Meal Equivalent Calculator

Fareed Khamitov

CIS4914, Senior Project
Department of CISE
University of Florida

Advisor: Dr. Borum, *email*: prb@ufl.edu

Abstract

Hippocrates (400 BC) said “Let thy food be thy medicine and medicine be thy food” and more than two millennia later we are trying to do that. Like the popular ketogenic diet, Precision Ketogenic Therapy induces nutritional ketosis. However, PKT is a medical therapy – not simply a special diet. Compared to the typical diet, PKT reduces intake of carbohydrates and replaces those calories with fatty acids. Compared to other ketogenic diets used to treat epilepsy, PKT is more precise in administration and monitoring. Current way of making recipes for the PKT patients is only through excel sheets. Meal Equivalent calculator web application will be a next step in helping patients and their families. Meal equivalent calculator will be a multi-page web application that will be hosted on the ResVault (HiperGator) that is going to help clinical consultants create meal equivalences for patients using patients diet prescription. The application will be build using python (Flask), PostgreSQL, SQLAlchemy in the back end with using postgreSQL database for storage, retrieval and querying the data. Python (Jinja) and HTML/CSS will be used for the main structure and style of the front-end with JavaScript, AJAX and JQuery performing calculations and proper functionality of the application.

1. Introduction.

The need for the updated web application arose from unreliability of the Excel spreadsheet where the current meal equivalent calculator is stored and operated. Besides being slow, due to the usage of a large foodonomics database, and limit on certain functionality, current meal equivalent calculator is also prone to human error where one wrong click can mess up the function in the cell that eventually can cause wrong calculations in the whole recipe.

My plan for this semester is to create user-friendly, robust, and reliable application with a thought that it will be keep updating, growing, and become useful application for the clinical consultants as well as other professionals. My main motivations behind taking this project were to help people in need with their well-being heavily relies on their diet (being husband of type 1 diabetics showed me first-hand what proper nutrition can do), as well as learn web-development. Having experience with web design, I lacked the proper training in technical aspects of web development, its practices, techniques, and terminology.

1.1. Problem Domain.

As more patients come for help from clinical consultants, students, and professors fast and reliable recipe/meal equivalence creating process. When completed meal equivalent calculator the application will be useful not only for UF team, but also other medical professionals. The meal equivalent web application falls into the field of web development and software engineering.

1.2. Previous Work (Literature Search).

To determine state-of-the art, I investigated several web-applications and websites such as MyFitnessPal, MayoClinic, WebMD and omniCalculator to get the idea of how to better plan the layout of the app. Then, I considered the existing meal equivalence excel spreadsheet, its mathematical functions, database relationships, and complexity. This let me properly design and get a plan of implementation.

Since I was required to work with Flask (a web framework for Python) which was connected to the PostgreSQL database system and I had no prior experience in either I had to use online courses to get basic concepts fast. Firstly, [Portilla] discussed basic set up and implementation of Flask project and “how-to” create small, one-page application. After getting basic understanding I moved to more complex examples. [Grinberg] implements more complex blog application where he describes how to connect to the database, web-forms and basic JavaScript, AJAX. [Grider] gave basic understanding on how to use PostgreSQL and operate pgAdmin 4 which simplified a lot of tasks.

In the second case, the author [Medina] dives more in depth with WTForms, which is a major part of the web-application. [Herbert] gave better understanding of how properly design the application to achieve “clean code” and implementation that will be understandable by the future developers and deepened my understanding of SQL-Alchemy and JavaScript. Finally, mathematical functions and equations implementation in the meal equivalent combined with the PostgreSQL tables requires complex JavaScript, JQuery and AJAX functions discussed in [Davies] article.

2. Technical Approach (Solution).

In response to the lack of reliable or robust recipe calculator that would work fast on multiple machines, I have developed meal equivalent calculator we have developed *Astute*, an object-oriented, GUI-driven web application for clinical consultants to create, view and edit meal equivalences. It is written in python (Flask), HTML/CSS and JavaScript/JQuery. It also uses postgreSQL and SQLAlchemy for database management. Meal equivalent calculator allows patient lookup, and data entry with on-page calculations using calculator interface with relational database constructed using PostgreSQL and SQLAlchemy, as well as viewing previously made equivalences presented as a list.

Meal Equivalent is part of the bigger PKT application. The UML diagram and the design of the application is shown in the Figure 1. Elaborations of this design and each part of the application are omitted in this project for purposes for simplicity.

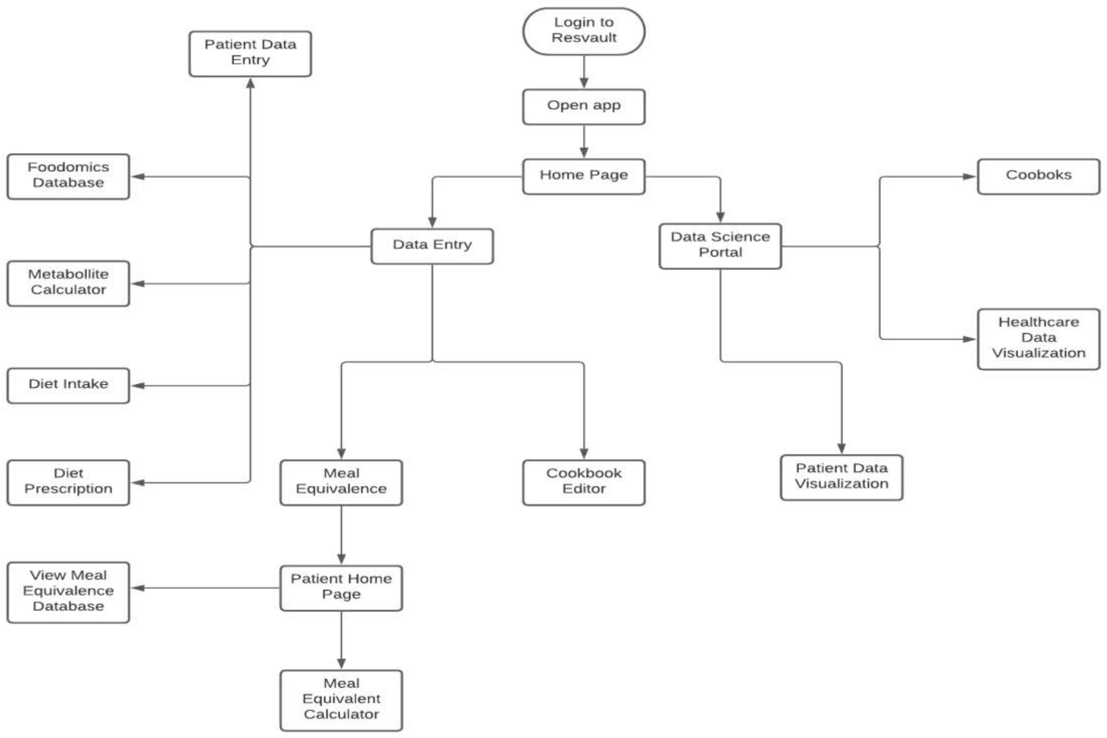

Figure 1. UML diagram of PKT application

In order to make application fast and user friendly to use for everyone I tried to make sure that every page that user is landed on is simple and easy to understand what it does. Each page designed to do one or two things but do them fast and efficient. Flowchart of the meal equivalent calculator is displayed nn Figure 2.

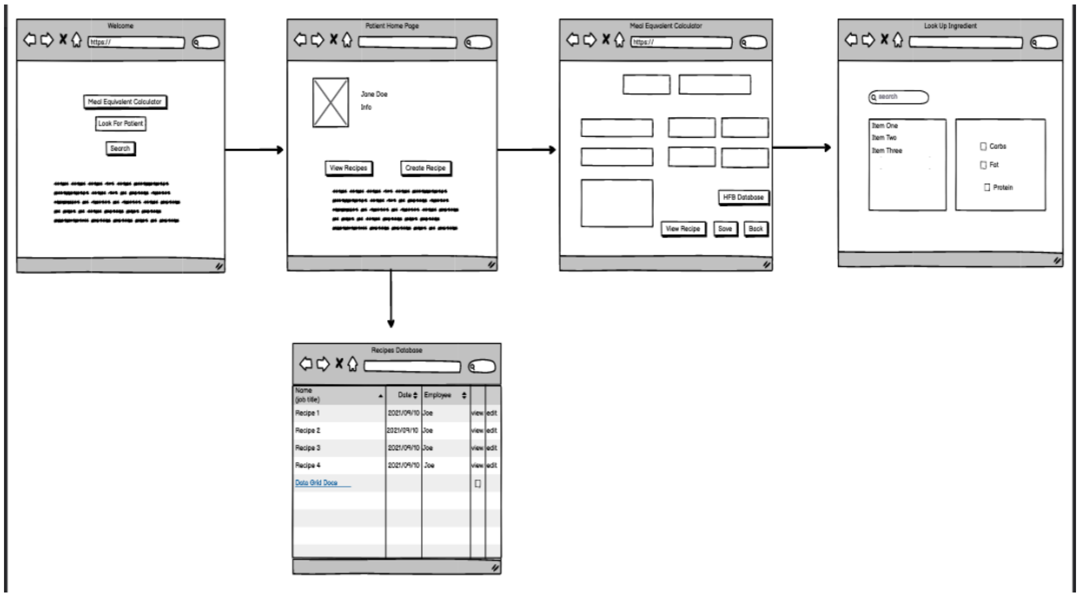

Figure 2. Meal Equivalent calculator flowchart

Besides using multiple database tables that are created by teammates and those that already exist in database, calculator needs its own tables to be able store each meal equivalence for future reference and usage. To properly store and retrieve such an information, it is necessary for tables to be retrieved parsed through and displayed fast in the front end with ability to create a lot of ingredients in one recipe/meal equivalence. This is done by creating two tables: meal equivalence and ingredients with one-to-many relationship between them. The ER diagram used for the meal equivalent calculator is displayed in Figure 3. As stated before this is only one part of a bigger PKT application and the whole ER diagram is not shown.

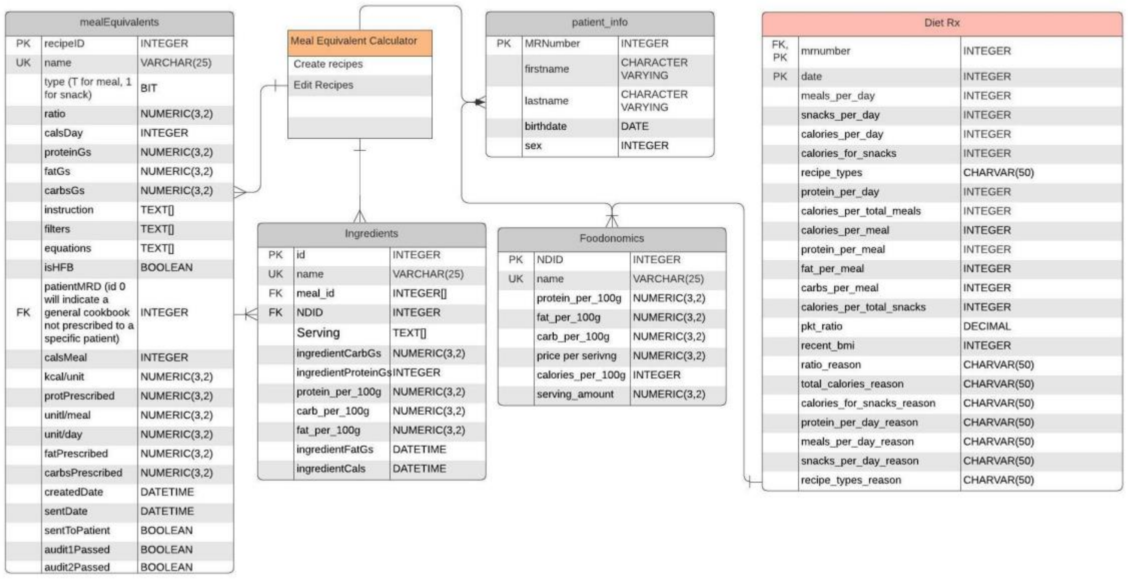

Figure 3. ER diagram for meal equivalent calculator

Software development at the database level incurred few challenges or problems, due to the inconsistency in developments between teammates. Several of us used python (Flask) to create and manage database tables, while other created it using command line. However, no major problems were encountered linking the database to the GUI. Only thing that might possibly produce bugs in the future is if other developer of the lab, who is working on database that meal equivalent is actively using, decides to change name or the type of the column it might cause interruptions and inconsistency. PostgreSQL and Flask are connected to each other with SQLAlchemy, a toolkit and Object Relational Mapper. This simplifies a lot of processes and tasks by localizing most functionality to one place. Each page in M.E. calculator is either using, creating or updating database tables, so the integrity of each table is extremely important for full functionality. For example, all pages use diet prescription table, patient home patient is also using patient information, while calculator page uses all of them as well as writes into the meal equivalent and ingredients tables. Besides SQLAlchemy, JSON is used for a fast information exchange between back and front end of the application.

Application uses a lot of equations using both user input and data pulled from database tables. Most of the equations and functions are implemented in the front-end using

JavaScript and JQuery. The functions and equations implemented in the prototype version of the calculator are displayed in the Table 1.

Name	Equation
Grams of Macronutrient in food	(Amount of macronutrient per 100g/100) * Amount of food in grams
Kcal/unit	(Meal Ratio * 9) + 4
Units/day	Calories per total meals * kcal/unit
Units/meal	(Units/day) / Number of meals
Prescribed Fat per meal	Units/meal * meal ratio
Prescribed Carbohydrates per meal	Units/meal – prescribed protein per meal
Total amount of ingredient	((adj. Protein * 100) / percentage of protein in ingredient) + ((adj. fat * 100) / percentage of fat in ingredient) + ((adj. carb * 100) / percentage of protein in ingredient)

All equations were pulled from the excel sheets and are coded and tested in JavaScript and JQuery. Since the plan is to enhance the prototype in the future, all calculation are done "live" in the front-end which makes it easier for them to be changed and updated if needed.

Testing and debugging of the calculator prototype took the most of the semester since most of the processes and type of functionality were new to me. For form handling Flask library WTForms allowed a lot of required functionality and is fully compatible with JQuery function for creating, updating, and storing forms.

3. Results

The result is fully functional multi-page web application, the UI of each page is fully functional and has been tested. The calculator prototype tested successfully, performing all the equations listed in Section 2 as well as creating, viewing and updating the information in the database within the testing environment. The prototype still needs to be tested against the meal equivalent excel spreadsheet in depth to cover the edge cases and outliers.

All tests were conducted using the information copied from the meal equivalent excel sheets that was stored in the "dummy" databases, so further testing with real database table implemented on ResVault (HiperGator) is required. The time and space complexity tests were not performed due to the same reasons that I do not have access to the actual patient and foodonomics databases in order to properly test load and calculations time.

4. Technical Challenges and Solutions

As stated previously, biggest challenge for this project was starting from 0 knowledge in both python and JavaScript and building the whole application from there. In order to complete if not all at least most of the goal I had to dive deep into the research right away. I took several Udemy courses online, had one on one tutoring sessions, and a lot of documentation reading.

During the development process two major roadblocks became the issue: incontinency in database development in the team and complex functionality in the calculator prototype using JavaScript. Firstly, I had to make sure that the table names and columns that I use are the ones that will be developed and added to the ResVault. From there solution for the problem is clear communication and documentation within the team. If somebody updated their part of the project the rest needed to know about it, so they can adjust accordingly. The way Flask processes database tables are either it is created using Flask and SQLAlchemy and stored in the tables.py document where developers can clearly see what columns are existing in the table or create the via the command line. This adds more complication to those who use the tables since the information about the table is not easily visible. Since some team members used command line in the creation of the tables, I needed to accommodate those type of tables using psycopg 2 database driver. Since I used SQLAlchemy tables created in Flask the proper adjustment will be needed when projects are transferred to the ResVault.

In the front-end, I had spent a lot of time researching and developing proper way of cloning the ingredient form. Since each form field had to be unique, and each meal equivalence can have anywhere from 1 to 20 ingredients I could not create a form with set amount of field or copying the same form over and over. The solution to this problem came through use of JavaScript, JQuery and JSON. Since I already am using onchange handler an event listener in the form fields like NDID ingredient search, I had problem with making sure that every form that was copied also created unique id for each form field in order for me to utilize and use users input in calculations. The function in JQuery listens to the button 'onclick' event when it is clicked it creates new 'div' which is an exact copy of the parent form, but with updated form names, id and it properly appends itself to the ingredient database table after user is finished.

5. **Conclusions**

As a result, I have fully functional meal equivalence calculator application with several pages that work the way they were planned in the beginning. The home page displays instructions and has a field for patient MRnumber lookup which leads to the patient page that displays patient diet prescription and information. From there user can either see list of the created recipes for the patient or create new meal equivalence. The calculator prototype has most if not all functionalities as excel spreadsheet.

Through this process I learned way more than I expected. Besides learning python and postgreSQL which I knew I will have to learn how to work with in the beginning of the season, most of the things I learned was in the front-end. I came with no knowledge of how web development works and being able to create a working web application. I learned a lot of JavaScript and worked with JSON which is important in the industry right now.

Since I learned as I worked on the project and sometimes the development was more reactive than proactive due to my inexperience, I think that is also main weakness of the application. The prototype works properly and as I intended it to work, however I don't know if the way I implemented certain features is the proper and correct way in the long run.

The only functionality that was planned but is missing is ability to edit existing meal equivalences. That should be included in the future work and improvement of the meal equivalent calculator application in the future. Another functionality that was discussed but couldn't be included this semester is the healthy fat blends and profiles of different types of fats in the ingredient and recipe in general.

5 .Standards and Constraints

Standards: Python programming was done in PEP 8. For JavaScript I used Google JavaScript Standard.

Constraints: All software was required to run on an Intel i7 quad-core processor (or better) with latency less than 10 milliseconds.

6. Acknowledgements

The author would like to thank his advisor, Dr. Peggy Borum, for her guidance, advice, and encouragement toward successful completion of this project. Additional thanks go to my clinical consultants Paige Pressley, Emma Hammer and Renata Volonterio. Lastly, I want to thank my teammates, especially Octavio Ochoa, and Berlin Sankar for their constant help and advice in the process.

7. References

[Davies] Davies, Alan. "Using Python Flask and Ajax to Pass Information between the Client and Server." *Medium*, Towards Data Science, 18 June 2021, towardsdatascience.com/using-python-flask-and-ajax-to-pass-information-between-the-client-and-server-90670c64d688#309d.

[Grider] Grider, Stephen. "SQL and PostgreSQL: The Complete Developer's Guide." *Udemy*, Udemy, Feb. 2021, www.udemy.com/course/sql-and-postgresql/.

[Grinberg] Grinberg, Miguel. "The Flask Mega-Tutorial." *Miguelgrinberg.com*, blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world.

[Herbert] Herbert, Anthony. "Pretty Printed Flask Tutorials." *Pretty Printed*, 2015, prettyprinted.com/.

[Medina] Medina, Rafael. “Dynamic Fields with Flask-WTF.” *Rmed*, 2 Mar. 2019, www.rmedgar.com/blog/dynamic-fields-flask-wtf/.

[Portilla] Portilla, Jose. “Python and Flask Bootcamp .” *Udemy*, Udemy, 1 Feb. 2020, www.udemy.com/course/python-and-flask-bootcamp-create-websites-using-flask/.

8. Biography

Fareed Khamitov was born in Almaty, Kazakhstan. Being adopted at the age of 12 he moved to Lakeland Florida. He completed his secondary education at Lakeland Christian High School. Starting his academic career at Florida Southern College with major in graphic design, he transferred to Polk State College to finish his Associates degree and gain necessary pre-requisites to transfer to University of Florida majoring in Computer Science graduating on December 17th, 2021. During his time at UF he completed two internship and throughout his academic career he worked at the ACE aircraft refinishing. Fareed enjoys programming in C++ and Java as well as web design and development. He accepted employment with Pubilx Super Markets (Lakeland, FL) as a software engineer. He is also an avid artist and sculptor working with mixed media as well as book collecting, hiking and kayaking. He also hopes to pursue an advanced degree in Business Administration while employed in the software design and development industry.