Data Management

The folder **data_management** contain the programs which were responsible for gathering and organizing the nutrition data we pulled from the FDA's database. The program **scraper.py** was the program that downloaded all of the nutrition data onto my computer. It works by making successive GET requests to the URL http://ndb.nal.usda.gov/ndb/foods/show/####?reportfmt=csv which leads to a csv where ### represents the number associated with a particular food. The program then writes this csv file into memory. The program **insert_info.php** is relatively straightforward. It reads every nutrition csv nutrition file stored in the folder Food_Data. Then it takes the data from the third column which contains the nutrition information of the food per 100g and inserts that data into the MySQL table called food_data. **Insert_identifiers.php** works in a similar manner (this program was not combined into insert_info.php because we thought to add it later). This program reads the csv files and looks for the line that contains the phrase "Nutrient data for:". Once it has found this phrase it parses that cell and adds the result to the database. For example, if the line were "Nutrient data for: butter, salted, low-fat", then the program would insert "butter" into Id_1, "salted" into Id_2, and "low-fat" into Id_3, the remaining Id's would be left as "N/A." **Servings.php** reads the csv files and looks at the last column of the file to see if an sort of serving information is provided. The program then uses a number of predetermined keywords that act as serving types and compares them to whatever serving information is in the csv file. If the serving size is in grams and the type of the serving matches one of the keywords, that the relevant serving information will be added to the SQL columns, serv_type and serv_size. **Food_groups.php** adds the food group that many items belong in to the SQL database. All of the ranges in this program were generated by looking through the data to see how the items were grouped. We considered processing the Id information to determine what food group an item belonged in, but the Id's were too different for us to easily program in different cases that would allow us to assign a particular item a food group. The notable exceptions however were "Beverages" and "Alcoholic Beverages."

Web Management

In **add_item.php,** we give the user the option to add their own personal food items if they were unable to find the one they wanted through search.php. We had 2 forms for add_item, the first being add_item_form.php which is displayed every time someone reaches the page via a get request. In **add_item_form.php** we had several forms that would be submitted: Id_1, Id_2, calories, amount and food group. These would be each inserted into our database and were there for the user to be able to identify their own inputted food items. Id_1 and Id_2 are just descriptors for each food name as you can see if you look at the mysql table food_data. Amount and calories allowed the program to include this new food item when computing daily averages and such in nutrition.php. Food Groups are identifiers we used for recommendation.php (we will explain why we have them later). The javascript at the bottom of **add_item_form.php** just runs

error checking for the inputs of the form and prints an error to the page if the user fails to enter stuff in correctly. It's very important that the user enter their data into their food log (and our database) correctly because we want the user to be able to see everything they entered and for their added food items to be included in the computed health metrics. If the user entry passes all the tests then that information is sent to **add_item.php** via post. If the item_no of the food was already set (see line 15), then the item that the user requested to add was already in our database -- item numbers are not assigned until a food has been added to the database. If just the Id_1 was set, than we know it was a user submitted result. We use htmlspecialchars to prevent injection attacks and then if the food was entered in ounces we convert it into grams so that everything stays consistent when we do our calculations later on. We then find the calories per 100 grams for this food item so that the user can easily compare the caloric content between 2 items without needing to do conversions. After this, we insert this new food into our food_data sql table and set the item number equal to the id of the food. Once all of the table information has been updated, we render the form **add_item_success_form** which is basically the same as add_item_form but with text notifying the user that they successfully added food into their food log.

Home_page.php is a simple views file that welcomes the user to the website. It gives a brief summary of what Health+ does and what the purpose of the website is. There is also a brief slideshow of photos drawn from the "img" file that we made using javascript. We directly implemented the slideshow in the home page because we didn't want to put any of it in header because then every page would have that code that was only designated for **home_page.php**. We limited the slideshow to 500x300 pixels and had a new image display every 3 seconds through a recursive defined function slideit().

Search.php controls one of the most important aspects of the website, letting the user query the database. The code behind the search is pretty simple--we just take the user's search and then match it up against all the Id's in the table and the results displayed are shown according to SQL's ranking algorithm. The text in each search result can be clicked on to see more information. If a search returns no results (i.e. that count($foods ==0)), then text will appear that will contain a link to the add item page where the user can manually input a new food.

In **recommendations.php**, we first selected all the items that a particular user has logged. We then declare an array with each food group as a key value pair with its count (all initialized to be 0). We then went through a foreach loop to cycle through all the items that a user has logged to count every occurrence of a particular food group. We then determine which was the first and second highest occurring food group. We figured that the first and second highest occurring food groups would give us a relatively good idea of what the user likes to eat and so we could give recommendations based off of that. We then have a series of if statements to

determine what type of recommendation to give and what data to pull from the database. For the users most preferred type, if that is not alcohol or fast food, we give them a recommendation that is of the same type. If they like fast food, we extract fruits and vegetables from the database. If they like alcohol, we suggest to them other beverages. If they have no food preferences at all, we pull fruits and vegetables by default. We go through the same process for the second most liked food group. We then render **recommendations_view.php** and pass in the 6 (in total) food recommendations. In recommendations_view.php we have another series of if statements to deal with the multiple cases. If they like a particular food group that isn't fast food or alcohol we print out a particular message with their recommendations, and if they do like at least one of the 2 then we print out another message along with their recommendations. Again we did the same for their second most liked food group. If they had no preferred food group we just display fruits and vegetables by default.

If a user clicks on **All Food** in the menu bar or on **Food Log** below that, he will be sent to the page controlled by **food_log.php**. The program **food_log.php** will get all of the food the user has logged by querying the table **food_logs** in MySQL for all the items that match the user's id. This will then return the item number of every food the user has logged. Once we have the item number we query the main database (food_data) for the nutritional data associated with each item. These results are then sent to the view **food_log_result.php** with a "success code" of 1 where they are then displayed in a table. Each row of the table can be clicked on to see more information. If the success code is 0 and no results were found, the user will be alerted to this.

In **help_view.php** we have a simple page of in-page links that describe how to use the site. Nothing special.

scripts.js defines the function openWin(), which is the function that displays all of the more info popups. It takes in the item number and then requests the food_log_item page with the query ("q") equal to the item number.

helpers.php is where the logout(), render(), redirect() functions are defined. Their implementations are the same as those written by the CS50 staff. The one new addition is the function called render_pop_up() that styles popups differently than the rest of the website.

In **nutrition.php** we first render **nutrition_history.php** with a success code of 0 so that they will be prompted with 2 forms to get a start and end date for the desired timespan a user would like to observe. When submitted, that information is passed back to **nutrition.php** via post at which point we run checks to make sure the user has entered valid dates. After that we convert the start and end dates into proper date-time format. We also declare an array of key

value pairs that holds the recommended daily values for each nutrient and an array that holds all the proper units for the nutrients. After that we find the time interval between the two dates the user has entered. If the number of dates is 0 or negative, we will just change it to 1 so that everything will continue to work fine. Then we declare a series of variables, each assigned to keep track of the total amount of it that a user has consumed. $log keeps track of the food that was entered within this time frame, $nutrition holds all the nutrient data, and $nutrition_daily holds the daily averages. We then render nutrition_history.php with all the new information with a success code of 1. With a foreach loop we print out both our tables row by row. In our second table, nutrition summary, we also do some calculations to find the percent of daily recommended value using the totals we found and the daily recommended value from the FDA website. We have also embedded in-page links to make navigation of the page easier, especially since the tables can get fairly long. At the end of nutrition_history.php we run checks to validate the user's inputted dates.