UTRITION IV

TRANSITION DOCUMENT

Kiera Belford, Thomas Phelan, Jared Schneider, Stuti Shah, Brittany Shine
CSE 5915
Team HealthBucks

24 April 2021

Capstone Sponsor
Andee Peabody
Founder & CEO, Engage Health

Capstone Supervisor
Robert Vanderwall
The Ohio State University College of Engineering
Dept. Computer Science and Engineering

Table of Contents

1. Introduction

The following document is intended to provide future iterations of the Utrition sequence, as well as those involved with EngageHealth, an outline of the design decisions made by the Utrition IV team and to act as a transition document for contributors in following iterations. Those responsible for the efforts made in this project iteration are members of The Ohio State University's Class of 2022, who obtained their Bachelor of Science in Computer Science and Engineering. Team members include Kiera Belford, Tom Phelan, Jared Schneider, Stuti Shah, and Brittany Shine. The team was guided by project sponsor Andee Peabody, Founder and CEO of EngageHealth, and project supervisor Robert Vanderwall, Professor at The Ohio State University.

The document will highlight the redesigned Utrition database schema; scripts to create and populate the database tables; scripts to obtain genetics data and further populate said database; a script that serves as a testing suite and that should be expanded over future iterations of the project; and some query examples to demonstrate data retrievability. Information on setting up virtual environments, the database manager, and connection to user interface can be found in the Utrition Setup Manual. Additional information on the Utrition IV iteration can be found in the Software Design Document, Testing Manual, and Testing Coverage documentation.

A special and important note: all files and documentation referenced throughout this transition document can be found in the project's Google Drive folder at the parent path listed below. It should be noted that some files are listed in subfolders within the path.

- Nutrition > SP 22 > FINAL_DELIVERABLES

2. Database Schema

The following diagrams illustrate the database schema redesign. The Utrition IV team concluded that the originally proposed schema was flawed with data organization, relational mapping, referencing and normalization. The Utrition IV team first targeted the genetics side of the database and set new primary keys, adjusted foreign key referencing, and evaluated the need for certain composite keys by doing so.

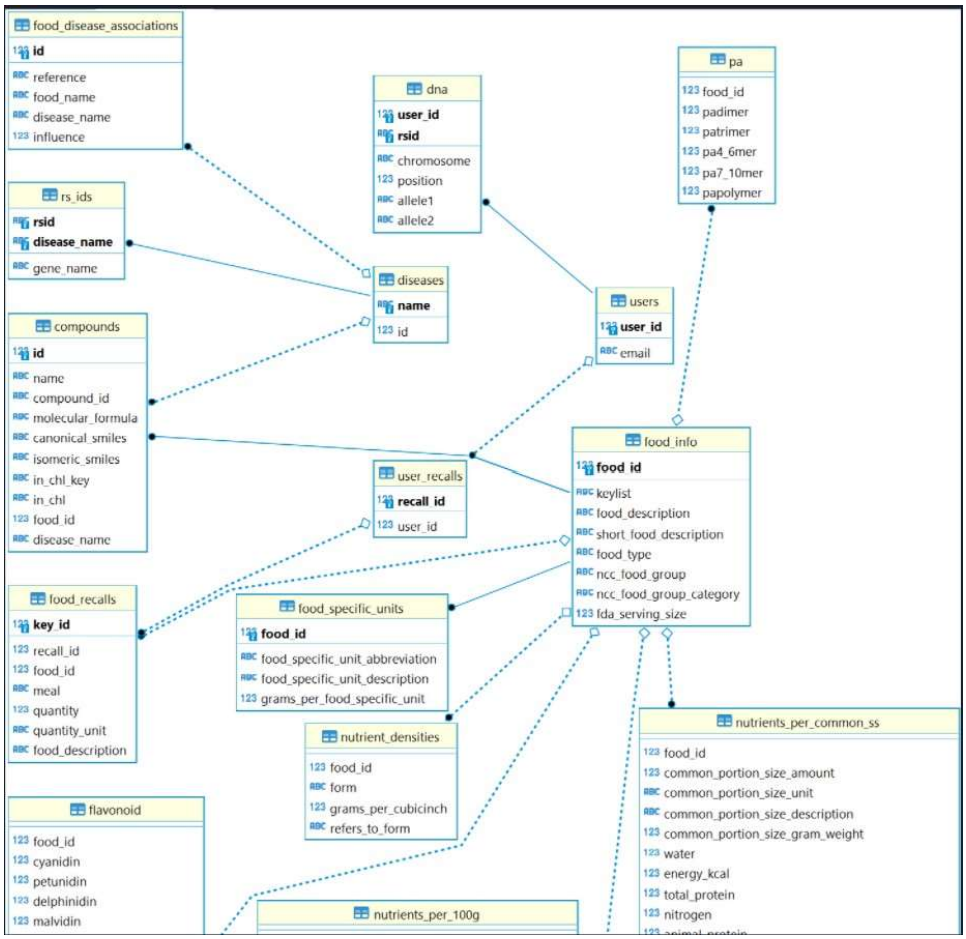

Figure 1. Schema from previous Utrition iteration.

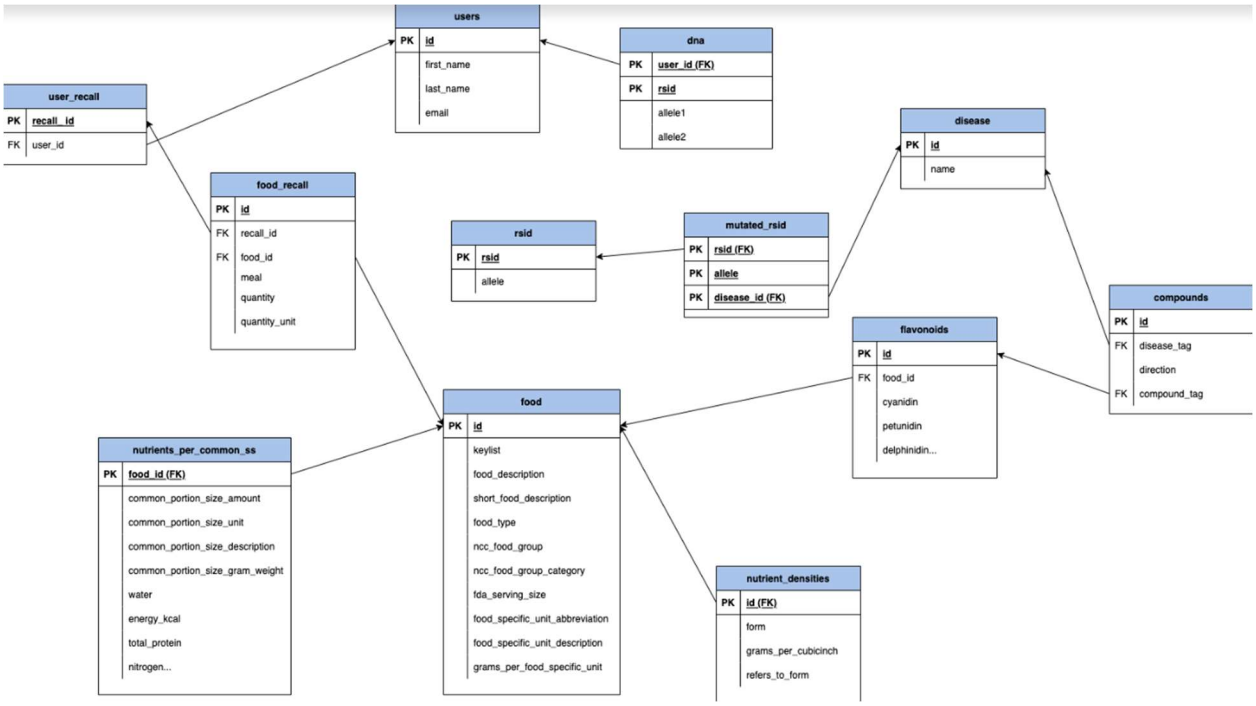

Figure 2. Revised and current schema proposed by Utrition IV.

As a general walkthrough, the user enters their information on the web via the user interface. Their information is logged, if a new user, to the users table. The user's dna file is uploaded, parsed, and inserted into the dna table. The dna table is a set of entries unique by the composite primary key user_id and rsid and contains allele information for said rsID. Since a user can, and will, have multiple rsIDs for a single dna file, it was critical to keep the composite primary key.

The database is pre-populated with data, including both data already provided (food / nutrition data loaded in populate script) as well as all data obtained from the web scrape. Once the data has been loaded into the database, the end-user / patient can obtain results.

The rsid table is a dataset of all rsIDs and their base alleles obtained from the web scrape. The mutated_rsid table includes all rsID mutations under the constraints that they are Single-Nucleotide Variant (SNV) and are within a threshold of specific clinical

significance selections. The filtering decisions made can be observed in 'esearch_webscraping.py'. Provided below is a snapshot of the same search filters, entered directly to National Center for Biotechnology Information (NCBI). This, of course, can be expanded to different selections in SNP Class, Clinical Significance, and a number of other criteria found at NCBI.

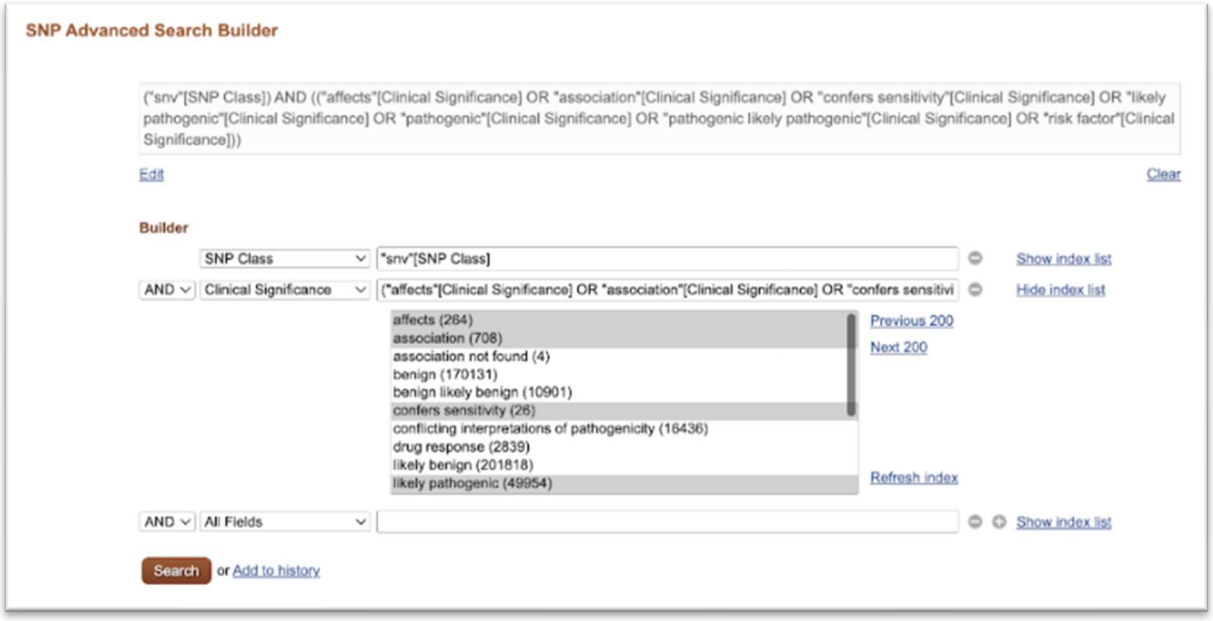

Figure 3. Search filters for mutations of type SNV and Clinical Significance selections.

Based on the genetic information provided by the user, and the web results for their genetic mutations, dietary recommendations are made to the user. Foods and food groups are suggested as part of a regular diet based on micronutrients found in said foods that downregulate for diseases the user may be at risk of obtaining. Similarly, certain foods and food groups that may upregulate for the same diseases will help the user understand what should be left out of a new diet / considered in better moderation. Listed below is a brief explanation of the food and nutritional data tables:

- **food**: information specific to a certain food
- **nutrients _per_common_ss**: holds information based on one common portion of a food
- **nutrient_densities**: holds density metrics specific to a food
- **flavonoids**: list of biological compounds found in foods
- **compounds**: the connection between diseases and foods based on common flavonoids; attribute 'direction' indicates a downregulation (1) or upregulation (2) of associated disease
- **food_recall**: table to contain information about a user's meal

3. Create & Populate Scripts

In correspondence to the redesigned Utrition database, scripts for creating and populating the database were updated. The create script includes a database creation and each table's create SQL command, with all specifications, constraints, and relations included. The SQL command series can be found in the 'create_db.txt' file. Included below is a snippet of the code.

Followed by the create script is the population script, which is responsible for loading food and nutritional data. These tables, once populated and once the web scrape is completed, will be what make nutritional and dietary recommendations possible. The details of this script can be found in 'populate_db.txt'. Additionally, a set of commands for populating sample data is included in 'populate_sample_gene_info.txt'. This is for testing purposes only as it serves as an alternative to user-entered data on the web application interface. More importantly, it allows future teams the ability to complete a full walkthrough of the analysis and recommendation process, less the recall aspect.

All three SQL command series are called via PSQL commands, found in 'setup_utrition.txt'. Once this set of SQL commands completes, web scraping can ensue, followed by end-to-end, full interaction from client to database, and back to client!

```
\connect utrition_final

CREATE TABLE IF NOT EXISTS users(
id serial PRIMARY KEY,
first_name varchar(20),
last_name varchar(20),
email text UNIQUE NOT NULL,
CONSTRAINT email CHECK (email SIMILAR TO '[a-zA-Z0-9][a-zA-Z0-
9_\-.]*@[a-zA-Z]+.[a-z][a-z][a-z]')
);

CREATE TABLE IF NOT EXISTS dna(
user_id int NOT NULL,
rsid varchar(20) NOT NULL,
allele1 char(1) NOT NULL,
allele2 char(1) NOT NULL,
PRIMARY KEY (user_id, rsid),
FOREIGN KEY (user_id) REFERENCES users(id),
CONSTRAINT rsid CHECK (rsid SIMILAR TO
'rs(0|1|2|3|4|5|6|7|8|9)+'),
CONSTRAINT allele1 CHECK
        (allele1 = 'A' OR allele1 = 'C' OR allele1 = 'G' OR
allele1 = 'T' OR allele1 = '-'),
CONSTRAINT allele2 CHECK
        (allele2 = 'A' OR allele2 = 'C' OR allele2 = 'G' OR
allele2 = 'T' OR allele2 = '-')
);
```

Figure 4. Code snippet from SQL command series in 'create_db.txt'.

4. Web Scraping Scripts

To obtain genetics data, the Utrition IV team utilized a shell script to connect to NCBI titled 'ncbi_webscraping.sh'. This international database has billions of genetic information, mutations related to specific rsIDs, and diseases that correlate to said mutations.

The script first connects to the Utrition IV database. The following parameters can be adjusted by future Utrition teams based on how many batches and the batch size they are wanting to obtain during the web scrape:

```
SEARCHBATCHSIZE=20
RETSTART=201
BATCHSIZE=10
```

Figure 5. Variables utilized in the 'ncbi_webscraping.sh' script.

The variable 'SEARCHBATCHSIZE' refers to the size of the batch of data; 'RETSTART' is an index variable for the web scrape to avoid repeated data collection; and 'BATCHSIZE' is as it is stated. Ideally, this specific example would iterate over 20 batches of 10 rsIDs, starting at an index of 201, relative to NCBI and the web scrape.

Nested in the shell script is a single 'esearch' connection. This grabs a web and query key to maintain the same session when fetching the batches of rsID data. This connection is made via the Python script titled 'esearch_webscraping.py'. Below is a snippet of the Python script showing the connection to the esearch web scrape and obtaining a dictionary object containing the resultant web and query keys.

```
# Search
log.debug("Beginning search")
eSearch = Entrez.esearch(db=db, term='("snv"[SCLS] AND
("risk factor"[CLIN] OR "pathogenic"[CLIN] OR "pathogenic likely pathogenic"[CLIN]
OR "likely pathogenic"[CLIN] OR "affects"[CLIN] OR "association"[CLIN]
OR "confers sensitivity"[CLIN]))', retstart=args.retstart,**paramEutils)
log.debug("Completed search")
```

Figure 6. Code snippet from Python script 'esearch_webscraping.py'.

Once the keys are obtained, the Utrition team can connect to the same session on each iteration of data collection. The ‘efetch’ command is run using the parameters previously defined in the shell script. Once an efetch run completes, the data is appended to a JSON file and parameters are incremented to fetch the next batch of data. The JSON file contains all found information for specific rsIDs, their corresponding mutations, and diseases related to those mutations. The ‘efetch_webscraping.py’ script returns to the shell script for continued iterations. A snippet of this code is included below.

```
while batchCounter < batchNum:
    # print(str(batchRetstart))
    efetch = Entrez.efetch(db=db, query_key=queryKey,
WebEnv=webEnv, retstart=batchRetstart, retmax=batchSize,
rettype='json', retmode='text')
    print(efetch.read())
    batchRetstart += batchSize
    batchCounter+=1

# print(jsonStr)
```

Figure 7. Code snippet from Python script ‘efetch_webscraping.py’.

After each iteration of the efetch script call completes, the JSON file is inserted to a JSON temp table. Upon completion of the entire web scrape (all found records from original esearch are accounted for), the temp table is parsed through; the shell script creates a SQL view; and pending each entry passes SQL constraints (duplicates, data format, etc.), it is inserted into the database tables in the following table order:

1. **disease**: Foreign key disease_id in table mutated_rsid depends on this table. Distinct diseases of current view only inserted if not existent in database.
2. **rsid**: Distinct rsIDs and their corresponding base alleles inserted if not existent in database.

3. **mutated_rsid**: Mutations collected for specific rsID inserted to table with corresponding rsID, mutated allele, and disease_id. This is a composite primary key due to the potential of a single rsID mutation mapping to several diseases.

Once table updates and insertions have completed, the temp table and SQL view are deleted. Step-by-step descriptions can be found in the comments of both the shell script and the Python scripts.

5. Testing Suite

The Utrition IV team focused nearly all efforts on the data science of this project. Database design, organization, and decision-making required an intensive focus and was only initiated almost halfway through the project upon realization for a fresh start.

Consequently, the testing suite completed focused on querying data and ensuring the results populated and returned as expected. For each table in the schema unrelated to food and nutritional data, a test series of SQL insertions was executed by utilization of a Python wrapper. Below is a snapshot of tests for the mutated_rsid table – both expected pass tests and fail tests - as well as a snapshot of the Python file that executes said tests.

```
-- 10 pass test cases
-- pass
INSERT INTO mutated_rsid (rsid,allele,disease_id) VALUES
('rs0','A',1);

-- pass
INSERT INTO mutated_rsid (rsid,allele,disease_id) VALUES
('rs0','C',1);
```

Figure 8. Snippet of test cases expected to pass for insertions into table: mutated_rsid.

```
-- 10 fail test cases
-- test 23andme's internal rsid label, fail
INSERT INTO mutated_rsid (rsid,allele,disease_id) VALUES
('i777','T',7);

-- check NOT NULL constraint on (rsid), fail
INSERT INTO mutated_rsid (rsid,allele,disease_id) VALUES
(null,'A',7);
```

Figure 9. Snippet of test cases expected to fail for insertions into table: mutated_rsid.

```
                                # if this case should have
passed, increment the number of cases passed. Otherwise, print an
error message
                                if i <= cases_should_pass:
                                        number_passed += 1
                                else:
                                        print("Error: The
following test case should have failed, but instead passed: ")
                                        print(test_case)
                        except:
                                # we will only get to this point
in the logic if the case fails.
                                # if this case should have
failed, increament the number of cases failed. Otherwise, print
an error message
                                if i > cases_should_pass:
                                        number_failed += 1
                                else:
                                        print("Error: The
following test case should have passed, but instead failed
instead: ")
                                        print(test_case)

                        i += 1
```

Figure 10. Snippet of Python test suite wrapper. Variables used to track number of test cases passed and test cases failed for any given test file. Error message displayed upon incorrect output for test case.

While the Utrition IV team has provided a compelling and intensive test suite, it is highly encouraged that testing is expanded and continued in future iterations of the project. Namely, testing on food / nutrition data collection and query returns should be incorporated next.

6. Sample Queries

The final component of the transition document Utrition IV has decided to include is a set of sample queries. Not only is this helpful for brainstorming future queries that may stem from what the team has included, but most importantly, it should serve as a guide for understanding the database schematics at a fundamental level. Included are snapshot of some queries the team ran at the beginning of database design. The queries are likely optimizable as the team improved code and database constraints throughout the course of the project, so it is worthwhile for future teams to consider testing reduced / nested, simpler queries.

```
SELECT users.email, COUNT(user_recall.recall_id) FROM users
INNER JOIN user_recall ON users.id = user_recall.user_id GROUP BY users.id;
```

Figure 11. The above query returns rows containing the number of times a user has entered data (new submission to UI) as well as the user's email address.

```
SELECT u.id, u.first_name, u.last_name, u.email,
d.rsid, r.allele, d.allele1, d.allele2, mr.allele, dis.name
FROM users AS u
    INNER JOIN dna as d  ON u.id = d.user_id
    INNER JOIN rsid as r ON d.rsid = r.rsid
    INNER JOIN mutated_rsid as mr ON mr.rsid = r.rsid
    INNER JOIN disease AS dis ON dis.id = mr.disease_id
        WHERE dis.id != 0
        AND (d.allele1 NOT LIKE r.allele OR d.allele2 NOT LIKE r.allele)
        AND (d.allele1 LIKE mr.allele OR d.allele2 LIKE mr.allele);
```

Figure 12. The above query returns rows containing user-related information and a full explanation of mutated allele from base allele on a specific rsID, as well as the disease associated. Can return multiple entries for one user, and multiple entries for one user's specific mutation (1:many mutation to disease).

7. Future Recommendations

The Utrition IV team has listed a summarized list of recommendations for future project iterations to consider implementing and enhancing. Some suggestions may be trivial adjustments, while others will prove critical to the integrity of the application and project. Regardless, this list encapsulates many of the ideas and optimizations Utrition IV was unable to take on during their capstone project.

- **Query Optimization.** The example query illustrated in Section 6, Figure 12 of this transition document is a working query that lists data for all found users' mutations and disease-risk associations. As future teams populate the database with tens of thousands of datapoints, however, this specific query would be deemed inefficient. Future teams should research query optimization and the utilization of query nesting to create time-savvy and cost-effective querying.
- **Artifacts and Documentation.** Thanks to the diligent work of Utrition IV, future iterations of the project have access to various resources and sources of documentation outlining steps needed to setup software; execute and access the Utrition web application; testing guidelines and design decisions; and several more helpful artifacts – including tutorial walk-thru videos. Utrition IV asks that the next team carry on this practice by contributing / enhancing said artifacts, to ensure all future iterations are well-prepared from day one.
- **ML / NLP**. Once a major goal for Utrition IV, machine learning and natural language processing is integral for the future success of this application. Teams should work to incorporate predictive language (autocomplete / word completion) into the app. For example, as a user begins to enter "br", foods such as broth, broccoli, and bread should

begin to appear as clickable suggestions. Additionally, when a user enters something like “apple”, the fruit should be the first suggestion- not foods like “apple juice” or “apple chips”.

- **Testing Suite Expansion.** Currently, the testing suite focuses on table insertions and constraint checking for database tables. As the project evolves to incorporate more UI/UX features and the involvement of ML/NLP, additional test coverage will be required. Additional recommendations include testing for SQL injection when users are entering data for food recall.
- **UI/UX Enhancements.** Utrition IV has a few recommendations with respect to enhancing the user interface and user experience:
    1. **Overlap in “What to Eat” and “What Not to Eat**”. In the dietary analysis, certain foods and substances are recommended to the user to consume based on a downregulation for a disease said user may be at risk for; additionally, the same item may upregulate another disease associated to the user. Future teams should utilize code to remove this overlap or find a way to further specify the situation.
    2. **Security and Account Profile**. All web applications are concerned with security. Utrition IV suggests introducing a password requirement to the web app, as well as an account profile page for editing / viewing a user’s profile and settings.
    3. **Recommendation Condensing**. Currently, foods such as peanuts appear in six or more variations, whether they be roasted / unroasted, salted /

unsalted, etc. Future iterations of Utrition should consider condensing repetitive recommendations.

4. **Uploads and Downloads**. Next iterations should enable the option for a user to download their dietary analysis. Additionally, users should have the option to reupload their DNA file or replace their current file with a different file.