and effortless logins.

In summary, a **FinTech mobile app is a software-based platform on your phone that provides a financial service in a faster, cheaper, and more user-friendly way than traditional methods.** It has fundamentally changed how we interact with and think about money.

productivity mobile app

- In the context of a mobile app, **productivity** refers to the app's ability to help users **organize their work, manage their time, complete tasks efficiently, and achieve their goals** with less effort and in less time.
- It's not about doing *more* work; it's about doing *meaningful* work more effectively, often by automating, streamlining, or organizing the processes around it.
- A productivity app acts as a digital tool that enhances your personal or professional output.

Key Pillars of Mobile Productivity Apps

Productivity apps generally focus on one or more of the following core functions:

1. Task and Project Management

These apps help you break down goals into actionable steps and track their progress.

- **Examples:** Todoist, Microsoft To Do, Trello, Asana, Monday.com
- **What they do:** Create to-do lists, set deadlines, assign tasks to others, and visualize workflows (e.g., with boards, lists, and calendars).

2. Note-Taking and Documentation

Apps that capture ideas, information, and knowledge in a structured and searchable way.

- **Examples:** Evernote, Notion, OneNote, Google Keep, Apple Notes
- **What they do:** Take text, audio, and image notes; organize them in notebooks or databases; sync across devices; and make information instantly retrievable.

3. Time Management and Focus

Apps designed to help you use your time intentionally and minimize distractions.

- **Examples:** Forest, Focus Keeper (Pomodoro technique), Clock (Google/Apple with timers & alarms), RescueTime

- **What they do:** Use techniques like the Pomodoro timer (25 min focus, 5 min break), block distracting websites, and track how you spend your time on your phone.

4. Communication and Collaboration

Apps that facilitate efficient teamwork and information sharing.

- **Examples:** Slack, Microsoft Teams, Zoom, Google Workspace (Docs, Sheets, Drive)
- **What they do:** Provide chat channels, video conferencing, and real-time collaborative document editing, all accessible from a phone.

5. Document Scanning and Storage

Apps that turn your phone's camera into a scanner and help you manage paperwork.

- **Examples:** Adobe Scan, CamScanner, Genius Scan
- **What they do:** Scan receipts, contracts, and whiteboards into high-quality PDFs, often with OCR (Optical Character Recognition) to make the text searchable and editable.

6. Calendar and Scheduling

Apps that go beyond the basic calendar to manage your time and coordinate with others.

- **Examples:** Google Calendar, Fantastical, Calendly

- **What they do:** Schedule appointments, set reminders, view multiple calendars at once, and allow others to book time on your calendar automatically.

In summary, a **productivity mobile app is a digital tool designed to optimize how you manage your time, tasks, and information, leveraging the unique advantages of a mobile device to help you work smarter, not harder.**

What is Native App Development?

- **Native app development** is the process of building software applications that are specifically designed to run on a single platform or operating system (OS), such as iOS or Android.
- They are called "native" because they are written in programming languages and use the development tools **native** to that specific platform.
- The core principle is that you build two separate, dedicated codebases for each platform you want to target.

Key Characteristics of Native Apps

1. **Platform-Specific Programming Languages:**
   - **iOS:** Developed primarily in **Swift** or **Objective-C**, using Apple's **Xcode** development environment.
   - **Android:** Developed primarily in **Kotlin** or **Java**, using **Android Studio**.
2. **Direct Access to Device Features:** Native apps have full, unimpeded access to all the device's hardware and software features via the platform's native APIs (Application Programming Interfaces). This includes:
   - Camera
   - GPS (Location Services)

- Microphone
- Accelerometer & Gyroscope
- Contacts
- File System
- Push Notifications

3. **High Performance:** The code is compiled directly into machine code for the device's CPU. This allows for:
   - Faster execution
   - Smoother animations and transitions
   - Highly responsive user input (touch, swipe, etc.)

- Efficient handling of complex computations and graphics (e.g., intense gaming apps).

4. **Superior User Experience (UX) and User Interface (UI):** Native apps adhere strictly to the official **UI guidelines** of their platform (Apple's **Human Interface Guidelines** for iOS and **Material Design** for Android). This means:
   - They look and feel like a natural part of the operating system.
   - Users are already familiar with the standard interactions (e.g., how a back button works on Android vs. a swipe gesture on iOS).
   - This leads to intuitive and comfortable user experiences.

5. **Distribution:** They are distributed through official app stores like the **Apple App Store** and **Google Play Store**, which provides security, trust, and a centralized discovery platform.

Native Apps: Pros

- Best possible performance
- Complete integration with OS features
- Excellent user experience (UX)
- Strong security

Native Apps: Cons

- Higher cost of development (separate teams)
- Maintenance challenges
- Slower cross-platform scalability

What is Hybrid App Development?

Hybrid app development is a practical and efficient strategy for building good-quality mobile apps that need to reach a wide audience across platforms quickly and affordably, trading off some performance and native feel for immense (huge) gains in development speed and cost.

- Built with HTML, CSS, JavaScript inside WebView
- Popular frameworks: Ionic, Cordova
- Essentially a web app packaged as a mobile app

Hybrid Apps: Pros & Cons

- ✅ Quick and low-cost development
- ✅ Single codebase for all platforms
- ❌ Lower performance than Native
- ❌ Limited user experience

What is Cross-Platform Development?

- Shared codebase across multiple platforms
- Examples: Flutter, React Native, Kotlin Multiplatform
- Balance between high performance and code reusability

What a "shared codebase across multiple platforms" means, specifically in the context of Flutter.

The Core Idea: "Write Once, Run Anywhere" (but better)

In traditional development, you need separate teams and codebases:

- An **Android team** writing in Kotlin/Java.
- **An iOS team** writing in Swift/Objective-C.
- **A web team** writing in JavaScript/TypeScript.
- **A desktop team** writing in C#/C++.

This is expensive, time-consuming, and hard to keep consistent.

Flutter's approach is different: **You write one application in one language (Dart) using one codebase.** The Flutter framework and engine then take this single codebase and **compile it, or "transpile" it, into native code for each target platform.**

What is traspile

The term "transpile" is a portmanteau (combination) of (transform + compile). It's a specific type of compilation, and understanding the difference is key to understanding how many modern frameworks work.

In other words,

Transpilation is the process of converting source code from one high-level programming language to another high-level programming language.

Why is Transpilation Useful?

Cross-Platform in 2025

- Now the global standard
- Reduces development cost by 40–60%
- Backed (Supported, Sponsored) by Google, Meta, JetBrains

Why Flutter Dominates

- Dart language → fast and easy to learn
- Supports Android, iOS, Web, Desktop
- Hot Reload for instant updates
- Near-native performance
- Rich customizable Widgets

Hot Reload for instant updates

Hot Reload is a development feature that lets you see the results of your code changes instantly, without losing your place in the app, making building and fixing things incredibly fast.

The Simple Analogy (Similarity, Likeness): A Word Document for Your App

Think of building your Flutter app like typing a document in Microsoft Word or Google Docs.

- **Without Hot Reload (Traditional Development):** You type a sentence, but to see how it looks, you have to **Save the file, close Word, and then re-open it**. This is slow and frustrating.
- **With Hot Reload (Flutter Development):** You type a sentence, and it **instantly appears on the screen** as you type. You can change the font, color, or size and see the result immediately without any interruption.

Hot Reload is the "live-typing" feature for your app's code.

Key Things to Remember:

- **It's Stateful:** This is the magic. Hot Reload tries to **preserve the current state** of your app. For example, if you're logged in and on a specific screen, you'll stay logged in on that same screen after the reload. Your text fields, scroll positions, and other data remain intact.
- **It's for UI and Logic:** You can change not just how things look (colors, text) but also app behavior (like the logic of a button press).
- **It Has Limits (Sometimes you need a Full Restart):** If your change is fundamental (like modifying the app's initializer, adding/removing a package, or changing core app state), you

might need a **Hot Restart**. A Hot Restart takes a few seconds longer as it fully resets the app to its initial state.

Near-native performance

Near-native performance means that a Flutter app isn't written in the device's "native language," but it runs so efficiently and smoothly that for the end-user, it feels just as fast, responsive, and smooth as an app that is.

It's the "you can't tell the difference" Level of performance.

The Simple Analogy:

A Translated Book vs. The Original

Imagine you have a famous novel written in English (the "native" language).

- **Native App:** This is like reading the **original English book**. The reader (your device's processor) understands it directly, so it's very fast and efficient.
- **Traditional Cross-Platform App (e.g., with a Web View):** This is like reading an **English book that has been translated twice**. First to Japanese, and then from Japanese back to English. The final story is the same, but the process is slower, clunkier, and

some meaning might be lost. This adds overhead and makes it feel slower.

- **Flutter App (Near-Native):** This is like reading a **perfect, direct translation** done by an expert linguist. It wasn't written in English originally, but the translation is so good and so direct that **you can't tell the difference** from the original. It feels just as fast and smooth.

What Does Flutter Do to Achieve This?

Flutter gets this "near-native" speed by skipping a slow middle step that other frameworks use.

1. **It Doesn't Use Native UI Components:** Most cross-platform frameworks (like React Native or Xamarin) use the device's built-in buttons, sliders, etc. They have to ask the device's OS to draw these, which adds communication overhead.
2. **It Paints Everything Itself:** Flutter brings its own set of widgets (buttons, menus, etc.) and **draws them directly onto the screen**. It's like the app has its own art supplies and doesn't need to ask the device for help.

This means:

- **No Communication Delay:** It doesn't have to wait for the native system to respond.
- **Consistent and Fast:** The UI is consistently fast on both old and new iOS & Android devices because it's not relying on the version of the OS.
- **60/120 Frames Per Second:** The goal is to be so smooth that it can run at the device's maximum refresh rate (60fps or 120fps), just like a native game or high-performance app.

Rich customizable Widgets

"Rich customizable Widgets" means Flutter gives you a massive, high-quality toolkit of pre-made UI building blocks. More importantly, every single block is designed to be easily modified and combined with others, allowing you to build any user interface you can imagine, without having to create everything from scratch.

It's the power and flexibility to go from a standard-looking app to a beautifully unique, branded design with complete creative freedom.

No Limits on Customization: Because Flutter draws its own widgets on the screen (instead of using the native Android/iOS ones), you are not limited by what the operating system provides. If you can imagine a UI, you can build it without fighting against the platform's default style.

Flutter in Numbers (2025)

- 2M+ active developers
- Used in Google apps (Ads, Pay)
- Adopted by BMW, Alibaba, eBay

Flutter vs React Native

- **Flutter:**
- Dart language
- Independent rendering engine (Skia)
- Higher performance
- **React Native:**
- JavaScript
- Relies on native bridges
- Larger community but performance issues

Independent rendering engine (Skia)

- An independent rendering engine (Skia) means Flutter brings its own powerful, portable "artist" to every device. This artist paints the app's interface directly onto the screen, ensuring it looks and performs exactly the same way, everywhere, and giving you, the developer, complete creative control.
- Imagine Flutter is a painter who needs to draw your app's screens, buttons, and animations.
- Skia is the painter's toolbox and brushes, It's a powerful, open-source 2D graphics library that does the actual low-level work of drawing pixels onto the screen.

The Simple Analogy: Your Own Portable Film Crew

Imagine you're a director making a movie that needs to look exactly the same in every cinema.

- **Native Apps & Some Cross-Platform Frameworks:** This is like using a **different film crew in each country**. You have to explain your vision to the American crew, the Japanese crew, and the French crew. While they're all professionals, they might interpret your instructions slightly differently, leading to variations in the final movie.
- **Flutter with Skia:** This is like **bringing your own film crew with you everywhere**. You have the same director, same camera operators, and same editors. No matter where in the world you

show the movie, it will look **exactly** as you directed it, with perfect consistency.

What Does Skia Actually Do?

Skia is a high-performance, open-source **2D graphics library** that does the drawing.

1. **It's the "Artist":** When your Flutter code describes a blue, rounded button with shadow, Skia is the one that takes those instructions and actually **paints the pixels** onto the screen to create that exact button.
2. **It's "Independent":** Flutter bundles Skia **inside your app**. This means your app doesn't have to ask the phone's operating system (iOS or Android) to draw the UI elements for it. It has its own personal artist ready to go.

Relies on native bridges

"Relies on native bridges" means React Native acts as a manager that gives instructions in JavaScript. A communication channel (the bridge) relays those instructions to the phone's native "workers" (iOS/Android) to build and control the user interface. This provides deep integration with the platform but can introduce a performance cost for heavy communication.

The Simple Analogy: A Translator Between Two Colleagues

Imagine you have a JavaScript developer and a native (iOS/Android) developer who need to work together to build an app. The problem is, they speak completely different languages.

- **JavaScript Developer:** Works in React Native, writing JavaScript/TypeScript code.
- **Native Developer:** Represents the phone's OS, which understands Objective-C/Swift (iOS) or Java/Kotlin (Android).

The **Native Bridge** is the **translator** that stands between them, passing messages back and forth.

What Actually Happens?

When your React Native app needs to do something that only the native side can do (like show a navigation bar, play a vibration, or use the camera), here's the process:

1. **Your JavaScript code** says, "Hey, I need to vibrate the phone for 500ms."
2. This request is **serialized into a JSON message** (turned into a standard text format).
3. The message is **sent across the "Bridge"** to the native side of the app.

4. The **native side (iOS/Android) receives the message**, understands it, and executes the actual native code to trigger the vibration.

5. If there's a response (like "the vibration is done" or "here is a photo from the camera"), the process reverses: the native side sends a message back across the Bridge to the JavaScript side.

This two-way communication channel is the "Native Bridge."

Flutter vs Kotlin Multiplatform

- Flutter: Cross-platform for both UI + Logic
- Kotlin Multiplatform (KMP): Shared logic only, native UI
- KMP suits large enterprises with separate Android/iOS teams

Kotlin Multiplatform (KMP): Shared logic only, native UI

- "Shared logic only": You write business logic (like calculations, data fetching, and game rules) once in Kotlin, and it runs on both iOS and Android.
- "Native UI": You separately build the user interface using the native tools for each platform SwiftUI for iOS and Jetpack Compose for Android. This gives you apps that look and feel 100% native on each device.

In one sentence:

KMP lets you share the brain of your app (the logic) while letting you build a custom native body (the UI) for each platform.

Hybrid vs. Native vs. Cross-Platform

Feature	Native	Hybrid	Cross-Platform (e.g., React Native, Flutter)
Codebase	Separate for each platform	**Single** codebase (Web tech)	**Single** codebase (JS, Dart, etc.)
Technology	Swift, Kotlin	**HTML, CSS, JavaScript**	React Native (JSX), Flutter (Dart)
Rendering Engine	Native UI Components	**WebView** (Browser Engine)	**Native UI Components** (Compiled)
Performance	**Best**	**Good for most apps**	**Very Good** (Near-Native)
Access to Native Features	Full, direct access	Via **Plugins/Bridge** (can be slower)	Via **Bridge** (often more efficient)

Feature	Native	Hybrid	Cross-Platform (e.g., React Native, Flutter)
Development Speed & Cost	Slowest, Highest Cost	**Faster, Lower Cost**	Fast, Moderate Cost
User Experience (UX)	Perfectly native	**Web-like**, can feel generic	Very close to native

Case Study 1: Alibaba (Flutter)

- E-commerce with millions of users
- Unified user experience across devices
- Faster release cycles

Case Study 2: BMW (Flutter)

- MyBMW app
- Available in 30+ languages, 45+ markets
- Faster multi-platform delivery

Case Study 3: Nubank (Flutter)

- Largest digital bank in Latin America
- Secure and flexible with Flutter
- 30% faster time-to-market

Case Study 4: Google Ads (Flutter)

- Official Google app
- Shows Google’s confidence in Flutter

Real-world Use Cases

- FinTech: Revolut, Nubank
- eCommerce: eBay, Alibaba
- Automotive: BMW, Toyota
- Social & Media: Philips Hue, Reflectly

Developer Experience

- Hot Reload = higher productivity
- Massive open-source libraries
- Strong global community and Google support

1. Hot Reload = Higher Productivity

Hot Reload is arguably (maybe) Flutter's killer feature for developer productivity, Explain (final question)

Hot Reload allows you to inject updated source code into a running Dart Virtual Machine (VM). This means your app updates *instantly* (typically in under a second) without requiring a full restart, losing its current state.

Why it creates higher productivity:

- **Instant Feedback Loop**: You see the result of your code changes *immediately*. This is a game-changer compared to native Android/iOS development where

a change can require a 30-second to 2-minute rebuild and redeploy cycle.

- **Experiment & Iterate Rapidly:** Tweaking (Change) UI elements like padding, colors, or animations becomes effortless. You can try ten different designs in the time it would take to do one the old way.
- **Debug UI Instantly:** You can fix a UI bug, hit save, and see if the fix worked right away.
- **Preserves State:** This is the magic. If you are logged into your app and on a specific screen, doing a Hot Reload after a code change will keep you logged in

and on that same screen. This allows you to quickly build and test flows that are deep within your app without having to navigate manually every time.

2. Massive Open-Source Libraries

- This refers to Flutter's incredible ecosystem of pre-built packages available on **pub.dev**, the official package repository for Dart and Flutter.
- **Pub.dev** is a massive marketplace of free, open-source code packages that solve common problems. Instead of building everything from scratch, you can easily plug these libraries into your app.

Why it boosts (increases) productivity and capability:

- **Don't Reinvent the Wheel:** Need to add Google Maps? Use the google_maps_flutter package. Need to handle HTTP API calls? Use http or dio. Need state management? Use provider, flutter_bloc, or riverpod. This saves an enormous amount of development time.
- **Access Native Features Easily:** Many packages provide a simple Flutter API to access complex native device features (camera, GPS, Bluetooth, sensors) through Platform Channels. The package author has

already done the hard work of writing the native Android and iOS code for you.

- **High Quality & Vetted:** Packages on **pub.dev** are scored based on health, maintenance, and popularity (pub points), making it easy to identify reliable and well-supported options.
- **UI Toolkits & Components:** There are entire packages dedicated to providing ready-made UI components like charts, calendars, video players, and advanced navigation drawers, drastically speeding up UI development.

3. Strong Global Community and Google Support

- This is the foundation that makes the first two points sustainable and successful.
- Flutter isn't a niche (place) framework from a small company. It's a major project backed by Google and adopted by a massive, passionate community of developers worldwide.

Challenges of Flutter

- Larger app size compared to Native
- Delayed support for newest APIs
- Smaller talent pool vs Java/Kotlin/Swift

Future of Mobile Apps (2025–2030)

- Cross-platform = new standard
- AI-powered apps will dominate
- Expansion into VR/AR
- Integration with WebAssembly

Summary

- Native = best performance, higher cost
- Hybrid = fast, limited UX
- Cross-platform = balance of cost & performance
- Flutter = leading framework (Android, iOS, Web, Desktop)
- Case studies: BMW, Alibaba, Nubank, Google Ads

Thank You!