CappuCoffee (https://github.com/JustGinger888/CappuCoffeeProject)

Hosting URL: https://cappucoffee-40101.web.app (https://cappucoffee-40101.web.app)
Github Repo: https://github.com/JustGinger888/CappuCoffeeProject (https://github.com/JustGinger888/CappuCoffeeProject)

Setup

Install Dependencies

```
npm install
```

Serving and running locally

```
npm run serve
```

Test Accounts

Username	Password	GroupID
tester@test.com	password	lyFJC18ZXh7gAFWeGFuf

Introduction

While interning at TouchBase ltd over the summer of 2019 one of my most frequent duties included preparing and serving coffee to the small team I was assigned to, a rather common task associated with the intern role in any field. The process would usually consist of me asking for their orders and noting them down on my phone, something that proved to be rather time-consuming as their orders would often differ on a day to day basis.

This proved to be a common occurrence not only for me but also for other friends who had the same experience in their internships. I ran a short survey among a focus group of ten interns and 8 of them said that they have been responsible for the preparation of coffee on multiple occurrences, a further 7 went on to agree that the team's orders would change frequently. The teams would often change the strength of their drinks and the amount of sugar they wanted based on the time of day and their overall feeling.

Hence why I decided to produce a web-based solution where employees can create and join teams to add and customize their respective coffee orders to a backend database. Allowing for easier access to the list for interns, as well as saving the time it would take for the team to discuss and give their orders to interns.

Design Decisions

I have decided to teach myself due to use as a framework for this project. This decision was mainly done due to me wanting to expand my own knowledge on single page web apps and their frameworks. But it does provide me with associated features such as:

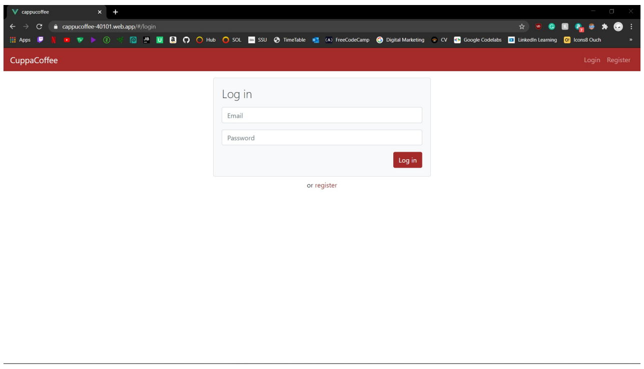
- No extra queries to the server to download pages.
- User-friendly design and navigation
- Performance improvement

As for my database decision, I opted to use firebase due to its ease of associated hosting and integrated authentication to make use of. Allowing me to more comfortably create associated user records and such. The security rule feature also makes it much more secure to use for the given use case when compared to solutions such as mongo atlas. Finally, the whole Firestore structure is Real-time and will load data dynamically no matter which user was to upload.

System Overview

Authentication

I have made use of firebases own auth services to allow for email and password login, this could in feature allow for easy OAuth methods with respective accounts. The reason for adding authentication is to improve security and create user-specific records that are unique to their needs. This also ensures that all groups can only be accessed and joined by registered members.



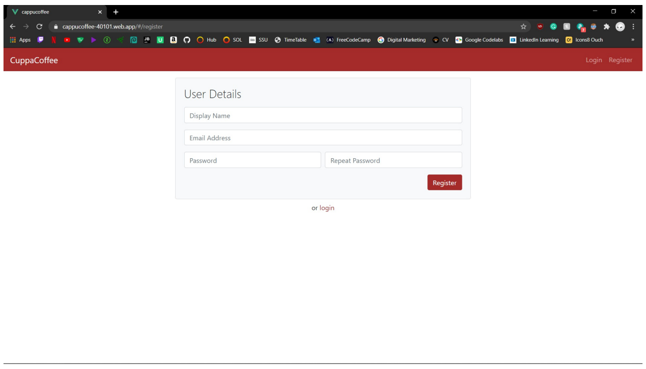
Log in

Email

Password

Log in

or register



User Details

Display Name

Email Address

Password | Repeat Password

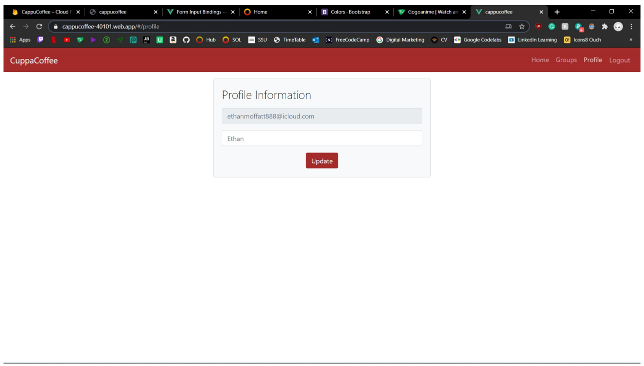
Register

or login

Web Interface

Profile

In the profile section, I make use of an update method to allow users to change their respective display name as required. This is then displayed as the new placeholder and on the home page above the introduction Log. This is how I make effective use of the update functionality found in the CRUD cycle.

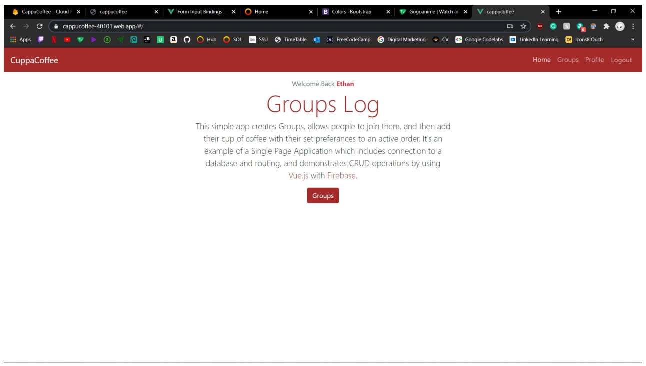


Profile Information

ethanmoffatt888@icloud.com

Ethan

Update

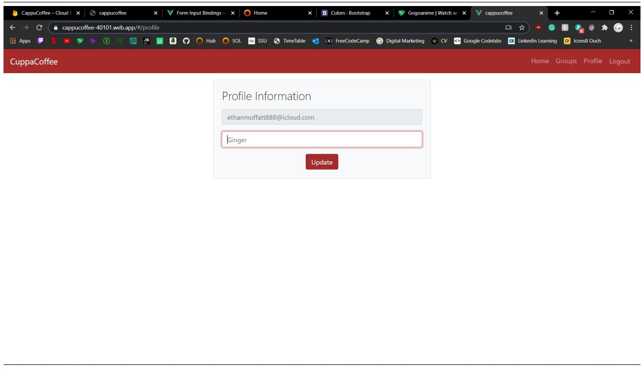


Welcome Back Ethan

Groups Log

This simple app creates Groups, allows people to join them, and then add their cup of coffee with their set preferances to an active order. It's an example of a Single Page Application which includes connection to a database and routing, and demonstrates CRUD operations by using Vue.js with Firebase.

Groups

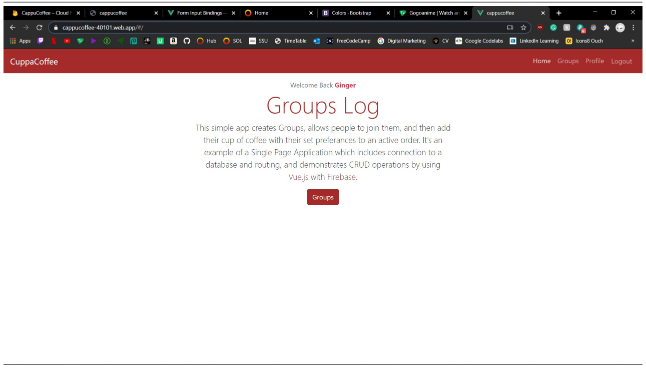


Profile Information

ethanmoffatt888@icloud.com

Ginger

Update



Welcome Back Ginger

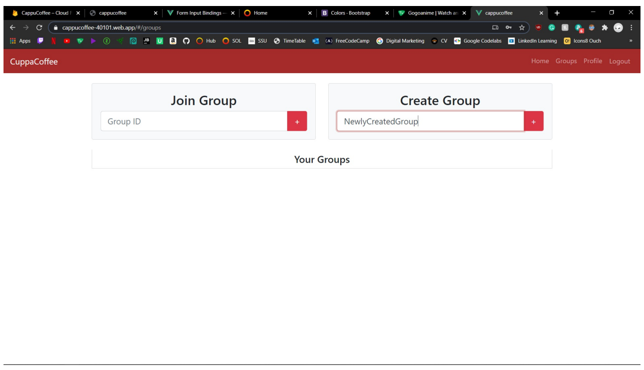
Groups Log

This simple app creates Groups, allows people to join them, and then add their cup of coffee with their set preferances to an active order. It's an example of a Single Page Application which includes connection to a database and routing, and demonstrates CRUD operations by using Vue.js with Firebase.

Groups

Groups

The group view is responsible for creating and joining a group. As well as listing them and their options below. The joining of a group would require you to know the ID of a group and then enter it to add it to the users joined group lists in the backend, displaying it after. Moreover, the creation of the group only requires you to add a name for the group and then it creates the reference in the user-groups collection and a new entry in the group's collection.



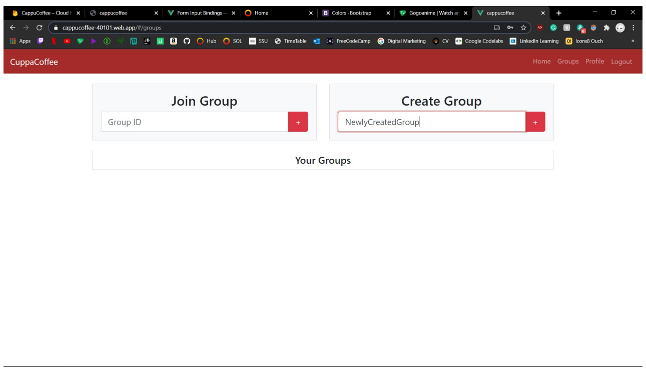
Join Group

Group ID +

Create Group

NewlyCreatedGroup +

Your Groups



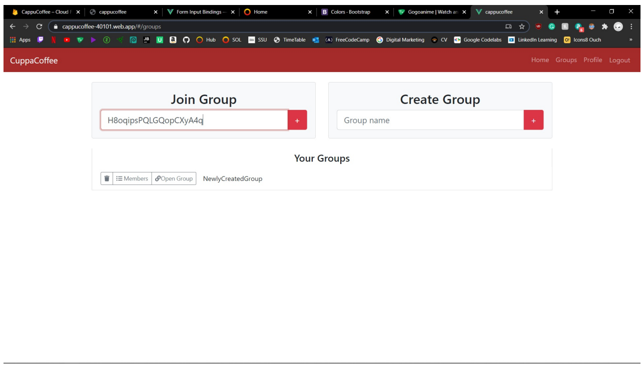
Join Group

Group ID +

Create Group

NewlyCreatedGroup +

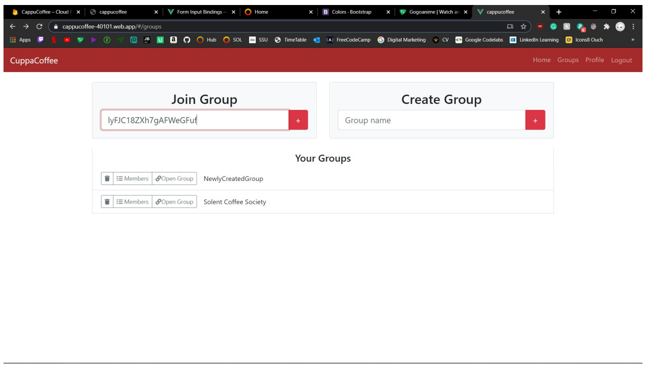
Your Groups



H8oqipsPQLGQopCXyA4q

Group name

NewlyCreatedGroup



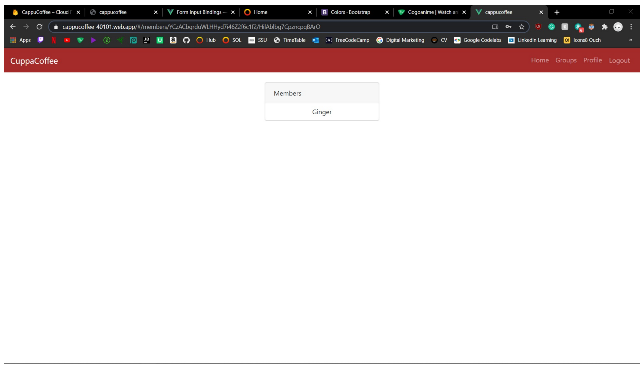
lyFJC18ZXh7gAFWeGFuf

Group name

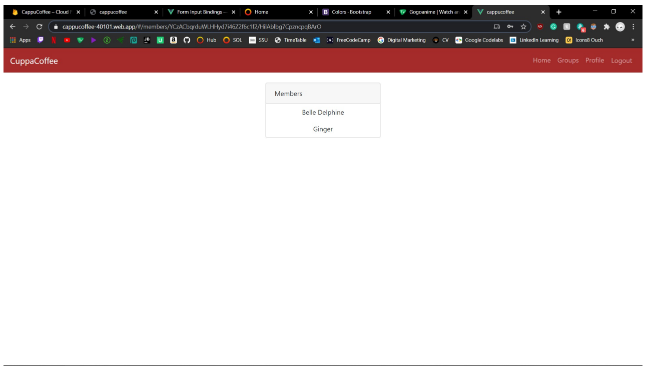
NewlyCreatedGroup

Solent Coffee Society

Moreover, each group that gets listed displays the name of the group and three buttons. The delete button is responsible for removing the group record from the user-groups collection and removes it from being displayed in the list. The member's button displays a list of users that are currently joined in the group and gets this from the groups-members collection. The view order gets the current user uid and the group uid to navigate to the associated drinks list of the group which will be explained next.



Ginger



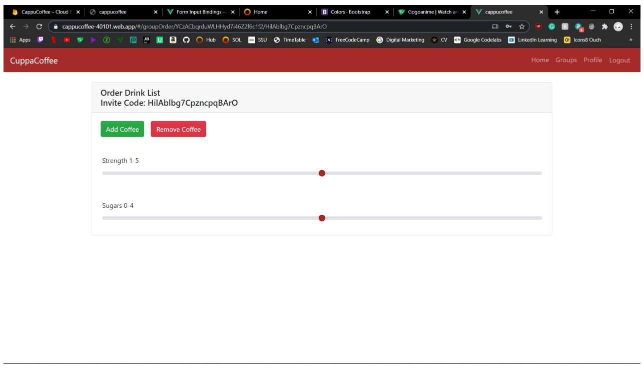
Belle Delphine

Ginger

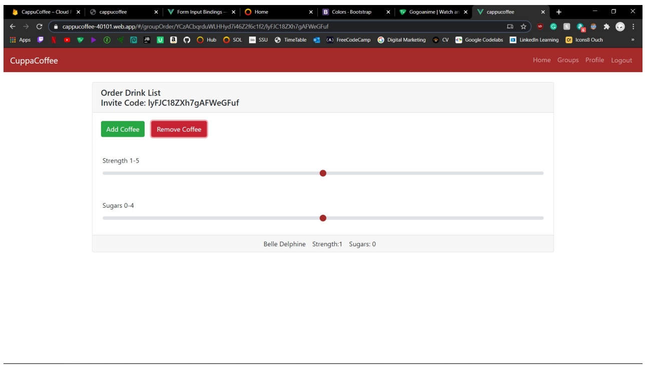
This page demonstrates the Create, Read and delete functionality of CRUD and is a clear demonstration thereof, further showing how two databases can be connected and create unique records for users.

Orders

The orders view uses, as mentioned above, the users UID and group UID to navigate to the associated view and display a drink list with two input fields to specify the strength and sugar of coffees. when these choices are made and the add button has been clicked the list gets populated with the drink accordingly. If any other changes are made to the drink the user would just have to click add again to dynamically update a drinks values.



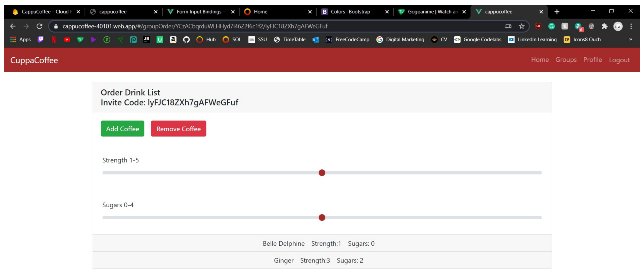
Order Drink List
Invite Code: HilAblbg7CpzncpqBArO



Order Drink List
Invite Code: lyFJC18ZXh7gAFWeGFuf

Belle Delphine Strength:1 Sugars: 0

The remove button on the other hand deletes a user's specific drink from the database and allows then dynamically updates the list for all users. This would mean that this view singlehandedly demonstrates all CRUD functionality and allows for a clear use case thereof.


CuppaCoffee

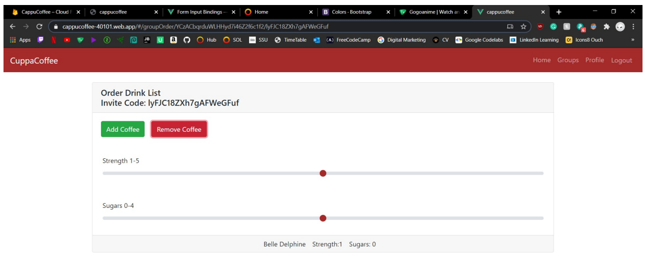
Order Drink List
Invite Code: lyFJC18ZXh7gAFWeGFuf

Strength 1-5

Sugars 0-4

Belle Delphine Strength:1 Sugars: 0

Ginger Strength:3 Sugars: 2


CuppaCoffee

Order Drink List
Invite Code: lyFJC18ZXh7gAFWeGFuf

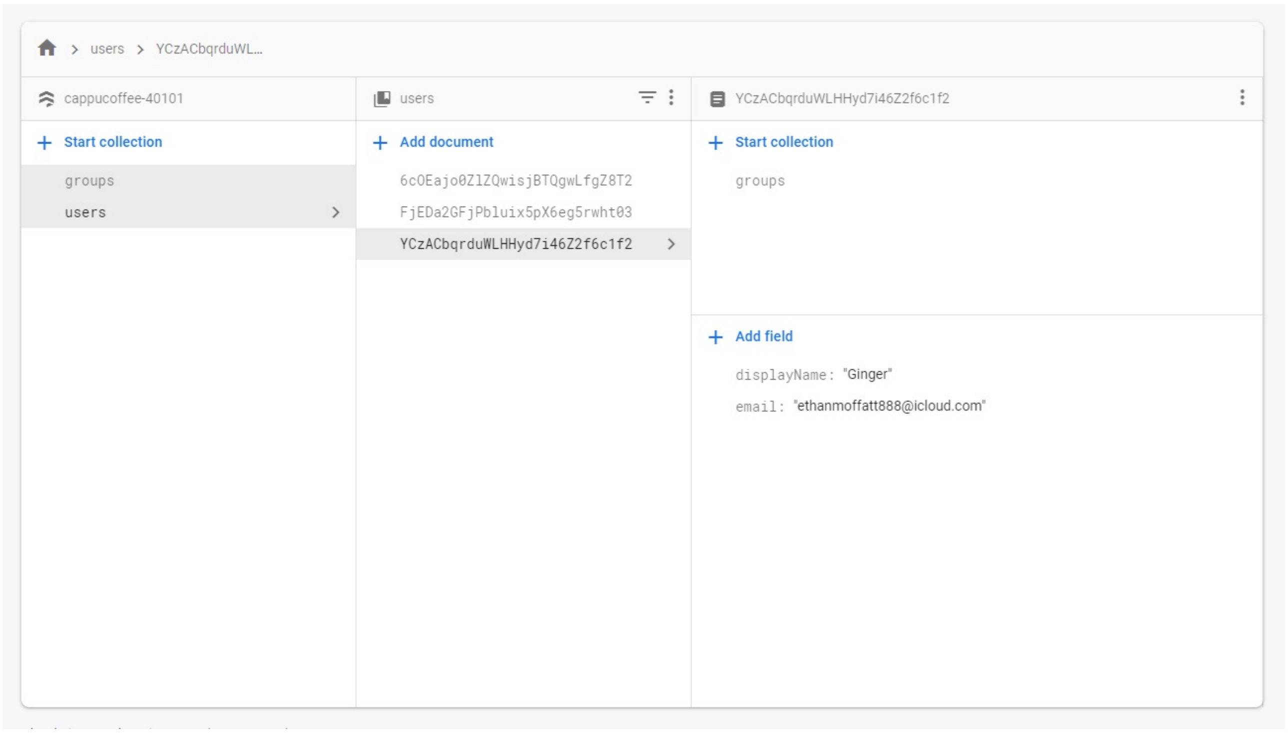
Strength 1-5

Sugars 0-4

Belle Delphine Strength:1 Sugars: 0

Database

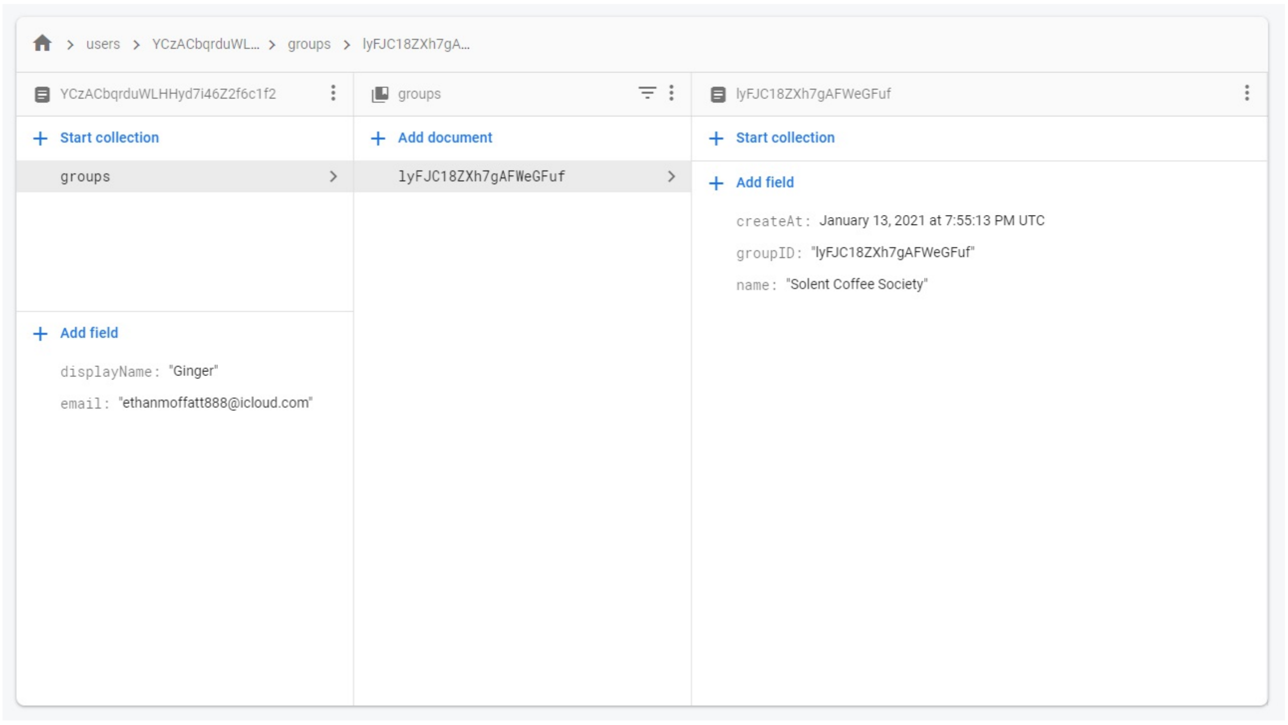
For this project, I decided to use firebase as a backend database solution, as it proved to be a secure and scalable solution with built-in hosting. The database itself consists of 2 main collections, as seen below, of which the "groups" collection relies on the "user" collection.



displayName: "Ginger"
email: "ethanmoffatt888@icloud.com"

User Collection

The user's collection creates a record upon a clients registration where it stores the user's name and email respectively in a document identified by their UID. The user collection also has a "groups" sub-collection where a newly created or joined groups information gets stored, the document itself is again identified by the UID of the group and stores its associated name for display purposes. There is data duplication due to the GroupID also being present, but I added the field only for demonstration purposes as it would be redundant to make use of it in the production format.


users > YCzACbqrduWL... > groups > lyFJC18ZXh7gA...

YCzACbqrduWLHHyd7i46Z2f6c1f2

groups

displayName: "Ginger"
email: "ethanmoffatt888@icloud.com"

groups

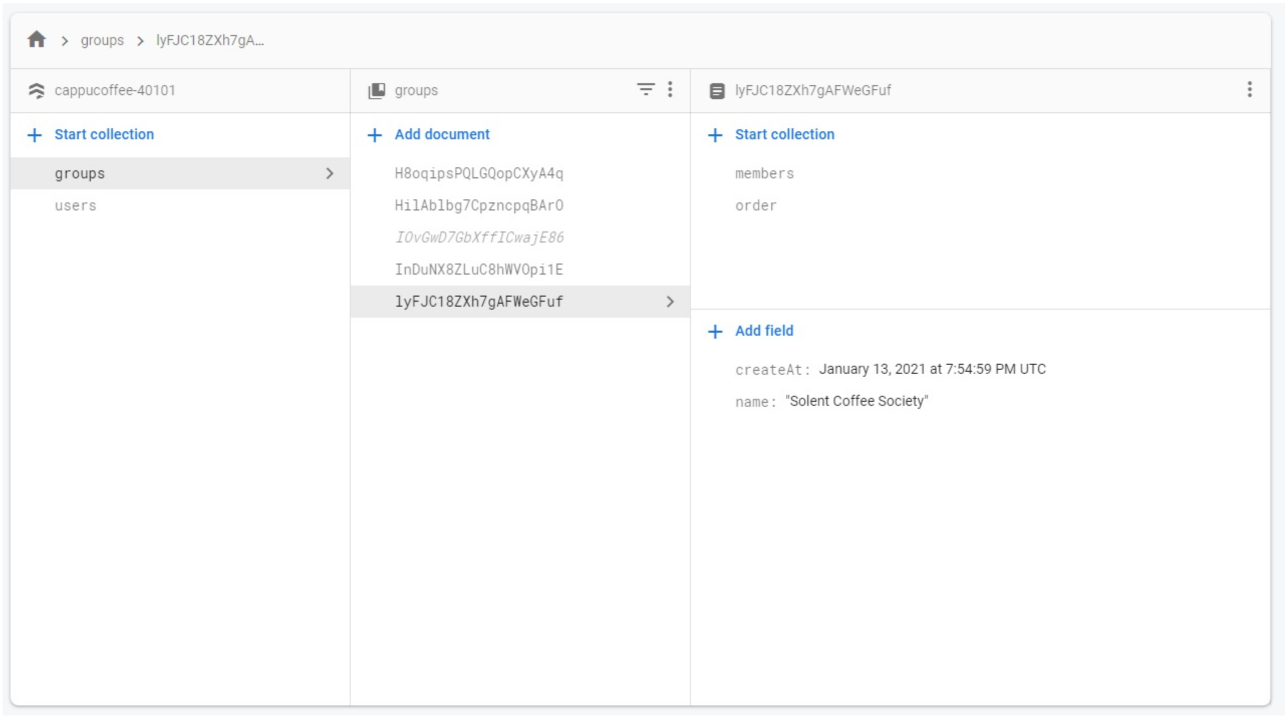
lyFJC18ZXh7gAFWeGFuf

lyFJC18ZXh7gAFWeGFuf

createAt: January 13, 2021 at 7:55:13 PM UTC
groupID: "lyFJC18ZXh7gAFWeGFuf"
name: "Solent Coffee Society"

Groups Collection

As for the "groups" collection, it stores data that is relevant to the group and makes use of 2 sub-collections to display associated members and orders. This allows me to keep users independent from in case I wanted to expand the associated information for them. While also allowing for multiple groups to exist independently even if the founding member were to leave.


groups > lyFJC18ZXh7gA...

cappucoffee-40101

groups
users

groups

H8oqipsPQLGQopCXyA4q
HilAblbg7CpzncpqBArO
IOvGwD7GbXffICwajE86
InDuNX8ZLuC8hWVOpi1E
lyFJC18ZXh7gAFWeGFuf

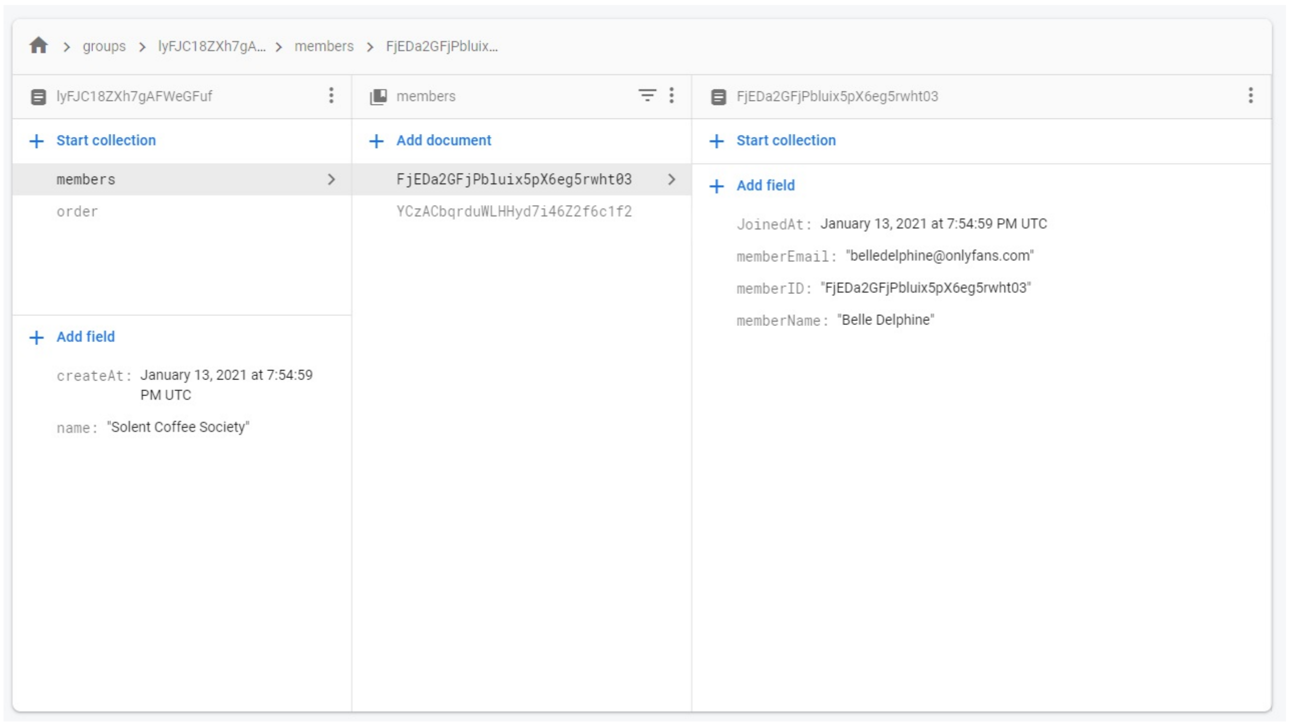
lyFJC18ZXh7gAFWeGFuf

members
order

createAt: January 13, 2021 at 7:54:59 PM UTC
name: "Solent Coffee Society"

Members Sub-Collection

The member's sub-collection lists all members that have joined the group, storing a bit of redundant data that is only there for my own testing and demonstration purposes. It is also clear that the documents are all uniquely identifies using the joined users UID. This allows for easy access and removal of their data once they have left.


groups > lyFJC18ZXh7gA... > members > FjEDa2GFjPbluix...

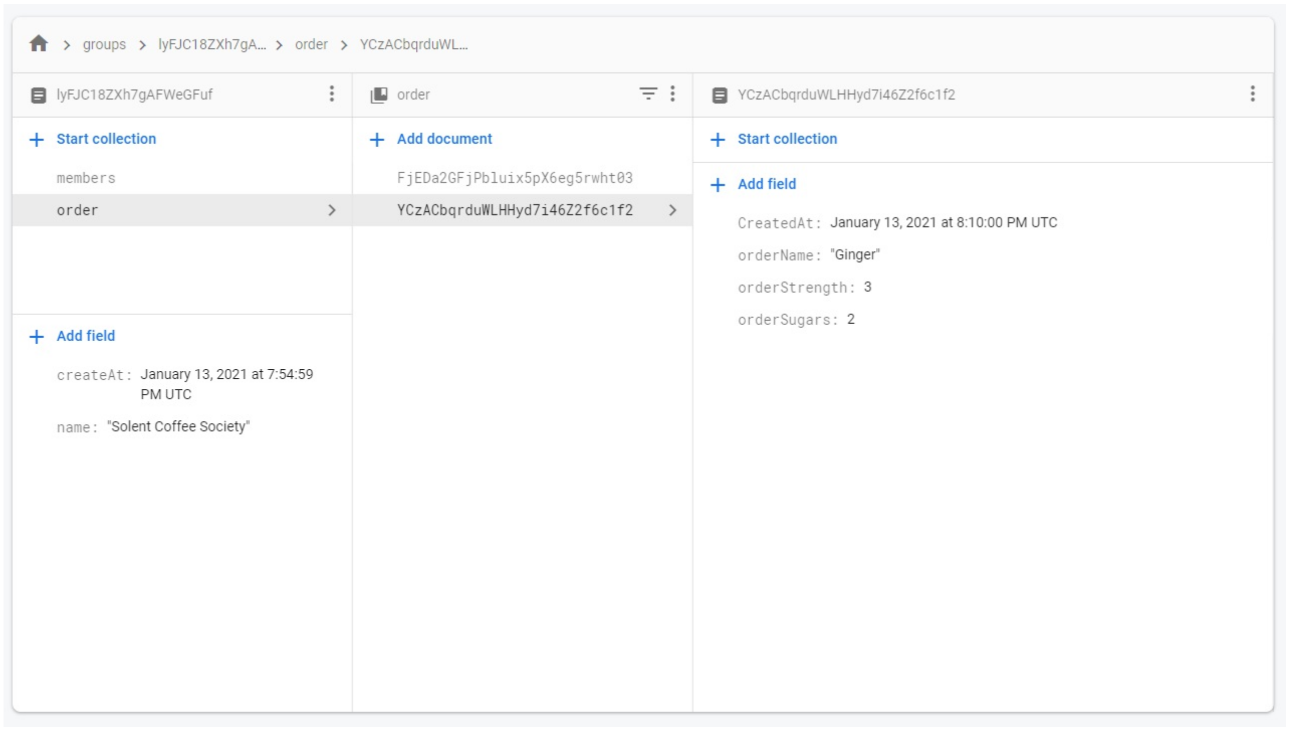
lyFJC18ZXh7gAFWeGFuf
Start collection
members
order
Add field
createAt: January 13, 2021 at 7:54:59 PM UTC
name: "Solent Coffee Society"

members
Add document
FjEDa2GFjPbluix5pX6eg5rwht03
YCzACbqrduWLHHyd7i46Z2f6c1f2

FjEDa2GFjPbluix5pX6eg5rwht03
Start collection
Add field
JoinedAt: January 13, 2021 at 7:54:59 PM UTC
memberEmail: "belledelphine@onlyfans.com"
memberID: "FjEDa2GFjPbluix5pX6eg5rwht03"
memberName: "Belle Delphine"

Orders Sub-Collection

This sub-collection contains all of the data associated with the coffee order the user has placed and again is stored in a document that contains the UID of the user to allow for easy updating or removal of data. Most importantly it stores the numbers of sugar and the strength of the drink to be displayed in the drink list.


groups > lyFJC18ZXh7gA... > order > YCzACbqrduWL...

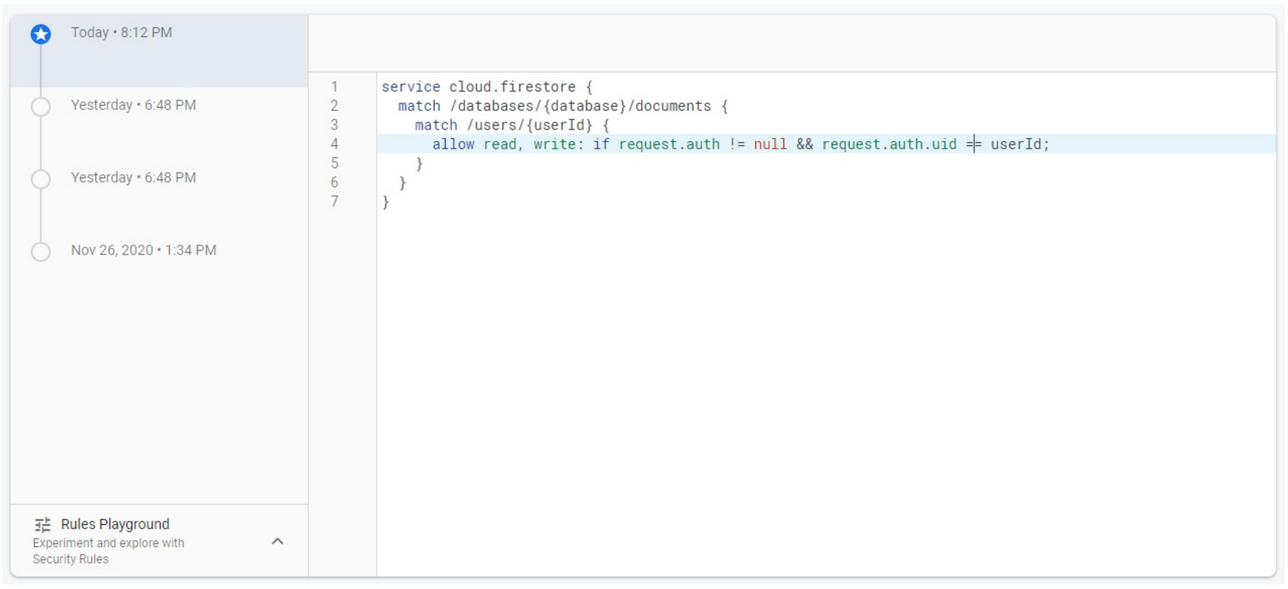
lyFJC18ZXh7gAFWeGFuf
Start collection
members
order
Add field
createAt: January 13, 2021 at 7:54:59 PM UTC
name: "Solent Coffee Society"

order
Add document
FjEDa2GFjPbluix5pX6eg5rwht03
YCzACbqrduWLHHyd7i46Z2f6c1f2

YCzACbqrduWLHHyd7i46Z2f6c1f2
Start collection
Add field
CreatedAt: January 13, 2021 at 8:10:00 PM UTC
orderName: "Ginger"
orderStrength: 3
orderSugars: 2

Security

To ensure that only logged-in and authenticated users can read and write data to the firestorm, I ensured to write my own security rules that only users that have logged in and have a UID that matches with their own record UID can post and update user records. Hence ensuring that no data can be leaked or accessed by malicious people.



Today • 8:12 PM

Yesterday • 6:48 PM

Yesterday • 6:48 PM

Nov 26, 2020 • 1:34 PM

```
service cloud.firestore {
  match /databases/{database}/documents {
    match /users/{userId} {
      allow read, write: if request.auth != null && request.auth.uid == userId;
    }
  }
}
```

Conclusion

I believe that I have successfully achieved my goal of creating an interface for the above problem. As it can create and join groups without a problem and create drink lists in real-time. Setting out to complete all my initially stated goals. This also demonstrates an advanced application of CRUD as I can join any group that has been created.

I did make use of Firebase functions to enhance security and better features but had to get rid of them due to the payment plan associated with firebase functions now. this did mean I lose some security checks when it comes to the processing of data, meaning they could potentially be changed before the writing process.

If I were to improve this project further I would have removed redundant data used to demonstrate and test my database. As well as add a better interface, though this would be out of scope for this module. The security rules would also need to be updated slightly to ensure there is no vulnerability within the system.