Team: HvZ App

Members: Santiago Rodriguez, Alex Hadley, Matthew Waddell, Kyra Clark

Project 2A.2 — Preliminary Architecture
GitHub: https://github.com/alexhad6/HvZ-app/blob/master/Administrative/Phase%202/architecture_2a.pdf
Primary author: Santiago

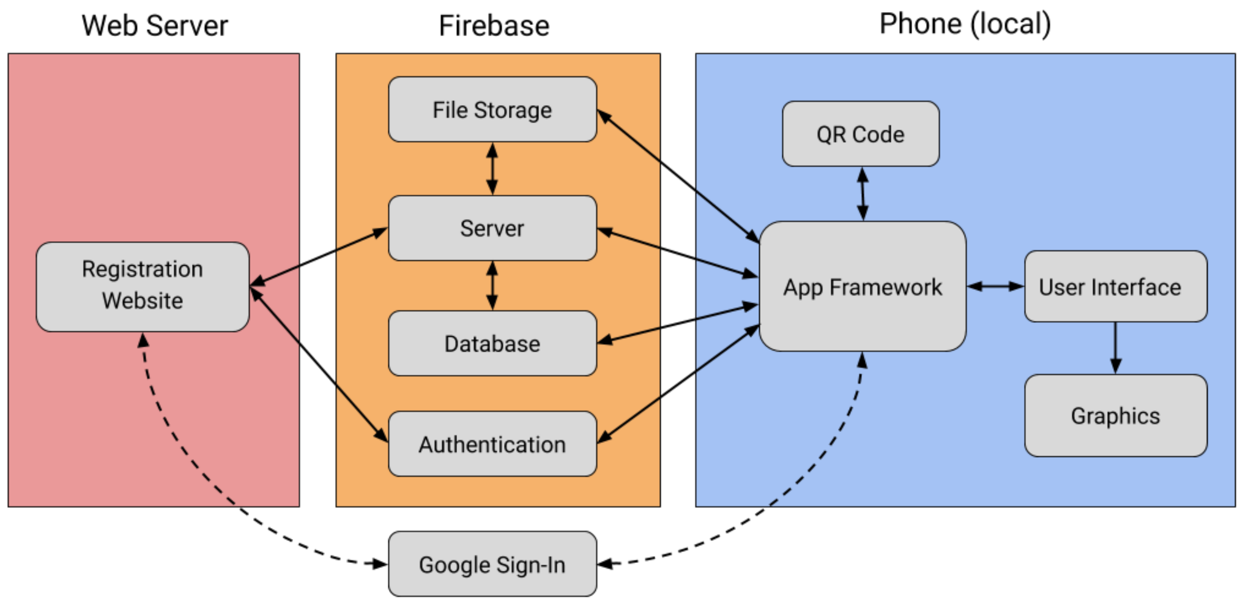
Slip days: 1 Kyra

HvZ App Architecture

Our app has three main domains in which the components fall. First, we have the app itself which is located on the phones of our users. Then we have our cloud products which are from the Firebase app development suite. Finally, we have our registration website which will lie on a separate web server.

Table of Contents

Component Diagram



Component Descriptions

App Framework

The Application Framework is the software used to implement the standard structure of the application software. It is the software which will create the app as in shown on the user's smartphone. It is the component that will function as a hub for all the other components to exist and connect with each other in a centralized place for the user. The app framework will also be the place for communication to occur between the users' device, the components of their device (such as notiviations, locations, internet, etc.) and the components within the app. Within the app, the framework will allow the user to access the QR Code scanner, and display the graphics and user interface. It will also connect to Firebase to allow the app to connect to the server, database, file storage, authentication, and Google authentication.

Creating a basic app framework is needed for the first implementation of this project in order to function as a hub for the remaining components of the app. We can test the individual connections of each component, but it is through the application framework that we can test how they all function together at the same time. It will also provide us with a prototype that can be understood and presented to users for basic initial testing.

User Interface

The user interface facilitates interaction between our users and the application. The user interface is responsible both for figuring out how to layout information from the app framework and for receiving input (primarily touch input) from the users.

Graphics

The graphics component actually renders what the user interface decides to display. This is the primary way that information will be communicated to the users.

QR Code

The QR code component is responsible for generating and scanning QR codes. This allows players to conveniently display and register feed codes, an integral part of the HvZ game.

Authentication

Authentication is responsible for verifying the identity of our users. Knowing who is using the app allows us to tailor the app experience to a particular user (e.g. by displaying their name and statistics) or to a particular subset of users (human players, zombie players, or moderators).

The authentication server will be able to read in user credentials and issue ID tokens that can be verified by other components to identify users. In particular, verification allows us to manage read and write permissions for our database and file storage system. For example, only moderators are allowed to alter mission information, and only zombies have permission to initiate brain transfers.

Google Sign-In

Google Sign-In (https://developers.google.com/identity) is a federated identity provider. When users log in to our app using their Google credentials, Google Sign-In can return to us an OAuth token which verifies their identity. This component is external since its operations are done on a Google server that we do not control.

Database

The database will be responsible for holding all of the relevant player data for each game. This includes player or moderator status, name, human or zombie status, number of brains (zombie currency), school, and other such information. Each player will have a unique ID so that they can be referenced when needed. The database also stores information for each game such as the number of players and which players are playing. It can store relationships among players such as Player A turned Player B into a zombie. Select information from the database will be accessible by other parts of the architecture.

File Storage

The file storage component will handle storing photos, videos, sound files, and other larger items. Moderators will be able to use the app to upload these files (for example, images relating to HvZ missions), and the files would then be downloaded and displayed to users on the app. The file storage system will be organized with folders for the active game and previous games.

Server

The server performs any computation that is not done locally on the app or registration website. For example, the server will communicate with the database and file storage to reorganize when a new game is initiated, and it will generate feed codes when players register.

Registration Website

The registration website allows players to register for the app or for an upcoming HvZ game. Users will be able to sign in or create an account (either with an email and password combination or Google Sign-In) from the registration website just like they can from the app. If they are creating a new account (they have not used the app before), they will be asked for additional information such as name, school, and class year. The registration website then communicates with the server to register the user for the upcoming HvZ game. This website would allow moderators to sign up users on the spot through a laptop computer at club fairs or in dining halls, the main way that users have signed up for the game in the past.

Component Connections

App Framework ⇹ User Interface

The app framework tells the user interface what information to display. A user interacts with the user interface to provide input to the app, and the user interface then tells the app framework what the user did. For mobile apps, the main form of input will be through a touch screen. So the user interface will process touch events and tell the app what the user is doing.

User Interface → Graphics

The user interface will tell the graphics renderer what to display (text, colors, shapes, images, etc.) and the layout of these components. The graphics renderer will then actually figure out how to display these items and will tell the phone what to display on the screen.

App Framework ⇹ QR Code

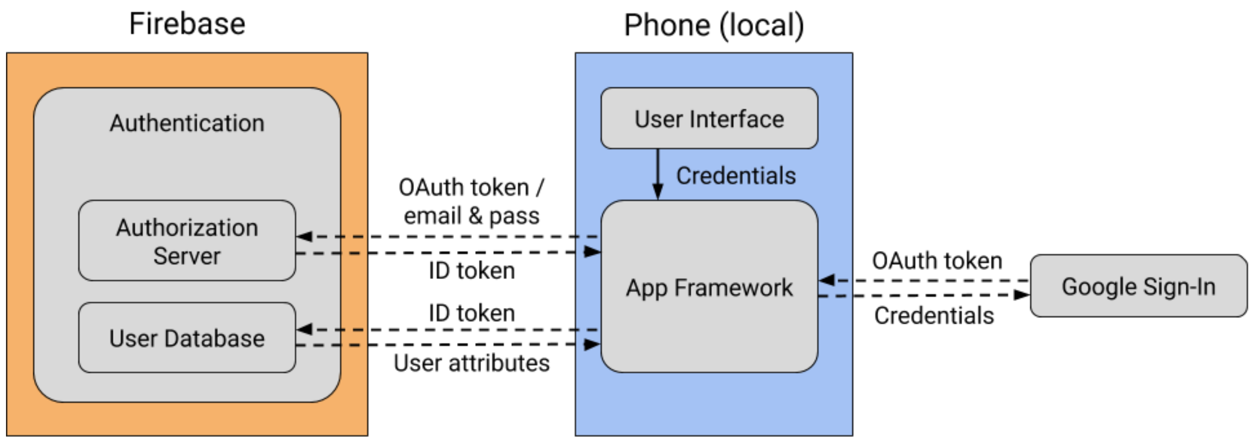
The QR code component serves two purposes: generating and scanning QR codes. For generating a QR code, the app framework will send the player's feed code to the QR code generator. The QR code generator will then render the feed code information as a QR code and send the result back to the app framework. For scanning a QR code, the app framework will send the QR code scanner an image of a QR code that the player has photographed. The QR code scanner will then process the image and send the feed code information back to the app.

App Framework ⇹ Google Sign-In

When logging in or signing up for the app, users will have the option to enter an email and password or to log in through Google Sign-In. If they log in through Google, their credentials are sent from the app to a Google server, and an OAuth token is returned to the app.

App Framework ⇹ Authentication

When a user logs in or signs up, either their email and password combination or the OAuth token from Google is sent to an authorization server within the authentication component. The authorization server verifies the login information and issues an ID token. This ID token is a signed JSON Web Token (https://jwt.io), or JWT. The ID token is sent back to the app. The app then communicates with the user database within the authentication component. Based on the ID token, the user database returns basic user attributes like email and name. The user is now logged in. The diagram below illustrates the authentication process.



App Framework ⇔ Database

The app framework sends a request to read from or write to the database (e.g. to view the number of humans or zombies or to update mission information, etc.), along with the user's ID, a signed JSON Web Token. The database can then verify the ID token to get the identity of the user requesting data. We will have permissions set up so that only certain users can perform particular operations. For example, only moderators can update mission information. The database will then send back the requested data or confirmation that data was changed.

App Framework ⇔ File Storage

The app framework will connect to file storage to request and display larger files. Moderators will be able to upload image or video files through the app and include them in mission or quest information. Only moderators will have permission to upload files to file storage, and just like the database these permissions will be managed by sending the moderator's ID token. Then when a user opens the mission page, the app framework will request to download these files from file storage. File storage will send the requested file, and it will be displayed in the user interface.

App Framework ⇔ Server

For basic database or file storage operations (such as requesting the current number of humans or downloading an image), the app framework will be able to access data directly from the relevant component. The server will be used for more complicated operations (e.g. brain transfer). The app framework will request to call a function on the server (and send the user's ID token when needed), and the server will complete the action, returning data that was requested or sending a confirmation that the action was completed.

Database ⇔ Server

The server will let the database know that a new game is being created (the database will then handle storing new game information in a different place than the old one). The server will also request the information of whether or not a player is already registered from the database. The database will tell the server that a player is registered, or it will tell the server that a player is not yet registered (and it will subsequently add the player to the database).

File Storage ⇔ Server

The server will let the file storage system know when a new game is being created. The file storage system will then move files for the current game to an archival folder and create a new folder for the new active game. The file storage system might need to communicate with the server if the files being uploaded need to be processed, for example if they need to be renamed, filed into a relevant subfolder, or if a reference to them needs to be added to the database.

Registration Website ⇔ Google Sign-In

A user can also log in or sign up for the app through the registration website. If they log in through Google, their credentials are sent from the registration website to a Google server, and an OAuth token is returned to the website.

Registration Website ⇔ Authentication

The process for logging in through the registration website is identical to the process for logging in through the app framework, outlined above. The registration website will send user credentials (Google OAuth token or email and password) to the authorization server, get back an ID token, send the ID token to the user database, and get back basic user attributes.

Registration Website ⇔ Server

After a player signs in to the registration website and is authenticated, they will fill in information such as class year and school. The registration website will then communicate with the server, calling a server function to register the user by adding this information to the database and generating a feed code. The server function will also send a confirmation email to the user with their generated feed code and additional game information. The server will return a confirmation to the registration website that registration was successful, and tell the registration website to sign the current user out so that someone else can register (when mods are registering people at club fairs or in dining halls).

Example Stories: Walking Through Components

Registering Through the Website

- A user or moderator navigates to the registration website URL.
- The user types in their login credentials (email and password or Google Sign-In).
  - If they used Google Sign-In, their credentials are sent to a Google server.
  - The Google server sends back an OAuth token.
- The website sends the email and password or the OAuth token to Firebase's authentication server.
- The authentication server verifies the credentials and returns a signed JSON Web Token with the user's ID.
- The website sends the user's ID token to Firebase's user database.
- The user database returns basic user attributes, such as their name.
- The registration website prompts the user for additional information, such as school and class year.
- The registration website sends the ID token and additional information to the server and calls a server function to register the user.
- The server verifies the ID token and identifies the user.
- The server adds the user's information to the database and marks them as registered in the upcoming game.
- The server generates a feed code and adds it to the database.
- The server sends a confirmation email to the user's email with game information.
- The server sends a confirmation to the registration website that registration is complete.
- The registration website tells the user that registration was successful.
- The registration website signs the user out and reloads for the next user to log in.

User Logs in to the App

- The user types in their login credentials (email and password or Google Sign-In) to the user interface.
- The user interface sends the login credentials to the app framework.
  - If they used Google Sign-In, their credentials are sent to a Google server.
  - The Google server sends back an OAuth token.
- The app sends the email and password or the OAuth token to Firebase's authentication server.
- The authentication server verifies the credentials and returns a signed JSON Web Token with the user's ID.
- The app sends the user's ID token to Firebase's user database.
- The user database returns basic user attributes, such as their name.
- The app sends the user's ID token to the database and requests more specific information, such as player or moderator status and human or zombie status.
- Based on this information, the app framework tells the user interface what to display. (There will be different layouts and options for humans, zombies, and mods.)

- The app tells the user interface to display a confirmation that the login was successful.
- The user can then interact with the app through the user interface.

Brain Transfer

- The user logs in to the app (see story above).
- Through the user interface, the user navigates to the brain transfer tab.
- User A (who has 2 brains) initiates a brain transfer of 2 brains to User B (who has 8 brains).
- The app framework calls the server function for brain transfer, sending the user's ID token, how many brains they want to transfer, and the user they want to transfer brains to.
- The server communicates with the database to check whether users A and B are both zombies.
- The server checks whether User A has sufficient brains to do the transfer.
    - If not, then the server tells the app that the transfer failed.
    - The app tells the user interface to say that there is an insufficient brain balance.
- If so, the server writes to the database and subtracts the given number of brains (2) from User A's balance (now 0) and adds the same number of brains to User B's balance (now 10).
- The server sends a confirmation of the transfer to the app, along with User A's new balance.
- The app tells the user interface to display a confirmation and User A's remaining balance.

Input Feed Code

- The zombie player first must access the application through the app framework.
- When logging in, the framework will connect to the authentication server, and return to the framework (see story above).
- After logging in, the application framework will display their particular user interface, which includes the QR scanner for zombie feedcodes. The UI displays information from the graphics.
- By selecting the particular component, via the app framework, the user can access the QR code scanner.
- The app framework will then communicate with the smartphone device to take a picture of the QR code.
- The app framework will send the image to the QR code scanner, which will then decode the image and return the feed code.
- Through the app framework, this code will be sent to the database and file storage to connect to a particular second player.
- When it is verified using the authentication token, the two players' information will be updated and will connect back to the app framework to update the player information.
- The player information will need to be updated to show that they consumed another human, and that a human is now turned into a zombie. The app framework will

communicate with graphics through the user interface to update and display the new information.

- For the newly turned zombie, the app framework will tell the user interface to change to reflect their transformation into a zombie.
- The app framework will also communicate with the database to update the global game statistics of all the players, which is done by connecting first to the app framework, then to UI and graphics.

Moderator starts a new game

- A moderator logs in to the app (see story above — process is the same for both players and moderators).
- The moderator interacts with the user interface to navigate to the game settings tab.
- The moderator taps the "new game" option.
- The app framework requests the moderator to log in again to confirm their identity (see story above).
- Through the user interface, the moderator inputs relevant information about the new game, such as its theme or its starting and ending dates.
- The moderator is prompted for a master admin password to confirm that they really want to reset the old game and start a new game.
- The app framework sends a request to the server to start a new game.
- The server resets player information (whether they are registered for the active game and human or zombie status) and increases an attribute for the number of previous games played.
- The server organizes files in the files storage into an archival folder and creates a new folder for the new game.
- The server resets all other information that might be in the database or file storage, such as mission or question information.
- The server sends a confirmation to the app that everything has been reset and that the new game was created.
- The app framework tells the user interface to display a confirmation that the new game was created.

Moderator updates information

- The mod will first have to open and access the app, using the application framework.
- When logging in, the framework will connect to the authentication server, and return to the framework.
- After logging in, the application framework will display their particular user interface, which includes more privileges and permissions. The UI displays information from the graphics.
- The mod will be navigating through the app framework, using the user interface, to update a piece of information.
- Along with the updated information, their authentication token will allow them to change information connecting to the file storage, and database.

- Through the database, the other players will be connected to receive the updated information.
- The new info will be displayed via the other players’ UIs (and graphics) and there app framework, via notifications, etc.

Feasibility

App Framework

We could use Flutter to create the application framework (https://flutter.dev). Flutter is a tool created by Google to assist developers in creating application frameworks. Because Flutter is made by Google, they offer many tools, resources, and even started code for us to understand and to later implement Flutter to the best of our abilities. In addition, Flutter is built to be compatible with both iOS and Android devices, allowing our application to be accessible to more students.

Flutter will be an important tool and resource in our development of this project because it also offers a clear connection to Firebase. Firebase is also made by Google, and therefore is built to be implemented and function together. They are very compatible resources, which should allow the implementation and connection of these two components to be feasible. In addition, Flutter offers strong resources and guidance to assist its developers in connecting Firebase and Flutter, which will allow for an even smoother process (https://firebase.flutter.dev). Because of these benefits, it is feasible for Flutter to allow us to connect to both Firebase, which houses the server, database, authentication, and file storage, as well as helped us easily create an elegant application framework.

UserInterface/Graphics

Charts in flutter_chart library will allow for the organized display of game stats. Flutter also self adapts to visuals to the given screen size and has its non-functional or non user interface components implemented by the LayoutBuilder class and functions managed by MediaQuery.of. Both components of user interface and graphics are organized as widget classes.

(https://flutter.dev/docs/development/ui/layout)

QR Code

Multiple plugins available for the QR reader. We can create a new flutter project and add this plugin with pubspec.yaml under the dependencies. Through the terminal we will download the necessary plugin component for our project, for example 'package:qr_flutter/qr_flutter.dart'. We then add the widget where the QR code is going to display. When adding a QR widget the only required property is data. There you must pass the data which need to be represented as a QR. Here I use some simple text. We can use whatever text we want for the feedcodes.

(https://medium.com/flutter-community/building-flutter-qr-code-generator-scanner-and-sharing-app-703e73b228d3)

Authentication

Authentication could be handled through Firebase (https://firebase.google.com/products/auth), a platform developed by Google for creating mobile and web applications. As the Firebase website states, “It can take months to set up your own auth system, and it requires an engineering team to maintain that system in the future.” On the other hand, Firebase allows you to “Set up the entire authentication system of your app in under 10 lines of code, even handling complex cases like account merging.” After looking through their website, it appears that this really is the case. Firebase is widely used for authentication (https://firebase.google.com/use-cases), and even provides a sign-in UI that can be easily added to any app. For example, it takes only around 10 lines of high-level code to add and customize FirebaseUI Auth in an iOS app (https://firebase.google.com/docs/auth/ios/firebaseui). FirebaseUI Auth is nice because it already handles things like creating an account for a new user, logging in through federated identity providers (like Google or Facebook), and displaying messages for incorrect passwords. However, it is also possible to further customize the login experience using the Firebase Authentication SDK.

Authentication through Firebase works as explained in the Component Connections section above. When a user signs in, Firebase authenticates their credentials (email and password or OAuth token), issues a Firebase ID Token (a signed JSON Web Token), retrieves basic user attributes from Firebase’s user database, and returns a user object (https://firebase.google.com/docs/auth/users). The user who signed in is then set as the “current user” in the Firebase Auth instance stored in the app framework. When the app framework requests information in the future from other Firebase services, such as Cloud Firestore or Cloud Storage, the ID token is sent with the request and can be verified in order to identify the user. The Firebase Auth instance persists the user’s state so that they are not signed out when the app is closed. When the user does sign out, the Auth instance stops keeping a reference to the user.

Google Sign-In

Google Sign-In (https://developers.google.com/identity) is a widely used identity provider. When the user sends their Google credentials, Google’s servers return an OAuth token that can be used to sign the user in. This is easy to implement using Firebase authentication.

Database

A database that fits our needs is 100% feasible. There are so many database services to choose from and so many different ways that they can be implemented (https://medium.com/@rwilliams_bv/intro-to-databases-for-people-who-dont-know-a-whole-lotabout-them-a64ae9af712). Oracle MySQL, Oracle NoSQL, and DynamoDB are just a few examples of database services ready to be used. We looked in particular at Cloud Firestore (https://firebase.google.com/docs/firestore). It is a document-based NoSQL database which allows us to store information in a simple way. Because our data needs are relatively simple, we don't need to worry about complex database operations. The service is a pay as needed type, and with our limited needs it will be free for a long while. Furthermore, it has libraries to directly interface with ios, android, and web applications. This allows us to consider building a website in the future with very little changes to our architecture. As a Firebase product, it interfaces easily with other Firebase products, such as Firebase Functions, which is the implementation of the server that we are considering. In summary, we plan to use an off-the-shelf product to implement our database, and we know that it will fit our needs and interface nicely with the other components of our architecture.

One problem with using a cloud database is that we don't have control of the servers it runs on. If they all went down for some reason, we would have to wait for the provider to fix them. However, because Cloud Firestore is run by Google, the database would probably be back up faster than if we had our own cloud database that crashed. Cloud Firestore also guarantees that our data is well backed-up.

File Storage

Similar to databases, there are a plethora of file storage systems out there. There are Microsoft Azure file storage, Panzura Global Cloud, and Amazon Cloud File Storage to name a few. We specifically looked at Cloud Storage for Firebase (https://firebase.google.com/docs/storage). It allows us to specify where in the storage we would like to store specific data, and it allows us to download it directly to ios, android, and web applications. It takes care of images, and should we need to expand to other file types in the future, it allows us to specify which type of file we are uploading. It integrates nicely with other Firebase products just as the database does, and the website has libraries to integrate it with other app frameworks should we choose to not use a Firebase product for that.

A problem that we might encounter is that only 5GB of data are allocated to our file storage system for free, so we may have to end up paying for the service at some point as HvZ uses lots of images and videos. The service is scalable so we have no worry of using too much space. Should the loss of the free nature of the service become a problem, we could fund it with club funding.

Server

Firebase allows an application to directly make requests to the database and file storage system. Their backend servers are able to verify the user's identity, and permissions can be managed through the online Firebase console (https://console.firebase.google.com). However, another Firebase service, Cloud Functions (https://firebase.google.com/products/functions), allows for computation outside of our app. Using this service is nice because it eliminates the need for us to find and maintain our own backend servers, which could cost money and be hard to maintain long term.

Firebase Cloud Functions is easy to integrate with other Firebase services, and can be triggered by events such as logging in through Firebase Authentication, or writing to the database. Functions can also be requested directly with an HTTP or HTTPS request from the app or the registration website. Functions are written in JavaScript and uploaded to a Firebase project through the Firebase CLI (command-line interface). They can then be called from an app or website using Firebase client SDK's, or they can listen for an event (e.g. write to database) and be triggered when the event occurs.

Registration Website

Firebase features, such as FirebaseUI Auth, can be added to websites just as easily as they can be added to applications (https://firebase.google.com/docs/auth/web/firebaseui). Since the registration website only needs to consist of static HTML, Javascript, and CSS files, we are confident that it will be easy to implement. While our team does not have very much app development experience, we do have prior experience creating simple web pages. The website could be hosted through Firebase Hosting (https://firebase.google.com/products/hosting). It could also be hosted through other free services, such as GitHub Pages (https://pages.github.com). Both of these services offer the option to add a custom domain name and HTTPS security. Buying a custom domain would cost money so we will not implement it at first, but perhaps the HvZ club would want this in the future.

Outstanding Issues

Locked Into Firebase

Most of our architecture fits the Firebase architectural model of an app. If we ever want to transition to something else, it may be difficult. However, our needs are relatively simple and seem like they will only incur minimal change over the years. And if our needs scale, so do the Firebase products (although it comes with a price tag). Firebase is backed by Google, so we have no worries of it being decommissioned any time soon.

Firebase Support for Flutter

However, in reference to integration with Flutter in particular, Google mentions that “Firebase supports frameworks like Flutter on a best-effort basis. These integrations are not covered by Firebase Support and may not have full feature parity with the official Firebase SDKs” (https://firebase.google.com/docs/flutter/setup).

Server vs. Local Computation

We are not completely sure which database operations will be done through a server function or directly between the app framework and the database. Flutter allows connection between the app framework and the database and can manage user permissions this way, so this sort of connection seems like a typical thing to do (https://firebase.google.com/products/firestore). However, for operations that require more computation (such as zombie brain transfer or feed code generation), we will have to decide what computation will be done locally on the app and what should be done through on an external server through Firebase’s “Cloud Functions” (https://firebase.google.com/products/functions).

Double Check That Everything is Free

From our research, it appears that everything is free. However, it is hard to know whether everything we need to do is covered by Firebase’s free plan. For example, Cloud Functions mentions that you need to “enable billing for your project” even for free use (https://firebase.google.com/docs/functions). (Although this may just be to use the Node.js 10 vs. 8 runtime.) This raises the question of what payment method we would use and how we would ensure that we do not go over the free plan’s limits and incur a charge.