Dustin Crosson, drc518
Justyn Pollard, jmp447

CMPT 436 Project Documentation

API's Used:

- Context API

Cloud Services Used:

- Google Firestore in version 1
- Back4app Parse in version 2

Language:

- React-Native

Other Tools:

- Expo Go was used to build and run the application on IOS devices

Testing:

Both versions were run on an IOS device and verified to be working at the time of handin.

Test Cases:

- User can sign up ==> test passed
- User can exit the app and log back in ==> test passed
- User cannot re-sign up with the same email ==> test passed
- User is taken to home screen upon successful login ==> test passed
- User can search for other members in the program ==> test passed
- User clicks on member, new chat window opens up ==> test passed
- 2 Users can chat back and forth ==> test passed
- From chat another user can be added to the chat ==> test passed
- Chat is maintained after closing and re-staring up the program ==> test passed
- parse version only(user can remove themselves from the chat) ==> test passed

Stepts to Run the Program:

- IOS:
    - Install the expo go app on device from apple store, or use IOS emulator
- Android:
    - Install the expo go app on device from google play store, or use andorid emulator
- navigate to version's main folder
- npm install //install dependancies
- npm start //will open up a window in the browser
- scan the QR code in the webrowser or terminal with your IOS device's camera. This will open the project in the Expo Go app or enter a to start on android emulator

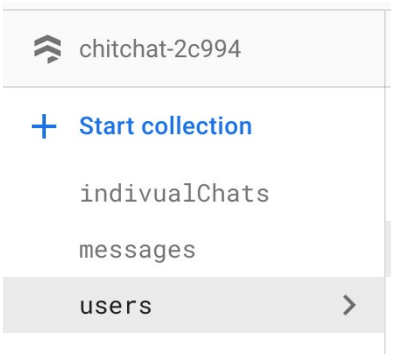
Front End Design:

The front end of the application was designed in react native for easy cross-platform building the mobile application. It can be written in JavaScript and then compiled to run natively for both Android and IOS. When data is required from each respective cloud data storage service, a query is sent. Each request is fulfilled with the data in JSON format, and the program uses the data or manipulates it and updates the cloud. Snapshot listeners are used in the firebase version, and livequeries are used in the Parse version to add event listeners to the backend. This notifies the front end of a change of state, providing an alternative to the front end from constant polling of the cloud.

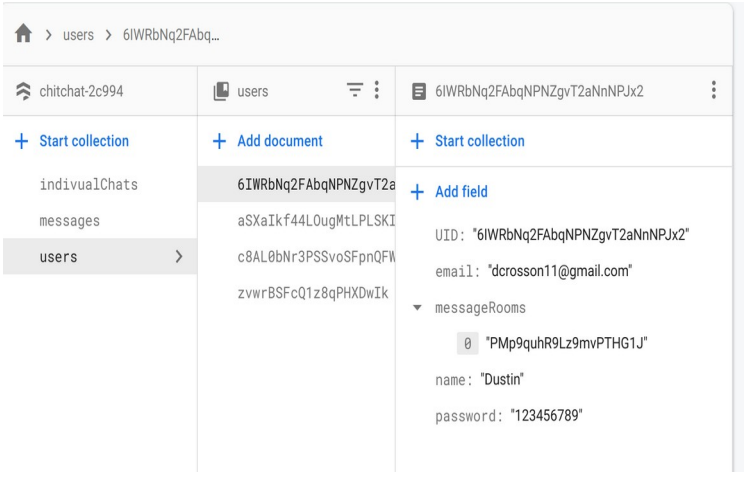
Back End Design: Both backend solutions are essentially Backend as a Service solutions. You sign up, and use querys from the front end to the backend, and the service takes care of everything in the backend.

1. **Google Firestore:**
   Firestore is a close-source NoSql document database cloud storage system created by Google. The main advantage is that realtime database which handles synchrizing all clients that are on the system.

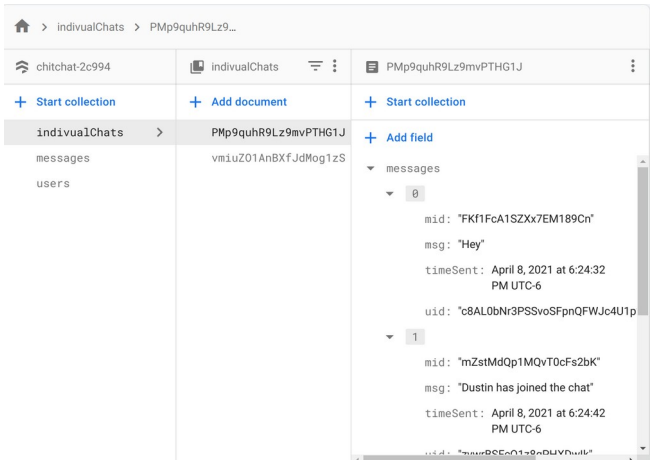
- Collections are used to store a series of documements which are also a collection, or a JSON object.



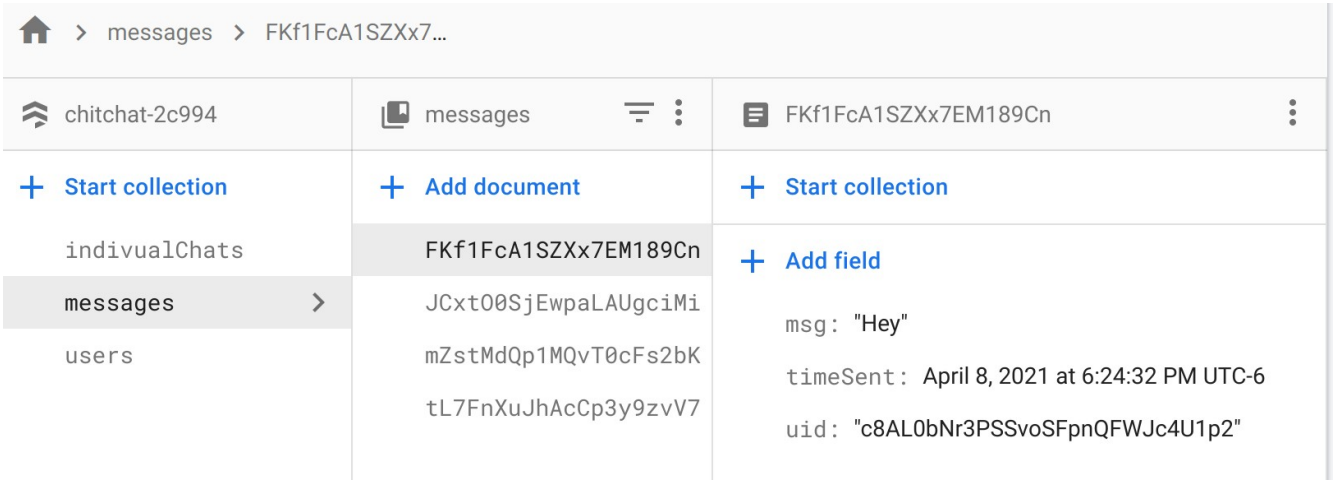
- User Collection: stores information pertaining to the specific user. It is not safeguarded againt being edited by other users .



- IndivualChats collection: stores individual chatRoom information in JSON format
    - also has an array of userIDs which is not seen in the picture

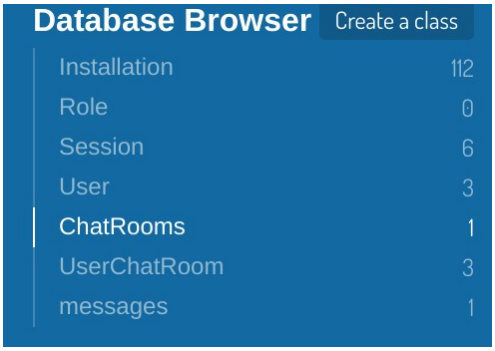


- Messages Collection: Each document is a singular message which is stored in the JSON format.

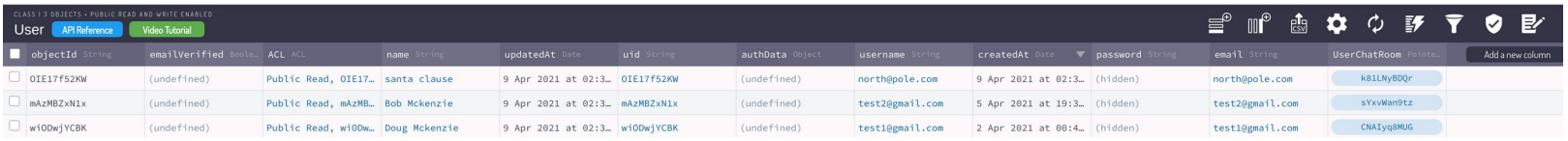


2. **Back4app Parse:**
   BParse is a open-source NoSql databse system created by Back4app. Its main advantage is that it open-source, giving the full control to the devleoper and preventing vendor lock-in.

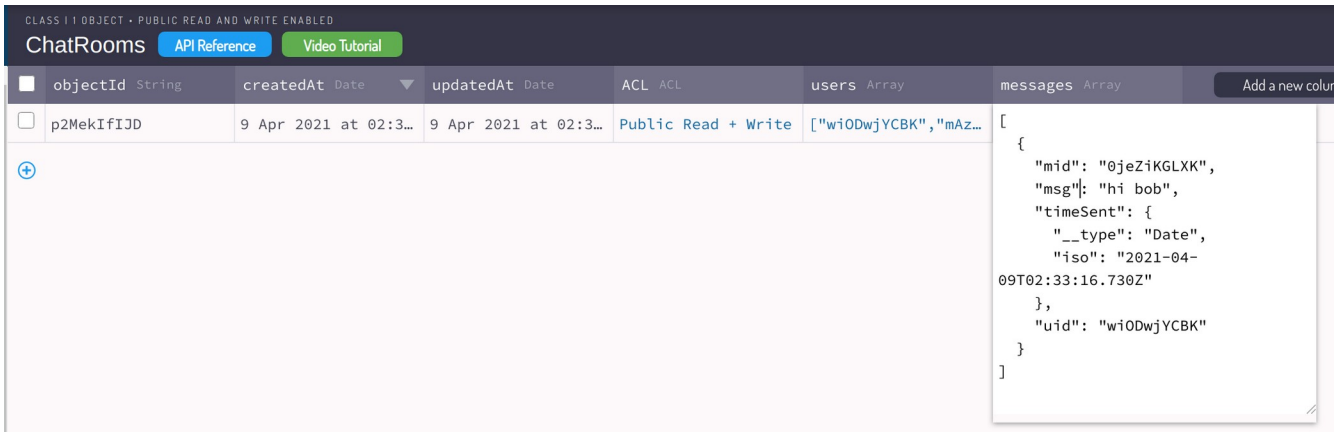
- Classes are created which are similar to documents in firestore. The Installation, Role, Session and User are created by default. User offers pre-defined methods to sign-up and log-in but can also have custom data assinged. Each class can be assigned permsisions in the ACL column. The User comes default that anybody can read it, but only the user that is logged in can edit it.



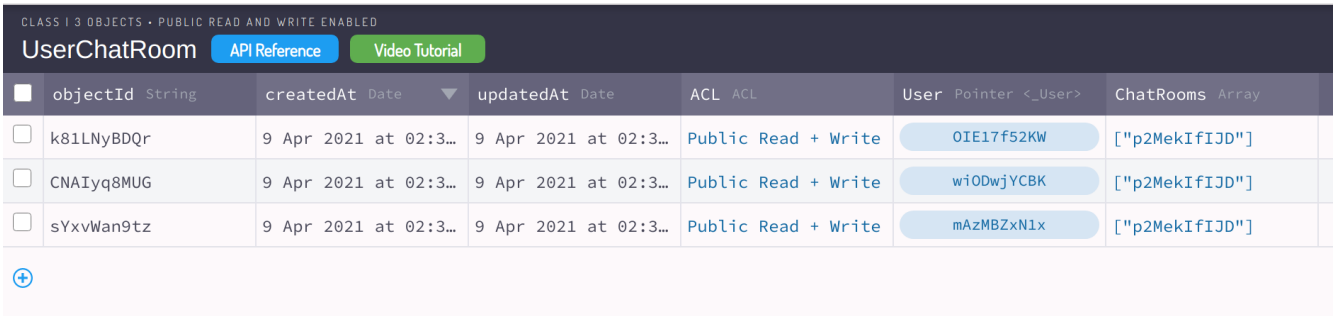
User Class: stores info specific to each user, used by login and sign up methods in front end
- objectId => unique ID assigned by the cloud
- ACL => permissions on that class
- name => assigned at sign-up
- uid => added so that other users would be able to see the id
- username => assigned at sign-up
- email => assigned at sign-up
- UserChatRoom => since users cannot be updated by others, each user is assigned their own pointer to a UserChatRoom where other users can add them into group chats



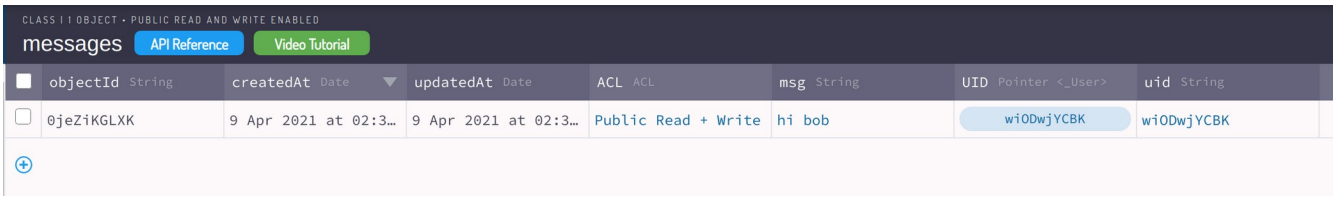
ChatRooms Class: holds all users and messages
- users: an array of userIDs that is stored in the room
- messages: an array of JSON objects that represent messages



UserChatRoom Class: class assigned to each user to hold their chatrooms they are part of
- User => a pointer to the user that is assigned this room
- ChatRooms => the chatrooms that the user is a part of, other users can write to this so there is a possible security concern, the only solution I could find was to use a cloud function.



messages Class:
- msg => the message
- uid => the sender of the message



Possible Improvements to the program:

- Centralized file that does all data queries to the service that can be called from any file -> would make switching cloud services easier as we would only have to edit 1 file instead of searching through all files and replacing query requests.

- Cloud functions -> all code would run in javascript on the cloud provider reducing mobile computing requriements. Disadvantage is that it makes our applciation tightly coupled with the coud service provider.