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 how to leverage 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.

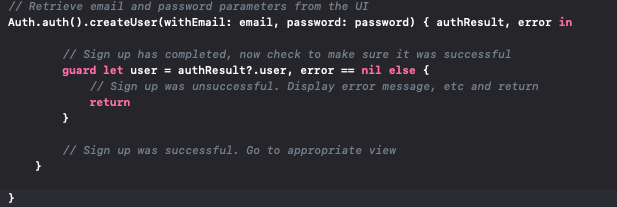


Fig. 2 SignUpViewController.swift: A code snippet highlighting the user sign up procedure

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.

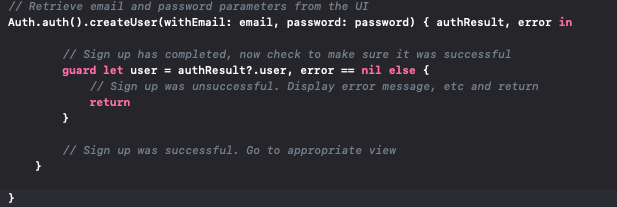
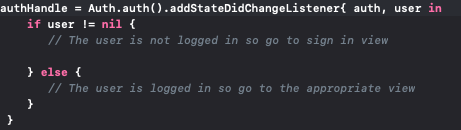


Fig. 3 SignInViewController.swift: A code snippet highlighting the user sign in procedure

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.

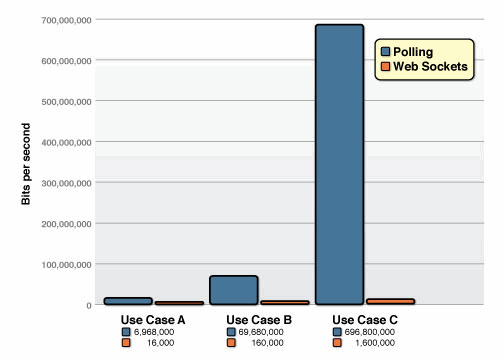


*Fig. 4 AppDelegate.swift: A code snippet depicting attaching a listener to perist authentication state across app launches/terminations*

1. Real Time 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].

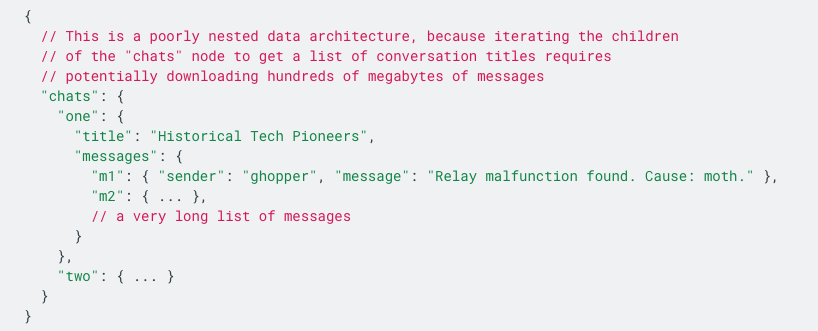


*Fig. 5 A chart illustrating unnecessary network throughput as a function of 1000, 10000, and 100000 clients receiving 1 message per second for HTTP Polling and WebSocket implementations.*

*A. Structuring Data*

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], so naturally it is in the developers’ best interest to structure data as efficiently as possible to minimize operating costs. The best practices for doing so go 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.*

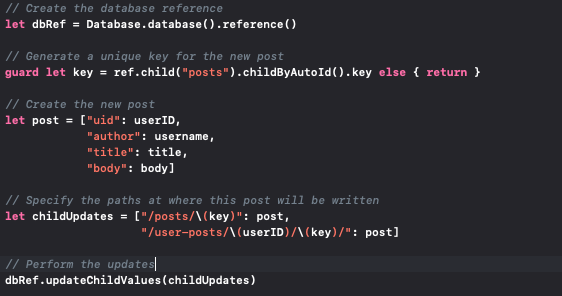
*B. Writing Data*

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

Your paper must use a page size corresponding to A4 which is 210mm (8.27") wide and 297mm (11.69") long. The margins must be set as follows:

* Top = 19mm (0.75")
* Bottom = 43mm (1.69")
* Left = Right = 14.32mm (0.56")

Your paper must be in two column format with a space of 4.22mm (0.17") between columns.

1. Page Style

All paragraphs must be indented. All paragraphs must be justified, i.e. both left-justified and right-justified.

1. Text Font of Entire Document

The entire document should be in Times New Roman or Times font. Type 3 fonts must not be used. Other font types may be used if needed for special purposes.

Recommended font sizes are shown in Table 1.

1. Title and Author Details

Title must be in 24 pt Regular font. Author name must be in 11 pt Regular font. Author affiliation must be in 10 pt Italic. Email address must be in 9 pt Courier Regular font.

TABLE I  
Font Sizes for Papers

|  |  |  |  |
| --- | --- | --- | --- |
| Font Size | Appearance (in Time New Roman or Times) | | |
| Regular | Bold | Italic |
| 8 | table caption (in Small Caps),  figure caption,  reference item |  | reference item (partial) |
| 9 | author email address (in Courier),  cell in a table | abstract body | abstract heading (also in Bold) |
| 10 | level-1 heading (in Small Caps),  paragraph |  | level-2 heading,  level-3 heading,  author affiliation |
| 11 | author name |  |  |
| 24 | title |  |  |

All title and author details must be in single-column format and must be centered.

Every word in a title must be capitalized except for short minor words such as “a”, “an”, “and”, “as”, “at”, “by”, “for”, “from”, “if”, “in”, “into”, “on”, “or”, “of”, “the”, “to”, “with”.

Author details must not show any professional title (e.g. Managing Director), any academic title (e.g. Dr.) or any membership of any professional organization (e.g. Senior Member IEEE).

To avoid confusion, the family name must be written as the last part of each author name (e.g. John A.K. Smith).

Each affiliation must include, at the very least, the name of the company and the name of the country where the author is based (e.g. Causal Productions Pty Ltd, Australia).

Email address is compulsory for the corresponding author.

1. Section Headings

No more than 3 levels of headings should be used. All headings must be in 10pt font. Every word in a heading must be capitalized except for short minor words as listed in Section III-B.

1. Level-1 Heading: A level-1 heading must be in Small Caps, centered and numbered using uppercase Roman numerals. For example, see heading “III. Page Style” of this document. The two level-1 headings which must not be numbered are “Acknowledgment” and “References”.
2. Level-2 Heading: A level-2 heading must be in Italic, left-justified and numbered using an uppercase alphabetic letter followed by a period. For example, see heading “C. Section Headings” above.
3. Level-3 Heading: A level-3 heading must be indented, in Italic and numbered with an Arabic numeral followed by a right parenthesis. The level-3 heading must end with a colon. The body of the level-3 section immediately follows the level-3 heading in the same paragraph. For example, this paragraph begins with a level-3 heading.
4. Figures and Tables

Figures and tables must be centered in the column. Large figures and tables may span across both columns. Any table or figure that takes up more than 1 column width must be positioned either at the top or at the bottom of the page.

Graphics may be full color. All colors will be retained on the CDROM. Graphics must not use stipple fill patterns because they may not be reproduced properly. Please use only *SOLID FILL* colors which contrast well both on screen and on a black-and-white hardcopy, as shown in Fig. 1.



Fig. 1 A sample line graph using colors which contrast well both on screen and on a black-and-white hardcopy

Fig. 2 shows an example of a low-resolution image which would not be acceptable, whereas Fig. 3 shows an example of an image with adequate resolution. Check that the resolution is adequate to reveal the important detail in the figure.

Please check all figures in your paper both on screen and on a black-and-white hardcopy. When you check your paper on a black-and-white hardcopy, please ensure that:

* the colors used in each figure contrast well,
* the image used in each figure is clear,
* all text labels in each figure are legible.

1. Figure Captions

Figures must be numbered using Arabic numerals. Figure captions must be in 8 pt Regular font. Captions of a single line (e.g. Fig. 2) must be centered whereas multi-line captions must be justified (e.g. Fig. 1). Captions with figure numbers must be placed after their associated figures, as shown in Fig. 1.



Fig. 2 Example of an unacceptable low-resolution image



Fig. 3 Example of an image with acceptable resolution

1. Table Captions

Tables must be numbered using uppercase Roman numerals. Table captions must be centred and in 8 pt Regular font with Small Caps. Every word in a table caption must be capitalized except for short minor words as listed in Section III-B. Captions with table numbers must be placed before their associated tables, as shown in Table 1.

1. Page Numbers, Headers and Footers

Page numbers, headers and footers must not be used.

1. Links and Bookmarks

All hypertext links and section bookmarks will be removed from papers during the processing of papers for publication. If you need to refer to an Internet email address or URL in your paper, you must type out the address or URL fully in Regular font.

1. References

The heading of the References section must not be numbered. All reference items must be in 8 pt font. Please use Regular and Italic styles to distinguish different fields as shown in the References section. Number the reference items consecutively in square brackets (e.g. [1]).

When referring to a reference item, please simply use the reference number, as in [2]. Do not use “Ref. [3]” or “Reference [3]” except at the beginning of a sentence, e.g. “Reference [3] shows …”. Multiple references are each numbered with separate brackets (e.g. [2], [3], [4]–[6]).

Examples of reference items of different categories shown in the References section include:

* example of a book in [1]
* example of a book in a series in [2]
* example of a journal article in [3]
* example of a conference paper in [4]
* example of a patent in [5]
* example of a website in [6]
* example of a web page in [7]
* example of a databook as a manual in [8]
* example of a datasheet in [9]
* example of a master’s thesis in [10]
* example of a technical report in [11]
* example of a standard in [12]

1. Conclusions

The version of this template is V2. Most of the formatting instructions in this document have been compiled by Causal Productions from the IEEE LaTeX style files. Causal Productions offers both A4 templates and US Letter templates for LaTeX and Microsoft Word. The LaTeX templates depend on the official IEEEtran.cls and IEEEtran.bst files, whereas the Microsoft Word templates are self-contained. Causal Productions has used its best efforts to ensure that the templates have the same appearance.

Causal Productions permits the distribution and revision of these templates on the condition that Causal Productions is credited in the revised template as follows: “original version of this template was provided by courtesy of Causal Productions (www.causalproductions.com)”.

Acknowledgment

The heading of the Acknowledgment section and the References section must not be numbered.

Causal Productions wishes to acknowledge Michael Shell and other contributors for developing and maintaining the IEEE LaTeX style files which have been used in the preparation of this template. To see the list of contributors, please refer to the top of file IEEETran.cls in the IEEE LaTeX distribution.

References

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.