Architecture solution for signing in through authentication providers

Mentorship System

by Bartek Pacia

Google Code-in 2019

task link

This document contains hyperlinks – it is intended to be read on an electronic device with Internet access. Do not print.

In this document, I will present my proposal for the implementation of Single Sign-On functionality in Systers Mentorship System.

Over the past few years, **Single Sign On** (SSO) has become very popular. It is a feature that enables users to securely authenticate with multiple web, mobile and desktop apps using a single set of credentials – usually, the ones used to login to tech giants accounts like Google and Facebook.
From the user's point of view, the whole sign up and sign in process SSO enables is incredibly convenient – it effectively comes down to tapping "Sign in with…". The rest is handled by the app developer, which (again) delegates most of the security-related stuff to an authentication provider, like Google.
This approach has many advantages, both to users and developers, that will be discussed later in this document.

Overview of the current auth system

Currently, authentication in the Mentorship System is handled in a very traditional, so to say, way – using email and password. Then those emails and hashed passwords are stored on our server hosted on Amazon Elastic Beanstalk. There is also an essential feature that definitely needs to be added - password reset.
As a result, we have full control over the whole process, from the dotted text field where the user enters the password to the implementation of the algorithm hashing users' passwords on the server. But do we need that full control? Or maybe it is that the problems it creates outweigh the flexibility it provides?
I think that the answer is no and I have good reasons to prove it:

- We must create a secure system from the ground up and protect it from breaches
- We have to manage users credentials securely
- We have to implement features like password reset manually

- Users have to remember yet another credentials (making a bit naive assumption that people do take security seriously and do not use the same password everywhere)

Wouldn't it be better to give away all those responsibilities to an entity that's generally considered as trustworthy and about which we are sure that will take care of credentials?
This leads us to Single Sign-On. It solves all of the aforementioned problems

- A secure system already exists and is battle-hardened. Credentials are securely stored.*
- The programmer only has to create the required UI.
- Users don't forget the credentials – at the end of the day, almost everyone remembers their Facebook password.

  * Of course, hacking into Google servers cannot be ruled out, but it has never happened to date. Google claims that the credentials are securely stored in its world-class system and if we want to use their services, we have to trust them.

Introducing Firebase

Firebase is the service that I am depending on in my implementation proposal, so I want you to have a general idea of what it is and what it offers.

Firebase is Google's mobile-first platform which is powered by Google Cloud Platform infrastructure. It offers developers a truckload of different services, such as databases (with the possibility to add triggers to them), cloud notifications, quality assurance, and testing utilities. It also provides detailed analytics data about users and their devices and authentication through many auth providers.
Most of Firebase services cost – a little or a lot, depending on how big the load put on them is. Fortunately, Firebase Authentication, which interests us the most, is completely free, and the quota limits of Firebase Cloud Functions more than satisfy our current needs. If you would like to see the full pricing, it is available at firebase.google.com/pricing.
Firebase Authentication is a very convenient end-to-end solution, which makes it possible to set up an entire secure authentication system in a matter of minutes.
It supports authenticating using auth providers and also a good old email + password combo. What is more, if one user logs in using Google and then decides to use email and password, it will be detected and the 2 accounts will be seamlessly merged. I think it is noteworthy Firebase Auth can handle even verification of the user's email. It would free us from managing the code responsible for that and let us remove it, which is a good thing.

There is also a package of libraries to make building UI login and signup flows fast and easy. It is called FirebaseUI and is available for Android, iOS, and the Web.

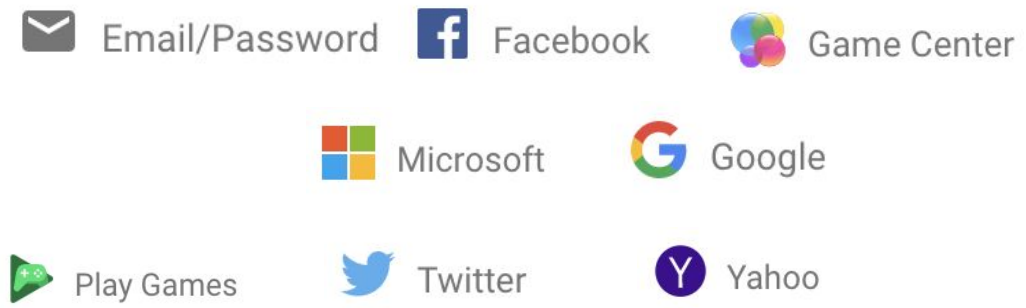

Figure 1. Some of the authentication providers that Firebase Auth supports

Firebase Cloud Functions is a service that allows executing JavaScript code in response to many events, like a change in Firebase Database, a new user being registered in Firebase Auth or an incoming HTTP request.

Solution using Firebase Authentication with Cloud Functions

You probably see where it goes. I would like to move the whole authentication system from our server to Firebase.

Firstly, I would like to dispel your fears and say that Firebase allows importing users from an existing database, with users' passwords hashed using the PKBDF2 SHA-256 algorithm, which is used in the current system. It will make that hypothetical transition much easier.

Let's talk about how it would work.

In the beginning, we could take advantage of Single Sign-on by **Google**, **Facebook**, **Twitter,** and **password + email,** because these are the most popular auth providers. Adding it to our Android client would be a breeze.

The problem is that this way, our server would know nothing about new users being registered in Firebase because Firebase Auth client libraries communicate directly with Firebase Auth backend. We can't just completely resign from storing users' data in the SQLite database on our backend. The diagram below visualizes this.

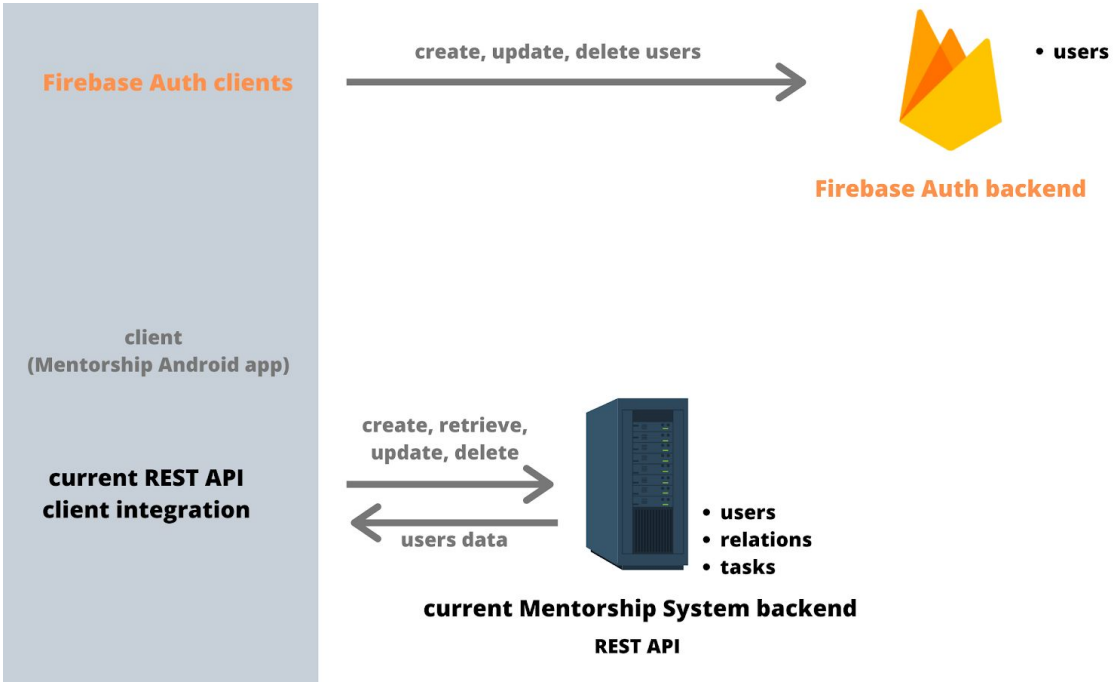

Figure 2. Firebase Auth integration without Cloud Functions (will not work)

Cloud Functions as the solution

As I wrote before, Cloud Functions allow for writing code that reacts to events happening in different Firebase services, including Authentication. Using CF, we could add triggers on Authentication whenever a user is created, updated or deleted and when such an event occurs, make an HTTP request to a certain (non-existent yet) endpoint in our API reflect the changes on the server.
Example: the user signs up in the app for the first time and selects to authenticate using their Google account.

- Firebase Auth client library handles everything related to creating a new account. Result: A new user is created in Firebase Auth.
- Cloud Function set up to listen for new user creation events fires up. It makes a `POST /users` request to Mentorship Backend with request body containing all new user's data. Result: the creation of a new user in the `users` table in the SQLite database. User's id would be assigned automatically, the user's name would be assigned from `displayName` property provided by Firebase.
  The problem arises with the `username` property of the user on the backend. Firebase Auth cannot ask the user for that during the authentication flow, so it would be null.
  One possible solution that comes to my mind is to make the user enter the username in-app if it is null (= after registration). Also, as long as the username is null, that user would be considered as *not initialized* (an equivalent of current *not verified*), i.e they won't show up in the Members tab in the app.

This architecture is presented by the diagram below.

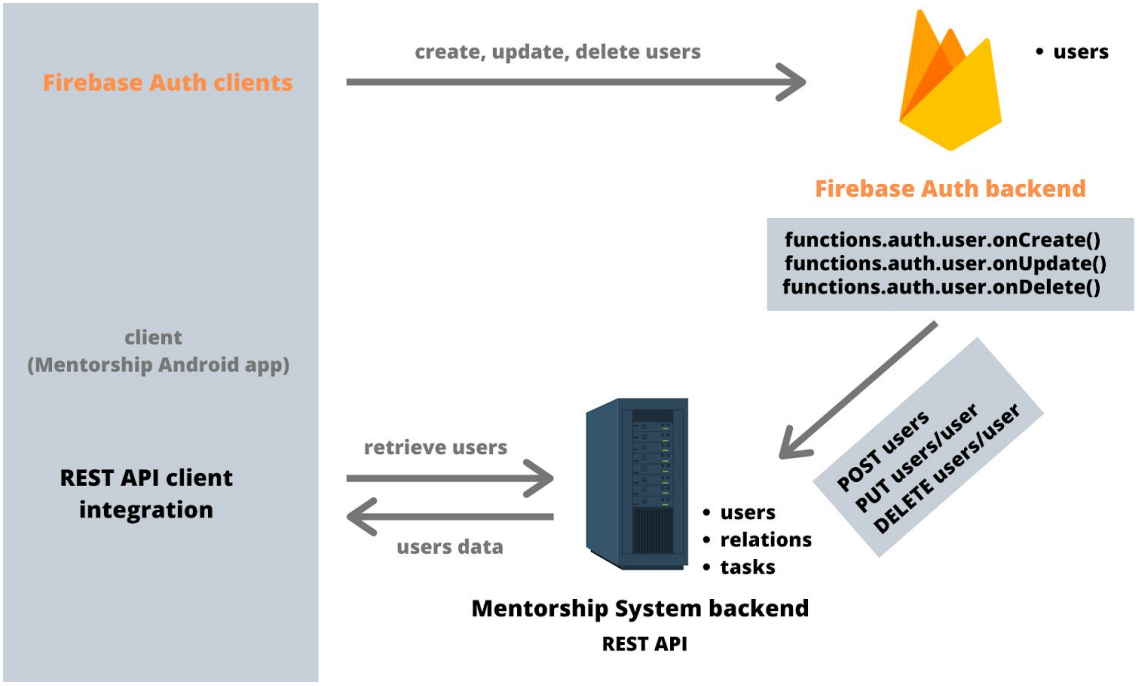

Figure 3. Firebase Auth integration using Cloud Functions to communicate with current Mentorship Backend.

Wrapping up

That's it for the integration with the backend. All in all, Firebase is an awesome platform that simplifies many complex things down to a few lines of code. I really don't see a reason why the Mentorship System would not take advantage of it.

Sharing content from social media

A part of this task is also to design a solution for sharing posts from social media. Example (given in the task description) was that if the user decided to connect their account with, for example, Facebook, and they shared a post on Facebook, the Mentorship System would know about that new post and would ask the user: "You recently shared this post. Do you also want to publish it in the Mentorship System?". Now, the question arises: *where* would the user share that post?

Mentorship System does not have a feature like a dashboard or a newsfeed where users could share such information. It is not part of the task to design it, but in order to accomplish the "posts sharing" functionality, it would be necessary. That's why I drafted a simple proposal for the dashboard feature. It is in the bottom part of this spreadsheet.

First of all, I want you to know that in one of my previous documents I already drafted the friends feature implementation, so I won't be repeating myself.

I would base the dashboard functionality on the friends functionality. Client apps would be able to send a simple POST /posts request to the backend…..

```
{
  "content": "Recently I discovered a ..."
}
```

...and that data would be put into the database.

posts

id [int]	publisher_id [int]	content [string]	sent_on [timestamp]
1	1	Recently I discovered...	15792…
2	1	I would like to add that...	15843…
3	2	This time last year, my ...	16241…

id and sent_on are assigned automatically by the backend, and publisher_id is retrieved from the authentication JWT.

Upon GET /posts, the backend returns *the list of posts from users who are friends of the user who made the request.*
It does so by getting all friendship relations of the user and getting ids of the users involved in that friendship. Then, the posts table is filtered to include only the posts whose publisher_id references a user who is a friend.
Then, the response is sent to the client.

Integration with social media

Now, I'd like to give a general concept of how I imagine sharing posts from other social media websites. General, because such a feature requires a tremendous amount of work. I will use Facebook as it's the biggest one and integration steps are similar to other social w
First, we, as system developers/administrators, need to register Systers on Facebook Developers. It will allow us to make requests to Facebook's Graph API – a system allowing for access to, among many others, to access Facebook user's public data.
Then we would ask users in the Mentorship App to enter a link to their Facebook accounts.
Now, on "new post" screen, there would be an option called "import from…", allowing to import posts from other social media websites (as text).
Contents in *import from* would be the list of posts user has published on Facebook - it would be fetched thought the Graph API.
It would be very convenient for user to just copy the contents of the post, and not to create a link to that post on Facebook, as it was suggested. What if Facebook

changes the API or the user deletes the post on Facebook? The link would become a *dead link.*

And that's it for the third part of this (hardI have to say) task. I described it rather generally, but I hope it's easy to understand.