CS 411 Final Project Report

By: Jeffrey, Kedar, Sam, and Neha

Our original proposal was a Sleep Calculator. Essentially, it would take in a variety of factors, such as age, height, weight, gender, and certain medical conditions, and then calculate how many hours of sleep they would each need each day. However, we ran into an issue where surprisingly we could not find as many thorough and detailed datasets as we thought we would. There were not many datasets that tracked users' sleep data along with information about them and their health. At the very least, they were not easily accessible or of a large enough size. As a result, we decided to pivot to diabetes tracking. Diabetes is a highly prevalent disease from which hundreds of millions (around 830 million) suffer and even more may be at risk, so the change of direction in our project is a worthwhile goal.

Ultimately, while our application is not as perfect or precise at warning individuals of diabetes as it could be, I absolutely believe that it does achieve a good deal in terms of its usefulness. It can track a variety of a user's metrics, from BMI to Hb1ac to blood glucose levels to whether or not they smoke or if they eat enough fruits and vegetables. One of the transactions can assess the first three factors mentioned and, utilizing the RiskAlerts table we created, send an alert to the user that they may be at risk for diabetes. We have a well-functioning frontend with a pleasant-looking and easy-to-use UI, allowing ease of use so that users can see if they are at risk of diabetes based on pre-existing conditions and lifestyle choices. Furthermore, our application encompasses several main services, including user creation and deletion, daily entry of data relating to a user's lifestyle or health, and monitoring both advanced large scale user health metrics and individual user data. The former two are made to register a new user to the application and update their health in a timely manner. The latter two features show a user / health industry professional more detailed and descriptive information about how at risk people are, as well as their personal health history. There were some liberties taken with these functionalities that wouldn't be present in a more robust application. For example, the first two features (user creation / daily entry management) should be kept separate from the latter two features that allow the application to query for global user information. This kind of functionality is better left in the hands of a healthcare professional using our application to monitor their patients at scale, or more closely observe a single patient. These features aren't isolated from each other in our current application, which may lead to some confusion for a new user. Having this level of isolation and additional quality of life features would've increased the utility of our application for a larger population of potential users.

Kaggle was the source of all of our datasets; it was where we found all three of our original tables for the Diabetes trackers and the Country dataset. Any additional tables were made by us, such as the RiskAlerts table to warn users who are in danger of contracting diabetes. This was ultimately the right choice as it gave us access to tables with thousands of rows of data, each with up to a dozen columns of detailed information.

At the start of stage 3 we trimmed the attributes that we'd included in our ER diagrams when setting up the database in Google Cloud Platform (GCP). For example, our Lifestyle entity, which became our LIFESTYLEINFO table in the database, first had an attribute for "Physical_activity". This was after cutting multiple other attributes from our initial ER diagram (before resubmission). We considered that this change would maintain most of the ideal functionality of each table while avoiding any issues of redundancy between and among tables. We removed the entire "Diabetic" table from the first iteration of the ER diagram because that information fit better in the user table. We added a new table, Country Info, with relevant information such as health care availability / spending, to add potential for more interesting queries and analyses. In our final database implementation we also added the tables UpdateLog, Incentives, and RiskAlerts which are affected by our transactions. These tables are uniquely identified by a ID and were included to provide additional interactivity with the application. This was in line with what we believed a user could be using the application for — to receive timely alerts on their diabetes risk, as well as motivation to stay on track.

Our advanced database queries add a great deal of functionality to our application. Transactions allow us to create new programs that can alert users if they are at risk of diabetes, stored procedures check for age range stats and also assess patients for diabetes, and triggers also ensure that the project is functioning as it should. For instance, the stored procedures GetAgeRangeStats() retrieves aggregate health statistics for non-smokers grouped by age ranges in decades GetHighRiskPatients procedure retrieves details of "High Risk" patients who are nonsmokers and have certain high-risk conditions like high blood pressure or heart disease. Ultimately, our project would not be the same or even functioning without all of our advanced programs and queries working together to allow users to see their various statistics and if they are at risk of getting diabetes.

Now presenting technical difficulties from each of our team members:

Jeffrey: One of the biggest technical challenges we had with this project was getting our database set up on GCP. We spent too much time troubleshooting an issue with getting people connected to the cloud database through public IP onto the MySQL workbench. There was some additional hassle with getting our data files uploaded to data tables and creating the entity relationships that we wanted, since our datasets were collected independently of each other. We compromised on sharing the account of the team member who created the GCP instance, as well as generating data to simulate a connected relationship between entities similar to what was specified in the ER diagram. Of these issues, GCP connectivity is the most pressing for future groups, as other groups brought it up during office hours.

Kedar: One of our technical challenges was deciding what transactions we should include in the project. I had to spend quite some time trying to figure out which functions the different transactions should accomplish. Ultimately, it took quite some time to realize that the best use for transactions required creating several new tables for RiskAlerts and UpdateLogs. Without the creation of new tables and several rounds of revision, I would not have been able to figure out what the best transactions to use were. I would advise future students to take their time considering how to implement transactions and similar complex functions in their database, considering making new tables or just expanding the database if necessary, and going through multiple rounds of revisions for the best accuracy.

Neha: One of our technical challenges was identifying which triggers would provide meaningful enhancements to the application. While I initially came up with several ideas for potential triggers, I had to make modifications based on the data types of the attributes in the database. Ensuring compatibility between the trigger logic and the specific data types was crucial to making sure that the triggers functioned as intended and did not cause unexpected behavior or errors.

Sam: One of the significant technical challenges our team faced was working with Node.js. Since most of us were not very familiar with it initially, there was a steep learning curve. Adapting the Node.js setup to our specific project also revealed issues that were not present in the original web app tutorial on Google Cloud Platform (GCP). This added another layer of complexity. Additionally, writing routes and designing the database backend to integrate seamlessly with the frontend was a challenging task, as it required a deeper understanding of Node.js and its interaction with other components For any future team attempting a similar project, here are some tips: Learn Node.js early and go to Office Hours early.

The application could have more functionality in terms of diabetes diagnosis in the future, particularly in the area of prediabetes. Right now, the functionality is mostly for warning a user if they are at risk of getting diabetes based on a variety of factors, but despite the fact that one of the tables offers an option for a prediabetes diagnosis, we have not utilized programs that could directly interact with patients who could be at risk for prediabetes. It is something we should absolutely add to future iterations of our application.

While our team occasionally did suffer from some communication issues, ultimately we were able to work together to each do our tasks and accomplish our ultimate goal. Neha worked on the frontend design, UI, early implementation, and the triggers. Jeffrey worked on setting up the database in GCP, implementing the frontend, and connecting it to the backend. Sam helped establish the backend and wrote the stored procedures. Kedar drafted the Project Proposal, assisted with some frontend implementation, and wrote the code for the transactions.