IJIRSET
International Journal of Innovative Research in
SCIENCE | ENGINEERING | TECHNOLOGY

e-ISSN: 2319-8753 | p-ISSN: 2347-6710


# INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

## IN SCIENCE | ENGINEERING | TECHNOLOGY

Volume 13, Issue 5, May 2024

ISSN INTERNATIONAL STANDARD SERIAL NUMBER INDIA

Impact Factor: 8.423

9940 572 462 | 6381 907 438 | ijirset@gmail.com | www.ijirset.com

International Journal of Innovative Research in Science, Engineering and Technology (IJIRSET)

|e-ISSN: 2319-8753, p-ISSN: 2347-6710| www.ijirset.com | Impact Factor: 8.423| A Monthly Peer Reviewed & Refereed Journal |

|| Volume 13, Issue 5, May 2024 ||

| DOI:10.15680/IJIRSET.2024.1305372 |

# Review of Android Activity Development: Best Practices, Challenges, and Future Directions

**Vasim J. Tamboli, Chanchal M. Patil**

Assistant professor, Department of Electronics and Computer Science Engineering, Padmabhooshan Vasantraodada Patil Institute of Technology, Sangli, Maharashtra, India

Assistant professor, Department of Electronics & Telecommunication Engineering, Padmabhooshan Vasantraodada Patil Institute of Technology, Sangli, Maharashtra, India

**ABSTRACT:** Android Activity development is crucial for building interactive and engaging mobile applications on the Android platform. Activities serve as the building blocks of an Android application, representing single screens with a user interface. Activities are responsible for presenting the user interface to the users. Through activities, developers can design and implement layouts using XML and Java/Kotlin, enabling the creation of visually appealing and intuitive interfaces. Activities facilitate user interaction by handling user inputs such as touch events, gestures, and key presses. Developers can implement event listeners and callbacks within activities to respond to user actions effectively.

Understanding the activity lifecycle is essential for managing the state of an Android application. Activities go through various lifecycle stages (such as onCreate, onStart, onResume, onPause, onStop, and onDestroy), and developers can override lifecycle methods to perform tasks like initializing UI components, saving and restoring instance state, and releasing resources. With the increasing prevalence of multitasking and multi-window support on Android devices, designing activities that can adapt to different screen sizes and configurations is essential. Developers can utilize features such as split-screen mode and picture-in-picture mode to enhance the usability of their applications. Android Activity development is essential for creating feature-rich, interactive, and user-friendly applications on the Android platform. By mastering activity development, developers can build applications that provide a seamless user experience and leverage the full capabilities of the Android ecosystem.

**KEYWORD:** Android, Activity, Activity lifecycle, activity development.

## I. INTRODUCTION

When considering the scope of a review for Android Activity development, it's essential to focus on various aspects to ensure thoroughness and effectiveness. Review the functionality implemented within the activity, ensuring that it aligns with the requirements specified in the project documentation or user stories. Verify that all intended features work as expected and that user interactions produce the desired outcomes. Evaluate the UI design of the activity, considering factors such as layout, navigation flow, visual aesthetics, and adherence to Android design guidelines (Material Design). Check for consistency in UI elements, appropriate use of colors and typography, and responsiveness across different screen sizes and orientations. Assess how users interact with the activity, including touch gestures, button clicks, text input, and other forms of input. Verify that event listeners are properly implemented and that user feedback is provided where necessary (e.g., progress indicators, error messages).

Check for robust error handling mechanisms within the activity to handle exceptions, edge cases, and invalid user inputs gracefully. Ensure that error messages are informative and user-friendly, guiding users on how to resolve issues effectively. Evaluate the performance of the activity, paying attention to factors such as responsiveness, loading times, memory usage, and battery consumption. Identify any performance bottlenecks or inefficiencies that could impact the user experience negatively and suggest optimizations where necessary. Review the activity for potential security vulnerabilities, such as input validation flaws, insecure data storage, or insufficient protection of sensitive information.
Ensure that best practices for Android security, such as using HTTPS for network communication and implementing proper permissions and encryption, are followed. Evaluate how the activity integrates with other components of the application, such as services, fragments, and data sources. Verify that appropriate unit tests, integration tests, and UI tests are written and executed to validate the behavior of the activity under different scenarios. By reviewing these

|e-ISSN: 2319-8753, p-ISSN: 2347-6710| www.ijirset.com | Impact Factor: 8.423| A Monthly Peer Reviewed & Refereed Journal |

|| Volume 13, Issue 5, May 2024 ||

| DOI:10.15680/IJIRSET.2024.1305372 |

aspects comprehensively, you can ensure that the Android activity meets the required standards of functionality, usability, performance, and security, resulting in a high-quality and user-friendly application.

## II. LITERATURE SURVEY

Key findings in a review of Android Activity development typically encompass crucial observations, issues, or insights that have a significant impact on the functionality, usability, performance, or maintainability of the application. The activity's UI becomes unresponsive during certain operations, such as loading large datasets or performing network requests synchronously, leading to a poor user experience. This finding highlights the need to optimize long-running tasks by offloading them to background threads or using asynchronous programming techniques. The activity retains references to objects that are no longer needed, resulting in memory leaks and potential performance degradation over time. Identifying and fixing memory leaks is crucial to prevent memory exhaustion and improve the overall stability of the application. The activity's UI elements exhibit inconsistent styling, layout, or behavior, deviating from the established design guidelines and causing confusion for users. Addressing these inconsistencies ensures a cohesive and visually appealing user experience across the application. The activity lacks proper error handling mechanisms, leading to abrupt crashes or unexpected behavior when encountering errors or exceptional conditions. Implementing robust error handling logic helps gracefully recover from failures and provides informative feedback to users. The activity consumes excessive system resources, such as CPU, memory, or battery, due to inefficient algorithms, resource-intensive operations, or unnecessary background tasks. Optimizing resource usage improves the application's performance, responsiveness, and battery life. The activity fails to meet accessibility standards, making it inaccessible or difficult to use for users with disabilities. Addressing accessibility issues ensures inclusivity and compliance with accessibility guidelines, enhancing the application's reach and usability. The activity exposes sensitive data or lacks proper security measures, making it vulnerable to unauthorized access, data breaches, or other security threats. Implementing appropriate security controls, such as encryption, authentication, and authorization, helps protect user data and safeguard the application against malicious attacks. The activity exhibits incorrect or inconsistent behavior during its lifecycle transitions, leading to state loss, resource leaks, or other unexpected outcomes. Ensuring proper lifecycle management helps maintain the activity's state and resources correctly across configuration changes and other lifecycle events. The activity faces difficulties integrating with other components of the application, such as services, fragments, or external APIs, due to compatibility issues, interface mismatches, or inadequate error handling. Resolving integration challenges ensures seamless communication and interoperability between different parts of the application. By identifying and addressing these key findings, developers can enhance the quality, reliability, and user experience of Android activities, contributing to the success of the overall application.

## III. FUNDAMENTAL

In Android development, an Activity represents a single screen with a user interface. It's a fundamental building block of Android applications, and each activity typically corresponds to one screen in the app. The user interacts with the app by moving between different activities, each offering a distinct UI and functionality.

The lifecycle of an Android activity describes the various states an activity goes through during its lifetime, from creation to destruction. Understanding this lifecycle is crucial for managing resources efficiently, handling state changes, and providing a seamless user experience.

|e-ISSN: 2319-8753, p-ISSN: 2347-6710| www.ijirset.com | Impact Factor: 8.423| A Monthly Peer Reviewed & Refereed Journal |

|| Volume 13, Issue 5, May 2024 ||

| DOI:10.15680/IJIRSET.2024.1305372 |

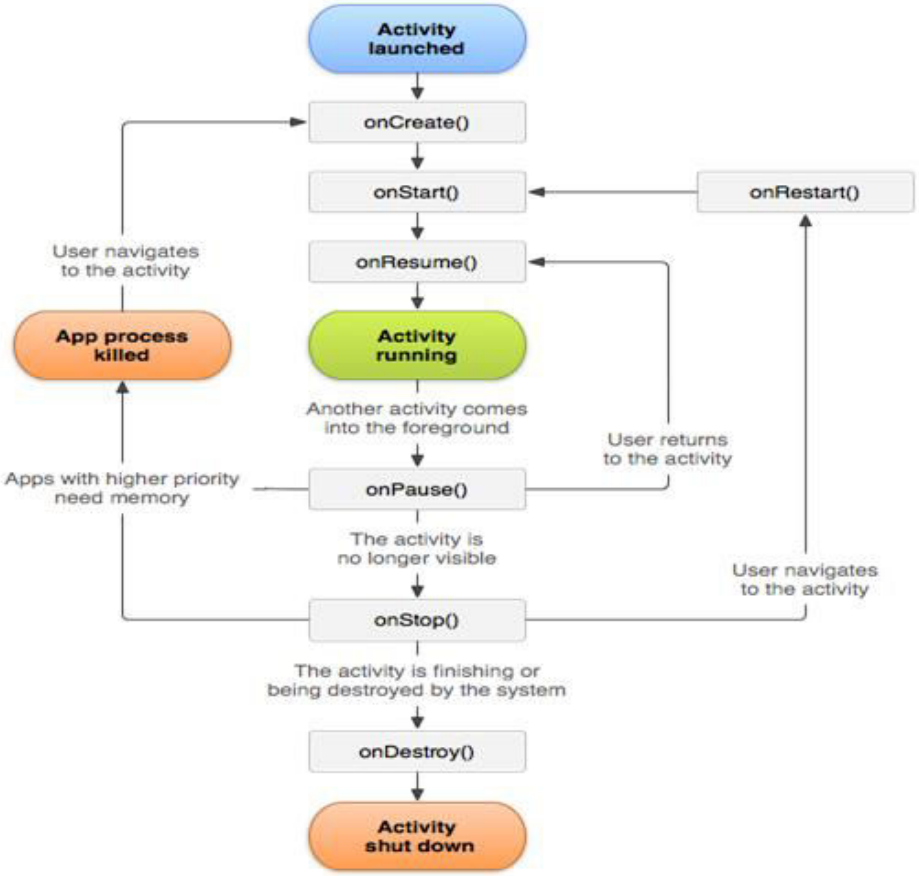


Fig. Android Activity lifecycle.

onCreate(): This method is called when the activity is first created. It's where initialization tasks, such as setting up the UI and initializing variables, should be performed. It receives the initial intent that triggered the activity creation.

onStart(): This method is called when the activity becomes visible to the user but isn't yet in the foreground. It's typically used to start animations or prepare resources that will be needed while the activity is running.

onResume(): This method is called when the activity is about to become visible and enter the foreground. It's where the activity should resume any actions that were paused in onPause() or onStop(), such as starting animations or refreshing data from external sources.

onPause(): This method is called when the activity is no longer in the foreground and is partially obscured by another activity. It's where you should pause ongoing actions that should not continue while the activity is not visible, such as animations or location updates.

onStop(): This method is called when the activity is no longer visible to the user. It's typically used to release resources that are no longer needed, such as stopping background services or unregistering broadcast receivers.

onDestroy(): This method is called when the activity is being destroyed either because the user navigated away from it or because the system is freeing up resources. It's where you should release all resources associated with the activity, such as unregistering listeners or closing database connections.

onRestart(): This method is called when the activity is restarting after being stopped. It's followed by onStart() and onResume() if the activity becomes visible again, or onDestroy() if it's being completely destroyed.

|e-ISSN: 2319-8753, p-ISSN: 2347-6710| www.ijirset.com | Impact Factor: 8.423| A Monthly Peer Reviewed & Refereed Journal |

|| Volume 13, Issue 5, May 2024 ||

| DOI:10.15680/IJIRSET.2024.1305372 |

By overriding these lifecycle methods, developers can manage the behavior of their activities at different stages of their lifecycle, ensuring efficient resource usage and providing a smooth user experience. Additionally, understanding the lifecycle is crucial for handling configuration changes, such as screen rotations, without losing the app's state.

Intents play a crucial role in launching activities and facilitating communication between different components within an Android application. They serve as a messaging mechanism that enables various components (such as activities, services, and broadcast receivers) to request actions from other components or to communicate data between them.

Intents are commonly used to start new activities or to communicate with existing ones. When an activity needs to be launched, an Intent is created with the target activity's class name or an action string identifying the desired action. This Intent is then passed to the Android system using methods like startActivity() or startActivityForResult(). The system then launches the appropriate activity based on the information provided in the Intent.

**Explicit Intents**: These Intents explicitly specify the target component by providing its class name.
**Implicit Intents**: These Intents don't specify the target component by class name but instead provide an action string, data type, or other criteria. The Android system resolves the Intent to the appropriate component based on the provided information.
**Passing Data between Activities**: Intents can also carry data between activities using extras. Extras are key-value pairs that are added to the Intent before it's passed to the system. The receiving activity can then retrieve this data from the Intent and use it as needed.

Overall, Intents provide a flexible and powerful mechanism for launching activities and facilitating communication between different components within an Android application. They enable developers to create modular, loosely coupled components that can interact with each other in a dynamic and efficient manner.

## IV. BEST PRACTICE

Designing Activities in Android involves various aspects ranging from creating responsive interfaces to optimizing performance and memory usage. Here are some best practices for each aspect:

**Creating Responsive and User-Friendly Interfaces:**
Use ConstraintLayout for designing UIs as it offers flexibility and responsiveness across different screen sizes. Optimize layouts for various screen densities and orientations to ensure consistency and usability.
Utilize RecyclerView for displaying large datasets efficiently with smooth scrolling.
Employ Material Design principles for intuitive UI elements and consistent user experience.
Provide feedback to user actions, such as displaying progress indicators during lengthy operations.

**Handling Configuration Changes Gracefully:**
Use ViewModel to store UI-related data that needs to survive configuration changes. ViewModel retains data across configuration changes, such as screen rotations, ensuring a seamless user experience.
Utilize onSaveInstanceState() and onRestoreInstanceState() methods to save and restore UI state respectively for simple UI-related data that doesn't require View Model. Design UIs to handle dynamic changes in configuration gracefully, such as adapting layouts for different screen sizes and orientations.

**Implementing Navigation Patterns:**
Consider using the Navigation Component provided by Android Jetpack for implementing navigation within your app. It simplifies navigation management, supports deep linking, and ensures consistent behavior across different Android versions.
Design a clear and intuitive navigation flow within your app, following user experience (UX) best practices to ensure users can easily navigate between screens.
Utilize features like Safe Args in Navigation Component to pass type-safe arguments between destinations, reducing the risk of runtime errors.

**Optimizing Performance and Memory Usage:**
Minimize UI rendering time by optimizing layouts and reducing view hierarchy complexity.
Implement lazy loading techniques, such as loading images and data asynchronously, to improve app responsiveness and reduce loading times.

|e-ISSN: 2319-8753, p-ISSN: 2347-6710| www.ijirset.com | Impact Factor: 8.423| A Monthly Peer Reviewed & Refereed Journal |

|| Volume 13, Issue 5, May 2024 ||

| DOI:10.15680/IJIRSET.2024.1305372 |

Profile your app's performance using tools like Android Profiler to identify performance bottlenecks and optimize critical sections of code.
Use memory-efficient data structures and caching mechanisms to minimize memory usage and prevent memory leaks.
Leverage background threading for CPU-intensive tasks to prevent UI freezes and ensure smooth user interactions.
By adhering to these best practices, you can create Activities in your Android app that offer responsive interfaces, gracefully handle configuration changes, implement intuitive navigation patterns, and optimize performance and memory usage for a seamless user experience.

## V. LIMITATION AND CHALLENGES

In Android Activity development, several challenges and limitations can arise, including:

**Fragmentation across Different Android Versions and Device Types:**
Fragmentation refers to the wide variety of Android versions and device types in the market, each with its own hardware specifications, screen sizes, and software capabilities. Developers need to ensure their activities are compatible with different versions of Android and various screen resolutions to provide a consistent user experience across devices. Testing on multiple devices and Android versions becomes crucial to identify and address compatibility issues effectively.

**Complexity in Managing Activity Lifecycle and State:**
The Android Activity lifecycle can be complex to manage, especially when dealing with multiple activities and transitions between them.
Developers must handle various lifecycle events, such as onCreate(), onStart(), onResume(), onPause(), onStop(), and onDestroy(), to ensure proper initialization, cleanup, and state preservation. Managing the state of activities, especially during configuration changes like screen rotations, can be challenging. Developers must use techniques like ViewModel, onSaveInstanceState(), and onRestoreInstanceState() to preserve UI state across configuration changes effectively.

**Balancing Performance with Feature Richness:**
Striking a balance between app performance and feature richness is a common challenge in Android Activity development. Adding more features to activities can lead to increased complexity, larger APK sizes, and slower performance. Developers need to prioritize essential features while optimizing performance-critical sections of the app to ensure smooth user experience, quick response times, and efficient resource usage.
Techniques such as lazy loading, background threading, and performance profiling can help identify and address performance bottlenecks without sacrificing essential features.
Addressing these challenges requires careful planning, effective use of Android development best practices, thorough testing across different devices and Android versions, and continuous optimization to deliver high-quality Android activities that offer a seamless user experience. Additionally, staying updated with the latest Android development tools and guidelines can help developers tackle emerging challenges more effectively.

## VI. FUTURE DIRECTIONS

Certainly! Here are some emerging trends and future directions in Android Activity development:

**Adoption of Jetpack Compose for Declarative UI Development:**
Jetpack Compose is a modern toolkit for building native Android UIs using a declarative programming model.
It allows developers to define UI components as functions that transform data into UI elements, simplifying UI development and making it more intuitive.
With features like state management, composables, and theming, Jetpack Compose offers improved productivity, code reusability, and UI consistency.
As Jetpack Compose continues to mature, it is expected to become the preferred choice for building dynamic and responsive UIs in Android activities, replacing the traditional XML-based layout system.

**Integration with Emerging Technologies (e.g., Augmented Reality, Foldable Devices):**
Android activities are expected to integrate more seamlessly with emerging technologies such as augmented reality (AR) and foldable devices.

|e-ISSN: 2319-8753, p-ISSN: 2347-6710| www.ijirset.com | Impact Factor: 8.423| A Monthly Peer Reviewed & Refereed Journal |

|| Volume 13, Issue 5, May 2024 ||

| DOI:10.15680/IJIRSET.2024.1305372 |

With the growing popularity of AR applications, activities may incorporate AR features for immersive user experiences, such as overlaying virtual objects onto the real world. Foldable devices present unique challenges and opportunities for activity development, requiring support for dynamic screen configurations and flexible layouts to adapt to different form factors.

**Enhancements in Tooling and Developer Experience:**

Google continues to invest in improving tooling and developer experience for Android activity development. Android Studio, the official IDE for Android development, receives regular updates with new features, performance improvements, and support for the latest Android APIs and tools.

Kotlin, the preferred programming language for Android development, continues to evolve with new language features and optimizations, enhancing developer productivity and code quality.

Google's efforts to streamline the development process, provide comprehensive documentation, and offer online resources and courses contribute to a better developer experience and foster innovation in Android activity development.

In summary, Android activity development is evolving with the adoption of Jetpack Compose for declarative UI development, integration with emerging technologies like AR and foldable devices, and continuous enhancements in tooling and developer experience. By embracing these trends, developers can build more dynamic, immersive, and user-friendly activities that leverage the latest advancements in Android development.

## VII. CONCLUSION

Staying updated with evolving best practices and technologies is crucial for Android Activity developers. Key insights include: prioritizing compatibility across devices and Android versions, meticulous management of activity lifecycle and state, balancing performance optimization with feature richness, integrating emerging technologies like AR and foldable devices, and embracing continuous learning to adapt to changes in the field. Keeping abreast of new developments ensures developers can create high-quality, innovative, and user-friendly Android activities that meet the demands of today's dynamic mobile landscape.

## REFERENCES

[1] ANDROID THE MOBILE OPERATING SYSTEM AND ARCHITECTURE 1Manishaben Jaiswal

[2] Research on Development of Android Applications Mrs. Prachi Sasankar1 . Mrs. Usha Kosarkar. 2 1 ( Prachi.sasankar@raisoni.net, BCA, Sadabai Raisoni Women's College,Nagpur SNDT Women's University,Mumbai , India.) 2 (Usha.kosarkar@raisoni.net , BCA, G.H.R.I.I.T.Nagpur, R.T.M.Nagpur University,Nagpur.,India).

[3] A Study on Android Application Development Dr. A. S. Kapse*, Vinayak Harlalka, Shripad Kulkarni Computer Science and Engineering Department, Sant Gadge Baba Amravati University, Chikhli Dist: Buldana, India

[4] Research on Android Architecture and Application Development To cite this article: Yuanyi Chen 2021 J. Phys.: Conf. Ser. 1992 022168.

[5] International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 16 (2018) pp. 12527-12530 © Research India Publications. http://www.ripublication.com 12527 Development of Native Mobile Application Using Android Studio for Cabs and Some Glimpse of Cross Platform Apps Neha Verma1 , Sarita Kansal2 , Huned Malvi3 2Department of Electronics and communication Engineering, Medi-caps University, Indore-453331, Madhya Pradesh, India. 3 Infonex Solutions, Indore -452001, Madhya Pradesh, India.

[6] A LITERATURE SURVEY OF ANDROID BASED EDUCATIONAL APPLICATION 1Akash Sharma, 2Gaurav Singh, 3Mehak 1Student, 2Student, 3Assistant Professor 1Department of Computer Science and Engineering, 1Lovely Professional University, Phagwara,Punjab, India.

[7] Research on Development of Android Applications Dr. D. M. Patel Prof. Dhruval Kachiya • Sharma Sagar • Gosai Dron • Gaikwad Ritesh • Baviskar Jay

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi cross ref

निस्केयर NISCAIR

# INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

## IN SCIENCE | ENGINEERING | TECHNOLOGY

9940 572 462 6381 907 438 ijirset@gmail.com

www.ijirset.com

Scan to save the contact details