**LINK TO PROJECT VIDEO:**
**https://drive.google.com/file/d/1h-HqES3b_oNxBVTZQaeXNsC1548Vud_w/view?usp=sharing**

*Note: For our project report we chose to split it up as responses under each required "topic" to be discussed in the project report for ease of grading and better assurance of completion of all required sections.*

**Please list out changes in the directions of your project if the final project is different from your original proposal (based on your stage 1 proposal submission).**
For the most part our project remained similar in scope, purpose, and real life usefulness other than a few small changes. In particular, we decided to have our application not be natural language processing based and rather be more structured, prompting users to type in their symptoms in a comma-separated list into the chat interface versus a more versatile chatting space. While the tradeoff was that the user functionality and freedom is reduced, it greatly simplified our application complexity, making it possible to complete within our timeframe while still retaining its main goal of providing users with possible disease diagnoses and medicines to take after they list their symptoms. In addition, we decided to not implement the creative body scanning functionality for our project, but still have a robust chat, history, individual user consultation, and account functionalities in the Sympchat application. Other than these changes, our project did not change in direction compared to our original proposal.

**Discuss what you think your application achieved or failed to achieve regarding its usefulness.**
Regarding the usefulness of our application, we believe that there are many practice usages that can be applied in the medical space. If we improve the interface and expand the use cases to include the working body chart, the application's target demographic can be expanded even more to quickly help younger patients and older patients diagnose their symptoms. I believe we achieved success in helping narrow down broad symptoms, as we have a very inclusive list of data points that highlight illnesses, diseases, and issues of all categories. I also believe that we excelled in the displaying of the cases, and the implementation of the doctor notes so the user can keep log of their symptoms over time in case repeat incidents occur. Over time, this tool can serve as the perfect health log as the user ages, serving as a journal of their medical history. Our goal to incorporate AI into the chatbot to generate more personalized responses to the user is one area that we lacked towards achieving, and this is definitely worth exploring more especially in the booming age of generative artificial intelligence. Overall, we are very proud of the usefulness that Sympchat was able to bring towards users of all ages suffering from common or even uncommon medical issues for users who are not as versed in deep web searching or effective medical searching practices and might be overwhelmed with that whole process.

**Discuss if you changed the schema or source of the data for your application**
We did change our source of data in order to conform to our data. When we initially planned our data sources, we had not yet created a schema, and therefore did not need to know what exact requirements we needed to fulfill. Once we created our schema, we realized that the two dataset that we had chosen would not be able to sufficiently fulfill our requirements and we needed to supplement our data with a combination of simulated data (particularly for generating User consultations and Users) along with significant amount of the data cleaning on the Symptoms dataset to clean incorrect and malformed text inputs. Altogether, although we ended up using both the datasets we chose, we needed to supplement these datasets with simulated data in order to properly adhere to the scheme we created. In the future, we would like to first develop our schema, then look for data that would best fit our schema to make this a smoother process.

**Discuss what you change to your ER diagram and/or your table implementations. What are some differences between the original design and the final design? Why? What do you think is a more suitable design?**
We actually did not change anything from our initially proposed ER diagram design since it fully fit our application needs and the information we wanted to retrieve and display for the user. We were able to effectively use all tables and join tables created when doing our advanced database and other application operations, and effectively utilized the attributes in the table that we chose to include. For example, we added a “Date” attribute in our UserConsultation table which was helpful to properly provide a time that the User logged their consultation and thus for our history feature, different sets of consultations will show up using the “Date” attribute if the user asks for their medical Sympchat history in the past 3 months vs. 6 months, for example. Another example is our Medicine table that had Medicine name, Usage/Instructions, and Side Effects, which we utilized in our main diagnosis functionality to not only show users what disease they may have but to provide them with a call-to-action and sufficient resources in next steps for curing themselves or easing their pain by displaying possible medicines too through the use of the Medicine table and its various attributes.

**Discuss what functionalities you added or removed. Why?**
As mentioned in the first question, we removed the NLP chatting and creative feature body scan functionalities from our Sympchat application for the sake of time in order to simplify the project while retaining other more integral user functionality that make or break the purpose of the application. As for added functionality, we did not add anything beyond our expansive and descriptive functionality that we outlined we were going to do in our Stage 1: Detailed Project Proposal; however, the functionality we outlined there did extend beyond just the features that would’ve been implemented if we only used the endpoints and SQL for our 4 advanced database queries, spread across 1 transaction and 1 stored procedure. For example, the trigger functionality added a functionality where the severity of a disease is changed based on the amount of Sympchat users that reported in their own user consultation that they had it, which is a

helpful additional functionality in terms of mass outbreak detection. For example, if users had been logging COVID-19 diagnoses, our application could quickly flag it from a mild disease to a severe one due to the sheer amount of individuals being diagnosed with it. We also altered our initially proposed UI design to make it more simple both for more simplified user experience and so that our focus could be on the advanced database functionality.

**Explain how you think your advanced database programs complement your application.** Our database programs work well within our application for a number of reasons, but primarily because our application is data driven. In order for users to access our database they first need to be signed in, which requires a database to hold their password and usernames. Moreover, User consultations are also saved within the database and can be referenced in the future. Lastly, the majority of our application revolves around using the symptom and disease data to make a prediction, so that part of our database is very necessary for the entire application. Overall, our database is a very important part of our application for pulling data that is relevant to the User, so that people can use the database to evaluate their health. Diving into specifics, our trigger alters our Disease table's "Severity" attribute when more than a certain threshold of users log User Consultations where their diagnoses was a certain disease, using our advanced database functionality to provide users betters information about the severity of diseases and flagging potential outbreaks. In addition, our transaction is used for our disease diagnosis feature, as it finds diseases matching input symptoms, and then finds similar diseases to the primary diagnosis to consider a more nuanced, expansive diagnosis and give users examples of X possible diseases they may have, important as a transaction since we want disease diagnosis to be an all or nothing procedure, and not show users incomplete diagnosis if the commands somehow stopped halfway through if not implemented within a transaction. As for the stored procedure, we combined 2 advanced database queries focusing on providing the user's health history to provide a comprehensive review of a user's recurring health issues, making it stored since this is a common functionality in our application that will be happening again and again.

**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.**

- Helena's response: A technical challenge our team encountered towards the beginning of our project implementation was figuring out how to properly set up, host, and configure the project. While the class recommended using GCP, it was still a bit finicky to properly set up GCP, connect it to our database, and set up everybody's permissions. In addition, initially connecting the frontend to the database through a backend was a bit hard to wrap our head around as a team initially, so we had to take a step back to understand how data flows through an application from frontend to backend to database without using an ORM to know what resources we should be looking up to learn how to properly configure it. While we were able to do it with success eventually, I would recommend

other teams set enough time aside for not just the actual project work but for the configuration portion as it is the base that the rest of your project is built on and can have errors across machines or from one's own understanding.

- Michael's response: One challenge we faced was creating an easy to use user interface that would successfully incorporate elements of our database to cater to a wide demographic of users. In order to create an impactful UI, we had to research other instances of pages that were similar to ours. Eventually, we were able to narrow down designs for the chat page, which we took inspiration from the modern LLM gpt interfaces, as well as browser interfaces to provide a simple, yet effective look that both young children and elderly adults can use. Additionally, we faced challenges in integrating AI into our project. We considered utilizing the OpenAI API to help enhance the personalized aspect within the chat. However, this caused issues in the connection between the frontend and the backend, so ultimately we abandoned this idea. In the future, with more time, I would most definitely look into modernizing the interface and streamlining it even more, while also fully incorporating generative AI to give an analysis of the diagnosis and the remedies for the user.
- Aman's response: One challenge we faced was accurately simulating database data. For purposes of this project, we had to use data in order to make different tables within the database and properly establish relationships. Due to having a niche topic, we were unable to find all the real world data that we wanted, and instead had to use simulated data to inject into our system alongside real data. Although this was not ideal, it did allow us to continue with our project for the time being, allowing us to check if our database queries were working correctly and if the other parts of our application were pulling the correct data. However, putting in and simulating accurate data was very challenging, as there were not many tools available in order to do this. In particular, text data such as consultation descriptions were very difficult. In the future, I recommend people use R/python to simulate numerical data, as it can be created according to a specific distribution that is set. For text data, AI tools such as chatgpt, claude, etc. can be very useful if given the correct input prompt data.
- Aarushi's response: One challenge we faced was standardizing symptom entry. We needed a single, intuitive format that could be applied consistently and logically across all tabs. We decided on a comma separated list to keep symptom entry uniform everywhere. A simple comma-separated list like "nausea, headache, sore throat" was hard-coded into every tab. Each input box now shows a faint hint like "enter symptoms, separated by commas" to make sure users follow the correct format.This helps standardize things, whether the user is in chat, logging a doctor's diagnosis, or clicking through history, they're always feeding the system the same clean, parse-ready string format.

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

We modified our chat feature to disinclude the LLM/AI aspect of personalized diagnosis and analysis. In order to focus on accuracy and ensure that the connection to the database was correct and mistake-free, we took out the generative analysis of the disease and symptom inputs to increase the accuracy of the illness matching. We also took out the creative modification aspect of our proposal, as creating the intricate interface proved to take too much time, and caused issues in other parts of our interface. The bugginess of the body diagram would often cause the application to crash, or engage in phantom clicks that would disrupt the otherwise streamlined smooth user experience. Other than that, our execution stuck mostly to plan and stayed relatively identical to the original proposal, with core functionality staying true to form.

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

To earn users' trust when handling sensitive and private information, SympChat could update their security features. All symptom logs, chat transcripts, and confirmed diagnoses could in the future be encrypted both in transit and at rest with field-level keys for any protected health information. A zero-knowledge architecture that keeps encryption keys on the client device further limits breach impact. Within the backend, role-based access controls and immutable audit trails prevent snooping, while differential-privacy techniques on analytics queries let the team refine the service without exposing individual records. Equally important is feeding the diagnostic engine with data that is broad, current, and clean. Merging the Kaggle and BioSnap symptom-disease tables into a single knowledge graph annotated with standard vocabularies can help eliminate synonym gaps and support probabilistic reasoning across sources as well as providing more nuanced diagnoses for different groups. A more advanced future feature we could add is a scheduled ETL pipeline that can scrape trusted sites such as Mayo Clinic and NIH daily, run deduplication and credibility scoring, and auto-flag anomalies for human review. As users confirm doctor-verified outcomes, those cases can flow into a semi-supervised retraining queue, giving the model fresh, demographically relevant examples while guarding against noise through quorum and reputation checks. High-quality, continuously curated data ensure that SympChat's recommendations remain accurate, reduce false positives, and adapt gracefully to emerging illnesses or new treatment guidelines.

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

Our team divided the workload so each member could focus on a certain aspect of the website while still being able to collaborate closely. Helena handled the implementation of the back end after the team developed the advanced queries, transactions, triggers, and stored procedures together, and she built the secure login, sign-up, and any other necessary backend endpoint for the applications initially described functionality and usage flow. Michael integrated the backend and frontend together using axios as well as designing the UI for the main Chat page. Aman populated the project's data layer with our gathered data sources and simulated records where necessary, implemented the sign-in flow, and laid out the overall page routing for the frontend of

the project. Lastly, Aarushi polished the user experience by designing the History and Consultation pages, ensuring past symptoms and doctor-confirmed diagnoses were recorded and displayed cleanly. Together, these contributions built our website, making sure everything was integrated from the database to the user interface while also ensuring everybody got experience in the topics that they wished to.