https://easymatch-53ed0.firebaseapp.com/

You can find the app's means of authentication within the site's login section. Authentication for both email and Google has been implemented. For the CRUD functions of the site within the code, you will find 2 create, 12 retrieve, 5 update, 2 delete. Of those, 1 create, 1 update and 4 reads are for images. The collective CRUD implementations are given in a high-level summary below.

Creation: when the user creates their account with the email option, or signs in with their Google account, Firebase will create that account within the "Authentication" section of the Firebase management console.

While signed into their account, the user can then create their profile. This is where the TopicDex functionality comes into play; users can configure their profile image (stored within Firebase Storage) and text-based information (stored in Firebase Database) such as gender, age, height, university, and major.

The initial message of a conversation creates that conversation within the inboxes of the sending and receiving users. This is explained on page-2.

Read: When the user visits the site, it will load into the gallery the profile images of the last 50 users that have most recently visited the site.

The search bar has some parameters that the user can set. The profiles matching those search parameters will be displayed within the gallery and will be sorted depending on the sorting method specified. Profiles without images will not appear in search results.

When the user opens their profile, or other people's profiles, it will show the profile's image along with the profile's information.

When the user opens their messages, it will show the profile images of the users that they are currently in contact with. Each profile image represents an ongoing conversation. When a conversation is opened, all messages within that conversation are retrieved and displayed.

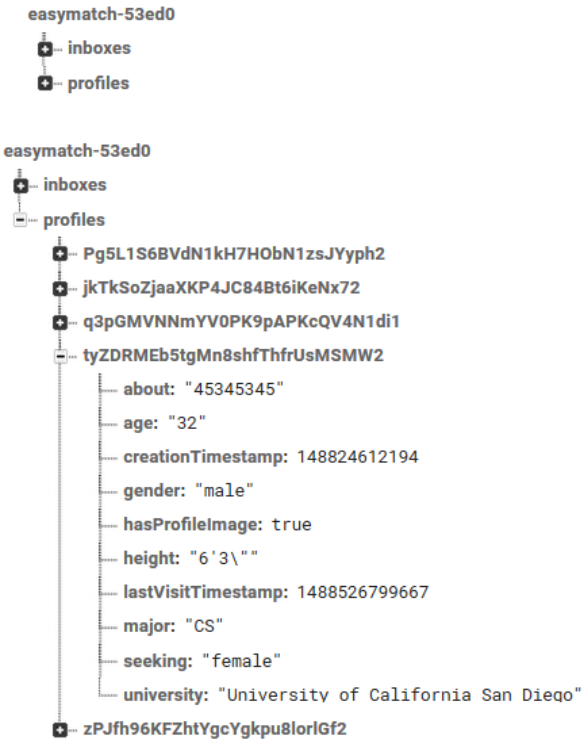
Update: When the user visits the site, their lastVisitTimestamp in the database is updated. This will place them within the top-50 list for an indeterminate amount of time.

Users can update their profile, which consists of one image and text-based information.

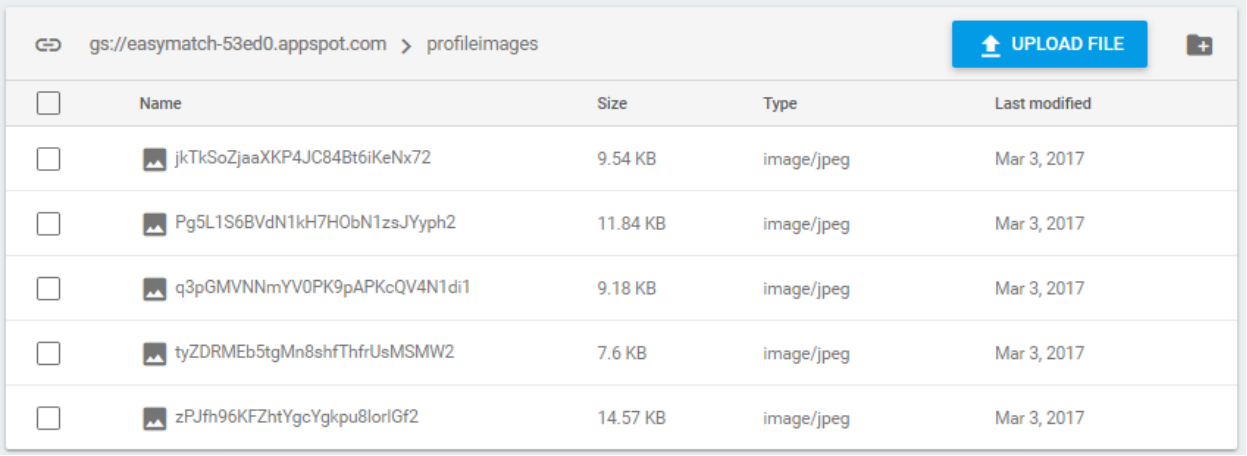
When users chat, messages are appended to that conversation within the inboxes of those users within the database. This is explained on page-2.

Delete: Users can remove conversations within their "messages" section. A copy of the conversation exists in each user's inbox, so either user removing the conversation from their inbox does not affect the copy of that conversation within the other user's inbox.

The database will contain an inbox and profile for each user. Inbox and profile keys are the Firebase UID of the user that owns them. A user's inbox maintains an index of conversations that the user is currently involved in, and the actual conversations. The key for each conversation is the Firebase UID of the user that this user is messaging, and the key of each message within a conversation is an epoch time stamp. A message object contains the message, timestamp, and Firebase UID of the sender of that message. Below is a screenshot of the database demonstrating how inboxes are structured. The screenshot also demonstrates how profiles are structured within the database; Profile keys are named based on the user's Firebase UID.



Each user's profile image is named based on their Firebase UID. When a new image is uploaded, the old image is simply overwritten by using that UID. Image dimensions and compression settings are intended to minimize file size for fast loading time without compromising image quality too much.



Challenges

The development of the database was a bit of a challenge; since the data is relational, a RDBMS would have been a more suitable means of storage, but Firebase does not offer such an option. Given that the Firebase Real-Time Database system is a No-SQL solution, an indexed design was used for structuring the data. The Firebase SDK provides limited querying capabilities which requires some client-side JavaScript logic for further operations on retrieved data before output can be shown to the user.

Images have a lot of dynamics to them: dimensions, orientation, EXIF data, and file size. In order to create a catch-all solution for profile images that will allow them to be universally compatible with all screen sizes, I implemented the following approach: profile images consist of a square that is center-cropped from the source image, the dimensions of which will be based on the smaller width or height dimension of the source image. EXIF data cannot be relied upon in giving the desired orientation of the image after cropping since the context in which the image was created cannot possibly be known, so a rotate button is provided that will allow users to orient their profile image to their liking. The extracted square images are resized to 250x250 pixels and compressed in "image/jpeg" format at 0.5 quality.

Initially, the JavaScript code was held within a script tag within the index.html file, but it grew to such size that it delayed the loading of the index.html page. I moved it to an external file so that the page will load quicker and the JavaScript will be loaded asynchronously.

Creating a responsive single-page design that promotes simplicity and whose primary focus is on visual media (images).

Optimizing image size so that loading times for search results are quick.