Cross-Platform Augmented Reality Asset Management via Hybrid Cloud Architectures

*Integrated Firebase Authentication and Supabase Storage Systems

Fuzail Ahmed
Reg No: 01-134231-025

Muhammad Ali
Reg No: 01-134231-078

Abstract—As Augmented Reality (AR) applications transition from static experiences to dynamic, content-driven platforms, the necessity for robust asset delivery and secure user management grows. This paper presents our development of "AR-Asset-Explorer," a Flutter-based application that integrates Firebase for identity management and Supabase for high-performance 3D asset storage. We detail our implementation of a custom AR model manager that validates GLB file integrity, manages local caching, and handles complex AR transformations including rotation accumulation and deferred scaling. Our findings suggest that decoupling authentication from asset delivery optimizes both security and rendering performance.

Index Terms—Augmented Reality, Flutter, Firebase, Supabase, GLB, ARCore, Cloud Storage, Mobile Computing.

I. Introduction

Augmented Reality (AR) on mobile devices requires high-fidelity 3D assets that are often too large to be bundled within the application binary. This necessitates a cloud-based delivery system. However, remote fetching introduces challenges regarding file validation, network latency, and session security. In this report, we outline our solution utilizing a hybrid cloud approach to decouple authentication from asset storage. By leveraging Firebase for identity and Supabase for storage, we minimize architectural bottlenecks and provide a seamless end-user experience.

II. System Architecture

We divided the system architecture into three primary layers to ensure modularity and scalability: the Authentication Layer, the Data Layer, and the Visualization Layer.

A. Authentication Layer

Firebase Authentication serves as our security backbone. Utilizing the firebase_auth package, we implemented secure sign-in and registration flows. This ensures that only authorized users can access the model library and initiate AR sessions, protecting both the user data and the cloud assets.

B. Data and Storage Layer

We utilized Supabase as our primary storage engine. Unlike traditional databases, Supabase Storage allowed us to manage large binary objects (BLOBs) with granular Row-Level Security (RLS).

- Model Management: We stored all 3D assets in the GLB (Binary glTF) format. This format is superior for mobile AR as it packs textures and geometry into a single binary file.
- Thumbnails: To optimize the user interface, we established a naming convention that maps basename.glb to basename.png. This allows the UI to fetch lightweight previews before the user commits to a large model download.

III. Technical Implementation

A. AR Model Manager and Validation

To prevent application instability during model injection, we engineered a custom service layer to perform preload validation.

$$V(f) = \begin{cases} 1 & \text{if Header} = \text{'glTF'} \wedge \text{JSON Chunk valid} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Our ARModelManager service utilizes flutter_cache_manager to ensure that assets are cached upon first download. If a file is corrupted or lacks the 'glTF' magic header, the system automatically clears the cache and alerts the user, preventing a Null Pointer Exception in the ARCore renderer.

B. Geometric Transformations and Logic

We leveraged the arcore_flutter_plugin for the visualization engine. A significant challenge in mobile AR is handling user input without jitter. We implemented:

- Rotation Accumulation: Instead of applying raw touch delta values directly, we accumulate angles $\theta_{total} = \sum \Delta\theta_{input}$ before updating the node's local transform matrix.
- Deferred Scaling: To maintain high frame rates (60 FPS), we deferred expensive mesh scaling operations. We apply a visual proxy during the gesture and finalize the high-resolution scale only after the user's fingers leave the screen.

C. State Persistence and Scene Recovery

We implemented a refresh mechanism to rebuild the scene graph from a local _nodeStates map. If the ARCore session encounters a tracking loss, the user can tap "Refresh" to reinstantiate the nodes at their last recorded coordinates without re-fetching data from Supabase.

IV. Experimental Results and Build Analysis

During our development and testing phases, we conducted performance audits on several Android devices.

- Performance: We achieved a consistent 60 FPS during model manipulation. Average download speed for a 5MB GLB model via Supabase was measured at 1.2 seconds on a standard 4G connection.
- Build Analysis: We observed resource conflicts in Android Release builds related to the lStar attribute within the ARCore library. We mitigated this by utilizing a Debug APK strategy which bypasses the aggressive resource shrinking that causes the API level mismatch.

V. Future Work

In future iterations, we aim to implement:

- Multi-User Synchronization: Using Supabase Realtime to allow two users to see the same AR object simultaneously.
- Occlusion Mapping: Implementing depth-sensing to allow virtual objects to be hidden behind real-world furniture.

VI. Conclusion

The AR-Asset-Explorer project demonstrates that a hybrid cloud approach effectively manages the complexities inherent in mobile AR. By separating user identity (Firebase) from high-volume data (Supabase), we achieved a scalable architecture that is both secure and performant.

References

1 Google Developers, "ARCore Overview and Device Compatibility," 2024.
2 Flutter Documentation, "Secure Authentication with Firebase," 2024.
3 Khronos Group, "glTF 2.0 Specification," 2023.
4 Supabase Docs, "Handling Large File Storage with RLS," 2024.