Discussion-6 What are some advanced techniques for optimizing performance in an Android app?

Discussion Topic:

Please choose one of the following questions to discuss in your initial post:

What are some advanced techniques for optimizing performance in an Android app?
How can you leverage background services and threads to perform long-running tasks in an Android app?
What are some strategies for implementing offline capabilities and synchronization in an Android app?
In your response to your peers, you can explore alternative perspectives or additional insights related to the answers provided by your classmates. Consider what other lessons or conclusions you can draw from your classmate's post.

My Post:

Hello Class,

Optimizing performance in an Android app goes beyond basic coding practices, it implies using strategies such as optimizing app startup, code & resource optimization, improving User Interface (UI) performance, managing background tasks, optimizing network operations, optimizing database performance, and monitoring the app performance. This post provides an overview of those strategies.

Optimizing App Startup

How an app starts, no one likes to look at a black or empty screen while waiting for the app to start, and the time it takes for an app to become fully functional after launch can significantly impact the user's perception of the application. Thus, optimizing an app's startup process and reducing its launch time are essential strategies for minimizing user frustration and providing a responsive and enjoyable user experience. This can be done by implementing a Startup Profile which is a subnet Baseline Profiles (Android Developers, n.d.a).

Baseline Profiles, which contain lists of classes and methods, are a mechanism that can improve the startup and runtime performance of apps and libraries (Android Developers, n.d.b). It can speed up code execution by about 30% from the first launch by avoiding interpretation and just-in-time (JIT) compilation steps for included code paths (Android Developers, n.d.c). They can be implemented in an app or library, and Android Runtime (ART) can then optimize specified code paths through Ahead-of-Time (AOT) compilation. This improves performance for every new user and every app update. Additionally, the Profile Guided Optimization (PGO) feature of the Baseline Profiles allows developers to optimize app startup. Similarly, Startup Profiles which are a subset of Baseline Profiles can be used to further optimize the classes and methods they contain, improving app startup time (faster) usually between 15% and 30% than with Baseline Profiles alone (Android Developers, n.d.c).

Optimizing UI Performance

Overdraw is an issue that occurs when an Android system draws the same pixel on the screen multiple times within a single frame, reducing rendering efficiency and draining battery life (Shrivastava, 2025). to minimize overdraw by flattening the view hierarchy by using layouts like *ConstraintLayout* instead of deeply nested *LinearLayouts*. Additionally, reducing the use of transparency like transparent views elements can also help to reduce overdraw (Engineering, 2024). Another effective technique for optimizing rendering is to implement a custom drawable that renders only the necessary pixels (Hadawale, 2024). Jetpack Compose, overdraw can be mediated by using Composables like *Box* that manages background colors and by utilizing the *clip* modifier that can clip drawing areas.

Managing Background Tasks

Managing background tasks can significantly improve the performance of an app. Android provides several APIs for managing those tasks. *WorkManage*, for example, can schedule tasks that need to be executed even if the app is closed preserving consistent app functionality. Additionally, it checks network availability, battery level, and device idle state to ensure that background tasks are executed at the optimal time (Mewada, 2024). *WorkManage* uses other background processes like *JobScheduler*, *AlarmManager*, and *BroadcastReceiver* providing an all-in-one easy-to-use API. This allows developers to optimize their apps' performance by managing long-running or resource-intensive operations, ensuring they run in the background instead of on the main UI thread.

Optimizing Network Operations

One of the fundamental aspects of network optimization is to reduce, as much as possible, the amount of data transmitted over the network. Reducing data transmission improves battery life and the overall responsiveness of the app, especially if the application is network-dependent. Utilizing protocols like *HTTP/2* can reduce data transmission (App Dev Insights, 2023), by ensuring that the server serving the application's backend has *GZIP* compression enabled for responses, by using offline compression tools like *Zopfli* or *7-zip* to compress data before transmitting it. For media files, using *WebP* format for images can provide better compression than *JPEG* or *PNG* (Android Developers, n.d.a). For videos, using codecs like *HEVC* can significantly reduce file sizes without significant quality loss.

Optimization Database Performance

It is important to write efficient *SQL* queries for data performance. One of the most important rules is to select only the necessary rows and columns from the database. In other words, it is better to avoid using *SELECT **(select all) and instead select only the columns required for the operation. Additionally, using parameters for query (e.g., '*?*') instead of strings not only prevents *SQL* injection but can also improve query performance by allowing the reuse of the query code (Kramer, 2024). Furthermore, when processing data within the *SQL* query itself, using aggregate functions like *COUNT*, *SUM*, *MIN*, *MAX*, *AVG*, and *GROUP_CONCAT*, is generally more efficient than retrieving the data and processing using the application code (e.g. Kotlin or Java). When using *Room*, Android's persistence library (an abstraction using *SQLite*), instead of raw *SQLite* developers should understand how *Room* utilizes *SQLite* to ensure that their *Room* is optimized for data access and manipulation.

Performance Monitoring

An essential aspect of optimizing performance is to continually monitor and profile the app's runtime behavior using tools like Android Profiler and Firebase Performance Monitoring to identify performance bottlenecks and address them. *Android Profiler* is a set of tools built in *Android Studio* that allows the monitoring and analysis of the performance and behavior of an Android app (Mudadla, 2023). *Firebase Performance Monitoring* collects performance data using traces which are reports that contain data captured between two points in time in an app (Firebase, n.d.). Other tools that can be used for performance monitoring are *BlazeMeter* for real-time app performance monitoring with simulation capabilities, *Apptim* for identifying performance issues early in development, and *New Relic Mobile* for mobile app analytics (KmDev, 2025).

Summary

Table 1

Strategies and Tools for Optimizing Android App Performance

Optimization	Techniques/Tools
Optimizing App Startup	Startup Profile (subset of Baseline Profiles), Baseline Profiles (lists of classes and methods), Ahead-of-Time (AOT) compilation, Profile Guided Optimization (PGO)
Optimizing UI Performance	Flattening view hierarchy (using ConstraintLayout), reducing transparency, implementing custom drawables, using Box composables and clip modifier in Jetpack Compose
Managing Background Tasks	WorkManager (utilizing JobScheduler, AlarmManager, and BroadcastReceiver), scheduling tasks based on network availability, battery level, and device idle state
Optimizing Network Operations	Reducing data transmission, utilizing HTTP/2, enabling GZIP compression on the server, using offline compression tools (Zopfli, 7-zip), using WebP format for images, using codecs like HEVC for videos
Optimization Database Performance	Writing efficient SQL queries (selecting necessary rows and columns, avoiding SELECT *), using parameters for queries, using aggregate functions in SQL, understanding Room's use of SQLite
Performance Monitoring	Android Profiler (built into Android Studio), Firebase Performance Monitoring (using traces), BlazeMeter, Apptim, New Relic Mobile

Note: The table lists and describes different strategies and tools for optimizing Android app performance.

As shown in Table 1, optimizing performance in an Android app goes beyond basic coding practices; it involves employing specific strategies and utilizing various tools. These strategies and tools can significantly improve the performance of an Android app, leading to a more functional and ultimately more satisfying user experience.

-Alex

References:

Android Developers (n.d.a). Best practices for app optimization. *Performance*. Android. https://developer.android.com/topic/performance/appstartup/best-practices

Android Developers (n.d.b). *Making apps blazing fast with baseline profiles*. YouTube. https://www.youtube.com/watch?v=yJm5On5Gp4c#:~:text=Baseline%20Profiles%20are%20a%20new%20way%20to,Runtime%20improves%20app%20performance%2C%20when%20a%20Baseline

Android Developers (n.d.c). Baseline Profiles overview. *Performance*. Android. https://developer.android.com/topic/performance/baselineprofiles/overview

Android Developers (n.d.d). Create Startup Profiles. *Performance*. Android. https://developer.android.com/topic/performance/baselineprofiles/dex-layout-optimizations

App Dev Insights. (2023, December 13). *How do you reduce battery consumption?* App Dev Insights - Medium. https://medium.com/@appdevinsights/how-do-you-reduce-battery-consumption-face63ba6329

Engineering (2024, July 10). *Android UI performance: GPU overdraw*. Think-it. https://think-it.io/insights/Android-Performance-GPU-overdraw

Firebase (n.d.). *Firebase Performance Monitoring*. Google. https://firebase.google.com/docs/perf-mon#:~:text=The%20collected%20performance%20data%20for,response%20time%20and%20payload%20size.

Hadawale, P. (2024, November 22). *Android JetPack Compose and GPU Overdraw: painting a performance masterpiece!* Medium. https://medium.com/@hadawalepranav/reducing-gpu-overdraw-in-android-jetpack-compose-5d0920a81958

KmDev. (2025, February 2*). Improving app performance: Profiling and debugging with Android Studio*. *Medium*. https://medium.com/@mkcode0323/improving-app-performance-profiling-and-debugging-with-android-studio-391f0f79b619

Kramer, N. (2024, March 14). *Query data efficiently: Best practices.* Daily Dev. https://daily.dev/blog/query-data-efficiently-best-practices

Mewada, p. (2024, January 18). *Background tasks in Android*. Scaler Topics. https://www.scaler.com/topics/android/background-tasks-in-android/

Mudadla, S. (2023, July 12). *What is Android profiler?* Medium. https://medium.com/@appdevinsights/how-do-you-reduce-battery-consumption-face63ba6329

Shrivastava, A. (2025, May 7). *Android performance profiling: From jank to smooth 120fps*. Quash. https://quashbugs.com/blog/android-performance-profiling-from-jank-to-smooth-120fps