# Development Environment Setup

The app is built using Flutter and Firebase and configured to support Android, Iphone, and Web devices. This document describes how to install dependencies to run and test the app, how to set up ESLint for cloud function development, and how to use Firebase emulators during testing.

- UI/Frontend - Flutter
- Hosting - Firebase
- Authentication - Firebase
- Database - Firebase

Our app is developed in VSCode and it is recommended that you use it as our ESLint process relies on VSCode and Flutter integrates very well with VSCode.

## Table of Contents



## Installing Dependencies

Install these dependencies in order. Additional packages used by the flutter project will automatically build when flutter builds. A list of these packages can be found in pubspec.yaml file. This is also where you should add any new packages that might be needed.

### Flutter Install:

- Install Git if you haven't already: https://git-scm.com/downloads/win
- Install Android Studio (required for Flutter install): https://developer.android.com/studio/install#windows
- Flutter:
    - Install flutter by selecting your operating system > android on the install page linked below.

    - It is recommended that you use the built in install process with the Flutter plugin in VSCode. All other install methods are untested by the team as of (4/22/2025).
    - If a virtual android device is desired, follow the steps after the Flutter install to set one up.
    - https://docs.flutter.dev/get-started/install
- OpenJDK install (if flutter install didn't install it): https://adoptium.net/

### Firebase Tools Install:

Firebase can be installed using either the Node Package Manager (NPM) from Node.js or by directly installing the binary. It is recommended that you use Node.js installed with a version manager.

- Firebase CLI (Firebase Tools):
    - Select the correct operating system, then follow the instructions at the install page linked below to install the CLI. You do not need to follow the steps for initializing a project since this one is already set up.
    - https://firebase.google.com/docs/cli#setup_update_cli
    - Helpful link for installing Node Version Manager: https://www.freecodecamp.org/news/node-version-manager-nvm-install-guide/
- Firebase Flutter Package Installation:
        - Note: Firebase packages can be added to a Flutter project to give it additional functionality. The project should already be configured with the required packages, and we will install those later. If you need to install further packages after setting up the app, follow the steps at this webpage: https://firebase.google.com/docs/flutter/setup?platform=android
    - If you haven't already, install the Firebase CLI.
    - Login to Firebase using your Google account by running the following command:
        - `firebase login`
    - Install the FlutterFire CLI by running the following command from any directory:
        - `dart pub global activate flutterfire_cli`

## Setting up and Running the App

Follow these steps to get the repo and install the app.

- Clone the repo: `git clone git@github.com:cddale/Baby_Words.git`
- Now run these commands from the flutter project root (<repo_dir>/baby_words_tracker):

    - `flutter pub get`
- Run the app:
    - Web: Chrome or Edge are required to run the app on the web (fast for debugging but not accurate to mobile screen layout). To use web simply run:
        - `flutter run`
    - Mobile: The android emulator is one easy way to test on mobile. Follow the steps in the flutter install page above to set up using Android Studio. Mobile is slower but will show you where your layout conflicts are. To run with mobile, start an android virtual device then run:
        - `flutter run`
    - Note: when calling run, you can specify the type of execution using -d
        - `flutter run -d chrome`
        - `flutter run -d mobile`

# ESLint for Cloud Function Development

Our cloud functions use ESLint for formatting. **Functions will not deploy if they do not match the ESLint rules.** Set up ESLint by doing the following:

1. In the <parent directory>/firebase_project/functions folder run the following command to install dependencies including ESLint:
    a. `npm install`
2. Install the ESLint extension for VSCode: https://marketplace.visualstudio.com/items/?itemName=dbaeumer.vscode-eslint
3. Add the following text to your VSCode settings.json file.

```
"editor.codeActionsOnSave": {
    "source.fixAll.eslint": true
},
"eslint.validate": ["javascript"]
```

    a. Access the file by pressing `ctrl + ,` to open settings and clicking the file icon with an arrow over it at the top right to open the json file.
    b. Or by pressing `ctrl + shift + p` and typing settings then looking for the options with (JSON) at the end.

Your index.js and other javascript files in the functions folder should now be automatically formatted on save.

## Updating Cloud Functions

Functions are updated by changing the index.js file in the functions folder and then running a firebase tools command. Any function that is exported from index.js will be treated as a cloud function. The steps are as follows

1. Modify and save index.js
2. From the firebase_project directory run: `firebase deploy --only functions`

    a. If you want to deploy other things as well, firebase deploy deploys all applicable things in the folder

## Firebase Emulators

Firebase emulators let you test code on your local machine as if it was interacting with a set of firebase services. This is very useful (probably necessary for safety) for testing firebase functions and database security rules outside of a production environment. In our repository, a firebase project with initialized emulators has already been created in the firebase-project folder in the main repository.

1. To run the emulator simply run `firebase emulators:start` in the firebase-project folder
2. You will need to connect the project to the emulators by adding code to main.dart. You should be able to use the `setupFirebaseEmulators()` function from check_emulators.dart in the util folder to automatically set up emulators. This function expects emulators to use their default ports and simply subscribes to an emulator if an open socket is detected on the corresponding port. Run the function using await below `Firebase.initializeApp()` and above `runApp()`.

If you want to run the project with some testing data the following commands are helpful:

1. Run emulators with import data:
    a. `firebase emulators:start --import ./emulator-source`s
2. Run emulators with imported data and automatically export it with changes at the end:
    a. `firebase emulators:start --export-on-exit ./emulator-sources --import ./emulator-sources`
3. Save changes to imported data while the emulators are running:
    a. `firebase emulators:export ./emulator-sources`

Notes:

Emulator setup is not covered above because our emulators are already configured. However, if you need to add another emulator you should rerun `firebase init emulators`. Emulator setup is straightforward. If possible, use the default ports for emulator setup as the check_emulators.dart util will expect that. Lastly, if you want check_emulators.dart to work with your added emulators, you will need to update the function to check the right port and subscribe to the correct emulator.

Lastly, emulators are not meant to run in mixed environments so you need to choose and run all emulators for features that you want available (for example you can't access the production database while using the firebase auth emulator.