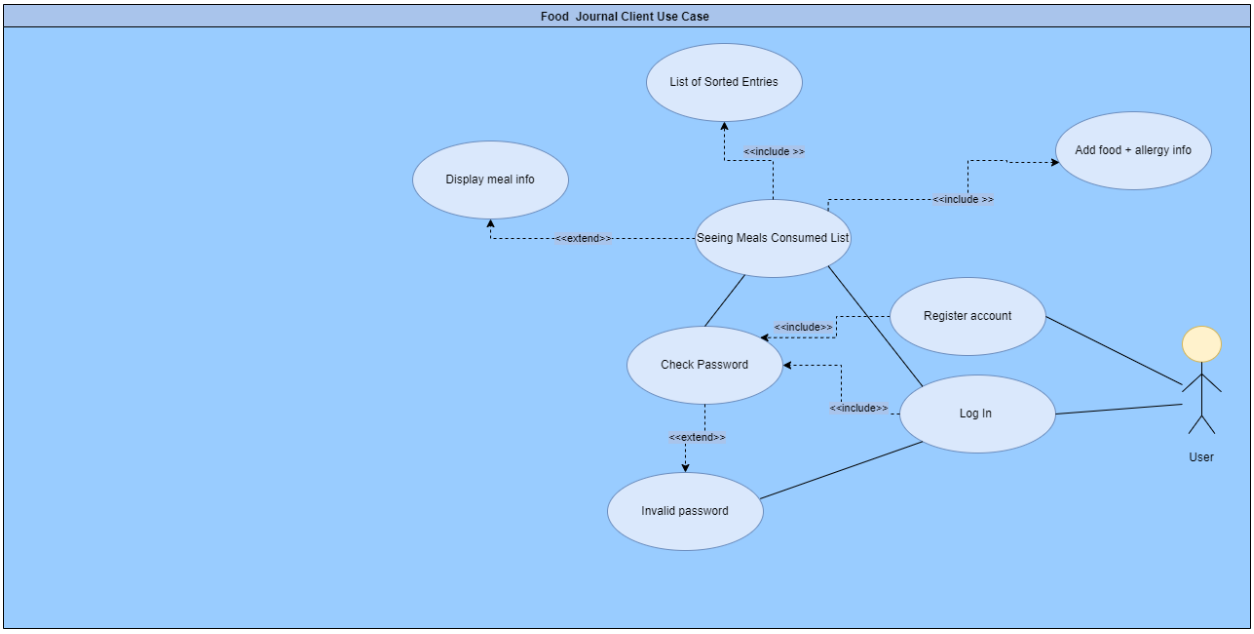
CRITERION B: DESIGN

USE CASE DIAGRAM:

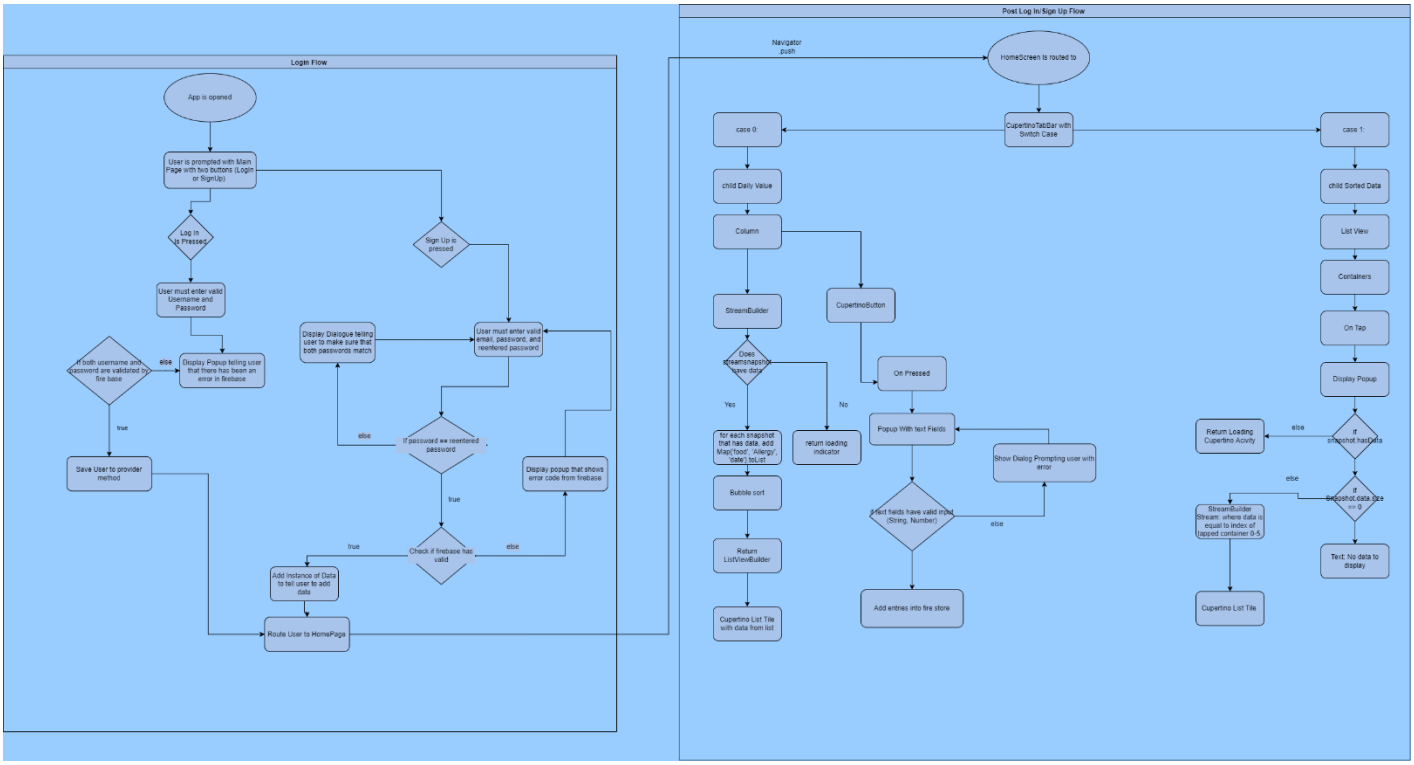


To see previous iterations of the Use Case Diagram, see **Appendix D.1.**

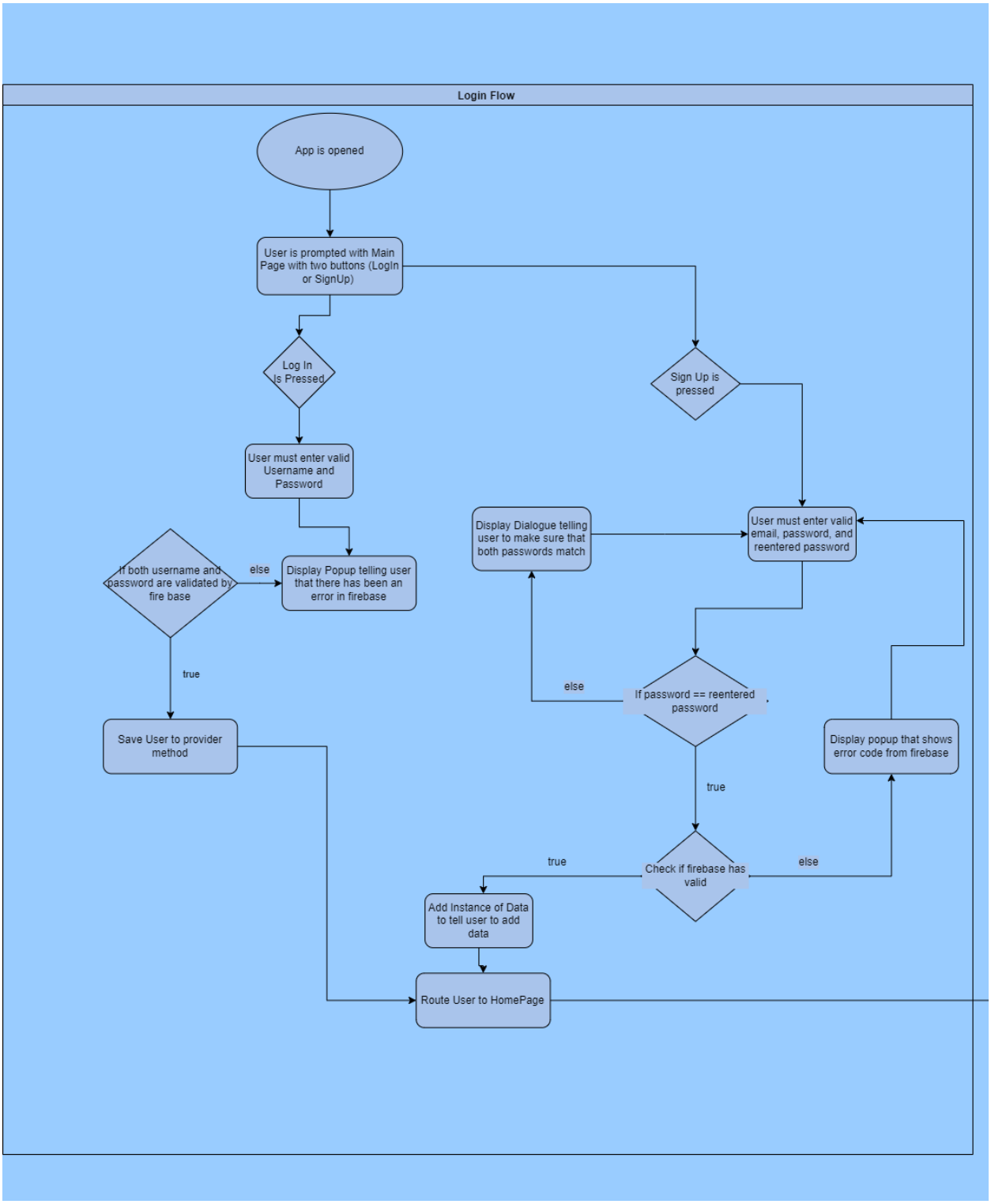
FLOW CHARTS

Below there are three different flow charts. The one titled “Overall” app flow, displays how the two flows should function together. It displays how process of logging in goes to the home screen where the user can see data and entry data. It demonstrates how the switch case code in the navbar connects the pages within the “Post Log In flow”. “Case 0:” and “Case 1:” both represent routing to different pages, when the correct icon or, “case” is selected. Versions for all of these documents can be found in **Appendix D.2.**

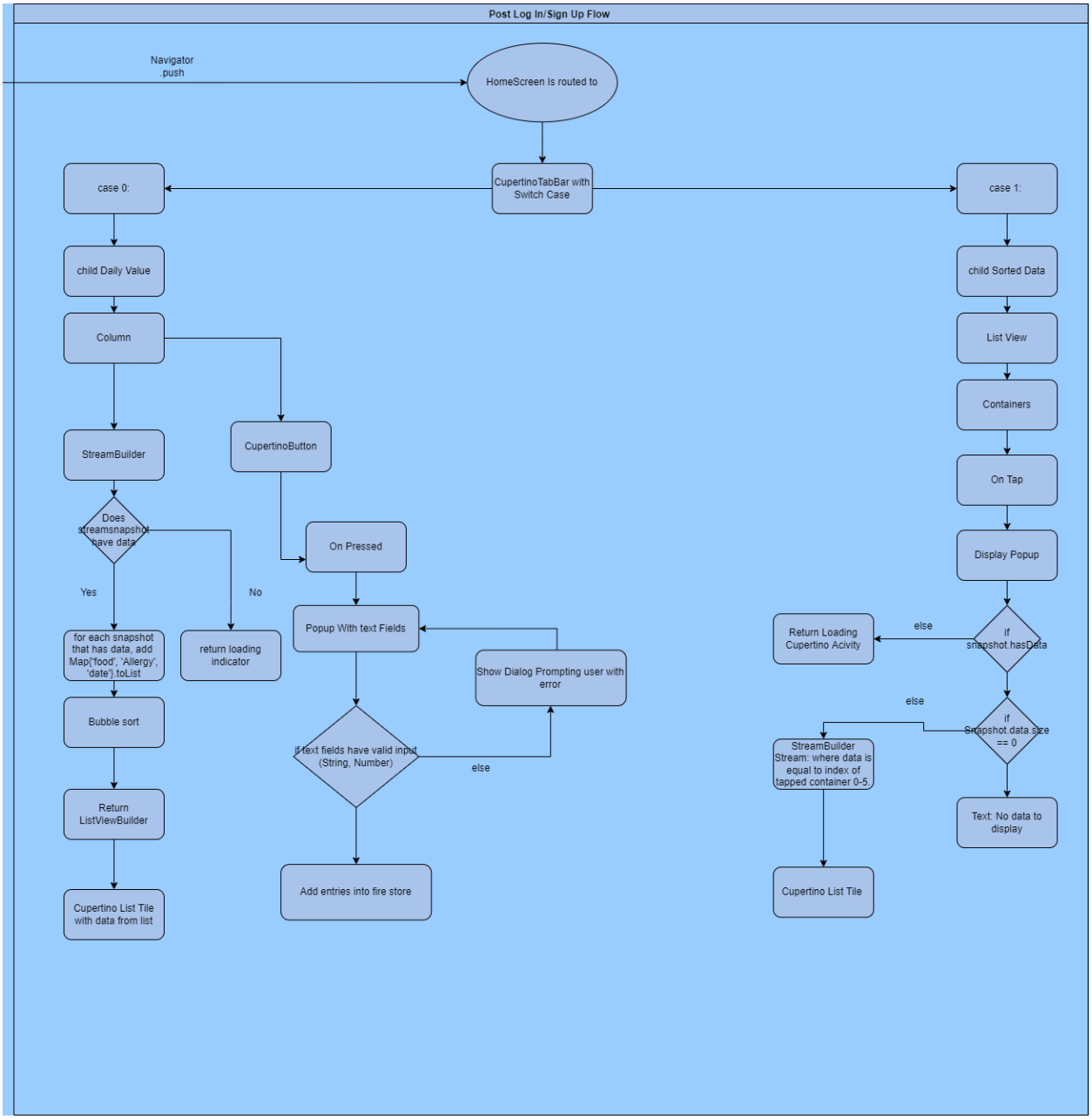
Overall App Flow:



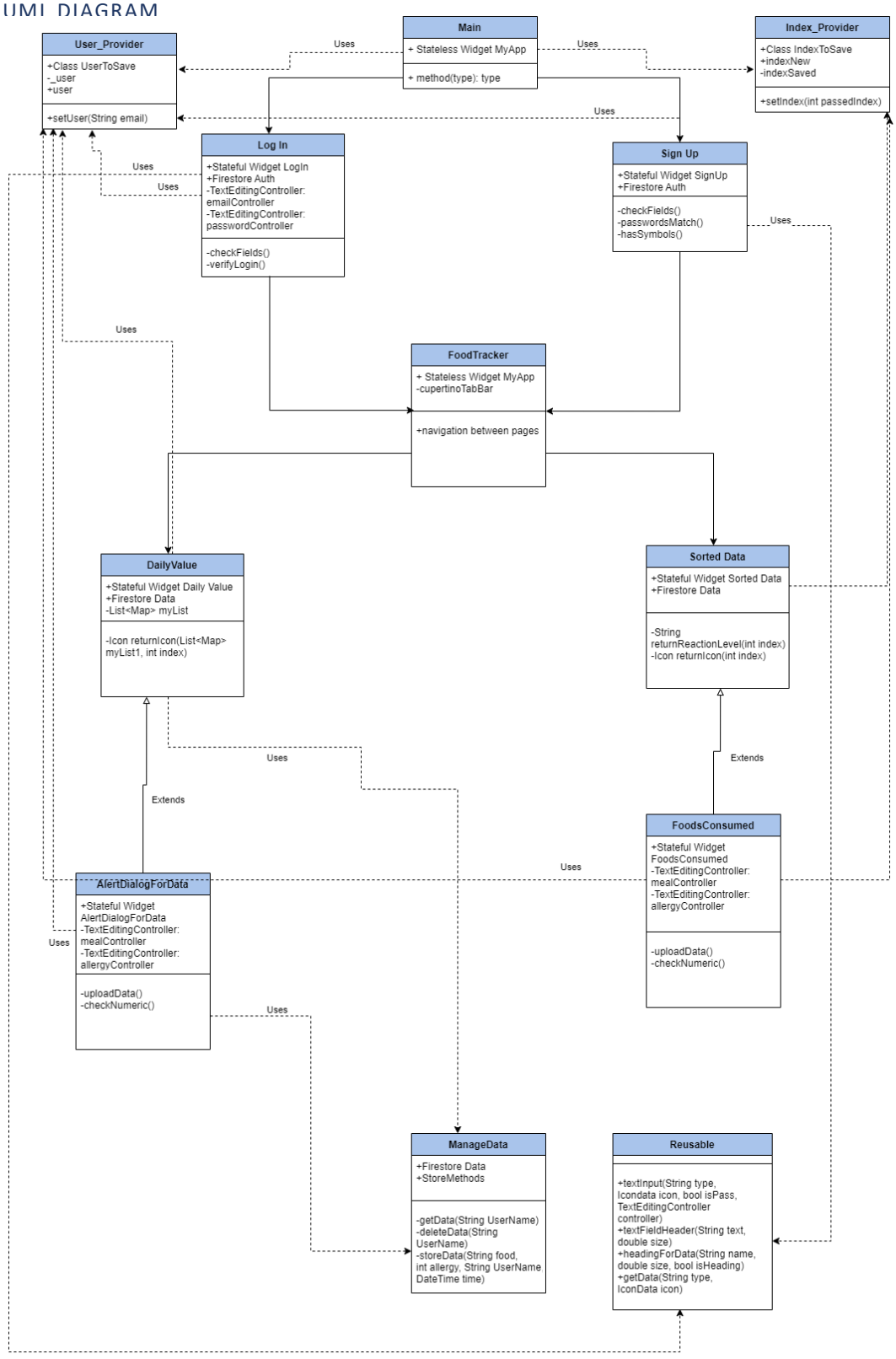
Log In/Sign Up Flow:



Post Log In/Sign Up Flow:



UML DIAGRAM



Previous Iterations can be seen at Appendix D.3

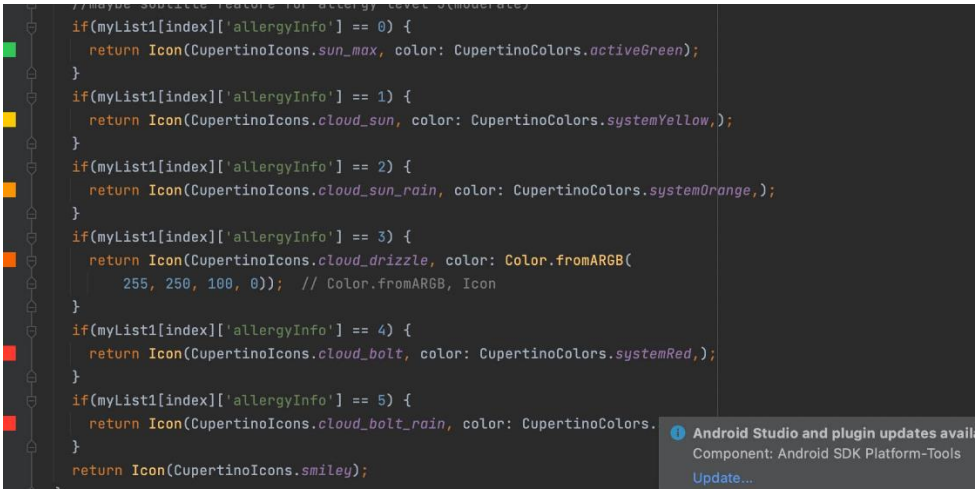
OBJECTS

Class	Purpose/Breakdown
Main	• Contains MyApp => has two buttons one that routes to signup one that routes to log in. MyApp contains routes [MainPage, LogIn, SignUp]. • MultiProviers[UserToSave, IndexToSave]
Log In	- TextEditing Controllers => emailController, password Controller, for text field validation • Contains LogIn => Cupertino Text Fields (From reusable widgets), Cupertino Button for submit • On Button Press => Validate textfields using firebase. If there's an error, pass it to CupertinoAlertDialog. Set UserProvider to email Controller. • Routes to MainPage
Sign Up	- TextEditing Controllers => emailController, password Controller, confirmPassController for text field validation • Contains SignUp => Cupertino Text Fields (From reusable widgets), Cupertino Button for submission of all fields. • On Button Press => Validate textfields using firebase. If there's an error, pass it to CupertinoAlertDialog. Set UserProvider to email Controller. • Create Entry of data to avoid errors with Streambuilder on home page. • Routes to MainPage
MainPage	• Contains CupertinoTabBar that routes user to Either SortedData or DailyValue, depending on which switch case value is returned.
Daily Value	- List <Map> myList = [];, for storage of maps from firebase - StoreMethods for returning data - For loop to add maps to myList - For loops for bubble sorting data from lowest allergy to highest allergic reaction • Contains Daily Value (widget) which has: StreamBuilder for returning data, list view for displaying organized myList, CupertinoTile (from library) to show data information => on tap displays information card with upload date • Cupertino Button => on pressed, displays AlertDialogForData as popup.
AlertDialog ForData	- TextEditingControllers: mealController and allergyController for data entry. - StoreMethods => for uploadData() to allow for storing data to firebase\ - Complex Selection with nested If's to add data or return error based on predetermined criteria

	- checkNumeric for making sure allergy level is a numeral value • Container for displaying TextFields, Submit Button with sleek UI
Sorted Data	- returnIcon returns icon based on passed index of container - returnReactionLevel returns string based on index of container, ensuring that the list builder displays correct text for each container • Series of containers that contain gesture detectors, that on pressed will navigate user to FoodsConsumed Popup
FoodsConsumed	- returnIcon returns icon based on passed index of container clicked • Container that contains a Streambuilder => Stream of data based on provider value of index, that is set when container is clicked. Utilizing Firestore's library that checks that value of allergyLevel is equal to that of the corresponding clicked container. • If app has no data for corresponding container, the text will display "no data"
ManageData	- Class called StoreMethods - Contains series of functions o storeData => Try-Catch, that uploads data to firebase, but if there's an error, it will return error. o getData => Get snapshots from user subcollection o deleteData => delete document (not implemented yet)
UserProvider	- Class called UserToSave - Contains provider statements for passing data between screens. - setUser => setUser to passed in "email (String)"
IndexProvider	- Class called IndexToSave - Contains provider statements for passing data between SortedData and popup - setIndex => setIndex to passed in "email (String)"

ALGORTHIMS

There are a multitude of algorithms used throughout the program, all for minor tasks, like if statements to set Icons or Text. Examples of those can be found in my object breakdown where I go tell you what they do. An example of one of those simple algorithms to set an Icon is below:



But as for the more complex and worthwhile algorithms, I have four main components.

1. A for loop to add a group key value pairs (map) to a List. The pairs are grabbed from firestore and are passed into the loop via the loops index (i).
2. A bubble sort that compares the map values of the certain index, until the entries are sorted from the lowest level of reaction to the highest level of reaction.
3. Adding data into Firestore via Future<void> method.
4. Sorting Firestore data via the .where() operator in the Frestore library.

All these algorithms are broken down in Criterion C.

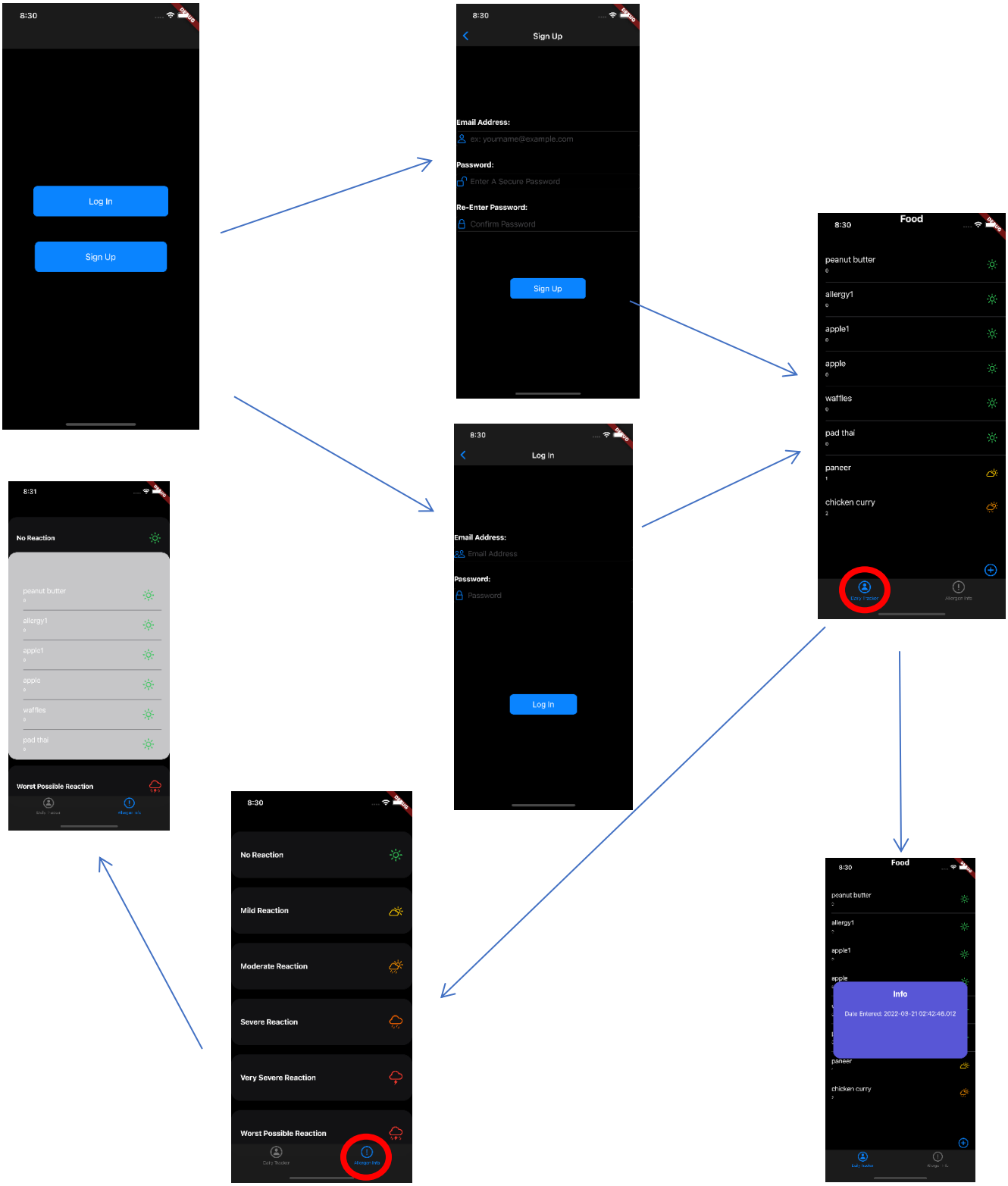
DATA STRUCTURES

My main data structure is the use of Google's Cloud Firestore. The data that I upload is user-specific via collection of users who all have their own sub collections, where their own data is present. I choose Cloud Firestore for the reason, that it is well documented in use with flutter, and is very simple to integrate. I figured given the time frame and style of the app using Firestore would be the most optimal way to save my data. In succession to this my main home page, as described above, takes all the data from cloud firestore and adds it as a map into a list which is then sorted. My rationale for this was that to meet success criteria 7, my client has specified that data should be stored from lowest allergic reaction to highest.

The specifics on how the data structure is set up can be seen within Criterion C, where I go in depth on the sub collections and collections aspect of my application.

UI FLOW

UI Flow Demonstrating how user could potentially navigate the app. Displays all tabs. Progression of UI can be seen in Appendix C. (Appendix C.1, Appendix C.2)



TEST PLAN

Success Criteria	Test Case
1. Ability for user to input meals/food consumed, in addition to inputting, viewing meals from across multiple dates should be easy and clean.	While testing the app the user should be able to input meals consumed. The data should return in a timely matter (< 5 seconds). If the food displayed is tapped, the date should be displayed in a popup, without any errors.
2. Having a separate page where user can see foods that have caused reactions.	The user can see a list of foods organized by the intensity of their reaction, via the Sorted Foods Tab. The data should display immediately and should be separate from the list of all the entries. If there are no entries for the desired intensity, the app should not crash.
3. User is able to create an account, where all their data is stored. They can input a username and password, to log in. This data can be transferred across devices. Setting up an account should require an email, and a password that is at least 6 characters.	During testing this phase will be heavily focused on. I will demonstrate that a user can: a. Create an account via the Sign-Up Page. b. Log in to their created account using username and password c. Show that multiple users can use the same device, but display different data (substitute for shared across different devices) d. Show that app handles bad data entry for log in (no email, no password, incorrect password, incorrectly formatted email, no user, not matching passwords on reenter password, entering a password less than 6 characters, user already present).
4. When the application is closed, the user should be logged out, and will have to relog in, however all their data will still be present.	While testing the app, I will run a hot restart to simulated leave the app, to save time, however the user will have to relog in. Upon relogging in the data entered on previous entries will persist and will still be under the user's profile.
5. Errors are handled, without the app crashing. If a user were to input an empty meal for example, the program would prompt the user to input a meal or cancel.	Since log in/sign up page errors would have already been accounted for, I will display many different ways a user can input bad data within the adding food tabs. This will include: a. Empty Food Value b. Empty reaction level c. Non-numeral entry for reaction level rather than expected integer d. Integer that is out of range (0-5) Each issue should result in a dialog box displaying that there has been an error, to be considered successful.

6. User is able to information about meals, such as the date uploaded, and allergy level.	Upon clicking on the desired food entry, the user will be able to see the date of the entry. The reaction level will be displayed with the food's name.
7. The user's inputted meals should be sorted by reaction level to provide an idea of how they have reacted to different allergens.	Data that has been added will be sorted from lowest to highest reaction levels. As data is added, it will remain sorted in this order without erroring out.