

Instant App-like Experiences in iOS: On-Demand Functionality and Its Implications

Gavrav Sharma¹, Abhishek Negi¹, Tribhuvan Chauhan³

Associate Professor¹, Students^{2,3}

Suryadatta Institute of Technology, Pune, India

Maharaja Sayajirao University, Vadodara

Email ID: Gavravsharma558@gmail.com¹, abhishek24negi@yahoo.com²

Abstract

The mobile application landscape is rapidly evolving, with user expectations shifting towards lightweight, instant, and context-aware experiences. In iOS, Apple's implementation of on-demand functionality enables applications to deliver specific features instantly, without requiring full installation. This concept, analogous to Android Instant Apps, is transforming user engagement, app retention, and performance optimization. This paper provides a comprehensive review of instant app-like experiences in iOS, including the technical framework, design considerations, use cases, benefits, limitations, and future directions. By analyzing current industry practices and developer strategies, this study identifies the critical factors influencing successful implementation and adoption of on-demand functionality. The findings highlight how modularization, app thinning, and intelligent caching mechanisms contribute to improved user experience while minimizing resource overhead.

Keywords: *iOS, Instant Apps, On-Demand Functionality, App Thinning, Modularization, User Experience, Swift, App Store, App Retention*

1. Introduction

Mobile applications have become indispensable tools for daily activities, ranging from communication and productivity to entertainment and e-commerce. Traditional apps, however, often demand large storage space, lengthy download times, and frequent updates, which can

create friction for users. To address these challenges, Apple introduced **on-demand resources (ODR)** and **app thinning** as part of iOS optimization strategies, allowing users to access app features or content without installing the full app bundle.

Instant app-like experiences in iOS leverage these technologies to provide **on-demand functionality**, enabling a subset of an application to be downloaded and executed immediately. This paradigm enhances user engagement by reducing friction, offering personalized interactions, and maintaining the benefits of full-featured apps without requiring permanent installation.

This paper reviews the technical architecture, development techniques, usability considerations, and real-world implications of instant app-like experiences in iOS.

2. Background

The concept of instant app-like experiences in iOS is deeply rooted in Apple's continuous efforts to optimize app distribution, reduce device storage consumption, and enhance user experience. Traditional mobile applications often require users to download the full app package, which can be large in size due to high-resolution assets, media files, and multiple functionalities. This approach not only increases the barrier for first-time users but also impacts app performance, launch speed, and storage management. To address these challenges, Apple introduced **on-demand functionality** as part of its broader **App Thinning strategy** starting with iOS 9. This section explores the evolution of these technologies and the principles underlying instant app-like experiences in iOS.

2.1 Evolution of On-Demand Functionality in iOS

On-demand functionality in iOS evolved in response to increasing user expectations for faster app access, smaller downloads, and context-aware interactions. Apple introduced **App Thinning** in iOS 9, a set of optimization techniques that reduce app footprint and enable dynamic delivery of app content. App Thinning consists of three main components:

1. Slicing

Slicing allows developers to create device-specific versions of the app, containing only the resources required for that particular device. For instance, an iPhone with a standard Retina display will receive only the images optimized for its resolution, while an iPad Pro will receive

higher-resolution assets. Similarly, binaries can be optimized for different processor architectures (e.g., ARM64 vs. ARMv7). This prevents users from downloading unnecessary assets, reducing initial app size and improving installation times.

2. Bitcode

Bitcode is an intermediate representation of an app's compiled code submitted to the App Store instead of the traditional binary. Apple can recompile apps from bitcode to optimize for specific devices and future hardware, ensuring better performance without requiring developers to resubmit updates. Bitcode enables efficient device targeting, dynamic optimization, and smaller app bundles, which indirectly supports instant app-like experiences.

3. On-Demand Resources (ODR)

ODR is a framework that allows developers to tag non-essential assets as **on-demand**, meaning they are hosted remotely on the App Store and downloaded only when needed. Developers can specify resource tags and assign priority levels, ensuring that essential assets are available immediately while optional content is fetched dynamically. For example, in a travel app, basic functionality like search and booking may be part of the core app, while high-resolution images of tourist destinations or promotional videos are delivered on-demand.

Collectively, these features reduce the app's initial size, enable modular downloads, and improve launch times, creating a foundation for instant app-like experiences. Apple's approach mirrors **Android Instant Apps**, but with iOS-specific considerations such as tighter sandboxing, App Store hosting of resources, and stricter guidelines on ODR usage. The evolution of these technologies highlights a trend toward **modular, context-aware app delivery**, enabling developers to provide tailored user experiences without burdening devices with unnecessary content.

2.2 Principles of Instant App-like Experiences

Instant app-like experiences in iOS are built on a set of design and architectural principles that guide both developers and user experience designers. These principles ensure that on-demand functionality is efficient, seamless, and engaging:

1. Modularity

At the core of on-demand functionality is modular design. Applications are divided into **independent modules** or features, each capable of operating in isolation. This allows users to interact with a specific functionality of the app without downloading the full package. For

example, a messaging app could allow users to view their inbox and send messages instantly, while advanced features like theme customization, file attachments, or media galleries are downloaded only when required. Modularization also improves code maintainability and facilitates incremental updates to individual features.

2. Resource Management

Effective resource management ensures that devices are not burdened with unnecessary data. Unused or temporary resources are automatically purged from storage, freeing space and maintaining optimal device performance. On-demand resources, caching mechanisms, and priority-based downloads are key strategies for managing app assets. For instance, a media app can remove video previews that have already been watched while retaining high-demand content locally.

3. Context Awareness

Instant app-like experiences are most effective when they are **contextually relevant**. Features are delivered based on user behavior, location, time, or specific triggers. For example, a transportation app may only fetch ticketing modules when the user is near a transit station, or a shopping app may preload seasonal promotions during festive periods. Context-aware resource fetching reduces unnecessary network usage and enhances user satisfaction by providing timely and relevant functionality.

4. Seamless Integration

The ultimate goal is to make the on-demand experience **indistinguishable from the full app experience**. Users should not feel that they are interacting with a lightweight or temporary module. This requires careful attention to UI/UX consistency, smooth transitions during resource downloads, and minimal latency. Proper preloading, predictive fetching, and responsive error handling contribute to seamless integration.

By adhering to these principles, developers can create **instant app-like experiences** that are efficient, user-centric, and aligned with iOS platform capabilities. These principles not only optimize performance and resource usage but also enhance user engagement, retention, and satisfaction.

3. Technical Framework

Implementing instant app-like experiences in iOS requires a solid understanding of Apple's **technical framework**, which revolves around App Thinning, On-Demand Resources (ODR), modular app architecture, and intelligent caching strategies. These components work together

to optimize app size, improve launch performance, and deliver features dynamically based on user context.

3.1 App Thinning and On-Demand Resources (ODR)

App Thinning is a set of optimization techniques introduced by Apple in **iOS 9** to reduce the initial download size of apps and deliver resources more efficiently. It enables developers to distribute apps in a device-specific, modular, and performance-optimized manner. App Thinning comprises three primary components: **Slicing**, **Bitcode**, and **On-Demand Resources (ODR)**.

1. Slicing

Slicing allows developers to generate multiple variants of an app, each containing only the assets required for a particular device. For example:

- An iPhone 14 Pro with a Super Retina XDR display will receive **high-resolution images** and optimized binaries.
- An iPhone SE with a smaller screen and lower resolution will receive **lightweight assets**.

By delivering only device-specific resources, slicing reduces app size, download time, and storage usage. Slicing also ensures that performance-critical resources, such as graphics and animations, are optimized for the target device architecture.

2. Bitcode

Bitcode is an intermediate representation of the app submitted to the App Store, rather than a precompiled binary. This allows Apple to **recompile and optimize the app** for different hardware devices, screen resolutions, or operating system versions without requiring developers to submit new builds. Benefits include:

- Reduced app size through hardware-specific optimizations.
- Support for future devices and architectures automatically.
- Improved performance and energy efficiency due to optimized code execution.

3. On-Demand Resources (ODR)

ODR enables developers to tag certain assets as **on-demand**, meaning they are not included in the initial download of the app bundle. These resources are hosted on the App Store and can be dynamically fetched when the user accesses a specific feature or content.

Key concepts of ODR include:

- **Resource Tags:** Developers define tags for asset groups, such as "galleryImages", "tutorialVideos", or "premiumFeatures". These tags are referenced in the app code when requesting assets.
- **Priority Levels:** Developers can assign priority values to resources (high, medium, low) to control which assets should be fetched first. Critical resources can be preloaded, while less important assets can be downloaded in the background.
- **Caching and Purging:** The system caches downloaded resources temporarily, and unused assets are automatically purged when the device is low on storage, ensuring efficient resource management.
-

Table 1: App Thinning Components and Functions

Component	Function
Slicing	Device-specific binaries and resources
Bitcode	Optimized recompilation for device architecture
On-Demand Resources	Dynamic downloading of assets when required
App Size Metrics	Analytics to monitor downloaded resources and cache usage

ODR uses **resource tags**, which are referenced in code, enabling precise download of assets such as images, videos, or UI elements. Developers can prioritize downloads, preload critical resources, and remove unused content.

3.2 Modular Architecture

Modular architecture is a core principle for implementing instant app-like experiences in iOS. By breaking an application into **independent modules**, developers can deliver specific features on-demand, reducing initial download size and improving performance. Modularization also simplifies code maintenance, testing, and updates, allowing developers to modify one module without affecting others.

Components of a Modular Architecture:

1. Core Module

The Core Module includes essential functionality that must be available immediately after app installation. This typically contains:

- User authentication and onboarding workflows

- Navigation structures and main UI frameworks
- Basic business logic required to interact with the app
- Minimal assets for app branding and default screens

For example, in a news application, the Core Module may include login, main feed navigation, and text-based article previews. Heavy resources such as video content, interactive graphics, or premium features can be separated into feature modules.

2. Feature Modules

Feature Modules are independent components that can be **loaded on-demand** using App Thinning and ODR. Each module focuses on a specific functionality and may include its own assets, view controllers, and business logic.

Examples of Feature Modules:

- A shopping app could have separate modules for product gallery, promotions, and user reviews.
- A fitness app could load workout routines and video tutorials only when the user selects them.
- A travel app could fetch hotel images, map overlays, and tour guides dynamically.

Feature Modules improve app scalability by allowing developers to incrementally add or update functionalities without requiring a full app update.

3. Shared Services Module

The Shared Services Module manages resources and services that are used across multiple modules. This may include:

- Networking and API communication
- Authentication and user session management
- Caching mechanisms for on-demand resources
- Analytics and telemetry services

This centralized approach prevents duplication of code and ensures consistent behavior across the app. For example, caching logic implemented in the Shared Services Module ensures that any module fetching resources via ODR can reuse the same cached data efficiently.

Tools for Modularization:

- **Swift Package Manager (SPM):** Facilitates dependency management and module integration, enabling developers to define separate Swift packages for each feature module.
- **Cocoapods:** Allows modularization through pods, which can encapsulate specific functionalities while providing version control and dependency management.

Advantages of Modular Architecture:

- Reduced initial app size, enabling faster downloads and installations.
- Easier testing and debugging since modules are isolated.
- Better scalability, allowing new features to be added independently.
- Enhanced user experience as only relevant features are loaded dynamically.

3.3 Intelligent Caching and Preloading

Caching and preloading are essential for ensuring a smooth user experience in instant app-like experiences. Without caching, on-demand resources would need to be downloaded every time a feature is accessed, leading to delays and higher network usage. Intelligent caching strategies address this issue by predicting user behavior, storing frequently used assets, and preloading critical resources.

Key Strategies for Intelligent Caching:

1. Predictive Preloading Based on User Behavior

By analyzing user interactions, developers can **predict which resources the user is likely to access next** and preload them in the background.

Example:

- In a news app, if a user frequently reads sports articles, the app can preload images and video highlights for upcoming sports stories before the user clicks on them.
- In an e-commerce app, the app could preload the top-selling product images in anticipation of user browsing patterns.

Predictive preloading can leverage machine learning or simple rule-based analytics to optimize network usage and reduce latency.

2. LRU (Least Recently Used) Caching

LRU caching is a strategy to store recently accessed resources temporarily while discarding older, unused assets when storage limits are reached.

Implementation:

- ODR caches downloaded assets automatically, but developers can implement custom LRU logic for feature modules to manage memory and storage efficiently.
- This ensures that frequently used assets remain available, while rarely used resources are removed to free space.

3. Background Downloads with NSURLSessionDownloadTask

Background downloading ensures that critical resources are fetched without blocking the main UI thread, improving responsiveness.



Figure 1: Instant App Resource Flow in iOS

4. Use Cases

4.1 Retail and E-commerce

Instant app-like features allow users to explore product catalogs, check-out, or view promotions without full app installation. This reduces friction for first-time users and increases conversion rates.

4.2 Travel and Navigation

On-demand modules for booking tickets, viewing maps, or checking real-time updates enhance user experience while minimizing storage consumption.

4.3 Media Streaming

Media apps can offer previews, trailers, or sample content instantly without downloading the entire app or media library.

4.4 Enterprise Applications

Businesses can deploy on-demand modules for specific workflows, ensuring employees only download tools required for their roles.

5. Benefits

1. **Reduced App Size:** Core apps remain lightweight.
2. **Enhanced Engagement:** Users can instantly interact with features without commitment.
3. **Optimized Resource Usage:** On-demand downloads minimize storage and network overhead.
4. **Flexible Updates:** Modules can be updated independently without full app updates.

Table 2: Benefits vs. Traditional Apps

Feature	Traditional App	Instant App-like iOS App
Initial Download Size	Large	Small (Core Only)
First-time User Engagement	Moderate	High
Storage Management	Manual	Automatic via ODR
Update Frequency	Full App Updates	Module-specific

6. Challenges and Limitations

1. **Network Dependency:** On-demand resources require stable connectivity.
2. **Complexity in Development:** Modular design increases development and testing complexity.
3. **Latency Issues:** Fetching resources may introduce slight delays if not optimized.
4. **App Store Guidelines:** Apple imposes strict rules on ODR usage and content delivery.

7. Design Considerations

1. **Predictive Loading:** Anticipate user actions to reduce perceived latency.
2. **Minimal Core Module:** Include only critical functionalities at launch.
3. **Error Handling:** Gracefully manage failed downloads and network interruptions.
4. **Analytics Integration:** Monitor resource usage, module access, and user retention.

8. Future Directions

- **AI-Driven Resource Fetching:** Predict user intent to pre-download modules dynamically.

- **Cross-App Modularization:** Sharing on-demand modules across apps for ecosystem consistency.
- **Integration with SwiftUI:** Dynamic UI loading for smoother instant experiences.
- **Edge Caching:** Using device or network edge servers to reduce download latency.

9. Conclusion

Instant app-like experiences in iOS present a paradigm shift in mobile application delivery, balancing user convenience with efficient resource utilization. By leveraging app thinning, ODR, and modular architecture, developers can deliver lightweight, responsive, and contextually relevant features. While challenges such as network dependency and development complexity exist, proper design strategies, predictive caching, and analytics integration can mitigate these issues. The future of iOS apps lies in more intelligent, adaptive, and modular experiences, aligning with growing user expectations for instant, frictionless interactions.

References

1. Apple Developer Documentation. *App Thinning and On-Demand Resources*. Apple Inc., 2025.
2. Guo, Y., & Zhang, L. *Modular Mobile App Architectures*. Journal of Mobile Computing, 2023.
3. Kim, H. *Instant Apps in Mobile Ecosystems: Lessons for iOS*. Mobile HCI Journal, 2022.
4. Singh, R., & Joshi, P. *Performance Optimization in iOS Applications*. International Journal of Computer Science, 2024.
5. Lee, S., et al. *Predictive Resource Loading for Instant Mobile Experiences*. ACM Mobile Systems, 2023.
6. Patel, D., & Mehta, A. *On-Demand Functionality in Enterprise iOS Apps*. Journal of Software Engineering, 2022.
7. Apple Developer Forums. *Best Practices for On-Demand Resources*. Apple Inc., 2025.
8. Sharma, V., & Rao, K. *User Engagement in Modular Mobile Applications*. International Journal of HCI, 2023.

9. Kumar, A., & Verma, S. *SwiftUI and Dynamic Module Loading*. iOS Developer Conference Proceedings, 2024.
10. Android Developers. *Instant Apps Architecture*. Google, 2022. (*For comparative context*)