Web Front-End Development
Week 7: Firebase Authentication

Many apps want to identify users. Knowing a user's identity lets you authenticate users, save custom data, and provide the same personalized experience across all of the user's devices.

Firebase Authentication
Firebase supports authentication using passwords, phone/SMS, Google, Facebook, Twitter, and Github. https://firebase.google.com/docs/auth/
To sign a user in:

1. Get authentication credentials from the user
2. Pass these credentials to Firebase Authentication
3. Firebase will verify the credentials and return a response to the client

Authentication
In the Firebase console for your project go into the Authentication section.
In the sign-in method tab enable the sign-in method you want to use.
When you enable Google it might require you to enter a project support email address.
Under Authorized domains add creative.colorado.edu as an authorized domain.
You should also have localhost as an authorized domain if you're using a local server.

Secure Data
https://firebase.google.com/docs/firestore/security/get-started?authuser=0
Firestore lets you define the security rules for the collections and documents in your database.

Structure security rules
https://firebase.google.com/docs/firestore/security/rules-structure?authuser=0
Firestore security rules always begin with the following declaration:

```
service cloud.firestore {
  match /databases/{database}/documents {
    // ...
  }
}
```

Basic rules consist of a `match` statement specifying a document path and an `allow` expression detailing when reading the specified data is allowed:
The `match /databases/{database}/documents` declaration specifies that rules should match any Cloud Firestore database in the project. Currently each project has only a single database named `(default)`.
For all documents in all collections:

```
match /{document=**}
```

All match statements should point to documents, not collections. A match statement can point to a specific document, as in `match /cities/SF` or use the wildcard {} to point to any document in the specified path, as in `match /cities/{city}`.

Writing conditions for rules

https://firebase.google.com/docs/firestore/security/rules-conditions?authuser=0
You can set up conditions for your security rules. A condition is a boolean expression that determines whether a particular operation should be allowed or denied. Use security rules for conditions that check user authentication, validate incoming data, or access other parts of your database.
Allow statements let you target your rules for read, write, delete, etc.
This rule allows authenticated users to read and write all documents in the cities collection:

```
service cloud.firestore {
  match /databases/{database}/documents {
    match /cities/{city} {
      allow read, write: if request.auth.uid != null;
    }
  }
}
```

When we set our database up in test mode it opened read and write access open to the public for all documents in our database.
Now that we're going to want to use authentication let's set up our security rules so a user must be authenticated to write to the database. We'll continue to allow public access to read from the database.
In the console in Firestore Database go into the Rules tab to change the security rules.

```
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read;
      allow write: if request.auth.uid != null;
    }
  }
}
```

Don't forget to publish!
You can test your Firestore security rules in the console. In the database rules tab there is a simulator you can use to test different types of actions on different parts of your database with different authentication rules.

Authentication with Google sign-in

To use authentication you have to add the firebase authentication library to your project.

```
<script src="https://www.gstatic.com/firebasejs/8.10.0/firebase-
auth.js"></script>
```

https://firebase.google.com/docs/auth/web/google-signin
To use Google authentication create an instance of the Google provider object:

```
let provider = new firebase.auth.GoogleAuthProvider();
```

To authenticate using Google you can either sign-in with a pop-up window or redirect them to a sign-in page. To sign-in with a pop-up window call

```
firebase.auth().signInWithPopup(provider).then(function(result)
{ })
```

To sign in by redirecting to the sign-in page call
`firebase.auth().getRedirectResult().then(function(result){ })`
The redirect is preferred on mobile devices.

You can also let the user sign-out by calling
`firebase.auth().signOut().then(function() { })`

Google also lets you handle the sign-in flow manually.

Manage Users
https://firebase.google.com/docs/auth/web/manage-users
After a user signs in for the first time, a new user account is created and linked to the credentials—that is, the username and password, or auth provider information—the user signed in with. This new account is stored as part of your Firebase project, and can be used to identify a user across every app in your project, regardless of how the user signs in.
The recommended way to get the current user is by setting an observer on the Auth object.

```
firebase.auth().onAuthStateChanged(function(user) {
  if (user) {
    // User is signed in.
  } else {
    // No user is signed in.
  }
});
```

You can get the currently signed-in user by using the currentUser property. This will give you access to their profile information which you can display, update, or use in your application.

```
let user = firebase.auth().currentUser;
```

Example:

- firebase_form_auth_cfs.html, firebase_tickets_auth_cfs.js
  - uses Google redirect sign-in method
- firebase_form_auth2.html, firebase_tickets_auth_popup_cfs.js
  - uses Google pop-up sign-in method