Team: HvZ App

Members: Kyra Clark, Santiago Rodriguez, Alex Hadley, Matthew Waddell
Project 3E.2 — Final Component Design

Github Link:
https://github.com/alexhad6/HvZ-app/blob/master/Administrative/Phase%203/design_3e_KC.txt
Primary author: Kyra

Component Design: Database

The Database component is a Firebase Cloud Firestore (see documentation here: https://firebase.flutter.dev/docs/firestore/overview). The Cloud Firestore database will be fully set up on the server within the app project. It will be set up to store two separate collections on the same database. The first collection, ‘users’ will store the user information, and the second collection ‘files’ will store all the file information for the file storage component. Thus, the component will be made up of three classes: UserData, FileData, and DataErrorMessage. UserData and FileData are singleton classes. When a User is successfully signed into the application, a UserData and a File Data objects are constructed. In their construction, an instance will be declared for both like so:

```
CollectionReference users =
FirebaseFirestore.instance.collection('users');
CollectionReference files =
FirebaseFirestore.instance.collection(‘files’);
```

This will create a local instance of the documents of the database, so the user will not have to call the database directly everytime. Furthermore, this will protect against the error of poor internet connection. When these instances are called, they will be protected by the Authentication’s functionality for connectivity. If the database cannot be reached or reached the time_out limit, the poor connection or no internet error will be thrown. This way, the app will not initially break down if there is no internet connection.

In the construction of these singleton objects of UserData, FileData, each will also declare a DataErrorMessage object for that particular user. In addition, the field variables of that User will be stored as local instance variables for quick and easy access. Calls to retrieve data for other Users will have to go through the actual getter functions. But, the user’s own data (which will likely be called the most) will be stored locally in the instance variables. Then, listeners will be set up to update the actual database every time a change occurs on the local instance.

```
class UserData:
```

The UserData class will be used to access the stored collection of users currently playing the game. There is a local instance of the database to limit connection problems as well as the user's own data stored in instance variables to optimize and limit excessive calls to the database. In the construction of this class, each of the below instance variables will be declared by calling their correct get function once, after testing for connectivity and other errors. There are the instance field variables included:

```
String name;
String team;
bool modStatus;
int brains;
string feedCode;
bool legStatus;
String school;
int classYear;
String dorm;
String taggedBy;
List<String> tagged;
```

- Read Functionality
    - 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

```
ArrayList<String> getPlayers()
     // initialize a new empty string array
     // use the .get() and the QuerySnapShot functionality of
Firestore Dart
     // 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
```

```
      // declare DataErrorMessage success else:
      // catch for errors: ‘no-connection’, 'timeout', 'doc-nonexist',
'get-fail', or display the FirebaseException error
```

- 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

```
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 Dart
      // 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
      // declare DataErrorMessage success else:
      // catch for errors: ‘no-connection’, 'timeout', 'doc-nonexist',
      'get-fail', or display the FirebaseException error
```

- Get specific player information
  - This will involve various methods for each kind of info stored (name, 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)

```
String getName(String userID)
```

```
String getTeam(String userID)
bool getModStatus(String userID)
int getBrains(String userID)
String getFeedCode(String userID)
bool getLegStatus(String userID)
String getSchool(String userID)
int getClassYear(String userID)
String getDorm(String userID)
String getTaggedBy(String userID)
List<String> getTagged(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 Dart 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', 'team', etc. depending on correct function)
     // declare DataErrorMessage success else:
     // catch for errors: 'doc-nonexist', 'get-fail', or display the
FirebaseException error
```

- 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. It also sets all the other parameters into the default setting.
        - Called By: Sign in, Registration
        - Input: name, user ID, and feed code
        - Output: NA (a new user of Player ID is created with their associated information)

```
Future<void> addUser(String uid, String name, String feedcode)
     // the Future<void> class is a Dart class used when needing to
change information asynchronously from the Database
```

```
      // calls the Firestore Dart .add({}) function to ascribe all the
correct information
      // the default settings are:
      // ‘team’ : “Human”
      // ‘modStatus’ : False
      // ‘brains’ : 0
      // ‘legStatus’ : False
      // ‘school’ : “NA”
      // ‘classYear’ : 0
      // ‘dorm’ : “NA”
      // ‘tagged’ : []
      // ‘taggedBy’ : “NA”

      // when the user is added, the database automatically ascribes it
a randomized and unique document ID
      // declare DataErrorMessage success else:
      // catch for errors: ‘no-connection’, 'timeout', 'doc-exists',
'add-fail', ‘wrong input’, or display the FirebaseException error
```

- 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)

```
Future<void> changeLegStatus(String userId, bool status)
      // users.doc(userID)
      // similar to this users.doc(userID).set({‘legStatus’ : status})
      // calls the .set() function on above to change the status
      // declare DataErrorMessage success, else:
      // catch for errors: ‘no-connection’, 'timeout', 'doc-nonexist',
'set-fail', or display the FirebaseException error
```

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

```
Future<void> resurrectZombie(String userId)
      // in a similar process as described above,
      // changes ‘team’ to “Human”
      // declare DataErrorMessage success, else:
      // catch for errors: ‘no-connection’, 'timeout', 'doc-nonexist',
'set-fail', ‘res-human’, or display the FirebaseException error
```

- Eat a Human

        - 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)

```
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
     // calls setTeam() to Zombie for the eaten_userID
     // calls setTaggedBy() of the eaten_userID to the tagger eater_userID
     // calls addTagged()to add the eaten_userID to the eater_userID's list of tagged players
     // declare DataErrorMessage success, else:
     // catch for errors: 'no-connection', 'timeout', 'doc-nonexist', 'set-fail', 'eat-zombie', 'human-eat', or display the FirebaseException error
```

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

```
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)
     // use the .update() Firestore Dart function to update the existing brain count, rather than completely replace it with .set()
     // in .update(), call the Firestore Dart function FieldValue.increment(add) to change the 'brains' field value by the number set as add.
     // declare DataErrorMessage success, else:
     // catch for errors: 'no-connection', 'timeout', 'doc-nonexist', 'set-fail', or display the FirebaseException error
```

    - 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)

```
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
     // declare DataErrorMessage success, else:
     // catch for errors: 'no-connection', 'timeout', 'doc-nonexist',
'set-fail', or display the FirebaseException error
```

- Set specific player information
  - This will involve various methods for each kind of info stored (team, school, class year, dorm) which will simply set that stored information for a single player. This will look like many various functions appearing similar to, "setSchool([Player ID])." This will be collected in a survey prompted by the application, and stored for statistics.
  - Called By: Application (to display stats)
  - Input: Player ID, desired variable change as a string
  - Output: NA

```
Future<DataErrorMessage> setTeam(String userID, String tmHZ)
Future<DataErrorMessage> setTaggedBy(String userID, String tagger)
Future<DataErrorMessage> setSchool(String userID, String schl)
Future<DataErrorMessage> setClassYear(String userID, String year)
Future<DataErrorMessage> setDorm(String userID, String drm)
     // all take in the string userID of the user whose information we
want to set, and a new string for the variable we want to change
     // each function goes through a nearly identical process
     // calls the .set() functionality to change the given status with
the new desired string

Future<void> addTagged(String tagger_id, String tagged_id)
     // use the getTagged function to retrieve the tagger_id's current
list of tagged players
     // append the list with the tagged_id (Dart .add() function)
     // use the .set() Firestore Dart function to replace the field
with the new list

     // declare DataErrorMessage success, else:
     // catch for errors: 'no-connection', 'timeout', 'doc-nonexist',
'set-fail', or display the FirebaseException error
```

- 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)

```
Future<DataErrorMessage> 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
     // called the private method incrementBrains twice:
     // incrementBrains(taker_userID, brns)
     // incrementBrains(giver_userID, -brns)
     // declare DataErrorMessage success, else:
     // catch for errors: 'no-connection', 'timeout', 'doc-nonexist', 'set-fail', 'transf-human', 'brain-limit', 'brain-neg', or display the FirebaseException error
```

- 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

```
Future<DataErrorMessage> deleteUsers()
     // calls the Firestore Dart .collection().doc().delete() function
     // transverses through all the documents in the 'user' database and deletes each one
     // declare DataErrorMessage success, else:
     // catch for errors: 'no-connection', 'timeout', 'data-reset-fail', or display the FirebaseException error
```

`class FileData:`

The FileData 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 URL to that file in file storage. First, it will declare the object of the database ‘files’ from Firestore in a local instance to protect against connectivity issues. The fields stored for each document here will be `String missionNum, String fileID,` and `String url`. But because we are not only working with one file mostly, we will not be using the instance variable technique like we did for the class UserData.

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

```
Future<void> addFile(String missionNum, String fileID, String url)
     // Calls the Firestore Dart .add({}) function
     // collects correct missionID and URL into document field, from
File Storage component
     // uses the .catchError() function to output an error if the user
fails to be added
     // declare DataErrorMessage success else:
     // catch for errors: ‘no-connection’, 'timeout', 'doc-exists',
'add-fail', ‘wrong input’, or display the FirebaseException error
```

- Read Functionality
    - Return a single File URL from File ID
        - This function will search through all the URLs to find that one associated with the correct file id, and return its associated file.
        - Called By: File Storage
        - Input: File ID
        - Output: File URL as a String

```
String getFile(String fileID)
     // takes in a string fileID from file storage
     // transverses through all the stored documents and checks their
fileID
     // when it find the matching FileID, returns its associated ‘URL’
     // uses the Firestore .get() Dart functionality and the Firestore
QuerySnapShot Dart functionality
```

```
     // transverses using the Firestore querySnapshot.docs.forEach()
Dart method
     // declare DataErrorMessage success else:
     // catch for errors: ‘no-connection’, 'timeout', 'doc-nonexist',
'get-fail', or display the FirebaseException error
```

- Return a list of all File URLs associated under one mission
  - This function will search through all the URLs to find the group that are all a part of the same mission number, and return the list of files.
  - Called By: File Storage
  - Input: Mission Number
  - Output: File URLs as List of Strings

```
List<String> getMissionFiles(String missionNum)
     // takes in a string missionNum from file storage
     // transverses through all the stored documents and checks their
missionID
     // every match of Mission Number
     // uses the Firestore .get() Dart functionality and the Firestore
QuerySnapShot Dart functionality
     // transverses using the Firestore querySnapshot.docs.forEach()
Dart method
     // collect in list and return
     // declare DataErrorMessage success else:
     // catch for errors: ‘no-connection’, 'timeout', 'doc-nonexist',
'get-fail', or display the FirebaseException error
```

- 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

```
Future<DataErrorMessage> deleteFiles()
     // calls the Firestore .collection().doc().delete() Dart function
     // transverses through all the documents in the ‘files’ database
and deletes each one
     // declare DataErrorMessage success, else:
     // catch for errors: ‘no-connection’, 'timeout',
'data-reset-fail', or display the FirebaseException error
```

```
class DataErrorMessage:
```

The final class is called DataErrorMessage, and it is used to collect and transmit accurate error messages. Being able to catch any errors will allow users to continue to interact with the application if an error occurs; the application should be able to handle errors without completely breaking. Furthermore, for each error, it will provide a custom error message, which will allow the programmers as well as future users and moderators to receive helpful feedback when interacting with the app. It has the instance variables:

```
static const Map<String, String> _errorMessages = {
   'no-connection': 'No internet connection',
   'timeout': 'Poor internet connection',
   'wrong-input' : 'Input was empty string',
   'add-fail' : 'The document was not added to database',
   'data-reset-fail' : 'The database is not clear',
   'doc-nonexist' : 'This document does not exist',
   'doc-exists' : 'This document already exists in the database',
   'set-fail' : 'Could not set field',
   'get-fail' : 'Count not get field',
   'eat-zombie' : 'Cannot eat Zombie player',
   'human-eat' : 'Human player cannot eat',
   'transf-human' : 'Cannot transfer brains to/from Human player',
   'brain-limit' : 'Not enough brains for successful transfer',
   'brain-neg' : 'Cannot transfer negative brains',
   'res-human' : 'Cannot resurrect Human player'
 }; // these are all the possible error messages that we are going to
catch
bool success; // whether the Database operation was successful or not
String errorMessage; // the error message from the above list of
possibilities
String _errorCode; // the error code the maps to the associated
message
```

- Throw an error message
    - If we can catch an error, then we want to throw the appropriate error message to assist the users
        - Called By: other Database classes
        - Input: the success bool, the error code to find the appropriate error message
        - Output: the correct error message is thrown

```
AuthMessage(this.success, this._errorCode)
   // If success is true, then set errorMessage to '' (empty string)
   // If _errorMessages contains the _errorCode as a key:
```

```
//   Set errorMessage to the corresponding value for the _errorCode
// Else:
//   Throw an error (this accounts for [FirebaseAuthException]s
```

Functionality References

Firebase Cloud Firestore:
https://firebase.flutter.dev/docs/firestore/overview
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
To use the Cloud Firestore package, use this:
import 'package:cloud_firestore/cloud_firestore.dart';