Team: HvZ App

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

Project 2C.1 — Final Architecture Proposal

GitHub: https://github.com/alexhad6/HvZ-app/blob/master/Administrative/Phase%202/architecture_2c.pdf

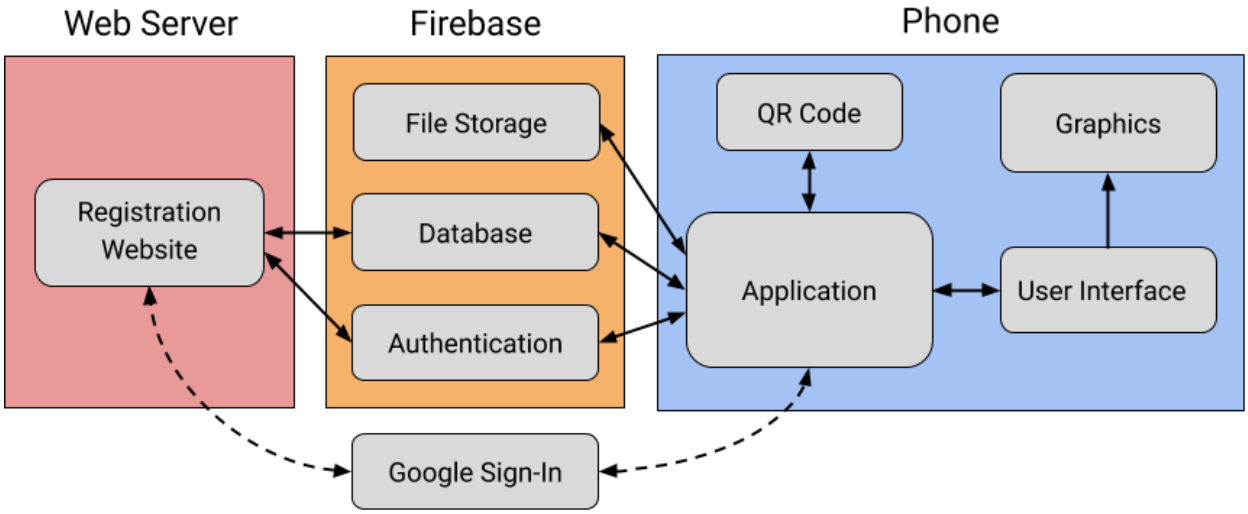
Primary author: Matthew

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

Application

The application is the software which exists locally 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 application 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. The application 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 database, file storage, authentication, and Google authentication.

Creating a basic application 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 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 application 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.

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 database 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. It will also generate feed codes when players register.

Component Connections

Application ⇿ User Interface

The application 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 application 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.

Application ⇿ QR Code

Rendering FeedCode Image: The QR flutter plugin has to receive an id as input in order to generate a QR Code. An id is then passed to the state manager which passes the id to a custom qr painter. The QRimage is saved by wrapping it inside the repaintBoundrary and assigning it a global key. This will allow the UI to search for the saved QRimage of the user's feed code. Scanning FeedCode: The scan function can be embedded into a button in the user interface. When called, the bardcode_scanner plugin will use its barcode scanner class to start the user's camera and begin searching for a QR code. If found, the barcode will be stored as an instance variable inside the setstate function. If not found the barcode will be stored as an error message inside the setstate function. From the setstate function we may either save the id in a widget inside the app for later use or send to a different component for storage.

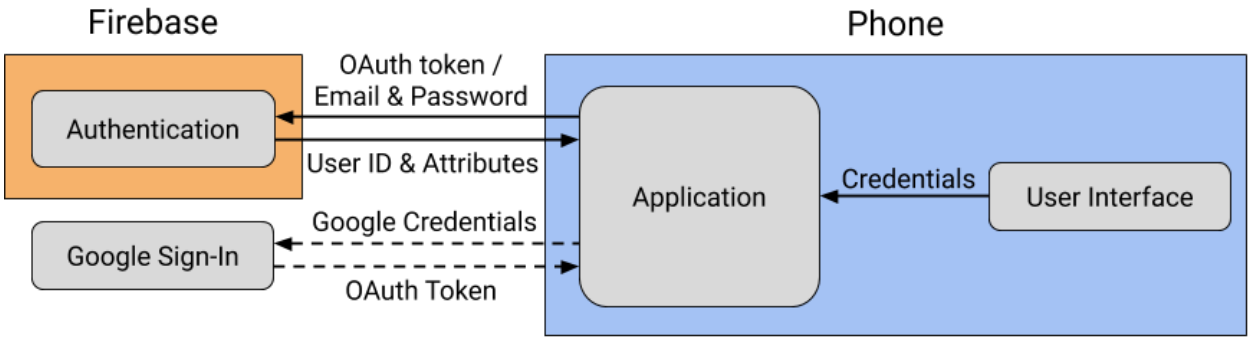
Application ⇿ Google Sign-In

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.

Application ⇿ Authentication

When a user logs in or signs up, either their email and password combination or the OAuth token from Google is sent to the Authentication component in Firebase. The Authentication component verifies the login information and issues a token containing the user's ID and basic information, such as their name and email. The token is a signed JSON Web Token (https://jwt.io), or JWT, and the basic information is stored in a special user database within the

Authentication component. This information is sent back to the app, and the user is now logged in. The diagram below illustrates the authentication process.



Application ⇿ Database

The application 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. The application will also reorganize the database when a new game is being created (the database will then handle storing new game information in a different place than the old one).

Application ⇿ File Storage

The application 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 application 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. The app 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.

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 application, 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 ↔ Database

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 register the user by adding this information to the database and generating a feed code. The website will also send a confirmation email to the user with their generated feed code and additional game information. The website will check for confirmation that registration was successful, and it will 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 authentication server returns the user's basic attributes, such as their name, to the website.
- The registration website prompts the user for additional information, such as school and class year.
- The registration website generates a feed code for the user.
- The registration website sends the user's additional information and feed code to the database, along with their ID token.
- The database verifies the ID token, identifies the user, and adds their additional information and feed code.
- The registration website tells the database to mark the user as registered for the upcoming game.
- The registration website sends a confirmation email to the user's email with game information.
- The registration website tells the user that registration was successful.
- The registration website signs the user out and reloads for the next user to register.

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 application.
    - 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 and basic attributes (name and email).
- 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 application 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 application sends User A's ID token to the database, along with a request for the player types of users A and B and the number of brains of User A and User B.
- The database verifies the user's ID token and returns the player types and brain counts.
- The app checks whether users A and B are both zombies.
- The app checks whether User A has sufficient brains to do the transfer.
    - If not, then the app says that the transfer failed.
    - The app tells the user interface to say that there is an insufficient brain balance.
- If so, the application calculates the new brain counts for each user: 0 for User A and 10 for User B.
- The application tells the database to update User A's brain count to 0 and User B's brain count to 10.
- The app tells the user interface to display a confirmation and User A's remaining balance.

Input Feed Code

- The zombie user logs in to the app (see story above).
- After logging in, the application 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 application, the user can access the QR code scanner.
- The application will then communicate with the smartphone device to take a picture of the QR code.
- The application will send the image to the QR code scanner, which will then decode the image and return the feed code.
- Through the application, 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 application 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 application will communicate with graphics through the user interface to update and display the new information.
- For the newly turned zombie, the application will tell the user interface to change to reflect their transformation into a zombie.
- The application will also communicate with the database to update the global game statistics of all the players, which is done by connecting first to the application, 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 application requests the moderator to log in again to confirm their identity (see story above).
    - Typically, a user’s credentials will be persisted by the Firebase auth object so that they only have to log in once, which is how most apps work. However, since starting a new game is an action which modifies large amounts of data, we will require that the moderator’s user credentials are recently issued.
    - This might prevent someone from getting hold of a moderator’s phone and starting a new game, or it might stop an old moderator from staying logged in forever and messing with the game.
    - Additionally, this extra step will prevent moderators from accidentally starting a new game.
- Through the user interface, the moderator inputs relevant information about the new game, such as its theme or its starting and ending dates.
- The application sends the moderator’s user ID to the database, along with requests to reset player information (whether they are registered for the active game and human or zombie status) and increase an attribute for the number of previous games played.
- The database verifies the moderator ID token and updates the data.
- The application sends the moderator’s user ID to file storage, along with requests to organize the current game files into a new archival folder and create a new folder for the new game.
- The application resets all other information that is in the database or file storage, such as mission or question information.
- The application tells the user interface to display a confirmation that the new game was created.

Moderator Updates Information

- A moderator logs in to the app (see story above).
- After logging in, the application 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 application, 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 their application, via notifications, etc.

Feasibility

Application

We could use Flutter to create the application (https://flutter.dev). Flutter is a tool created by Google to assist developers in creating iOS, Android, and web applications. 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. For the website, Flutter web is currently in beta. We can repackage existing Flutter code to work on the web, but there are some caveats while web support is still under development. However, building the website component is not relevant to our version 1.0 and it still offers the extensive support for iOS, Android, and the Google/Firebase tools we will be implementing into the app.

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 database, authentication, and file storage, as well as help us easily create an elegant app.

User Interface/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. See https://flutter.dev/docs/development/ui/layout for more information.

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. See https://medium.com/flutter-community/building-flutter-qr-code-generator-scanner-and-sharing-app-703e73b228d3 for more information on how to implement specific QR plugins.

Authentication

Authentication can 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 application. When the application 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-lot-about-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 Cloud Storage, which is the implementation of the file storage system 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.

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.

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).

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, which we were originally considering using, mentions that you need to “enable billing for your project” even for free use (https://firebase.google.com/docs/functions). 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. It is possible that other Firebase services could require a paid plan in the future, or that our app might reach the free usage limits.

Addressing Feedback

Feedback from Professor Kampe

You discussed the decision of what to put in the server (vs in the app) in terms of cycles. I would also be inclined to put complex operations (which could be done incorrectly or interrupted mid-way through and leave the database in an inconsistent state) into the (more controlled and more reliable) server.

- We conducted more research and found that it was completely feasible to have the app and database talk directly to each other. In fact, this research led us to omit the server component entirely because we found that it requires more code to use the server as an intermediary between the app and the database (and more money). The operations in the database are very simple, so we figured that we would not need a server to take care of those operations.
    - https://medium.com/firebase-developers/should-i-query-my-firebase-database-directly-or-use-cloud-functions-fbb3cd14118c
- To address the concern of database operations being interrupted, we are not worried about it because Firestore can implement a local cache on the app to update the desired information in the database when it is back online.
    - https://firebase.google.com/docs/firestore/manage-data/enable-offline
- Also, after March 15, 2021 Firebase's free plan will not include Cloud Functions. They are still cheap on the pay-as-you-go plan, although this would require us to add a payment method, which could be difficult and is unnecessary if we do not use Cloud Functions.
    - https://firebase.google.com/support/faq#expandable-13
- Change log of how we modified the architecture to not have the server
    - https://drive.google.com/file/d/1BK7k0hdub0u3Oar2NhBfSVYVn9BMu7hX/view?usp=sharing

I can understand confirmation before ending a game, but it was not clear to me why the moderator needed to log in a second time for new-game creation.

- Extra justification relating to security was added as sub-bullet points under the "Moderator Starts a New Game" story.
- We also deleted the step where the moderator must put in an additional master admin password, since this does seem unnecessary if they are already required to log in again.

Feedback from Team TBD Review Meeting

Website Component Implementation: It will be a painful and time-consuming process to build the website in basic HTML, CSS and JavaScript. The team might want to consider choosing a framework to build the website. For example, it is possible to simply just add a web app to the project using Flutter.

- We categorize this topic as an *issue*, and label it as a *should-fix*.

- **Change Rejected:** The is a suggestion of taste, style, and preference. While we will be conducting more research into Flutter, we are prepared to engage with the tool and this comment is not relevant to our architecture.

Justification of using Flutter as platform app development technology

- Although we believe that Flutter is a solid choice for the HvZ project, the team has not really justified their choice of Flutter, which is the major design decision in the preliminary architecture. It's important and useful to understand and discuss advantages and disadvantages of Flutter over its comparable platforms (i.e. React Native, Xamarin) when discussing feasibility.
    - We categorize this topic as an ***issue***, and label it as a ***should-fix***.
- **Change Rejected:** Perhaps the team missed this, but there are already two paragraphs of justification for why we chose Flutter in the Feasibility section of the project.

Security Protocol

- To ensure the fairness and safety of the game hosted by HvZ App, we believe that it's valuable to add or specify some level of security protocol. For example, it may be compulsory college/organization email verification, clearly defined moderator authority, monitoring abuse of the HvZ app. It matters more if the HvZ app features location sharing or other sensitive information services in the future.
    - We categorize this topic as a ***defect***, and label it as a ***should-fix***.
- **Change Rejected:** While we believe this feedback to be valuable for our implementation of the app, we find it to be out of the scope of architectural concerns. The tools for having security such as google sign in and specific moderator capabilities are outlined in our architecture, so we can already implement their specific suggestions if needed in the future.

QR Code

- We are not sure if QR Code is large enough to be a seperate component and we believe that description of QR Code component should contain more details, especially how the QR Code component interacts with other components of the program. For example, we are not sure if the QR Code component needs any service on the cloud or user authentication. We are also wondering how and where the QR codes will be stored and used later (if reused). In general, we think that QR Code descriptions need to be more clear.
    - We categorize this topic as a ***question***, and label it as a ***should-fix***.
- **Change Accepted:** Elaboration on how the QR code component works was needed.
    - Flutter plugin does not generate id, we need to use a random number generator and pass the number to the state manager at sign-up.
    - Rendering FeedCode Image: The QR flutter plugin has to receive an id as input in order to generate a QR Code. An id is then passed to the state manager which passes the id to a custom qr painter. The QRimage is saved by wrapping it inside the repaintBoundrary and assigning it a global key. This will allow the UI to search for the saved QRimage of the user's feed code.

    - Scanning FeedCode: The scan function can be embedded into a button in the user interface. When called, the bardcode_scanner plugin will use its barcode scanner class to start the user's camera and begin searching for a QR code. If found, the barcode will be stored as an instance variable inside the setstate function. If not found the barcode will be stored as an error message inside the setstate function. From the setstate function we may either save the id in a widget inside the app for later use or send to a different component for storage.

iOS/Android Difference

- We are not sure if the framework will be completely compatible with both iOS and Android devices and if the team should make slightly different designs given the difference of two operating systems. It will be useful to do some research on compatibility.
    - We categorize this topic as a ***question***, and label it as a ***should-fix***.
- **Change (semi-) Accepted:** It is stated clearly in our architecture and research that we know that Flutter is compatible with iOS and Android. There is no reason for the readers to doubt our research without justification. That said, we will add more detail about the compatibility with the website.

Google Sign-in and Authentication

- We believe that the authentication component should be more clear as it is currently encompassed by both the sign-in/registration process and the user data retrieval within Firebase. We believe that it is valuable to consider separation of components further or at least making the current components related to authentication clearer.
    - We categorize this topic as an ***issue***, and label it as a ***comment***.
- **Change Accepted:** The authentication diagram was updated to accurately reflect that reading and writing from the user database and credential verification are all done through Firebase Authentication, and not, as the old diagram suggested, with the app as an intermediary. In light of this, the diagram was also simplified to abstract away the internal components of the Authentication component which we will not need to think about when making the app.

Internet Access

- A large fraction of HvZ App rely on cloud services. Though we usually have strong Internet connections, especially on campus, it might be good to support some offline functionalities when some users accidentally lose Internet connections. However, it is not necessary at this moment.
    - We categorize this topic as an ***issue***, and label it as a ***comment***.
- **Change Rejected:** While many features on the app are cloud dependent, an app's limitations cannot be accounted for within the app. Therefore, we consider this not an issue to be accounted for by our architecture but instead addressed in game implementation.