Reflections

As no members of our group had any prior experience with developing web applications, many of the initially planned implementations had to be changed or completely replaced with new plans. This is because we understood the problems better as we attempted to solve them. As the system grew, we gained new insights into how to improve the system.

The first change in plans was the source of nutritional data. Originally, we had planned to have our own MySQL database that contained nutritional information. However, this database was very difficult to query and often returned the wrong result. A Google search for a nutritional database revealed API's that could be used as databases. Eventually, we ended up using the FatSecret REST API, which can access a database containing 500,000 food products, returns the right result, and is easy to use. In hindsight, we should have researched API's that could perform desired functions at the beginning of the project, but nobody in our group even knew what an API was.

Second, our originally planned user interfaces and interactions were changed, sometimes drastically. As we developed the frontend web pages, we were able to see the system from a user's perspective. Often, this new perspective gave us new insights that we used to make the frontend more user friendly. If we were to redo this project, we would likely have planned more user friendly interfaces, use cases, and scenarios from the very beginning.

Another major change in plans was the recommendation algorithm, both for general and registered users. Originally, we had planned to use an algorithm that factored in the user's height, age, medical condition(s), preferences, and gender. However, while developing the algorithm, we realized that such an algorithm would exceed the scope and goals and our web application. Our application is meant to be used as a simple, convenient guide to healthier eating, but asking a user to enter such private and extensive information to be used in a recommendation would risk turning the application into a source of medical advice. That would greatly exceed the scope of our system because there was no input from a medical professional in the making of the system.

One area that we struggled with was testing because our application was not object oriented. As a result, we were limited to what we could test. Testing mostly consisted of unit testing: as soon as we finished developing a subsystem, it was tested to ensure that the outputs were correct. However, such an approach does not guarantee a bug-free design. Another method of testing was acceptance testing. By the time we remembered to implement acceptance testing, we were nearly finished

with building our application. Eventually, we were able to wrap some of our implementations into testable classes and we were able to do some regression, integration and system testing. In retrospect, we should have taken a test-driven development approach at the start of the project.