

Single Activity Architecture

vs.

Multiple Activities Architecture

A Comprehensive Guide for Modern App
Development

SWIPE



Esmaeeil Moradi



Single Activity Architecture in Android:

A Comprehensive Guide for Modern App Development

Choosing between Single Activity and Multiple Activities architectures is a pivotal decision in modern Android development. This document synthesizes insights from recent talks (like Android Dev Summit 2018), articles, and real-world developer experiences to offer a fresh, practical perspective on these topics:

KEEP SWIPING

Esmaeail Moradi



1. When Should You Use Single Activity Architecture?

03/08

- **Complex Navigation Flows:** Single Activity, combined with the Navigation Component and Fragments, simplifies deep linking, shared element transitions, and navigation logic—especially in apps with multi-step or nested flows.
- **Shared State Across Screens:** Managing state at the Activity level avoids redundant data loading and enables seamless communication between screens.
- **Real-Time or Highly Interactive Apps:** Apps like chat clients or live dashboards benefit from centralized lifecycle and state management.
- **Exceptions (When Multiple Activities Still Make Sense):**
- **System-Level Features:** Features like Picture-in-Picture (PiP) or full-screen video playback may require separate Activities for proper integration.
- **Isolated Modules:** Highly decoupled modules (e.g., payment flows) can benefit from their own Activity for better encapsulation and security.

KEEP SWIPING

Esmaeail Moradi



2. Why Single Activity?

Key Advantages

- **Optimized Memory Usage:** Reduces the overhead of launching multiple Activities (saving up to 40% in memory usage in some cases).
- **Simplified Navigation:** Navigation Component allows for intuitive back stack management and deep linking without complex Intent chains.
- **Enhanced Testability:** Centralized navigation and state management make it easier to mock and test navigation flows.
- **Seamless Jetpack Compose Integration:** Single Activity architecture naturally fits with modern UI paradigms like Jetpack Compose and unidirectional data flow patterns.

KEEP SWIPING



3. Challenges and Practical Solutions

- **Complex Lifecycle Management:**
- Use ViewModel and SavedStateHandle to persist UI state across configuration changes and process death.
- **Debugging Difficulties:** Leverage Android Studio's Layout Inspector and Navigation Editor for visualizing UI and navigation graphs.
- **Performance Optimization:** Avoid unnecessary Fragment transactions and use setMaxLifecycle() to optimize resource usage.

KEEP SWIPING



4. Technical Comparison: Single Activity vs. Multiple Activities

06/08

Criteria	Single Activity	Multiple Activities
Memory Usage	~12-15MB (average)	~20-25MB (with 3 Activities)
Startup Time	400-500ms	600-700ms
Deep Link Handling	Simple with Navigation Graph	Requires complex Intent Filters
Centralized Updates	Easy to implement	Must be replicated in each Activity

(DATA BASED ON PRACTICAL BENCHMARKS AND DEVELOPER REPORTS)

KEEP SWIPING

Conclusion:

When to Use Each Architecture?

- **Single Activity:**

- Apps with unified user flows (e.g., social networks, modern dashboards)
- Projects using Jetpack Compose or advanced navigation
- Smaller teams seeking rapid iteration and maintainability

- **Multiple Activities:**

- Modular apps with independent sections (e.g., marketplaces)
- Apps needing system-level features or legacy codebases
- Projects requiring strong module isolation

KEEP SWIPING



“There is no one-size-fits-all architecture.

The best choice is the one that aligns with your project’s unique needs and constraints.”

— Android Developer Docs

