Adding to the Project

Intro

This document contains short descriptions of how to add to each section of the project written by the creators of those sections. Also, a short list of goals for each section. See the backlog for a more detailed list of Todo items.

Updating UI

Overview

The pages for UI are housed in the pages folder. To add a new page simply add a new file and follow the format already existing. YouTube is an incredible resource for understanding different dart elements. There are so many short, effective dart tutorials there. **UI should be self-contained**. If you need storage between widgets or complex logic, you should create a service. See the **Adding Functionality (Services)** section for details.

Normal Structure:

Each of the existing pages follow a basic structure to ensure that the app theme matches throughout. This is done by using a scaffold that implements both the top bar and navigation bar. The exact structure can be seen below. The body section is where the page specific widgets should be implemented. Widgets are the building blocks of any flutter app and simply reference any element in the UI for more details on widgets please see this documentation: https://docs.flutter.dev/ui/widgets

For each new page added, the page should also be added to the navigation bar. The navigation bar file can be found at the path:
<parent_dir>/baby_words_tracker/lib/pages/shared/bottom_bar.dart

```
return Scaffold(
        backgroundColor: const Color(0xFF828A8F),
        appBar: TopBar(pageName:
localizationService.translate("<page_name>")),
        bottomNavigationBar: bottomBar(context, "<page_name>"),
        Body: <rest of the UI for the app>
```

TODOs:

1. We need to add a user onboarding process consisting of at least a terms and conditions page and possibly some intake surveys.
2. In the same vein, a tutorial process or intro process should be added that guides users through the process of creating their first child and adding their first word.

Adding Functionality (Services)

Your UI should be separate from your logic. This project uses Provider, a dart package, to enable that separation. If a feature requires storage that is accessible between two widgets, it should be implemented as a service and accessed using Provider. A service is just a standalone class, usually extending ChangeNotifier, that performs some logic. See any of the services in the project already as reference (AuthenticationService in the auth folder and CurrentChildrenService in the util folder are great examples). Once your service is created, it should be initialized in main as a provider and then accessed in your widgets using a consumer widget. **Please read the Provider documentation here:** https://docs.flutter.dev/data-and-backend/state-mgmt/simple (and maybe read this too: https://pub.dev/packages/provider) to understand Provider. It is a very important tool for bringing the app together and it makes data sharing between widgets easy. However, it does take some understanding to get used to. YouTube also has tutorials on this.

TODOS:

1. There will most likely be required services to enable the intro processes described in the TODOs: section of Updating UI.

Authentication

Auth is handled by the user model and authentication services. It uses roles assigned using custom claims. The roles are represented in an enum in utils and a corresponding enum in roles.js in the functions/auth in the firebase project folder. The user types are represented by an enum in utils.

Adding New Roles

To add new roles, update the enum in utils and in the javascript file. Make sure the string representations in both match or the app will not recognize the new roles.

TODOs:

1. User type needs to be revisited. Currently it uses different collections in the database, but this system can be clunky at times. We should consider using a single collection for users and just indicating type using custom claims.
2. User roles need to be updated when they are changed. Currently, as I understand it, updates to user roles will not propagate to the user on the frontend immediately. This needs to be verified. If that is the case, we need to use FCM messages, a Firebase feature, to trigger a refresh when roles are updated. Alternatively, users can just sign out and sign back in. I do not think this is a security vulnerability.

Statistics

The current statistics availible to the user on the home and statistics page are a good start, but much more rich information could be provided to the user with only the data already stored. The main limiting factor for new stats will be database calls, as efficiently querying childrens' histories has repeatedly raised concerns about the numbers of reads necessary. The issue of read count could be addressed by having live updating database-side statistics, such as words learned this week or number of each part of speech.

Adding New Graphs on the Stats Page:

In order to add a new graph that works in the existing system, the programmer must:

1. Add a new properly named value to the GraphType Enum, located in <root_directory>\baby_words_tracker\lib\util\graph_type.dart

    a. Make sure to add its option name and display name, so that the graph will be labelled properly in the UI
2. If the graph has an adjustable time horizon, add it to the graphsWithLength set at the top of <root_directory>\baby_words_tracker\lib\pages\stats.dart
3. Create a data retrieval function to act as the future for the new graph's FutureBuilder
    a. Ensure that this function uses the caching structure to prevent overquerying
4. Create a FutureBuilder that creates a syncfusion graph or other figure to represent that data
5. Add a call to that FutureBuilder to the graphSwitcher, associated with the proper GraphyTyope

TODOs:

1. Add more interesting statistics for the end user
    a. e.g. Features relating to the frequency of the words the child is learning
        i. i.e. Your child knows 500 of the top 3000 most common words!
2. Add database-end statistics to reduce querying
3. Add a caching system that either saves graphs for the entire session, for a set amount of time, or until the relevant information in the graph changes
    a. The current caching implementation only saves data for the current instance of the stats page – extremely naive
    b. Large potential for query saving here, too.

Cloud Functions

Cloud functions and how to update them are explained in detail in the Feature Documentation doc. Please reference it for any information on updating functions or the current functionality they have.

TODOs:

There are currently no TODO items for cloud functions.

Data

Database Structure:

The firestore database structure is not enforced on the firebase side because of the NoSQL nature. This means that all changes to the structure need to be done in the database models and these changes will be reflected in the database for all new documents. However, this does not retroactively fix documents that are in the database already. Unfortunately, you will either

need to manually update the existing documents or write a script to update the documents with the new structure. When updating using a script, the set repository function should be used since that merges with current data and does not override like the update function. Note, the update option is only available for documents that allow updates from users. You can see which documents allow for updates in the rules section of the firebase console. For more on firestore rules please see the following documentation:
https://firebase.google.com/docs/firestore/security/get-started

Database Repository and Services:

The database repository and services should be updated when you need to implement a new CRUD functionality. The repository should implement any new direct calls to the database. Documentation on the firestore API that is used throughout can be found here:
https://firebase.google.com/docs/firestore/

Services specifically implement the business logic for each of the models. These functions should make calls to the firestore_repository to implement CRUD functionality. When writing these functions beware of the number of reads and writes that will be called because once 50k reads are done in a day the cost increases substantially. For reads that need all of the docs from a specific collection, batch calls should be used. These are implemented in the functions that start with get_multiple.

TODOs:

1.) Currently the researcher home page is very read heavily and will easily hit the max number of reads with several calls. The fix to this is to use paging. However, due to time constraints, this has yet to be implemented.
2.) There are currently no processes for deleting parents, children, or researchers from the database. Deleting children is a feature that parents will likely want. However, consideration needs to be taken for what data should be deleted and which should be kept. For research purposes, it may be advantageous to keep child data in many cases.

Localization:

The main things that can be changed in localization are adding new phrases and new languages. New phrases can be added by adding a new key-value pair to the all_localizations file in the l10n folder (that is the standard abbreviation for localization). The key should be the same for all languages and then the value should reflect the text that should be displayed for each language. If a key is passed to the translation function that does not exist within the all_localizations file, the plain text of the key will be displayed instead of the correct translation. Adding new languages should also be straightforward. Simply add another language code to the all_localizations file and add the correct translations for that language. However, adding the

setting to switch to a new language could provide difficulty, but the code for adding a list selection widget can be seen in the “Add Child” feature and should provide some insight on how to move forward.

TODOs:

1. A low priority issue is the formatting of the all_localizations file. This file is essentially one long list of every piece of text in the app at the moment. It might be advantageous to switch to having a different structure, such as a separate localization map for each page.

Video

The video upload, download, and streaming features are where you are likely to need to make the most changes. The functions used for video upload and download can be found in the video folder. The implementation of video upload is on the home page. The user can shoes a video through the functionality from the file_picker library. When they submit a video, the video is uploaded using a signed url and then the word tracker is updated with the filename. Currently each word will only be associated with the most recent video uploaded for that word. To display the videos back to the user, we are using a local download to a temporary directory.The back end includes getting a list of all word trackers for the selected child that have a video associated with them and displaying that list in a drop down menu to the user. This functionality can be found in the video_display.dart file under pages.

For any updates to the current video upload and download functionality it is likely that the firebase functions will need to be updated. The ones associated with video upload and download are clearly labeled within the index.js file in the firebase-project folder. You will need to have eslint working on your machine because perfect formatting is enforced when deploying the firebase functions. These functions are cloud based, so you must deploy them using the command “firebase deploy –only functions” to see your changes reflected in the functionality of the app. For some tasks, you may also need to upgrade the service accounts permissions. You can do this in the google cloud console. The service account for the google cloud bucket is the account that uses the “developer.gserviceaccount” domain. The full email is not listed here for security purposes.

TODOs:

1. Video display currently does not work. When attempting to access the temporary directory a file not found error is returned. We are confident that we are pulling the directory that was made, so we are unsure of the origin of this error.
2. Video compress was not done because the standard library for this lost support from the company who developed it and they subsequently deleted the binaries that would have

allowed the functionality without directly accessing the library. Finding a suitable alternative to this is essential to maintaining low costs for this project.

3. Using a download for video display is not ideal. The better strategy is to use video streaming. To do this you would need to upload the file in streaming format and then use the signed url to stream the video. This is the next step after video download begins working