International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2016): 79.57 | Impact Factor (2015): 6.391

Energy - Saving Design Patterns for Mobile Applications

Jagadeesh Duggirala

Software Engineer, Rakuten, Japan
Email: *jag4364u[at]gmail.com*

Abstract: ***In the rapidly evolving landscape of mobile applications, energy efficiency has become a critical concern for developers, users, and device manufacturers. Mobile devices are powered by batteries with finite capacity, and the demand for energy - efficient applications is increasing as users seek to maximize the battery life of their devices. This paper explores various energy - saving design patterns for mobile applications, aiming to provide developers with actionable insights and best practices to optimize their applications for energy efficiency.***

Keywords: android applications, battery, energy saving, batching requests, work manager, background services

1. Introduction

The significance of energy efficiency in mobile applications cannot be overstated. With the proliferation of smartphones and mobile applications, the demand for longer battery life has surged. Energy - hungry applications can lead to frequent recharging, reduced device performance, and user dissatisfaction. Understanding the factors that contribute to energy consumption in mobile applications is essential for designing energy - efficient solutions.

Key Factors Influencing Energy Consumption

1) **Display**: The screen is one of the most significant energy consumers in a mobile device. Brightness levels, screen resolution, and usage duration all impact battery life.
2) **Processing**: CPU and GPU usage for running applications and rendering graphics can drain the battery quickly.
3) **Networking**: Wireless communication (Wi - Fi, cellular data, Bluetooth) consumes a considerable amount of energy, especially during data transfer.
4) **Sensors**: GPS, accelerometer, gyroscope, and other sensors continuously collect data, which can lead to increased energy consumption.
5) **Background Services**: Applications running in the background can consume energy without the user's awareness.

Energy - Saving Design Patterns

1) Lazy Initialization Pattern
- **Concept**: Initialize objects only when they are needed rather than at the start of the application.
- **Implementation**: Use lazy properties in Kotlin (Lazy or lateinit) and similar mechanisms in other languages.
- **Benefit**: Reduces initial load time and saves energy by avoiding unnecessary object creation.

2) Observer Pattern
- **Concept**: Use observer or listener mechanisms to update UI components only when there is a change in data.
- **Implementation**: Utilize LiveData or Flow in Android to observe data changes and update the UI accordingly.
- **Benefit**: Minimizes the number of UI refreshes and reduces CPU usage.

3) Batch Processing Pattern
- **Concept**: Accumulate multiple operations and process them in a batch rather than individually.
- **Implementation**: Group network requests or database transactions and execute them together.
- **Benefit**: Reduces the frequency of resource - intensive operations like network access and I/O operations.

4) Adaptive Rate Pattern
- **Concept**: Adjust the rate of operations based on the current context such as battery level, network conditions, or user activity.
- **Implementation**: Use adaptive algorithms to change the polling rate or update frequency.
- **Benefit**: Saves energy by reducing the intensity of operations during low power conditions.

5) Resource Pooling Pattern
- **Concept**: Reuse expensive resources such as database connections, threads, and objects instead of creating new ones.
- **Implementation**: Implement object pools, connection pools, and thread pools.
- **Benefit**: Reduces the overhead of creating and destroying resources, leading to energy savings.

6) Geofencing Pattern
- **Concept**: Use geofencing to perform location - based tasks only when the user enters or exits specified areas.
- **Implementation**: Use geofencing APIs provided by platforms like Android and iOS.
- **Benefit**: Reduces the need for continuous location tracking, saving energy.

7) Data Caching Pattern
- **Concept**: Store frequently accessed data locally to minimize repetitive network requests.
- **Implementation**: Use caching mechanisms provided by libraries such as Glide for images and Room for databases.
- **Benefit**: Reduces network usage and speeds up data access, saving energy.

8) Deferred Work Pattern
- **Concept**: Postpone non - critical work to a later time when the device is plugged in or has sufficient battery.
- **Implementation**: Use job scheduling APIs like WorkManager in Android to defer tasks.

- **Benefit**: Ensures that energy - intensive tasks are performed only when appropriate, conserving battery life.

9) Dark Mode Pattern

- **Concept**: Provide a dark theme option to reduce the energy consumption of devices with OLED screens.
- **Implementation**: Design and implement themes for both dark and light modes.
- **Benefit**: Saves energy on OLED screens where black pixels consume less power.

10) Event Throttling/Debouncing Pattern

- **Concept**: Limit the rate at which event handlers are called to avoid excessive processing.
- **Implementation**: Implement throttling and debouncing techniques using libraries like RxJava or custom implementations.
- **Benefit**: Reduces the number of operations performed in response to rapid events, saving CPU cycles and energy.

11) Sensor Fusion Pattern

- **Concept**: Combine data from multiple sensors to reduce the frequency and complexity of sensor readings.
- **Implementation**: Use sensor fusion APIs to integrate accelerometer, gyroscope, and other sensor data.
- **Benefit**: Improves accuracy while reducing the overall energy consumption of sensor operations.

12) Efficient Bitmap Handling Pattern

- **Concept**: Optimize the loading and processing of images to reduce memory and CPU usage.
- **Implementation**: Use image loading libraries like Glide or Picasso and resize images appropriately.
- **Benefit**: Reduces the energy required for image processing and memory management.

13) Idle Detection Pattern

- **Concept**: Detect periods of user inactivity to reduce the frequency of updates and background operations.
- **Implementation**: Monitor user interactions and adjust the activity of background tasks accordingly.
- **Benefit**: Saves energy by minimizing unnecessary operations during idle periods.

14) Efficient Navigation Pattern

- **Concept**: Optimize navigation flow to reduce the creation of unnecessary activities or fragments.
- **Implementation**: Use single - activity architecture with navigation components.
- **Benefit**: Reduces memory usage and improves performance, leading to energy savings.

15) Incremental Data Loading Pattern

- **Concept**: Load data in chunks or pages instead of all at once to minimize memory usage and processing time.
- **Implementation**: Implement pagination for lists and other large data sets.
- **Benefit**: Saves energy by reducing the amount of data processed at a time and the frequency of data loading operations.

By employing these energy - saving design patterns, developers can create mobile applications that are both efficient and user - friendly, leading to longer battery life and better overall performance.

2. Conclusion

Energy efficiency is a crucial aspect of modern mobile application development. By implementing the design patterns and best practices discussed in this paper, developers can create applications that are not only functional and user - friendly but also optimized for energy efficiency. As technology continues to advance, the importance of energy - saving design patterns will only grow, making it essential for developers to stay informed and proactive in their approach to energy - efficient mobile app development.

References

[1] Abed, S., Ferzli, R., & Abed, F. (2016). Energy - efficient mobile applications design: Theoretical overview and practical recommendations. *Journal of Mobile Networks and Applications*, 21 (5), 765 - 780.

[2] Carroll, A., & Heiser, G. (2010). An analysis of power consumption in a smartphone. *Proceedings of the 2010 USENIX Annual Technical Conference*.

[3] Ding, Y., Zheng, H., & Kanhere, S. S. (2013). Energy - efficient scheduling for mobile cloud computing. *Proceedings of the 12th ACM International Conference on Mobile and Ubiquitous Systems*.

[4] Ferreira, D., Dey, A. K., & Kostakos, V. (2011). Understanding human - smartphone concerns: A study of battery life. *Proceedings of the 9th International Conference on Pervasive Computing*.

[5] Google. (n. d.). Battery Historian. Retrieved from https://developer.android.com/studio/profile/battery - historian