Utilizing Firebase for Application Development

Garrett Hagen

Department of Computer Engineering

University of Pittsburgh Swanson School of Engineering  
Pittsburgh, USA

gbh8@pitt.edu

Abstract— For the development of user applications that require internet based datastore and authentication functionality, Google Firebase serves as a flexible, highly abstracted solution. This paper analyses the various services Firebase offers and their effectiveness in realizing a solution to common features required in today’s user applications.

Keywords— Firebase; Google; Database; Realtime; Authentication; Internet; Storage; JSON; iOS; Android; Web

1. Introduction

With the fast-paced and ever changing nature of application development, producing flexible, scalable software is an essential factor in lasting success. As of recent years, there has been a growing trend of developing consumer software as a web based application which consequently has created an increased demand on robust backend solutions.

While developing one’s own backend services for their intended application is a viable option, it requires significant up-front effort and frequent maintenance - both of which inhibit development teams from placing their focus on the functionality of the application at hand. For start-ups and teams with limited resources, Firebase makes integrating backend features quick and simple by providing developers with a plethora of tools and services to help them develop high-quality apps, grow their user base, and increase profit [1]. This prevents the need to build many of these common tools from scratch, allocating development resources to craft a great user experience.

Additionally, it has technologies that can be integrated into the application that will help its growth through referrals, linking, and more. It has an easy-to-use Advertising API that can be dropped into the application to start earning, and importantly, the whole platform is tied together by analytics [2]. Firebase services can be divided into three pillars: Develop, Grow and Earn as shown in Fig. 1.



Fig. 1 A chart illustrating the various services that Firebase offers and the pillars they are grouped under.

This paper will focus heavily on the Develop pillar, as its goal is to provide a comprehensive outline on leveraging these various tools and services for application development. It will be organized into four main sections: Authentication, Real Time Database, Storage, and Crash Reporting. Each of these main sections will contain a detailed overview and guidelines to integrating these features into the application. Finally, there will be a Conclusion section that ties everything together.

1. Authentication

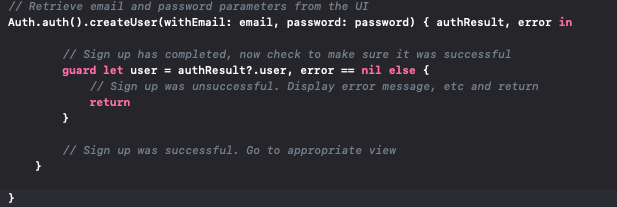
Writing a custom authentication system is a task that requires extensive development resources, as it must be extremely secure, handle all edge cases, conform to industry standards, and subsequently thoroughly tested. Firebase Authentication abstracts all of these implementation details through an API that supports OAuth 2.0 and provides both login and sign up services via (email/phone)/password, Google, Facebook, Twitter, GitHub, as well as others [3]. It integrates directly with the Real-Time Database service so data access can be restricted to authenticated users.

The state of a user can also be monitored by attaching a listener to the current user of the application, which allows authentication state to persist across application terminations and launches. This eliminates the repetitive task of requiring the user to log in every time the application is launched. Firebase Authentication also provides additional convenient services such as email/text verification, password resets, and user credential management – all of which can be implemented directly into the application.

1. User Sign Up

As mentioned previously, creating new Firebase users can be accomplished in a multitude of ways, but for the sake of simplicity we will look at the traditional approach of creating a user via email and password. The typical workflow involves collecting the email and password information from the user interface and then passing it into the code snippet displayed in Fig. 2.

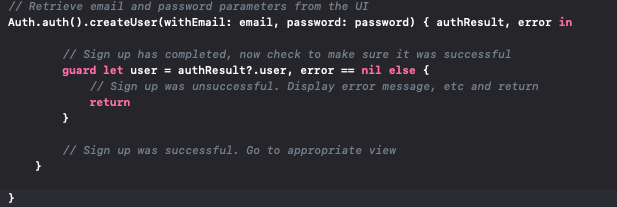
*Fig. 2 SignUpViewController.swift: A code snippet highlighting the user sign up procedure*



1. User Sign In

Users can be authenticated via any of the credentials that they specified during sign up, but again for the sake of simplicity we will use email and password verification. The workflow to sign in is nearly identical to that of sign up, the only difference being that a Firebase user object is returned in the callback instead of an authResult as illustrated in Fig. 3.

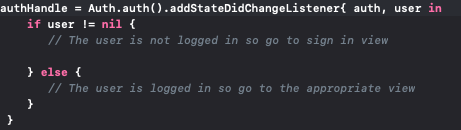
*Fig. 3 SignInViewController.swift: A code snippet highlighting the user sign in procedure*



1. Authentication State Persistence

Firebase listeners allow the application to be notified of any events that take place at a certain location. Using this feature, we can listen for any changes to the authentication state of the current user and then act accordingly. In iOS applications, the AppDelegate essentially serves as the highest level of logic for the application and executes whenever the application is launched. A simple approach shown in Fig. 4 is to create this listener in the AppDelegate, and then segue to the sign in view if the current user logs out/doesn’t exist, or segue to the initial view if the user exists and is logged in.

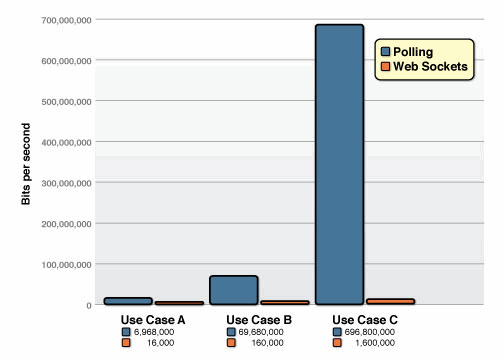
*Fig. 4 AppDelegate.swift: A code snippet depicting attaching a listener to perist authentication state across app launches/terminations*



1. Realtime Database

The Firebase Realtime Database is a cloud-hosted database. Data is stored as JSON and synchronized in realtime to every connected client. It supports cross-platform apps for iOS, Android, and JavaScript. All connected clients share one Realtime Database instance and automatically receive updates with the newest data [3].

Unlike traditional databases that require many individual HTTP calls to retrieve data, Firebase syncs data through a single WebSocket as fast as the client’s network can carry it [4]. As shown in Fig. 4, WebSockets provide a dramatic reduction in unnecessary network traffic compared to HTTP polling and thus, is much faster [5].



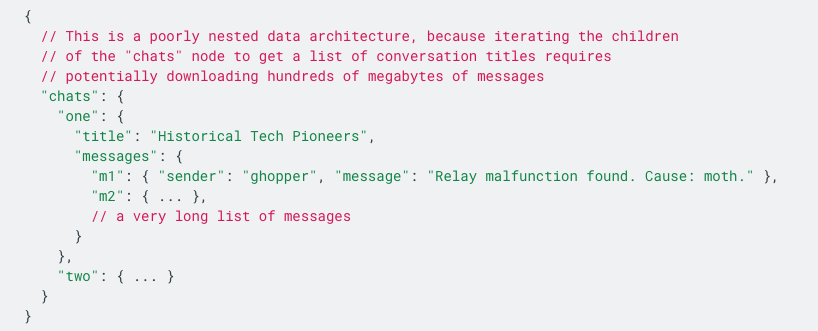
*Fig. 5 A chart illustrating unnecessary network throughput as a function of 1000, 10000, and 100000 clients receiving 1 message per second for HTTP Polling and WebSocket implementations.*

*A. Structuring Data*

Realtime Database is an effective solution for most applications, but it can quickly become costly with a growing client base, especially for applications that are data heavy in nature. Thus, it is extremely important that data is structured efficiently to minimize operating costs.

Firebase charges $0.18/GB for data stored, $0.18/100KB for document writes, $0.06/100KB for document reads, and $0.02/100KB for document deletes [3]. The best practices for storing data are as follows:

1. *Avoid Nesting Data:* When you fetch data at a location in your database, you also retrieve all of its child nodes. In addition, when you grant someone read or write access at a node in your database, you also grant them access to all data under that node. Therefore, in practice, it's best to keep your data structure as flat as possible [3]. As illustrated in Fig. 6, every time a chat node is accessed, the entire list of messages will be downloaded with it. This is would be problematic and inefficient in the event that the application simply wanted to iterate through all of the chats and display the title of each. *Fig 6. An example of a poorly structured, nested data structure for a chat application*



1. *Denormalization:* By splitting the data into separate paths, we can solve the above problem and access the chat components separately. This is illustrated in Fig. 7.



*Fig 7. An example of properly structured, denormalized data structure for*

*a chat application.*

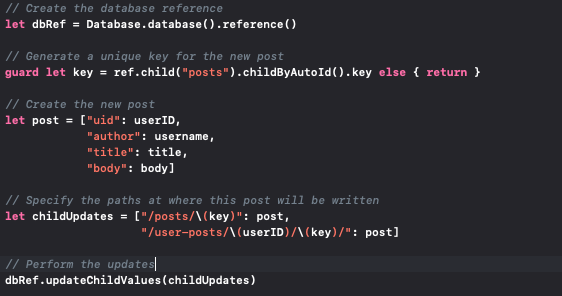
*B. Writing Data*

Firebase data is written as a dictionary structure by attaching a DatabaseReference instance to a certain node and then performing a write operation. A setValue write operation will replace any existing data, including child nodes, with the new data while an updateChildValues operation will only update the specified children of the node.

As shown in Fig. 8, if data is stored in multiple locations, a list of updates can be created at different locations and then applied simultaneously.

Data can also be saved as a transaction which is useful for avoiding potential data corruption that results from a race hazard condition such as the number of likes on a post being updated at the same time by two separate users.

*Fig 8. CreateNewPostViewController.swift: An example of writing a post to multiple locations in a social media application*



C. Reading Data

Firebase utilizes **listeners** to watch for changes in a DatabaseReference instance that is attached at a specified node. Using the observe operation, whenever changes in that node's data occur, the listener automatically provides the application updated data, called a **snapshot**. The application can then use information from the snapshot to update the user interface [6].

As shown in Fig. 9, it is considered a best practice to attach listeners when the view requesting the data appears and then remove them when the view disappears. This prevents unnecessary data retrieval for a view that is not currently visible, again minimizing operating costs.



*Fig 9. PostFeedViewController.swift: An example of correctly attaching and detaching listeners to update a view with the most recent posts*

Queries can also be used to filter and/or sort the data that is returned from Firebase. These are especially useful when working with lists of data, as they not only free the client of doing the work, but also can minimize costs by filtering out unnecessary data.

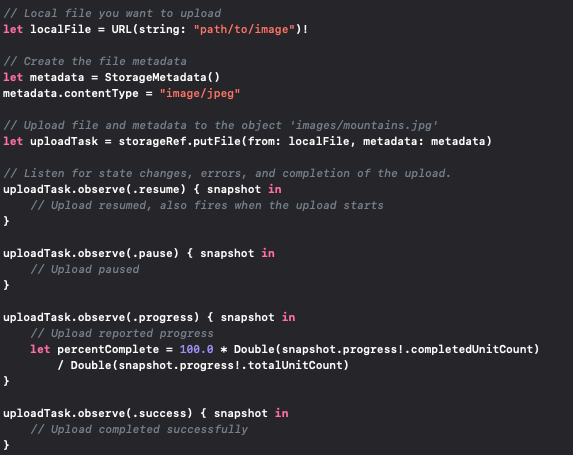
1. *Storage*

Cloud Storage for Firebase is a simple and cost-effective object storage service built for scale. The Firebase SDKs for Cloud Storage add Google security to file uploads and downloads for Firebase apps, regardless of network quality. The SDKs provide functionality to store and retrieve images, audio, video, or other user-generated content [3].

Data operations are very similar to those of the Realtime Database, the only significant difference being the type of data involved, and the absence of real time updates.

* 1. *Uploading Files*

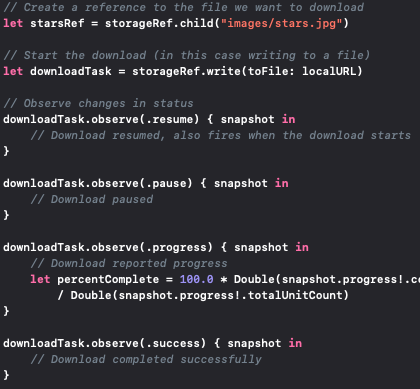
Like the Realtime Database, uploading a file involves creating a reference to a specified location in storage, but offers more refined control over tasks by adding the ability to pause and resume uploads as well as monitor their progress. Metadata such as content descriptions, last time updated, whether the data should be cached on the client’s device, etc. is included as well. Fig. 10 shows the complete implementation for uploading an image to storage complete with task management and progress monitoring.



*Fig 10. ImageUploadViewController.swift: An example of uploading an image with task management and progress monitoring.*

* 1. *Downloading Files*

Storage files can be downloaded to a local file on the client’s device or directly into memory. As shown in Fig. 11, the process for doing so is very similar to uploading files, in the sense that tasks can be managed and progress can be monitored in the same fashion.



Reference

1. Geeky Ants, “An Introduction to Firebase,” *Hackernoon*, Dec. 2017.
2. L. Moroney, “An Introduction to Firebase,” *The Definitive Guide to Firebase*, pp. 1–24, 2017.
3. Google Inc., Ed., “https://firebase.google.com/docs/,” *Firebase Documentation*, Jun. 2016.
4. Chris Espin, “What is Firebase?,” *How to Firebase*, Oct. 2016.
5. P. L. Kaazing Corporation and F. G. Kaazing Corporation, “HTML5 WebSocket: A Quantum Leap in Scalability for the Web.”
6. E. Inc., “Firebase: Reading Data and Event Listeners,” *Data Persistence*, Mar. 2019.

Fig 11. ImageDownloadViewController.swift: An example of downloadng an image with task management and progress monitoring.

1. *Conclusion*

This paper realizes the benefits and methods of using Firebase as an all-in-one tool to drastically reduce the effort needed to get an application from initial design to launch. There are many additional tools built around it such as FirebaseUI for iOS, Android, and Web Applications, that implement some of the common procedures discussed in this paper, reducing development time even further.

By utilizing the constantly expanding tool set provided by Firebase, the barrier to entry for application development is reduced significantly, allowing developers to focus on building a higher quality product in a shorter amount of time.