

Comp 4911 Project Report
“Bioactive Dash Web App”
Nobutaka Kim
T00057442

Contents:

1. Executive Summary	p. 3
2. Manual Testing	p. 4
2.1 Databases	pp. 5 to 6
2.2 UI	pp. 7 to 11
3. App User-Flow (shown in accompanying Youtube video)	(video)
4. Outstanding Issues	p. 12
5. Conclusion	p. 13

Executive Summary:

I achieved what I set out to do which was to make a web app that tracks the consumption of micronutrients and macronutrients based on foods in the Canadian Nutrient File Database [0.1] using a MongoDB version of it [0.2] and to encourage the consumption of plant-based foods by providing alternative ingredients and recipes [0.3]. However, I came across a major stumbling block that renders it impossible to further develop the app before the submission deadline with a working model so what I have now is the last milestone before a required research period during which I need to put on my nutritionist/biologist/chemist hat and learn about the individual nutrients in more detail in order to redesign and enhance the app. There is plenty of material on the subject for free but it is heavily cited and the actual data is usually drawn from the very last source document in the chain of citations [0.4]

The app itself was written in Python using Plotly Dash which is one of the most popular libraries for data science web applications and it was easy to see why. There is tight coupling with Pandas which of course is based on Numpy so this naturally leads to easy integration of Python's ML and other scientific computing libraries such as Biopython for Bioinformatics [0.5]. It is 100 percent asynchronous so what took me many lines of PHP, JavaScript, HTML and CSS code in Comp 3540 can be accomplished with a quarter of the work load, especially in terms of developing a prototype rapidly where style and other idiosyncrasies are less important. Because I can add external CSS and JavaScript or TypeScript files to my Dash app, and because Dash is piggy-backing on Flask, the possibilities are endless. Adding MySQL tables also let me learn the basics of SQLAlchemy using selections, updates and insertion queries where as Pandas has methods for uploading dataframes directly to MySQL databases as tables as well as to run SQL queries. Thus I feel as though I have run the gamut in terms of full stack web development and am grateful that this prepared me to continue enhancing my app and to take on roles as a web developer.

I am using a Youtube video and links to my repo in lieu of providing flow charts and in the case that you need to clone my repo and run the app locally to follow along, a Google Drive link with all the files is provided [0.6].

To be honest, my app is completely outclassed by nutritionix's natural language processing capabilities [0.7] so moving forward, I may opt to use it instead of the CNF database and my simple UI since there is no way that I can include nearly 1 million foods including brands and restaurants. As a simple comparison, the CNF contains about 5,600 foods and the USDA equivalent about 8,500. Thus, I am learning quickly that starting a company is not cheap and if I fall on my face, I am going to be out for all the API calls that are made to nutritionix by my users and I believe they have a tiered pricing system so I may have to commit to a certain number for a certain period of time but I plan on negotiating using the Founder Institute's clout covering my back. However, I will make at least a few attempts at using open source NLP neural networks to transfer learn my own.

Also, I need to redesign the alternative ingredients engine since the Bon API [0.3] is inconsistent and their database is not quite as up-to-date nor relevant to the North American market as I wished. Thus, this probably means digging around Github for an open source alternative ingredient recommendation engine although a quick Google search did not show any meaning I probably need to roll-my-own which again comes back to the need for a research period during which I learn enough about phytonutrients such as flavonoids, anti-oxidants, isoflavones and others so my engine can make recommendations based on the profiles of these for foods rather than say just based on names of foods.

For the reasons above, deployment is delayed until late January at the earliest which gives me about a month to collect enough user data and feedback before my Founder Institute program begins in March, 2021.

Manual Testing:

Manual testing of functions was performed in Jupyter Notebooks of which there are five in /1_foodsByNutrients [1] and integration testing was performed by me, again manually, by using the app as a “normal” user [2]. There were several reasons why I chose manual testing over automated testing at this point but the main one was that there are many upcoming changes to the UI and backend MySQL and Mongo databases for which I have over 20 open issues on Jira.

I stumbled upon manual testing while reading the Dash Testing documentation and about it on the Dash community forum [3]. It is heavily based on PyTest, Selenium and Chrome Web Driver of which I have tested the previous iteration of this app using PyTest [4] and it just seemed to be overkill as the app itself is a work-in-progress. Furthermore exploratory testing for corner cases, testing small changes in UI, feedback on human-computer interaction, ie. usability and adhoc testing is more easily performed manually [5].

The following were tested extensively in the Jupyter notebooks:

- database interactions for select, insert, update, create table, alter table in MySQL.
- User authentication using Flask-User and MongoDB
- User authentication using Flask-Login and MySQL
- Pandas dataframe manipulations, SQL queries and uploading dataframes to MySQL
- sqlalchemy basic functionality (note: foreign keys are not used in my current schema but rather separate queries made to each table which adds to execution time)
- parsing and editing data from Canadian nutrient file, Recommended Daily Allowance for nutrients from the USDA and calculating percentages for ingredients, meals and time periods

The gist of it is that my app is very simple at this point. Thus I believe manual testing was sufficient as Plotly Dash fails gracefully in most cases as does Pandas and the weak links in the chain that can cause the app to crash such as the sqlalchemy queries were learned from a DataCamp mini-course I enrolled in as part of the Github Student Developer pack that provided a free three month trial.

Databases:

As mentioned in emails, I switched the template on which I was building twice going from an edited repo of [0.2] to the Gentellela dashboard UI template of [4] to the current simplified prototype-esque UI of [6]. I was also very keen on learning no-SQL which added a significant hurdle early on as I could no longer count on writing simple SQL queries in the form of “SELECT * FROM table” to pull data into dataframes and then using my slightly better knowledge of Pandas to slice and concatenate the query results into dataframe “tables”. Thus it was extremely frustrating at first but this was a great lesson in improvisation and looking for alternative solutions that leverage my current knowledge as I copied the Mongo queries that pre-existed in [0.2] in routes.py that was filtering query results and then looked at each of the respective Jinja templates, basically HTML mixed with Python, that iterated query results for presentation in the view. I additionally wrote a few queries but these were enough to do most of the work in terms of interacting with MongoDB to get the CNF data I wanted into my Python files where I could convert JSON to dictionaries making life much easier. Although my strengths were more in Numpy and my coding “style” was over-reliant on nested arrays before, now I feel as though dictionaries are much more efficient in some cases, thus I feel that MongoDB is worth another look as collections are basically nested dictionaries. On the down side though, the documentation for Flask MongoEngine, MongoEngine left something to be desired so moving forward I would probably use the official Python Mongo driver which is PyMongo.

Once I found nutriana [7] I ended up abandoning my RDI (recommended daily intake) models and Mongo collections that I had written in favor of MySQL after discovering how easy it was to call:

```
#upload dataframe df to MySQL:
df.to_sql("table_name", con=engine, if_exists='append', index=False)

#download table from MySQL:
sql = "SELECT * from " + table_name
pd.read_sql_query(sql, engine)
```

The above is in Pandas but I saw the pitfall of over-relying on MySQL as a temporary storage that can be used to pass data from one page to another within an app. I do this once with the responses for my API calls to Bon API alternative ingredient API where I store them in MySQL the previous ingredient and alternate ingredient in the same rows and then these can be queried in the “Make Meal” page so the user has a table of possible replacements to search while inputting a recipe or meal description. In a production environment where latency of SQL queries might be an issue, this is probably not recommended but since my use-case fits in the use-flow of my app where a users realize from looking at their past meals that they need to replace some items to make up for nutritional deficiencies and then being able to have these on the screen when entering their next meal seemed convenient.

Cleaning the data and matching two sources, in this case nutrients from foods in the CNF database and RDI (recommended daily intake) from the USDA, both measured in units such as g/day, mg/day, ug/day and L/day and with varying names including adjectives like “total” or “(scientific names)” vs. common names and minor discrepancies such as “vitamin b12” vs “vitamin b-12” would have been easy to handle using regular expressions but unbelievably these official sources of data had inconsistencies that made applying regular expressions impossible at my level of understanding so I resorted to hard-coding some of the translations although moving forward these will be refined to be

done programatically as I intend on increasing the number of tracked nutrients to the entire collection of nutrients listed for foods in the CNF database including individual amino acids.

There is an entire notebook dedicated to this called `uploadRdiCsvMySQL.ipynb` in the directory mentioned earlier [8].

Finally, you will notice that my bar charts are missing some bars for the nutrients listed on the x-axis but that is because I have not been able to match up the CNF and RDI data yet for example, vitamin E is alpha-tocopherol, linoleic acid and alpha-linolenic acid have a few variants as well as carotenoids of which there are over 600 types. Also, I just discovered that I should be doing something like:

```
if rdi_nutrient in cnf_nutrient
```

where both variables are strings, rather than

```
if cnf_nutrient in rdi_nutrient
```

to check for matches since the CNF name is usually longer, contains extra words as mentioned earlier or other modifiers. Thus I am currently working on combining matches from searches in both directions.

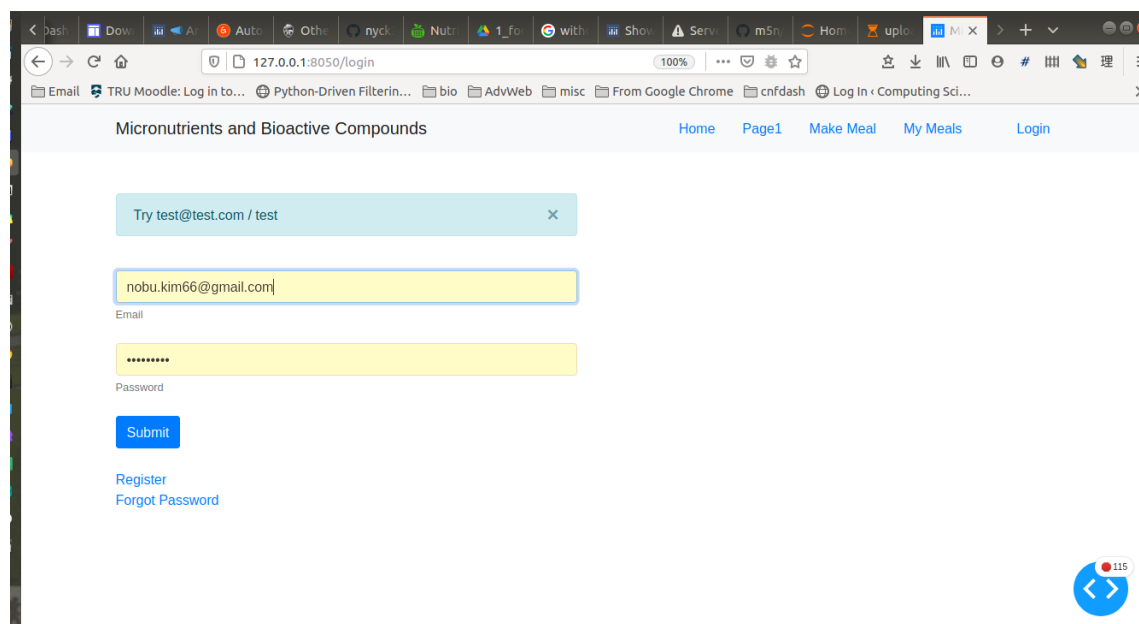
Please check folder `databaseSqlFiles` for the .sql files that can recreate the MySQL databases and all tables used for local testing of the app. `phpMyAdmin` [11] was used extensively as the GUI and `MySQL Workbench` [12] for schema creation and alterations.

To recreate the MongoDB database for Canadian Nutrient File nutrient info, please follow instructions on [0.2] repo. After creation of database 'cnf', connection on localhost port 27017 (standard for MongoDB) should be possible from the app. `MongoDB Compass` was an excellent GUI tool for examining the schema in table format [13].

UI:

Login:

Flask-Login is used and I believe it is possible to download the HTML templates, as I did for Flask-User in the previous version of the app following the Flask documentation, so the login page below can be edited beyond having a nav bar to provide whatever type of UX is preferred.



The screenshot shows a web browser window with the address bar displaying '127.0.0.1:8050/login'. The page title is 'Micronutrients and Bioactive Compounds'. The navigation bar includes links for 'Home', 'Page1', 'Make Meal', 'My Meals', and 'Login'. Below the navigation bar, there is a light blue box with the text 'Try test@test.com / test' and a close button. Below this, there is a yellow input field for 'Email' containing 'nobi.kim66@gmail.com'. Below the email field, there is a yellow input field for 'Password' with masked characters '*****'. Below the password field, there is a blue 'Submit' button. Below the 'Submit' button, there are two links: 'Register' and 'Forgot Password'. In the bottom right corner, there is a blue circular icon with a white double arrow and a red notification bubble with the number '115'.

The “Forgot Password” functionality is not yet implemented, though it was with the Flask Mail integration in Flask-User in the old version, but once logged in the password can be changed by clicking your name that appears in the nav bar. But first registration of your name and email can be done by clicking “Register”.

The screenshot shows a web browser window with the address bar displaying '127.0.0.1:8050/register'. The page title is 'Micronutrients and Bioactive Compounds'. The navigation bar includes links for 'Home', 'Page1', 'Make Meal', 'My Meals', and 'Login'. The registration form contains the following fields: 'First' (text input), 'Last' (text input), 'Email' (text input), 'Password' (password input), and 'Confirm password' (password input). A blue 'Submit' button is located at the bottom left of the form. A blue circular icon with a white arrow and the number '115' is visible in the bottom right corner.

Once logged in, click your name in the nav bar and you are taken to a page where you can change your name and password but not the email.

The screenshot shows a web browser window with the address bar displaying '127.0.0.1:8050/profile'. The page title is 'Micronutrients and Bioactive Compounds'. The navigation bar includes links for 'Home', 'Page1', 'Make Meal', 'My Meals', 'Nobutaka', and 'Logout'. The profile page is titled 'Profile' and contains the following sections: 'First:' with a text input field labeled 'Change first name...'; 'Last:' with a text input field labeled 'Change last name...'; 'Email:' with the text 'Cannot change email'; and 'Change password' with a text input field labeled 'Change password...' and a 'Change password' label below it. A blue circular icon with a white arrow and the number '115' is visible in the bottom right corner.

Plotly Dash has a `dbc_components` library (Dash Bootstrap Components) so I was able to use the familiar grid system of Containers, Rows and Cols to organize my UI. There are some parts that are still quite awkward for instance when the user searches foods by nutrient type, the resulting table is in the right-hand column extends the Row so some of the UI in the left Col (both of width=6), which is in the subsequent Row gets pushed down.

Before:

The screenshot shows a web browser window with the URL `127.0.0.1:8050/make-meal`. The page title is "Micronutrients and Bioactive Compounds". The navigation bar includes links: Home, Page1, Make Meal, My Meals, Nobutaka, and Logout. The main content area is titled "Selected Ingredients" and contains the following sections:

- 1. Choose Ingredient by Name**: A text input field with the placeholder "Enter food name", a blue "Search Ingredient" button, and the text "Or Choose Ingredient by Nutrient".
- Or Choose Ingredient by Nutrient**: A dropdown menu with "Total_Kcal" selected, a blue "Search by Nutrient" button, and a yellow "get my alternates" button.
- 2. Amount Units**: A dropdown menu with "Select..." selected.
- 3. Quantity**: A text input field with a spinner icon, and a green "Update Nutrient Table" button.

A blue circular button with left and right arrows and a notification badge "115" is located on the right side of the page.

After:

Selected Ingredients

1. Choose Ingredient by Name

Enter food name

Search Ingredient

Or Choose Ingredient by Nutrient

calcium

Search by Nutrient

get my alternates

Description	Measure	Calcium, Ca(mg)Per
filter data...		
<input type="radio"/> Cheese, swiss	1.0 cups, diced	1175
<input type="radio"/> Whey, sweet, dried	1.0 cups	1154
<input type="radio"/> Cheese, pasteurized process, swiss	1.0 cups, diced	1081
<input type="radio"/> Cheese, provolone	1.0 cups, diced	998
<input type="radio"/> Cheese, muenster	1.0 cups, diced	946
<input type="radio"/> Cheese, cheddar (Includes foods for USDA's Food Distribution Program)	1.0 cups, diced	937
<input type="radio"/> Cheese, mozzarella, low moisture, part-skim	1.0 cups, diced	920
<input type="radio"/> Tofu, raw, firm,		

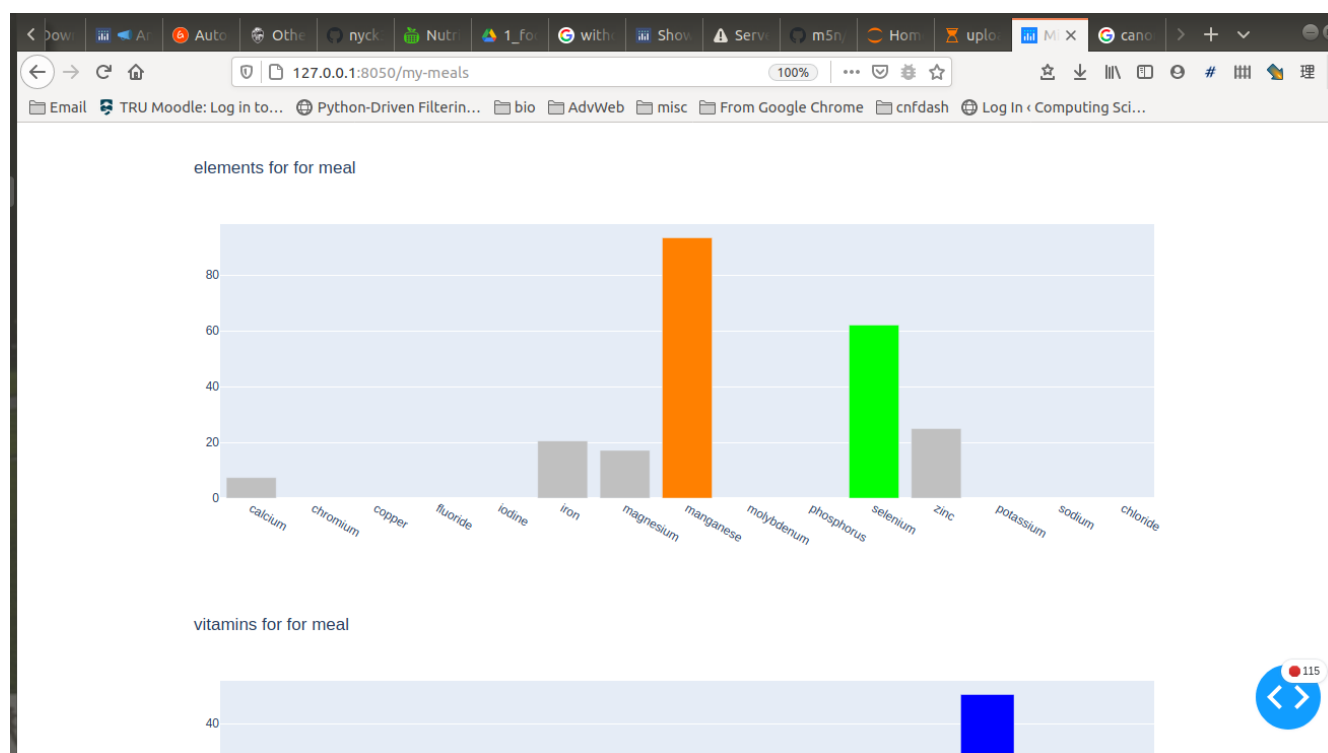
In the above screenshot, I am only showing 8 results per page of the table but it still seems to be too many. Reducing it to 5 is a small improvement but with almost 200 table pages, I should find another way to present the data although as you can see faintly in the top row are input boxes to filter the rows. Although the table looks selectable, I have yet to implement the feature here to be able to save “foods by nutrients” to a “My foods-by-nutrients” table. I am already letting users save all their meals and ingredients to MySQL so I have to investigate ways of compressing data while not losing the ability to make queries quickly as I can imagine that storage costs would add up quickly and possibly faster than linear to the number of users since I do not limit the number of meals nor ingredients per day.

Next, I am going to have to find some test users, ie. I had no alpha-testing nor beta-testing period this time around, reason being that I am in the middle with regards to the East-West contradiction in preferred UI styles with East preferring a more cluttered look than West so either I can make claim to a universal style or one which neither East nor West really comfortable with. However, this is only a prototype so some of the functions I currently write in Python could easily be moved to JavaScript or TypeScript later. On this note, I wish I had looked into TypeScript earlier as it seems perfect, having the canonical data structure names rather than pure JS where a class is a function() which for me reduces readability.

Also I forgot to mention in the video but ideally I should have a search function where users can select items after they search foods by nutrient name which then searches the CNF or USDA database for those foods and fills the input box with those items. The search by nutrient tables were taken from USDA so I would imagine switching in the USDA foods table of over 8,000 items vs about 5,500 for

CNF would facilitate the matches. At the current time users can only refer to the foods by nutrient table that pops up on the right hand column to filter their own inputs to the “search ingredient” input box.

Furthermore, I want to standardize the colors used in my UI to provide a consistent UX and for branding purposes. Currently my bar chart bars are colored closer to red as a nutrient intake level meets or exceeds 100 percent of the RDI/RDA, then orange, green, blue with grey being the lowest but this just looks awkward and messy. rather than informative so I am going to switch to red, blue and grey only with red over 100 percent and possibly for very low intake levels such as under 30 percent, blue 80 to 100 and grey for the rest.



All of the UI components shown above and in the video [2] are from Plotly Dash’s html_components, dcc_components and dbc_components with no other HTML, CSS or JS files.

Outstanding Issues:

These may overlap with or append issues brought up in this report and are exported from my Jira [9].

Please also note for the video part 1 [2]:

When I say, "The point is that it fails gracefully", actually the error was caused by the fact that I tried to save an ingredient with no amount or units and it remained in the cache for the data table of cumulative ingredients.

However, I do clean that up when I recreate a dataframe from the ingredients added to the recipe with the add-to-recipe button so in production that error will be hidden and will not have any effect as the demo video also shows that it did not affect what is actually saved to the database.

Conclusion:

To be honest, I was flying high being accepted to Founder Institute again for basically a similar idea I had 3 years ago which I outlined in the Software Engineering class I took in the summer with you but now that I have seen what nutritionix already has and how even Bon API is trying to integrate calculations of RDI percentages met in their responses, not to mention the abundance of apps with similar ideas on Github that look abandoned, I am having doubts about getting this to seriously work meaning earning revenue. Thus, I am going to go back into my assignments for Software Engineering to see what my software evolution plan was in order to try to piece together a concrete plan for success.

In the process of review, I have also learned about the popularity of fake gluten meat called Seitan in Italy. Yes, Italy. They are using it as topping for pizzas and as substitute ingredients for many of their favorite meat dishes. Furthermore, some Italians are reporting weight losses of 20 kgs or more based on the Seitan diet so I feel as though new opportunities present themselves, the more committed I am to plant-based eating which is becoming trendy even in North America. And to be honest, Beyond Meat just does not taste like real food at all. It is over-processed, contains too many unknown ingredients and definitely tastes has the after-taste of non-food like what I would imagine eating play-do is like. It reminds me of some of the bread I ate in China which tasted similar and probably was made of non-food ingredients as “foods” made entirely from chemicals is a major health problem there.

From a computer science perspective, I feel that this project, having the freedom to learn what I wanted, really rounded out my education since I felt as though I as over-specialized in computational photography, deep learning and AI for Robotics algorithms which I learned online but now with my web development skills, I have a way to present my work to potential clients and business partners. I am also less intimidated with data pre-processing and database interactions which is are important skills in many fields.

Being able to write an entire web app in Python with no other languages was the best thing about this project as I honed my Python skills such as list and dictionary comprehensions, new methods such as `isinstance(var, type)` and learning how Python imports really work by debugging errors when importing from other directories and circular imports. I also learned new encoding such as Base 64 when uploading files to a Plotly Dash app and more advanced syntactic sugar for functions that denote the parameter and return types.

Moving forward, further research into nutrients is the first priority so I can track more of them on my “CNF vs RDI” charts. Next would be re-examining whether to adopt the nutritionix NLP API or to build my own with the latter having a slight edge for educational purposes and cost factors. Third is the addition of even more bioactive peptides and compounds for which the data exists but needs to be matched carefully against any list of foods in order to ensure accuracy. This is going to involve emailing academics from Japan, China, Poland, Russia and the US to begin with who already have websites and databases in this field. Fourth, I am also going to look at the ecofootprint aspect which I had started out on but abandoned quite early due to the wide scope and depth of data that is available. But as we are seeing with covid, it is extremely difficult to convince citizens of Western countries to take individual actions that cumulatively have a positive effect on the good of the group. Thus, I should translate my app to East Asian languages and present it as an app to help prevent future pandemics since factory farms are pandemic factories [10] as well as mitigating climate change.