Health and Nutrition Web Application

Capstone Project Prepared for

The Degree of Bachelor Science

*Sebastian Pannuto*

Advised by

*Ron Sarner*

May 2019

**Contents**

**1.0 Introduction…………………………………………………………………………………4**

1.1 Objective………………………………………………………………………………4

1.2 Overview………………………………………………………………………………4

**2.0 Design**…………………………………………………………………………………………4

2.1 Design Layout…………………………………………………………………………4

2.2 MySQL Database………………………………………………………………...……5

2.2.1 Construction and Layout……………………………………………….……6

2.3 Connection to Java Web Application…………………………………………………6

2.3.1 Problem connection to the Java Web Application……………………..……7

**3.0 Features**……………………………………………………………………….………………8

3.1 Login System………………………………………………………………………….8

3.2 Food Lookup…………………………………………………………………………..9

3.2.1 Food Lookup Problems and Bugs………………………………………….10

3.3 User Home Page……………………………………………………………………..11

3.4 Meal Plans……………………………………………………………………………12

3.4.1 Creating a Meal Plan……………………………………………………….13

3.4.2 Editing the Meal Plan…………………………………………………...….13

3.4.3 Viewing the Meal Plan…………………………………………………….15

3.4.4 Meal Plan Bugs…………………………………………………………….16

3.5 Goals…………………………………………………………………………………18

**4.0 Deployment on AWS**……………………………………………………………………….19

**1.0 Introduction:**

**1.1 Objective:**

Create a health and nutrition application that allows people to create and track meals as wells as set goals to achieve a desired nutritional food intake. Users will be able to track their meal progress as well as their nutritional intake.

**1.2 Overview:**

Over the semester I worked on and developed a web application that allows users to help them watch their weight and get on the right track to living a healthier lifestyle. The application has four main features: food lookup, weight watching, meal plan set up and tracking, and goal setting. When starting the application, you are brought to a login screen where the user must enter his or her credentials into the system in order to retrieve the user data from the database. Afterwards they are brought to their user home page which acts as a hub for navigating throughout the entire application. From the home page the user can access everything they need such as Food Lookup, Meal Plan viewing and tracking, and goal setting.

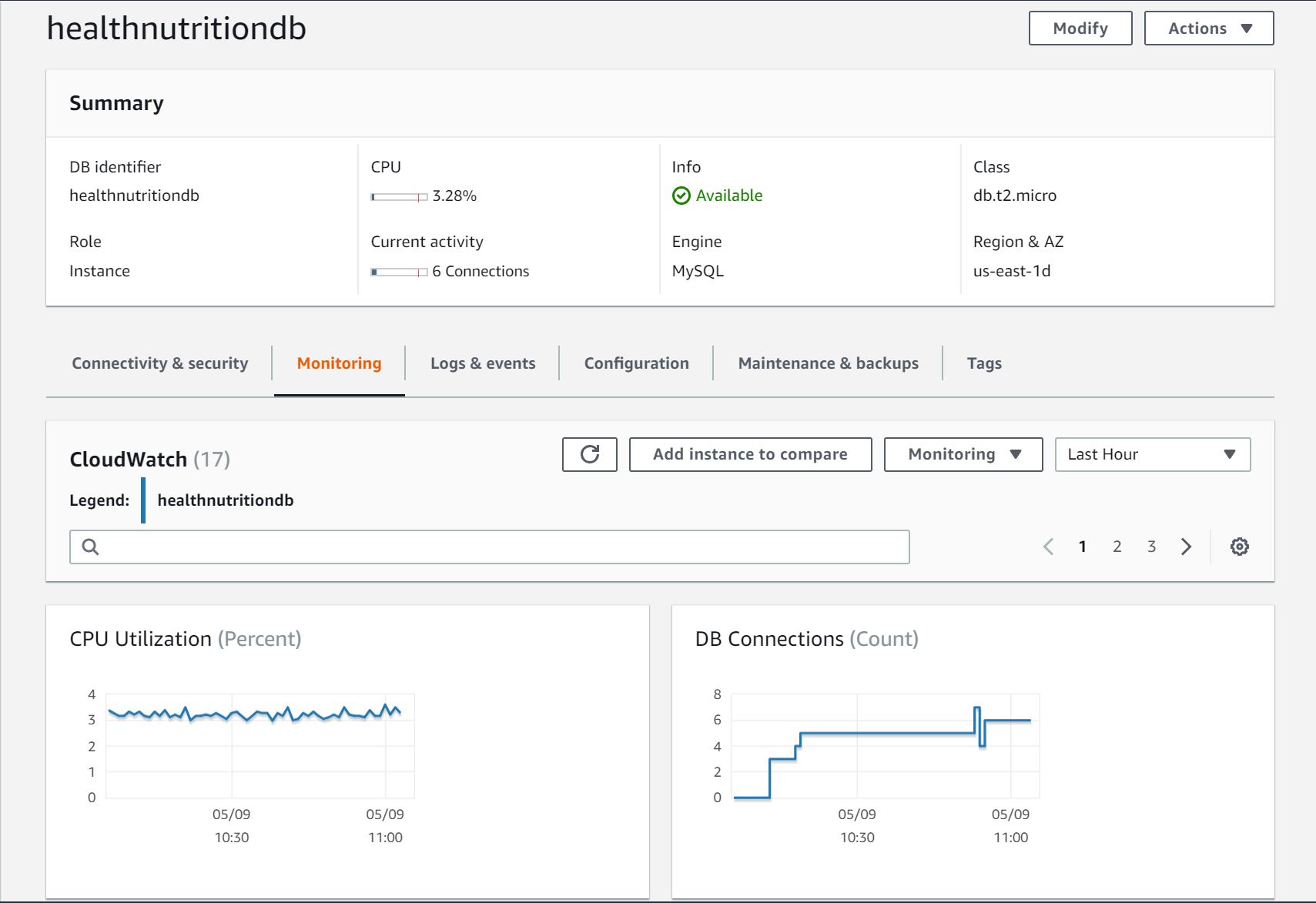
**2.0 Design:**

**2.1 Design Layout:**

The application is designed on many different levels with a lot of moving parts. On the top most layer is the JSF files which are the layout for the user interface. This is the part that the user sees while they use the application. It is also responsible for making calls to the next layer which is the Java aspect. This is the main aspect of the application as it is responsible for making calculations and running the actual program itself. On the next layer is the MySQL database which is responsible for storing data from the application and having it ready to use. The database stores user information as well as all the foods that can be used in the application. And the last layer is the server-side aspects. The SQL server is running on an Amazon Web Services (AWS) Relational Database service (RDS) instance while the actual web application is run on an Apache Tomcat server launched on an AWS Elastic Beanstalk instance.

**2.2 MySQL Database:**

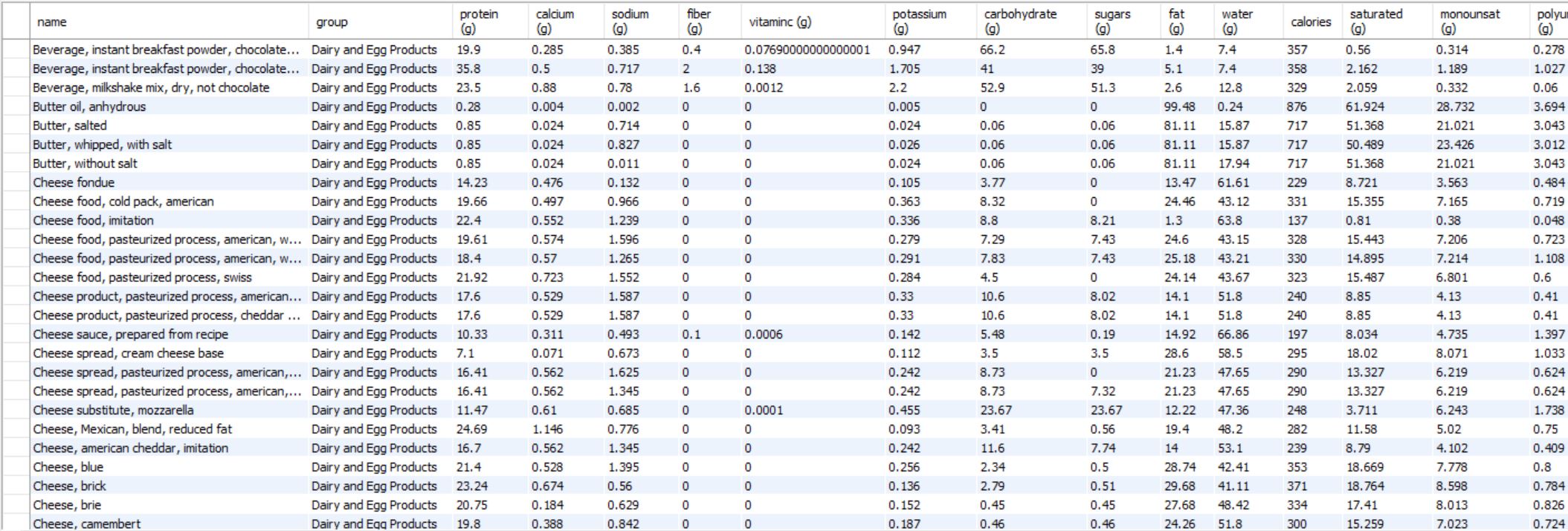
The database is stored on an AWS RDS instance and is stored on my AWS Virtual Private Network. It is run on the US East N. Virginia server. And can be monitored directly from the RDS dashboard in order to track how much data is being used as well as viewing how many connections are being made to the database.



(Figure: AWS RDS Dashboard)

**2.2.1 Construction and Layout:**

The database consists of two tables. One for a table called “Users” for storing information about each user that uses the application and another table called “FoodData” which stores all nutritional information of over 7,637 different foods. The data about the food was used from Nutrient Explorer at <http://bl.ocks.org/syntagmatic/raw/3150059/>. The User table has a column for the user id, username, password, weight, height, meal plan data, meal plan name and a column for every nutritional goal. The FoodData table has a column for the name of the food, food group, protein, calcium, sodium, fiber, vitamin C, potassium, sugars, fats, water, calories, saturated fat, monounsaturated fat, polyunsaturated fat and a unique id.



(Figure: The FoodData table as stored in MySQL. Not seen is the Polyunsaturated Fat column and id column)

**2.3 Connection to Java Web Application:**

The SQL database is connected to the Java application using the JDBC connector which is a library that helps connect the SQL database to Java application as well as be able to run queries to pull information from the database.

**2.3.1 Problems connecting to the Java Web Application:**

Trying to connect the SQL database to the Java Web Application was one of the hardest, and most frustrating tasks I have done to date. When I first started out, I thought that the IDE I was using, Eclipse JEE, had the JBDC connecter already built into. I thought this because Eclipse had AWS support built into it and since my database was stored on AWS, I thought that it would work without having the .jar file in the project file. I quickly realized that the .jar file was not prebuilt into the IDE. After downloading it and placing it into the project folder I did more research on how to properly connect it including coding examples that showed how to access the driver and then establish a proper connection with the database. After doing all that, I began writing code around it there was just one problem; I never tested it to see if it worked. When I ran my first test code I was stumped, I could not understand why the code was not working. So, I ripped everything apart, I checked my database to make sure that it was working properly, I tripled checked the IP address for the database, I went over my code examples to make sure that my code was right. After about 3 days of ripping my hair out trying to find the problem, I stumbled across a video that was showing how to install libraries in IDEs. It talked about when installing an external library to make sure that you set the build path to the destination of the .jar file. I felt so stupid that I could mess up something so basic as to set the build path for a library. After I did that everything worked properly, and I was able to connect my database to the Java Web Application.

**3.0 Features:**

**3.1 Login System:**

The login system’s intent was to come up with a way for the user to be able to retrieve and save their data when using the application. This system is probably the most important for the application as it can recognize whose data is being stored and where to store that data when new data entries are sent in. It works very simply by storing a username and password in the database and when the user enters his credentials the information is checked in the database to make sure that they are correct. If they are correct then, their information that they have saved is automatically loaded into the application for use.



(Figure 1: Login page)

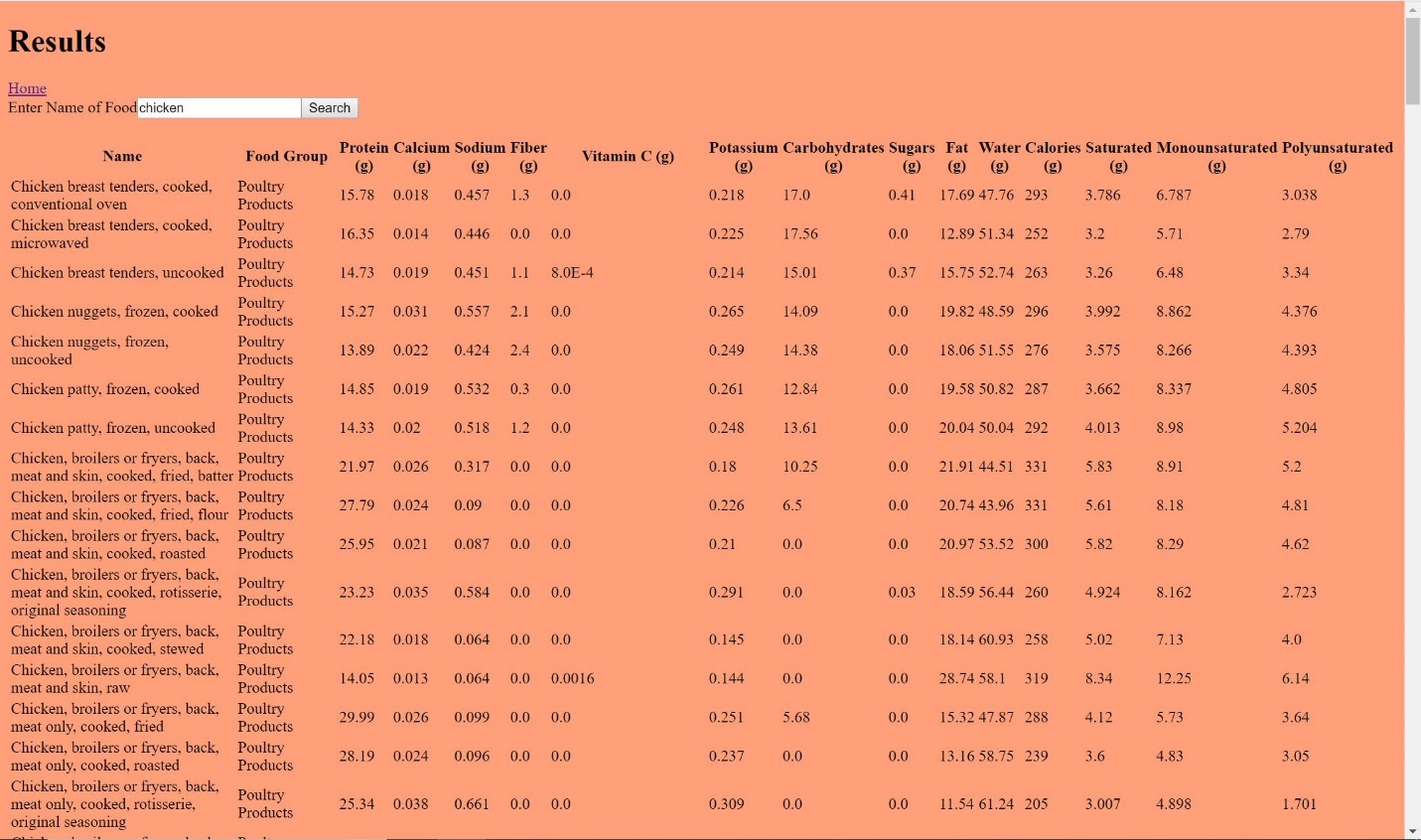
When designing this system, I had some different attempts as to what technique I wanted to use for it. At first, I wondered if there was a way to store an array of “Users” in the SQL database. Where each entry in the array would be a “User” object that had a username, password, and special id for the system to recognize. After doing further research however I concluded that it would be too difficult to store a whole array in a single column and then easily retrieve that information. The system I found to work best was to have a column in the table for username, password, and id and store this information for each User in a row. When the user logs in those fields are pulled from the database and then stored in the program to create a User object for use within the application.

**3.2 Food Lookup**

The food lookup system allows users to search for foods within the database and view their nutritional facts. The system is very straight forward as the user will enter what they want to search for in the search bar. It then checks the database for foods with a similar name to what the user entered and displays the information on the results page.



(Figure 2.1: Search bar for Food Lookup)



(Figure 2.2: Results for executing the search)

**3.2.1 Food Lookup Problems and Bugs**

The was one of the first things I developed as I knew that it would be one of the hardest things to get right as I had very little experience working with an SQL database in a Java application. I encountered loads of different problems when trying to perfect the search. At first, I was thinking I would have to run a JavaScript script in the .xhtml document on the results page in order to be able to print each result line. I gave up on this idea quickly as I learned that JSF had a datatable function allows information to be displayed neatly in a table. Setting up the table was quite easy once I learned how this function worked.

The hard part was having the information somewhat organized in the datatable. I tried to make the table look nice by having lines and different colors so the user could easily distinguish what they were looking at. I learned pretty quickly that there was a field to import a .css file that would act as the way to make the table look however you wanted. However, I encountered quite a big problem here that I still do not really know why it does not work. For some reason the datatable would not pull information from the .css file but it would recognize that the file existed and said that it was pulling the correct fields from it. This obviously was not the case as when the table would be printed it did not have any of the styles in which I verified. At first I thought it was because of the location in which the file was stored but it was stored in the location as the web page so it shouldn’t have any problem retrieving the file. I made sure that the path was correct and still to no avail was it pulling the information from the .css file. And lastly I made sure that the format was correct in the .css file and after studying it for many hours I found no problems with the way that the file was formatted. At this point I gave up and could not logically find a reason as to why the table was not being formatted according the specifications of the .css file. So unfortunately, I had to leave the table the way it was.

**3.3 User Home Page**

When a user signs into the system upon entry it brings that user to their user home page. This acts as a central hub where the user can access every feature that there is on the application. On the top the user can track their weight and height by entering those in those values to then produce their body mass index (BMI). Below that is the portal for accessing all of the features in the application such as the Food Lookup, Meal Plan viewer, a field to create a meal plan, and a place where you can set all of your nutritional goals.



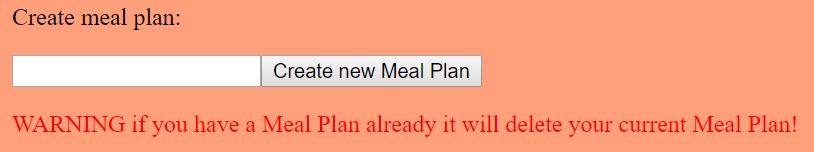
(Figure 3: User home page)

**3.4 Meal Plans**

The Meal Plan system was a system I developed that allows users to save certain foods that they searched for and store them in their meal plan. Using part of the food lookup I added an extra column that could be checked to determine which food the user wanted to add to their meal plan. On the top it shows the foods that they have added already to their meal plan while the bottom half of the page shows their search results from the users most recent search. There are three main parts to way that this meal plan system works: Creating a meal plan, editing a meal plan, and viewing the meal plan.

**3.4.1 Creating a Meal Plan**

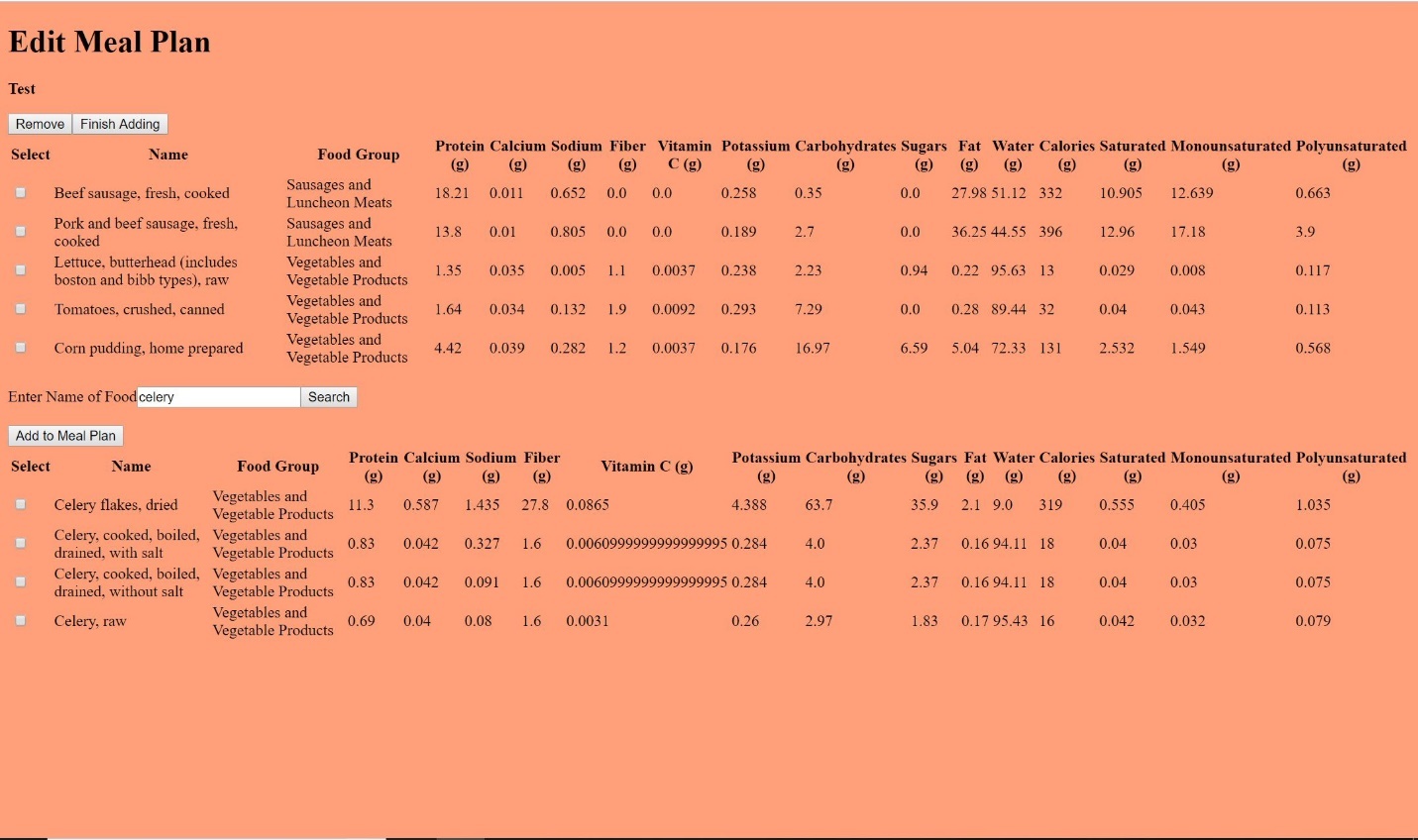
A user can create a meal plan by navigating to their user home page, and then finding the box that says “Create new Meal Plan”. The user must then enter the name of their new meal plan before continuing to add food items to their plan. On the backend once the user clicks the button to create a meal plan it creates a MealPlan object which contains the current user, the name of that meal plan and an ArrayList of food items that they choose to add to it. It then sets a flag to let the Controller class know that the system is going to be creating a meal plan and that no other functions can be called except for the ones corresponding to create that plan. It then moves on to the next phase which is actually editing the meal plan.



(Figure 4.1: Creating a meal plan)

**3.4.2 Editing the Meal Plan**

This part of the meal plan creation uses the same page and same functions for when one creates a meal plan for the first time and when the user wants to edit a current meal plan. Once the user has chosen to edit or create their meal plan it brings them to editing page. From their it will display on the current foods on the top half of the web page as well as two buttons above it. One for removing items that have been added to the meal plan and another button to finish adding foods to the meal plan. On the bottom half of the page it displays the search bar and search results of foods. Right below the search bar is the button for confirming to add foods to the plan.

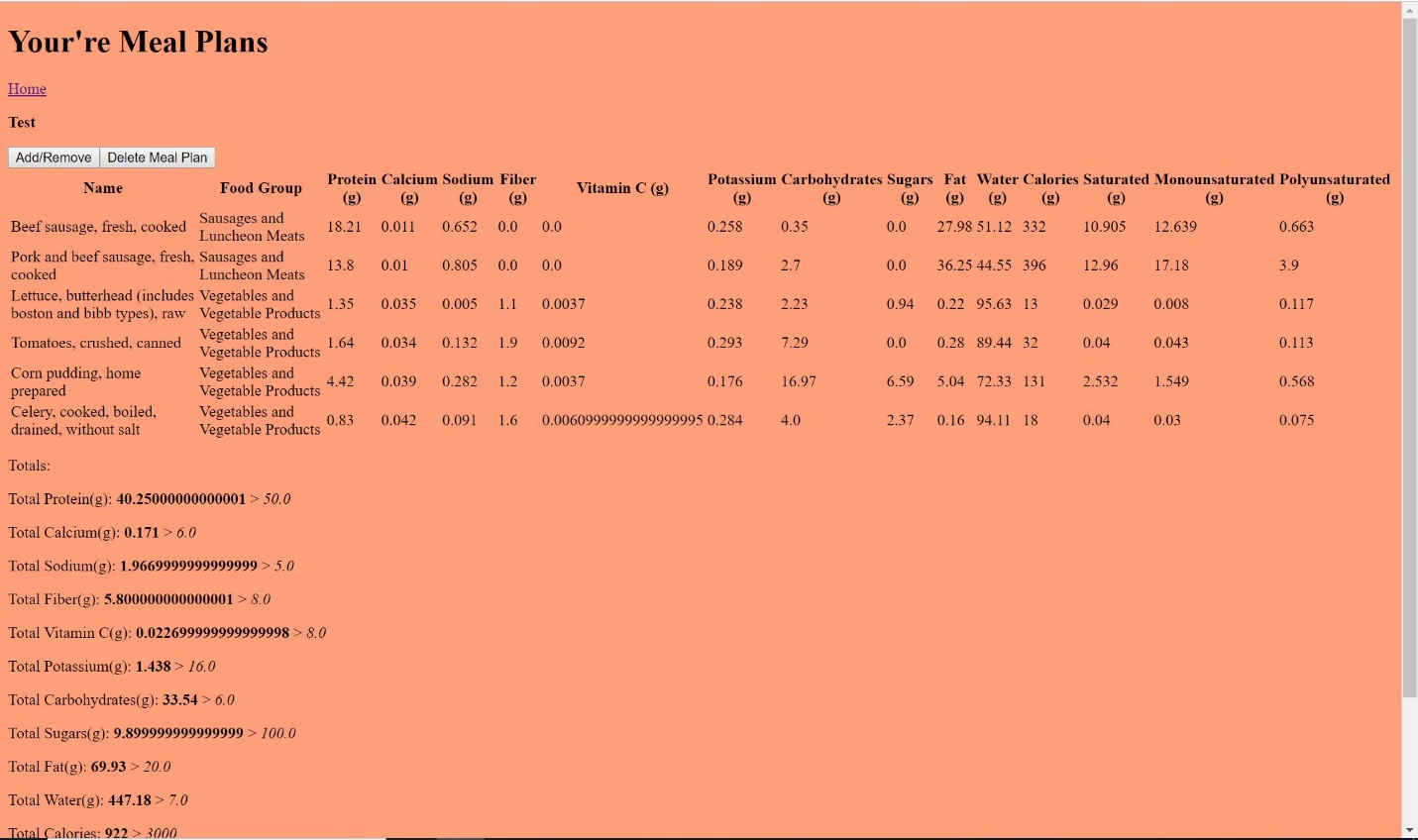


(Figure 4.2: Editing Meal Plan)

The system for this feature was quite intricate and probably one of the most complex features to design in the application. The way it works is that the user searches for foods and then checks of the ones that he wants and then adding them to his meal plan before searching for a new food item to add. The way that the checking system works is by using an Integer Hash Map. Each food item in the database has an id number that is set in ascending order. When the list of foods is brought up from the search results it adds the id from the food into the Hash Map as well as a Boolean value for whether the box to add it has been checked or not. When the user clicks on the “Add to Meal Plan” button it goes through the Map and finds the values that are set to true. When they are found it pulls the id and matches it up to the remaining items from the original search result list in order to pull the rest of the information about that food. The selected foods are added to a temporary list until the editing process is complete. Once the user is finished adding foods to his/her plan, they click the finish adding button which finalizes the meal plan. The first part of that includes taking the foods from the temporary list and transferring them to a finalized list that is stored in the MealPlan object which is stored finally in the User object. After adding the MealPlan object to the User object it takes all the foods from the User’s meal plan and converts them all into a single string. This string is then sent to the database in order to be stored for future use. When the user logs back into the application later, the string is retrieved from the database and converted into each food item ready to be used again in the application.

**3.4.3 Viewing the Meal Plan**

The last part is viewing the Meal Plan that the user has just created. This is very basic as it just retrieves the list of foods from the current user object and displays its contents on the web page. Below the listed food items are the totals for each nutritional category (protein, calories, fat, etc.) and to the right of it are the user’s goals that they have set for each category. I had originally planned for there to be an indicator when the total goes over the projected goal it is listed in red to signify that the user is exceeding their goal. However, I could not find a logical way to implement this and decided on just displaying the numbers. Above the list of foods are two buttons: one for editing the current meal plan, and the other for deleting the entire meal plan.



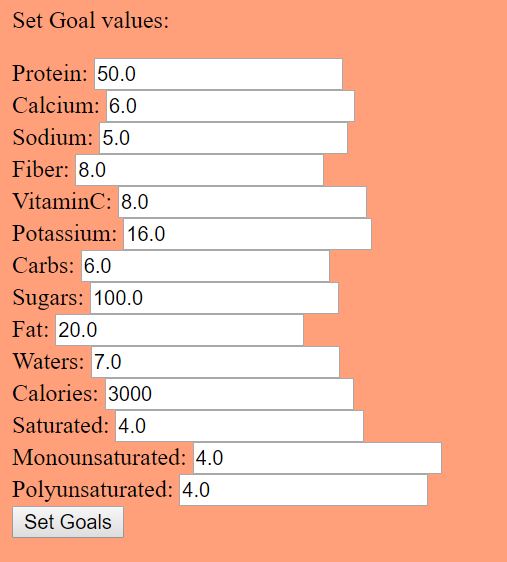
(Figure 4.3: Meal Plan viewing page)

**3.4.4 Meal Plan Bugs**

This feature was very difficult to design as it was very hard to be able to map the checkboxes to the actual food item since I could not get it work with my actual Food Object class. While designing this I ran into a quite burdensome bug that made me have to renter my entire FoodData database. Originally, when I imported the data about the food into the database, I had to do it in two batches because I lost connection about half way through upload. This caused me to go in the database and try and find what the last thing uploaded was so I could upload the rest of it. The source from which I was importing the data from had a predetermined id so when I uploaded the second batch it messed up the numerical order of the ids of each row. Now I didn’t think this would be a problem but when designing the adding of foods to the meal plan, I discovered in the search results that the ids of the foods were mixed up for certain foods. Now because of the way I programmed it, it relied on the foods being in numerical order and if it wasn’t it would not add the food to the meal plan. This was a very frustrating bug which made me choose between one of two choices: either rewrite the code so it doesn’t rely on the ids being in numerical order or reimport the data into the database in one batch so that the ids are in the right order. Ultimately, I chose to just reimport the data as it was much easier than rewriting part of a code that took me quite a while to get right.

**3.5 Goals**

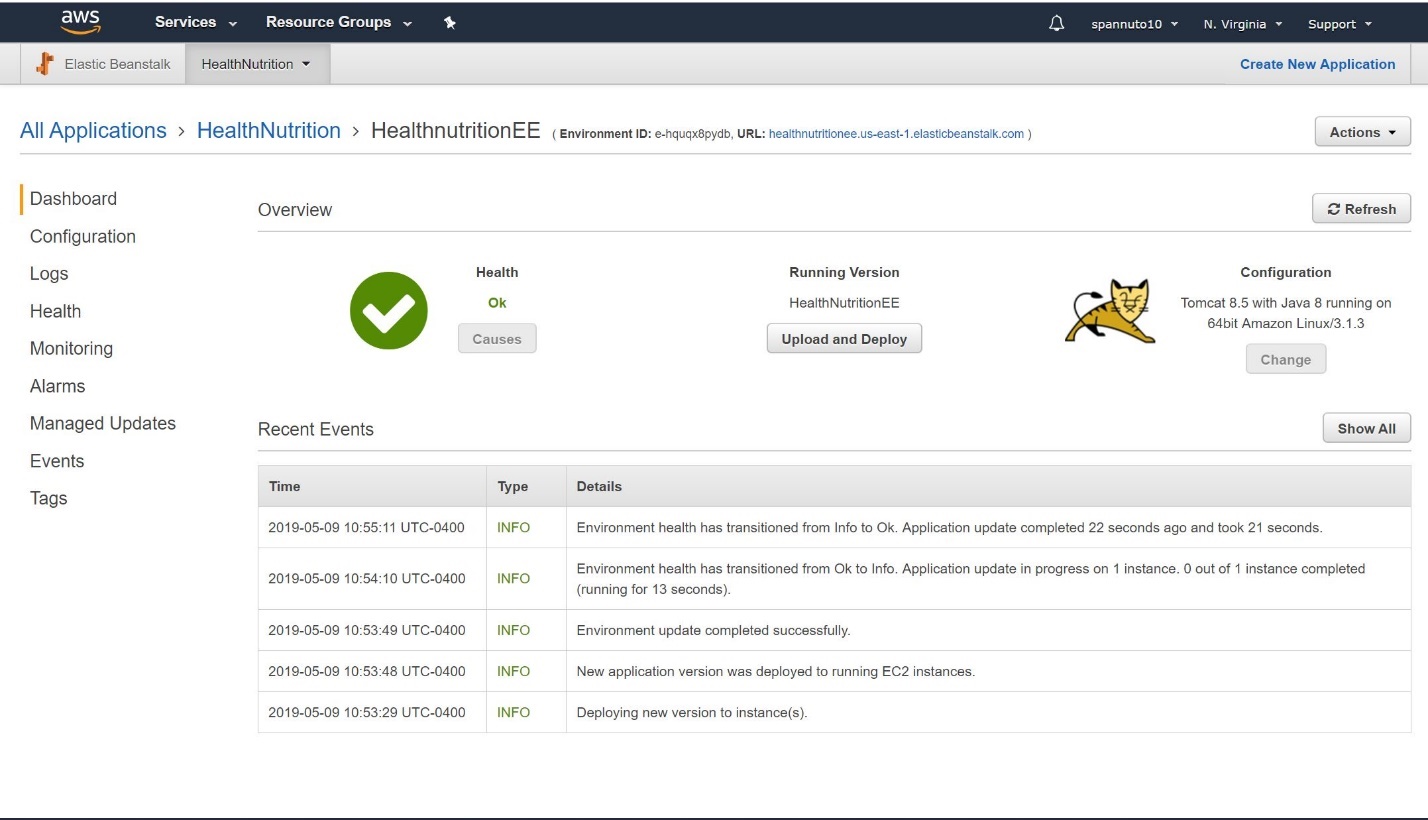
The last main feature is the ability to set goals. On the usage side this the most crucial part of the programming giving the user a purpose for using the application. This feature allows users to set certain goals on nutritional values that they want to achieve. There’s a goal for every nutritional fact on the application this includes protein, calcium, sodium, fiber, vitamin C, potassium, carbohydrates, sugars, fats, waters, calories, saturated fat, monounsaturated fat, and polyunsaturated fat. Each user can set these goals at any time and all the goals are stored in the database so that they are saved. Users can view their goals either on their user home page or on the meal plan view page. There it shows if their total intake is greater than what they would have like to take in.



(Figure 5: Setting goal values)

**4.0 Deployment on AWS**

The last step of my project was to deploy the application on an AWS Elastic Beanstalk instance so that it could be accessed by people other than myself. The application itself was running on an Apache Tomcat server, so I created an environment in Elastic Beanstalk to so that it could handle that server. This was not that difficult as all it involved me doing was setting up the environment, and then uploading all of the files to an Amazon S3 Bucket so that it could access all of them. And now the application can be accessed at: <http://healthnutritionee.us-east-1.elasticbeanstalk.com/index.xhtml>



(Figure: Deployment in AWS Elastic Beanstalk, dashboard)