Please list out changes in directions of your project if the final project is different from your original proposal (based on your stage 1 proposal submission).

From the very beginning, our goal is to create an app that could recommend treatments and identify conditions based on the users' input symptoms. Some minor changes were made on minor functionalities, however the main direction has not changed.

Discuss what you think your application achieved or failed to achieve regarding its usefulness.

To discuss the usefulness of the app, we would first have to talk about the dataset we used. In the Chronic Illness Dataset on Kaggle, conditions, symptoms and treatments are all stored as "trackables." For example, if user A was exhibiting symptoms including headache, runny nose, sore throat and coughing, the diagnostic from the doctor showed that the conditions of user A were the flu and allergy, and the treatment user A received were cough syrup, allergy pills and Antipyretics, these data would be stored with this format:

userID	trackable type	trackable name
A	symptoms	headache
A	symptoms	runny nose
A	symptoms	sore throat
A	symptoms	coughing
A	conditions	flu
A	conditions	allergy
A	treatments	cough syrup
A	treatments	allergy pills
A	treatments	Antipyretics

From our experience, we can quickly identify what symptoms are caused by a condition and what treatment is used to mitigate a symptom. However, with the large amount of data, this process could not and should not be done manually. But looking at the data source, there are no other columns containing information that would allow us to perform a JOIN and link up the relations between conditions and symptoms, or symptoms and treatments. In other words, we won't know what the treatments are for or what conditions are causing these symptoms.

In this case, we would have to come up with a new way to suggest conditions/treatments based on the input symptoms, since we can't do something like

SELECT Conditions, Treatments

FROM Symptoms JOIN Conditions JOIN Treatment on user_id

WHERE Symptoms = "some symptom"

The final method that we have come up with is, first find all userIDs with the same symptom input by the user, then group these users, finally look at the most common treatments taken by these users(grouped) or the most common conditions. This query could only give passable results. For example, the no.4 or 5 most common treatments taken by users with headaches, may not be directly related to headahce. We believe that our usefulness has

taken a very slight hit due to the way the reference data is stored. However, the app serves its purpose well if one is only looking at the very top suggestions.

Discuss if you change the schema or source of the data for your application.

Despite having the issue mentioned above, we decided not to change our data source since a personal-health-related app gained the most interest among our group during a vote. Also, we are confident that we can work around this issue. The source of the data remained unchanged, however, we did have to make changes to our schema(specifically the treatments, conditions, and symptoms) tables in order to have the correct columns that would be suitable and more efficient for the application we intended to build as the treatments, conditions, and symptoms data were stored into 3 separate columns(trackable_name, trackable_type, trackable_value and we could not restructure those columns in a different way.

Discuss what you change to your ER diagram and/or your table implementations.

In the ER diagram we submitted in stage 2, we had a table named Diagnostic, and it was meant to be the centerpiece of our app, as it would store the symptom-condition-treatment relation, and all future queries would reference this table. However, we could not think of any ways to efficiently join the three main attributes together, therefore we adopt the aforementioned alternate query. This is also described more in detail in our technical challenges section as we needed to restructure our ER diagram in order to make it more suitable for our application.

What are some differences between the original design and the final design? Why? What do you think is a more suitable design?

In the original design we had a Diagnostic table, but we couldn't find a way to create it. This would be the only difference between the original design and the final design. The reason behind this was that we had no key attributes to join symptoms, conditions and treatments together.

In reality, the new design is more suitable since it is the only way we can make our app work. But if we could have access to a better source dataset, joining symptoms, conditions and treatments and creating a Diagnostic table would give us better results.

Discuss what functionalities you added or removed. Why?

The feedback system was totally removed in the final version of the app. Since we could not get a lot of check-ins before the semester, naturally not many feedbacks would be made for us to improve the advanced query that we're using to give recommendations.

Other functionalities originally proposed as the creative components were not implemented as well. The advanced query was giving accurate enough recommendations so we didn't think a machine learning model is required to improve the results. Also, we felt that the API

for connecting drug info such as prices and brands was beyond the scope of this course and we decided to drop it.

The added functionalities will be introduced in the next paragraph.

Explain how you think your advanced database programs complement your application.

We chose a trigger+stored procedure as our advanced database programs, they are also additional functionalities that were not mentioned in stage 1. The trigger helped improve user experience by returning the user back to the login page after the user deleted his/hers last check-in. The stored procedure would output two queries that served as tutorials of the app for the new users.

Each team member should describe one technical challenge that the team encountered. This should be sufficiently detailed such that another future team could use this as helpful advice if they were to start a similar project or where to maintain your project.

Pranav: A technical challenge we encountered pertains to the stored procedure. We were having issues figuring out how to output the results we got from our stored procedure to our js script so we could display it on our web page. We initially thought of using the OUT parameters to get output variables but we ran into issues with this in our server.js file. We ended up using a table and inserting our values into this table to be used outside of the stored procedure. This solution worked well, but we had to make sure to delete the table after we were finished using it in order to declutter our database.

Darshan: A technical challenge that we encountered was the structure of our ER diagram. As we created our database as well as our tables, we realized that the tables and columns that we originally had would not be suitable for the application we had in mind. After we cleaned our data, we decided that a more efficient way of the database creation was to keep the structure of the treatments, conditions, and symptoms data the same with the same columns(trackable_name, trackable_type, trackable_value) as no modifications would not be made in these tables and data would only be read. In order to accommodate this, we created a checkins table and columns to essentially keep track of every time the user uses an instance of our application(inputting symptoms). This solution worked very well as our main column to connect all of our tables was the user's user id and allowed for smoother functionality as the structure is as follows:

Users table to keep track of the user's login information

Checkins table for each data entry or use of our application by the user

Treatments, conditions, and symptoms table for our main source of data

Roshan: A technical challenge that the team encountered was the implementation of our trigger. We initially wanted our trigger to delete an entry in the Users table if after inserting the User's information into the table would result in 2 or more of the same entries in the table, which would indicate a repeat user of our web app. To do this, we thought we could set a trigger after the insert into the Users table that would count how many entries there are with the userID that was entered, and if it was more than 1 entry, we would delete the entry that was just entered. We kept running into errors when testing this trigger though. We

decided to then look back at how to use a trigger in a CRUD application, and that's when we realized that you can't set a trigger on the same table that you are doing a CRUD operation on. So our idea was never going to work. To fix this issue, we decided that we could still delete a User's profile if they had 0 entries in the checkIns table after deleting prior symptoms. After implementing this feature, we realized that our trigger worked.

Bo-Yang: A technical challenge that we encountered was the special characters in the original dataset. During the process of importing csv files into the tables, the Treatment table could not be loaded properly. We later found out this was due to the existence of special characters in the data. Since the trackable names for treatment were input by users, there could be instances where the users typed wrong. For example, there were some commas accidentally typed in and caused problems while processing the csv files. Some characters were also mis-typed, for example some “a”s were in the fashion of “å”. We weren’t entirely certain about whether this was causing any problems, but after cleaning the data by getting rid of special characters, the csv file loaded in successfully. However, due to our cleaning method, which was deleting possible troublesome characters, some useful check-ins could become useless. For example, a check-in with symptoms of “heådåche” would become “headache”, thus losing its contribution when the user searches for “headache.” A future team could try to download the original data and try to address this problem.

Are there other things that changed comparing the final application with the original proposal?

At first we wanted the recommendation to be more personalized, and the queries for generating recommendations would have more filters. (e.g. age, sex, nationalities of the users) Later we found that our dataset doesn’t have enough data to support this many filters, since some symptoms are already very specific themselves. Therefore in the final version of the app the recommendation is determined only by input symptoms and the severity score.

Another change that we made was that we decided to hide the user history search, we still believe this is an essential part of the app and we still do store check-ins made by the users. However, we haven’t found a good way to display them.

Describe future work that you think, other than the interface, that the application can improve on.

We believe that if there is another dataset for which symptom-condition-treatments are correctly linked, we could integrate it into our app and improve our results. This would allow us to make the application more personalized, linking treatments and conditions directly with symptoms they are experiencing. In addition, we could add a user-history for each user, which we would need to refine our login page for to make it so users can see their information and their history in a second page. This would be beneficial as they could track their history over their time of using our application and use this to help with their at-home treatments.

Describe the final division of labor and how well you managed teamwork.

We believe that we divided the labor extremely well between all the stages. We all had internships during the summer and we were all located in different time zones, so we knew that communication was going to be vital. We used discord to communicate with each other, which turned out to be perfect since everyone would explain when they were free each day, what they were going to work on, if they were stuck on anything, and if they needed help. This communication allowed the entire group to be on the same page throughout the project. In terms of division of labor, we all worked together on everything in a group call for stages 1-3. Each of us came up with great ideas during these first 3 stages that helped us in stage 4 and 5. For stages 4 and 5, we would work in pairs for the most part to implement certain features in our web app. For example, Darshan and Roshan worked on the trigger while Darshan and Pranav worked on the Stored Procedure. However, there are things that we all worked on, such as updating the UI of the web app. For stage 6, Bo-Yang worked on the video for the most part, while the rest of us helped on the report.