https://github.com/SpiritualDemise/ChildrenValleyHospital

Summary: Ramiro and Gabriel progressed in implementing the UI skeleton for all the pages and functions ready to be implemented. For the front end, in developing a prototype dummy data was created using ngFor. In the database portion of the project Michael and Jason have implemented the main functions such as searching and updating the database. The team have started implementing the basic functions of webgazer which could project a prediction of where the user is looking at onto a computer window.

The main database functions of the code is in app.js. The functions render (Doc Name Here) will go into the database and create HTML elements such as li,span and div, and put data inside of them. We will then render them inside the doc. Afterwards we set an attribute to the created elements and connect it to the document we have passed inside the function. Finally we append the span and div elements to the li as well as append the li to catalyst. The use cases for this function allows us to get a snapshot of the collection. The cross element is used to then delete the data from the snapshot. Where we can take away an element of the collection with the div but in future plans we would be implementing an edit button for the doctor to his patients. The flow of events to implement the button would be to create another div element which its text content will be a pencil icon. The use case for the edit button would be to allow the doctor to change the specific parameters of the documents elements.

The getting data snapshot will go through our collection of PatientLogIn data and will create snapshots of each element in the table. This will then render all of the data in the doc and make it available to manipulate through the HTML commands. We then have a save event listener that is an event 'e'. First we make sure that the default choice for an event is not to automatically refresh the page, so we prevent this. Then we go to the database collection for PatientLogIn, and use the add function that will take in the text input from the HTML and when they save it by clicking the placeholder button it will add it to the table in our database. From here we reset the values that they user input to blank spaces using the form value equal to a blank space. From the cross element that we created called 'e'. We will create an event that is a click. We will first make sure to use the stop propagation function that will stop from refreshing the page when the event is true. Then will make made and id equal to the doc id. We will go through the collection of the Doc we are using, and use the id we want to delete in the doc, and use the delete function to remove it from the document.

For querying the data, we would be referencing one of our database collections such as ‘PatientLogIn”. To select and order the elements in the collection we will use orderBy which will reference an attribute of the collection such as ‘Username’. The next steps for querying the data would be startAt(querytext) where querytext is a variable that holds the string the user is searching with and startAt is a function where we order the elements based on comparing to the querytext first. We will then use endAt() function which will referencing querytext and anything afterwards. This makes it so that the only relevant searches that contain querytext and extra characters will be included and not the other irrelevant elements that do not contain querytext. This is a general explanation of the important functions we need to use in order to achieve all the outputs for our application. It is done using the PatientLogIn data as an example to show how we will manipulate the data for the other docs as well. There will be more fine tuning functions for each different docs that will only vary in either who can manipulate data, save data, or how the data can be accessed. Although some functions will have less manipulations within the method that it is in, this outlines the general idea of saving, deleting, editing, and sorting through the data.

In creating the front end, we created an array called dates, then using a *ngFor a built in function We displayed them using a grid, of which we proceeded to create a function that will allow the user to scroll. We created a function onDateChange(date) that for the moment displays when a different date is changed. In doing so a new bar chart of a person's progress is displayed.

```
onDateChange(date){

   console.log(date);

 }
```

We created the core. In building the core we have three different tabs. The tabs are as follows "home", "messages" and "settings". For the moment the function displays to the console this will be connected to the back end to retrieve data on certain dates. We created a chart using chart.js, this will display the progress of the user and be modified by the top buttons. A ngOnInit() built in function will display the chart when the user logs in. The settings page was created where we have generated four attributes to enhance user interaction and a About section to list developers' credit. The Account attribute will allow the users to edit their account information such as an address change, contact update, and or name change. The Security attribute will be able to allow users to enhance their security with 2 factor authentication, and allow users to change their passwords. The Notifications attribute will allow users to edit their notification preferences for alerts based off messages, appointments, exam progress and completion. The Help section will be a page of frequently asked questions of the app and provide guidance to the users for the best experience of the app usage.