**Portfolio: Google Cloud Storage File Backup Application**

Michael Maye

Colorado State University Global

CSC475: Platform-Based Development

Dr. Muhammad A Khan

08/06/2023

**Portfolio: Google Cloud Storage File Backup Application**

The applications use of technologies can be split between the three main views. The Home view contains a file directory of the device’s external storage along with a status of each file to advise the user if it is backed up in the Google Cloud Storage bucket on the users Google account. Implementation of this would require a user permission request for read and write of external storage, along with an implementation of the Google Cloud Storage API to allow the application to verify if the file exists in the bucket and if it doesn’t exist allow the user to upload the file. The Download view contains a file directory of files within the bucket itself, regardless of if they are on the device or not. This view allows users to directly download a file from the bucket to the device’s external storage. This will also require read/write access to the device, as well as file download authorization from the Google Cloud Storage API. The Login view allows the user to login with their Google Account associated with the Google Cloud Storage Bucket they want to access. This view uses the Google Sign-In API to allow the user to select their Google account email and log in and give the application authorization to access the bucket requested. From these views there are three main technology requirements: Read/Write permissions, Download/Upload capabilities to Google Cloud Storage, and Google Sign-in capabilities.

**Read/Write Permission**

Home view gives users a file directory of their device’s external storage. Implementation of this requires proper access rights from the device or user before it can run queries or operations on files. According to the Android Security Model, this means this application requires explicit user permissions to read files on the device, which can be handled in different ways. From Android 11 onwards, ‘READ\_EXTERNAL\_STORAGE’ which allows access to all files has been deprecated and would require the use of the Storage Access Framework (SAF). SAF gives the user an interface from which to select a directory or specific files to grant the application read/write permissions to. This allows the user to specify what directory/file to work with and prevents over-allowing permissions to the application. For devices running Android 10 or below, this can be accomplished by requesting the ‘READ\_EXTERNAL\_STORAGE’ permission from the user, which allows the application rights to read files from the device’s external storage. However, it is still recommended to user SAF when capable as it offers increased security and improved user experience overall.

**File Download/Upload to GCP Storage**

In order to allow the application the capability of downloading/uploading files to a Google Cloud Storage bucket, it would need to implement the Google Cloud Storage Client Libraries and API. These libraries allow the application to interact with Google Cloud Storage efficiently and give functionality like creating, configuring, and managing storage buckets, as well as performing operations on the data held within them. In order to allow access to the users bucket, the library seamlessly integrates with the Google Sign-in API and verifies authorization and authentication, implementing proper security and access controls to the users cloud account. As a point of order on this, the upload and download of a file inherently requires network operations. These operations can be quite time-consuming depending on the size of the file, not to mention the consumption of device resources during heavy operations. Methods like AsyncTasks, Thread, Handler, or utilizations of the java.util.concurrent package can offload these large tasks from the main thread and allow for the freeing up of resources for the remainder of user actions. Utilizing these operation techniques ensures that the user’s action of upload or download completes in a timely fashion while simultaneously avoiding any UI/Application stuttering or throttling in order to execute the action.

In addition to these requirements, the application will require permission to write to the device’s external data. In a similar way to the Home view the application will need to implement the SAF for devices that are using android 11, and for devices that are running Android 10 or below it will need to use ‘WRITE\_EXTERNAL\_STORAGE’.

**Google Sign-in authentication**

The Login view allows the user to log into their Google account and handles the request for the authentication to allow for access to the users Google Cloud Storage system. The application uses the GoogleSignIn libraries, specifically the GoogleSignInClient class, to perform this action. The class is created along with a GoogleSignInOptions object that would be configured to request the OAuth 2.0 access scop for Google Cloud Storage, typically by using the ’https://www.googleapis.com/auth/devstorage.read\_write’ scope. When the user successfully logs in the application receives back a GoogleSignInAccount object which can be used to create a GoogleSignInCredential. This credential grants the application authorization to perform operations within the users Google Cloud Platform accounts storage system, and this authorization is passed on to the Google Cloud Storage libraries used in the other views.

**Conclusion**

To ensure efficient and effective functionality, this application requires the integration of multiple technologies and. The Home view utilizes the Android Security Model and Storage Access Framework for access to the device’s external storage in order display directories and files to the user to examine. Devices using Android 10 and blow, ‘READ\_EXTERNAL\_DATA’ is utilized. The Download view allows the user to view and download files held within their Google Cloud Storage bucket by using the Google Cloud Storage Libraries and authentication from the Google Sign-in API. This process is optimized through offloading tasks to non-UI threads with AsyncTasks or java.util.concurrent package methods. The Login view implements the GoogleSignIn library to log the user into their Google account and request the OAuth 2.0 scope to utilize the users Google Cloud Storage bucket. Upon logging in, the application can create a GoogleSignInCredential, allowing it access to perform operations for the user. The implementation of these technologies allows the application to efficiently carry out the user’s requests while maintaining security and authorization requirements in the background on behalf of the user.

**References**

*Better Performance Through Threading  :  App quality  :  android developers*. Android Developers. (n.d.-a). https://developer.android.com/topic/performance/threads

*Data and File Storage Overview  :  android developers*. Android Developers. (n.d.-b). https://developer.android.com/training/data-storage

Google. (n.d.-a). *Cloud Storage Client libraries  |  google cloud*. Google. https://cloud.google.com/storage/docs/reference/libraries

Google. (n.d.-b). *Start integrating google sign-in into your Android app  |  authentication  |  google for developers*. Google. https://developers.google.com/identity/sign-in/android/start-integrating

*Processes and threads overview  :  App quality  :  android developers*. Android Developers. (n.d.-c). https://developer.android.com/guide/components/processes-and-threads