Team: HvZ App

Members: Kyra Clark, Santiago Rodriguez, Alex Hadley, Matthew Waddell
Project 3C.1 — Component Design

Github Link:
https://github.com/alexhad6/HvZ-app/blob/master/Administrative/Phase%203/design_3c_KC.txt
Primary author: Kyra

Component Design: Database

This component will be made up of two classes. The first class, PlayerDatabase extends the Firestore StatelessWidget class, and will be used to access the stored collection of users currently playing the game. First, it will declare the object of the database 'users' from Firestore. Then, it will include ten instance variables, used to access the fields of each document (user) stored in the database:

```
final String name;
final String email;
final String team;
final bool modStatus;
final int brains;
final string feedCode;
final bool legStatus;
final String school;
final int classYear;
final String dorm;
final ArrayList<String> allFeedCodes; // This will store every feedcode that has been made so far to make sure every randomly generated feedcode is unique.
```

The second class is called FileDatabase, which also extends the Firestore Stateless Widget class. This class is used to access the second collection of information in the database, 'files', which each document (file) stores the fields of missionID and the path to that file in file storage. First, it will declare the object of the database 'files' from Firestore. Then, it will include two instance variables, used to access the fields of each document stored in the database:

```
final String missionID;
final String path;
```

```
class PlayerDatabase:
```

- Read Methods
    - The database component will include several methods in order to access various information about the players to enable gameplay. For each player, the database

stores their Player ID, player or moderator status, human or zombie status, number of brains (zombie currency), legendary status, school, class year, and dorm. This information will be used both from gameplay in terms of keeping track of brains and other relevant statistical information.

- Get list of current users from database
  - This will be used for displaying information about who is still alive on the homepage of the application. This function will likely be used for various methods involving searching through or displaying the full list of players.
  - Called By: Application
  - Input: NA
  - Output: ArrayList of Player ID stored as Strings

```
public ArrayList<String> getPlayers()
     // initialize a new empty string array
     // use the .get() and the QuerySnapShot functionality of
Firestore
     // use the querySnapshot.docs.forEach() to transverse through all
the documents in the player database
     // collect the document id (doc.id) in the arraylist
     // return the array
```

- Get number of humans and zombies from database
  - This will be used similarly to the function which will get a list of all current users and players. This will be used to display the number of each kind of player. It will be used to display various information about live-game statistics on the front page of the application.
  - Called By: Application
  - Input: NA
  - Output: Two Integers as an array of the number of humans and zombies

```
public ArrayList<int> getNumberHZ()
     // initialize an empty integer array of length 2
     // initialize two integer counters (one for humans and one for
zombies)
     // use the .get() and the QuerySnapShot functionality of
Firestore
     // use the querySnapshot.docs.forEach() to transverse through all
the documents in the player database
     // check the field 'team' value of doc if it is a "Human" or a
"Zombie", then increment the appropriate counter
     // outside of the .forEach() (once it has gotten through all the
documents in the database), store the two counters into the original
array
```

```
      // return the array
```

- Get specific player information
  - This will involve various methods for each kind of info stored (name, email, school, human/zombie status, etc.) which will simply return that stored information for a single player. This will look like many various functions appearing similar to, “getSchool([Player ID]).” All stored information of players should be retrievable for statistics and data
  - Called By: Application (to display stats), File Storage (to correctly display information as files for each user), Authentication (to determine who can use or call other functions and methods)
  - Input: Player ID
  - Output: their specific information as a String or Integer (for zombie currency)

```
public String getName(String userID)
public String getEmail(String userID)
public String getTeam(String userID)
public bool getModStatus(String userID)
public int getBrains(String userID)
public String getFeedCode(String userID)
public bool getLegStatus(String userID)
public String getSchool(String userID)
public int getClassYear(String userID)
public String getDorm(String userID)
     // all take in the string userID of the user whose information we
want to get
     // each function goes through a nearly identical process
     // use the Firestore DocumentSnapshot functionality
     // create a snapshot of the document in the database (retrieved
from the userID)
     // collects all the data using the .get() function
     // returns it as data[‘customField’] (where ‘customField’ is the
‘name’, ‘email’, ‘team’, etc. depending on correct function)
```

- Write Functionality
  - As the game is played, often the moderators need to adjust player statistics from the database. This includes moderators updating legendary status for zombies, and resurrecting human players (changing their status from zombie back to human). Below is listed the various kinds of write functions there will be:

  - Write’s new user to database

        - This function writes in all the ascribed sign in information that is given when the person registers. This includes the user feed code and a randomized Player ID.
        - Called By: Sign in, Registration
        - Input: all the information needed inputted from the registration website
        - Output: NA (a new user of Player ID is created with their associated information)

```
public Future<void> addUser(this.fullName, this.email, this.team,
this.modStatus, this.brains, this.legStatus, this.school,
this.classYear, this.dorm)
     // the Future<void> class is a FireStore class used when needing
to change information asynchronously from the Database
     // calls the Firestore .add({}) function to ascribe all the
correct information
     // 'brains' : 0
     // the number of brains collected should always start at 0
     // 'feedCode' : calls a randomized string function in Firestore.
     // With the random 'feedCode', transverse through allFeedCodes to
make sure it is unique. If it is unique, add it to the user, else,
generate a new feedCode.
     // when the user is added, the database automatically ascribes it
a randomized and unique document ID
     // uses the .catchError() function to output an error if the user
fails to be added
```

    - Change 'Legendary' Status
        - Called By: Application, Moderators only (Authentication)
        - Input: String Player ID, boolean if legendary status
        - Output: NA (that Player ID's status was changed to the new given string)

```
public Future<void> changeLegStatus(String userId, bool status)
     // returns users.doc(userID)
     // similar to this users.doc(userID).set({'legStatus' : status})
     // calls the .set() function on above to change the status
```

    - Resurrect Zombie
        - Called By: Application, Moderators only (Authentication)
        - Input: Player ID
        - Output: NA (that Player ID's player status was changed to human)

```
public Future<void> resurrectZombie(String userId)
     // in a similar process as described above,
     // changes 'team' to "Human"
     // calls the catchError is case of unexpected behavior
```

- Change into Zombie
    - Called By: Application, during eating process by Zombie (Authentication)
    - Input: Player ID A (Zombie), Player ID B (Human)
    - Output: NA (Player B's player status was changed to zombie, Player A's brain currency increased by 1)

```
public Future<void> eatHuman(String eater_userID, String eaten_userID)
     // takes in two userIDs: eater_userID is the zombie that ate the human and gained another brain, eaten_userID is the human that was tagged by the zombie and now turns into a zombie
     // calls incrementBrains method on eater_userID
     // in a similar process as described above…
     // uses .set() on eaten_userID to change 'team' to "Zombie"
```

- Increment Brain Currency
    - Called By: Application, during Eat Human function
    - Input: Player ID (Zombie)
    - Output: NA (Player ID's brain currency increased by 1)

```
private Future<void> incrementBrains(String eater_userID, int add)
     // this method is private because it should only be called as part of other processes in this class
     // take in the eater_userID (the zombie that just ate a new brain) and an integer add (the number of brains they ate, which will usually always be called one at a time, meaning brains will be 1)
     // uses the .update() Firestore function to update the existing brain count, rather than completely replace it with .set()
     // in .update(), call the Firestore function FieldValue.increment(add) to change the 'brains' field value by the number set as add.
```

- Change Moderator Privileges
    - Called By: Application, by Moderators (Authentication)
    - Input: Player ID, boolean if they should have mod privileges
    - Output: NA (that Player ID's Mod status was changed to/from Moderator)

```
public Future<void> changeModStatus(String player_userID, bool isMod)
     // a similar process to changeLegStatus() or resurrectZombie(),
     // called .set to change the field 'mod' to the new bool isMod
```

- Zombie Brain Transfer
    - In the game, it is possible for zombie players to transfer their brain currency to another zombie. This entails changing the number of brains of Player A by x number of brains, and increasing the specified Player B's brains by x amount.
        - Called By: Application, Zombie players only (Authentication)

        - Input: Player A's ID (transfering), Player B's ID (receiving), Integer number of brains
        - Output: NA (that Player B's brains increase by the specified integer amount)

```
public Future<void> zombieBrainTransfer(String giver_userID, String taker_userID, int brns)
     // takes in two String userIDs, giver_userID is the player that initiated the transfer and is giving an int number of brains (int brns)to the other zombie player taker_userID
     // there is a variety of checks called in this function to .catchError in case they try to transfer more brains than are available to them, or if they try to transfer brains to a "Human" player
     // called the private method incrementBrains twice:
     // incrementBrains(taker_userID, brns)
     // incrementBrains(giver_userID, -brns)
```

- Reset all player data
    - When the week-long game ends, the database should be wiped clear for future games. This would entail completely deleting all the player information.
        - Called By: Application, Moderators only (Authentication)
        - Input: NA
        - Output: NA

```
public Future<void> deleteUsers()
     // calls the Firestore .collection().doc().delete() function
     // transverses through all the documents in the 'user' database and deletes each one
```

```
Class FileDatabase:
```

- File Storage
    - Write a Path to a File
        - The Database will store the paths as strings of all the objects in file storage. This function will write the file path and create a new object in the database.
        - Called By: File Storage
        - Input: File Path as a String, Mission Number
        - Output: NA (the file path will appear under a new entry in the database associated with the mission number.)

```
public Future<void> addFile(this.missionID, this.path)
     // Calls the Firestore .add({}) function
```

```
      // collects correct missionID and path into document field, from
File Storage component
      // uses the .catchError() function to output an error if the user
fails to be added
```

- Return a File Path
  - This function will search through all the paths to find that one associated with the correct mission number, and return its associated file.
  - Called By: File Storage
  - Input: Mission Number
  - Output: File Path as a String

```
public String getPath(String missionID)
      // takes in a string missionID from file storage
      // transverses through all the stored documents and checks their
missionID
      // when it find the matching mission ID, return's its associated
'path'
      // uses the Firestore .get() functionality and the Firestore
QuerySnapShot functionality
      // transverses using the Firestore querySnapshot.docs.forEach()
method
```

- Reset game file data
  - When the week-long game ends, the database should be wiped clear for future games. This would entail completely deleting all the player information.
    - Called By: Application, Moderators only (Authentication)
    - Input: NA
    - Output: NA

```
public Future<void> deleteFiles()
      // calls the Firestore .collection().doc().delete() function
      // transverses through all the documents in the 'files' database
and deletes each one
```

Functionality References

Document and Query Snapshots:
https://firebase.flutter.dev/docs/firestore/usage#document--query-snapshots
.get() for Getting Data:
https://firebase.flutter.dev/docs/firestore/usage#read-data
.add() for Writing Data:
https://firebase.flutter.dev/docs/firestore/usage#writing-data
.delete() for Removing Data:

https://firebase.flutter.dev/docs/firestore/usage#removing-data