Project Report: NutriChoice

Elizabeth Binkina, Simon Hu, Nynika Badam

Changes in Direction

Based on our stage 1 proposal submission, our project direction stayed mostly the same. The changes we made had to do with the user interface and how a user would interact with the website. Rather than just generating a list of grocery items that add up to the goal intake a user wants, a user is able to also casually search food items to learn about their nutritional values. A user is also able to do all this without making an account. They can ideally generate multiple lists of ingredients rather than just sticking to the first one generated. The initial proposal for the project allowed the user to add their input but ultimately, we settled for generating these lists ourselves while allowing the user to re-generate the results as they please. By doing so, our goal was to simplify the process for the user in the final application.

Successes and Failures

Some of the successes that we achieved include a complete front end UI that has user input, a search bar, drop-down menu, and redirected links. We also were able to implement our schema within Google Cloud Platform and import data from real datasets. The main failure was our inability to connect the front end to the backend using api calls as well as hosting the server on a local host. We ended up hosting the server on a private locale using ViteJs but because it was private, it was hard to access for the group. Since we were unable to connect the front end and back end, the user could not input data despite having the correct queries ready in the backend. We attempted multiple solutions such as exploring how to host the site on GirHub pages, following Vitejs tutorials to host a static site, using Postman to check API calls, and attending office hours to look over our backend server set up. Ultimately, we were unable to find a solution to our issues. .

Schema Changes

While creating the tables for our schema in Google Cloud Platform and when designing our frontend, we realized we neglected to create an entity that would hold a list of generated ingredients based on a user's goal. We named the table FoodGoalList; it has a many to one relationship with the UserInfo entity where every user can have multiple generated ingredient lists. Additionally, since the table is dependent on GoalIntake specified by a user, it has a one to one relationship with the Factors entity since one goal list corresponds to one goal intake. The updated scheme can be found here,

Additionally, as we were making the FoodItems and FoodGroup tables, we found that the dataset we chose to use actually listed multiple food groups for each food item rather than just one. This makes

it a many to many relationship instead of a many to one relationship, effectively removing the need for a foreign key within the FoodItems entity.

Finally, when laying out the login process for a user, we realized that a user would create their account with an email and password. We also wanted every email to be unique so we decided to make UserId = email and have password as a new attribute within UserInfo.

Functionality Changes

We removed checkboxes from sorting information because the automatically generated list will take into account how much vitamins or macro the user wants. If the user wishes to, they are still able to search for individual ingredients. In addition, since our application was going to automatically generate a list for the user, we didn't feel the need to add value sliders for specific fields such as "Calories" or "Protein" because we felt that our Goal_intake would replace that functionality. Initially, we wanted to display the current and goal intakes of the user using generated graphs from the user's data. However, due to time constraints and technical issues throughout the application process, we were unable to apply this functionality.

The advanced database programs significantly enhance NutriChoice application by ensuring personalized user experience. The transaction program ensures that the updates to user intake and goal settings for vitamins and macros are accurate and consistent. One of our main goals for this web program was to customize ingredients or grocery lists for users to advance in their goals based on factors such as macros and vitamins, and this transaction ensures that. The stored procedure adds customizable updates to user intake based on foods that they have consumed, as well as updated macro intake and goal intake variables for the user. Triggers further ensure that any changes to food items are reflected appropriately in related tables. Constraints enforce data quality and ensure non-negative values for intake and goals. Overall, these components make NutriChoice a user-friendly application that simplifies dietary and goal management and enhances overall user experience.

Technical Challenges

Simon - One technical challenge that we had was connecting the front end to the back end. We also had issues running our frontend on GCP so we moved towards running it on localhost where we also had issues. Eventually, we settled with ViteJs but ViteJs runs the servers on a private local that isn't accessible to other people outside of the creator. We tried different solutions by repeatedly trying to fix localhost on VSCode or running the site on Github pages but neither worked. For future reference, I would stick with trying to figure out how to use localhost because it is helpful to have working links that connect with the application when needed.

Nynika - Our group had trouble with writing API calls and having them call our database in GCP. At first, we had only written the calls and didn't integrate that functionality into the HTML components in our web app so even though the API calls might've been functional, the components weren't reflecting that. However, even after collecting data entered by a user by interacting with the components, our API calls still didn't end up working due to our inability to make Vitejs and GCP communicate successfully. We also used Postman to test the functionality of our API calls.

Elizabeth - Our main struggle with this project was figuring out how to fix technical issues when things were not loading, uploading, updating, etc. It was a challenge to fix these unexpected errors, such as importing the data into our database in GCP. We had trouble figuring out how to create tables in order to import our very large datasets that were in CSV format. We tried converting to SQL file, we tried conjoining files, until we figured out how to do it in the google console. After having issues with attempting to implement it in BigQuery, we finally figured out that we needed to create the tables within the cloud console. Then, we formatted our datasets so that it matched the table attributes and imported them manually. Google Cloud was definitely difficult to use at first due to it having so many features and components, running out of credits unexpectedly, and managing our data within it.

Future Work

Our application could improve its functionality in the future by allowing customers the ability to buy all their ingredients from their generated shopping list through online shopping. By connecting our application to online grocery chains, we are able to pull prices and even recommend users on how much they should spend trying to reach their health goals. By doing so, using our app will be a huge time saver and convenience for users since they won't have to manually go to individual shopping websites to browse their selection of ingredients. Another functionality that would be useful would be reading the user's data from current and goal intake and displaying a graph that shows the progress the user has made. With a physical graph that the user can compare their progress to, the user will be more inclined to continue their health goals and achieve our purpose of simplifying and supporting the user's nutritional health journey.

Division of Labor

Over the course of this summer class, we have divided up the work of the project into three main segments. Elizabeth worked on writing SQL queries such as transactions, triggers, and more. Nynika worked on implementing our schema within GCP and connecting to our web application. Lastly, Simon created the frontend UI and API calls as well as connecting it to the backend. Although the project was separated into three major segments, we all pitched in to assist one another in solving these issues because every step works in relation with another for our application. In order to update each other, we communicated through messages, zoom calls, and more to set goals for ourselves throughout the days and weeks we spent on this project. By doing so, we were able to work efficiently and assist one another

when needed, despite all the bugs and issues that arose. Although the division of labor did not follow our initial plan during the brainstorming phase, the important thing was that we all assisted one another when needed and were familiar with every aspect of the application process.

Work Cited

Vite | Next Generation Frontend Tooling, https://vitejs.dev/. Accessed 30 July 2024.

Stage 4 SQL Code (non-implemented)

Transaction:

```
START TRANSACTION;

SET UserId = ___;
SET VitaminName = ___;
SET MacroName = ___;

UPDATE Factors f
JOIN FoodItems fi ON fi.FoodId = f.FoodID JOIN Contains c ON c.FoodID = fi.FoodID JOIN Vitamins v ON v.VitaminID = c.VitaminID
SET f.Current_Intake = (
        SELECT SUM(v.VitaminAmount)
         FROM Contains c JOIN Vitamins v ON c.VitaminID = v.VitaminID  JOIN FoodItems fi ON c.FoodID = fi.FoodID
        WHERE f.UserID = ____ AND v.VitaminName = ______
)
WHERE f.UserID = _________;

UPDATE Factors f
JOIN FoodItems fi ON fi.FoodID = f.FoodID JOIN ConsistsOf co ON co.FoodID = fi.FoodID JOIN Macros m on m.MacroID = co.MacroID
SET f.Current_Intake = (
        SELECT SUM(m.MacAmount) FROM ConsistsOf co JOIN Macros m ON co.MacroID = m.MacroID JOIN FoodItems fi ON co.FoodID = fi.FoodID
 WHERE f.UserID = _____;

END IF;
COMMIT;
```

Stored Procedure:

```
DELIMITER //
```

```
CREATE PROCEDURE UpdateUserIntake(IN p_user_id INT)
BEGIN
   DECLARE v_new_goal_vitamin DECIMAL DEFAULT 500;
   DECLARE v_new_goal_macro DECIMAL DEFAULT 100;
   DECLARE v_vitamin_exists BOOLEAN DEFAULT FALSE;
   DECLARE v_macro_exists BOOLEAN DEFAULT FALSE;

      IF EXISTS (  //Check if the user has consumed any food with a specific vitamin

      SELECT 1
      FROM Contains c
      JOIN Vitamins v ON c.VitaminID = v.VitaminID
      JOIN FoodItems fi ON c.FoodID = fi.FoodID
      WHERE fi.FoodID IN (SELECT FoodID FROM Consumed WHERE UserID = p_user_id)
        AND v.VitaminName = 'Vitamin C'
   ) THEN

      UPDATE Factors f // Update vitamin intake for the user
      JOIN (
         SELECT v.VitaminName, SUM(v.VitAmount) AS TotalIntake
         FROM Contains c
         JOIN Vitamins v ON c.VitaminID = v.VitaminID
         JOIN FoodItems fi ON c.FoodID = fi.FoodID
         WHERE fi.FoodID IN (SELECT FoodID FROM Consumed WHERE UserID = p_user_id)
         GROUP BY v.VitaminName
      ) AS intake ON f.UserID = p_user_id AND intake.VitaminName = 'Vitamin C'
      SET f.Current_Intake = intake.TotalIntake
      WHERE f.UserID = p_user_id;

     UPDATE Factors //Set new goal intake for the vitamin

    SET Goal_Intake = v_new_goal_vitamin
      WHERE UserID = p_user_id
       AND EXISTS (
          SELECT 1
          FROM Contains c
          JOIN Vitamins v ON c.VitaminID = v.VitaminID
          JOIN FoodItems fi ON c.FoodID = fi.FoodID
          WHERE fi.FoodID IN (SELECT FoodID FROM Consumed WHERE UserID = p_user_id)
```

```
            AND v.VitaminName = 'Vitamin C'
        );
    END IF;

      IF EXISTS (  //Check if the user has consumed any food with a specific macro

        SELECT 1
        FROM ConsistsOf co
        JOIN Macros m ON co.MacroID = m.MacroID
        JOIN FoodItems fi ON co.FoodID = fi.FoodID
        WHERE fi.FoodID IN (SELECT FoodID FROM Consumed WHERE UserID = p_user_id)
          AND m.MacroName = 'Protein'
    ) THEN
        UPDATE Factors f //Update macro intake for the user

        JOIN (
            SELECT m.MacroName, SUM(m.MacAmount) AS TotalIntake
            FROM ConsistsOf co
            JOIN Macros m ON co.MacroID = m.MacroID
            JOIN FoodItems fi ON co.FoodID = fi.FoodID
            WHERE fi.FoodID IN (SELECT FoodID FROM Consumed WHERE UserID = p_user_id)
            GROUP BY m.MacroName
        ) AS intake ON f.UserID = p_user_id AND intake.MacroName = ______
        SET f.Current_Intake = intake.TotalIntake
        WHERE f.UserID = p_user_id;

         UPDATE Factors // Set new goal intake for the macro
        SET Goal_Intake = v_new_goal_macro
        WHERE UserID = p_user_id
          AND EXISTS (
            SELECT 1
            FROM ConsistsOf co
            JOIN Macros m ON co.MacroID = m.MacroID
            JOIN FoodItems fi ON co.FoodID = fi.FoodID
            WHERE fi.FoodID IN (SELECT FoodID FROM Consumed WHERE UserID = p_user_id)
              AND m.MacroName = _____
);
    END IF;
```

```
   COMMIT;
END //

DELIMITER ;
```

Trigger:

```
DELIMITER //

CREATE TRIGGER AfterFoodItemInsert
AFTER INSERT ON FoodItems
FOR EACH ROW
BEGIN
   IF (SELECT GroupName FROM FoodGroup WHERE GroupId = NEW.GroupID) != _____ THEN
      UPDATE FoodGroup
      SET GroupName = 'Update Group'
      WHERE GroupId = NEW.GroupID;
   END IF;
END //

DELIMITER ;


DELIMITER;
```

Constraints:

```
ALTER TABLE UserInfo
ADD CONSTRAINT UserID PRIMARY KEY (UserID);

ALTER TABLE FoodItems
ADD CONSTRAINT FoodID PRIMARY KEY (FoodID);
ADD CONSTRAINT FoodName UNIQUE (FoodName);

ALTER TABLE Factors
ADD CONSTRAINT ProgressID PRIMARY KEY (ProgressID),
ADD CONSTRAINT UserID FOREIGN KEY (UserID) REFERENCES UserInfo(UserID);

ALTER TABLE Factors
ADD CONSTRAINT NonNegativeIntake CHECK (Goal_Intake >= 0 AND Current_Intake >= 0);
ADD CONSTRAINT OverIntake CHECK (Current_Intake <= Goal_Intake);
```