UNIVERSITY OF LONDON

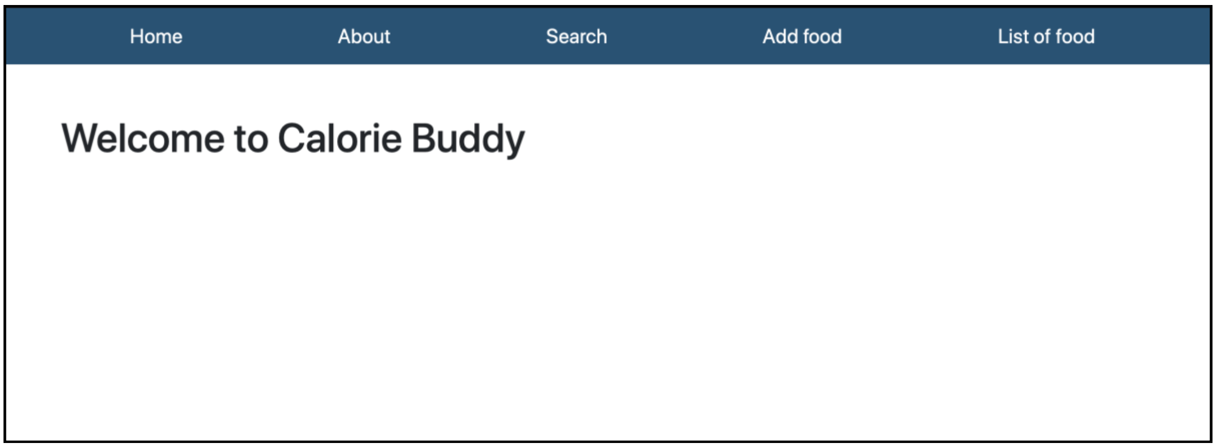
CM2040 DATABASES, NETWORKS AND THE WEB

Student Name:	Lawrence Ho Sheng Jin
Date Submitted:	25th January 2020
Degree Title:	Computer Science
Local Institution:	Singapore Institute of Management
Student ID:	10205927

Table of Contents

R1 – Home Page

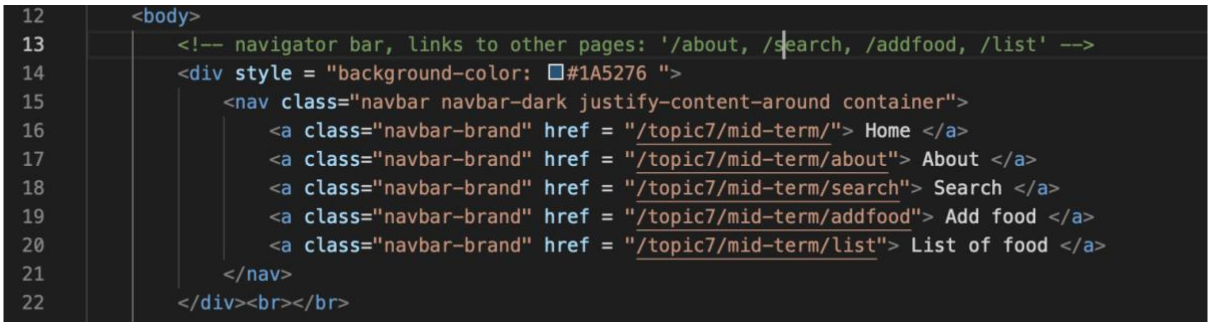
The rendered page view for home page resides in **'index.html'**, where there is simply a title and header indicating the home page of the site. There is a navigation bar on top of the page, will contains #href links to other pages. Other than that, the home page presents just a text body indicating 'Welcome to Calorie Buddy'.



Welcome to Calorie Buddy

Navigation bar:
The navigation bar contains links to other pages, namely '/about, /search, /addfood, and /list'. I have included bootstrap css styles to make the page look more presentable. **There will be a navigation bar present in every other page view.**

The source code for the navigation bar is identical in every single page:



```
    <body>
        <!-- navigator bar, links to other pages: '/about, /search, /addfood, /list' -->
        <div style = "background-color: #1A5276 ">
            <nav class="navbar navbar-dark justify-content-around container">
                <a class="navbar-brand" href = "/topic7/mid-term/"> Home </a>
                <a class="navbar-brand" href = "/topic7/mid-term/about"> About </a>
                <a class="navbar-brand" href = "/topic7/mid-term/search"> Search </a>
                <a class="navbar-brand" href = "/topic7/mid-term/addfood"> Add food </a>
                <a class="navbar-brand" href = "/topic7/mid-term/list"> List of food </a>
            </nav>
        </div><br></br>
```

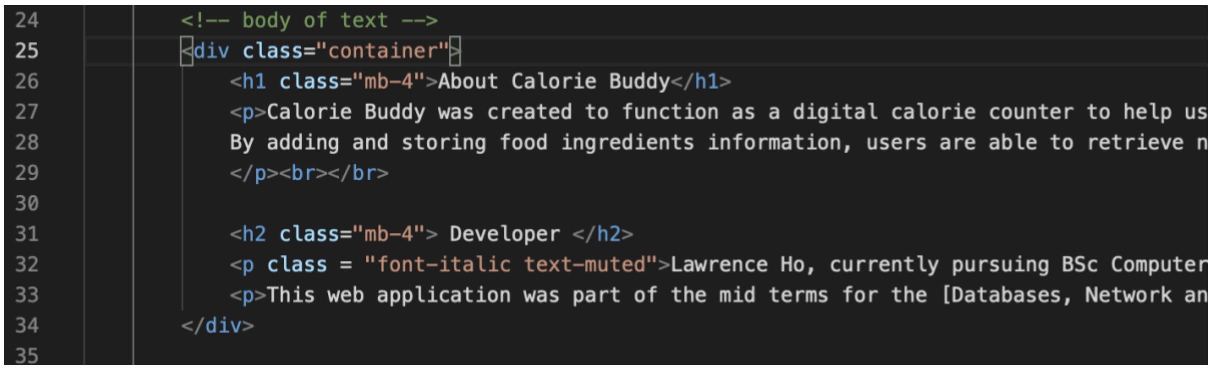
The middleware routing in **'main.js'** are as follows (lines 104-107 of 'main.js'):

```
// DESC: Render home page
    app.get("/",function(req, res){
        res.render("index.html")
    });
```

DESC – Description
For the rest of the report, I would be using this abbreviations to explain my code in comments.

R2 – About Page

The rendered page view is located in **'about.html'** and the body of the page simply contains a short description of the myself as the developer using header and paragraph tags.



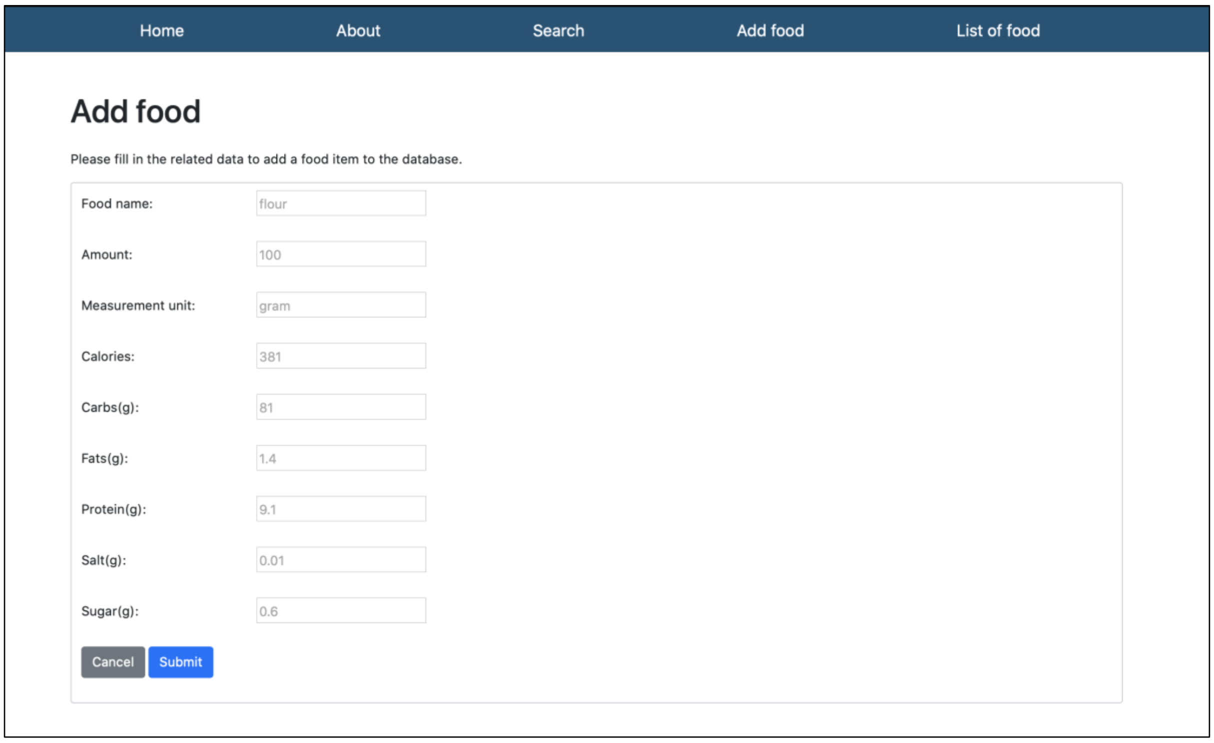
```
<!-- body of text -->
<div class="container">
    <h1 class="mb-4">About Calorie Buddy</h1>
    <p>Calorie Buddy was created to function as a digital calorie counter to help us
    By adding and storing food ingredients information, users are able to retrieve n
    </p><br></br>

    <h2 class="mb-4"> Developer </h2>
    <p class = "font-italic text-muted">Lawrence Ho, currently pursuing BSc Computer
    <p>This web application was part of the mid terms for the [Databases, Network an
</div>
```

R3 – Add Food Page

The rendered page view will be in the **‘addfood.html’**. This page displays a form for users to enter the necessary data fields to add a new food item to the database.

The page looks like this:



Add food

Please fill in the related data to add a food item to the database.

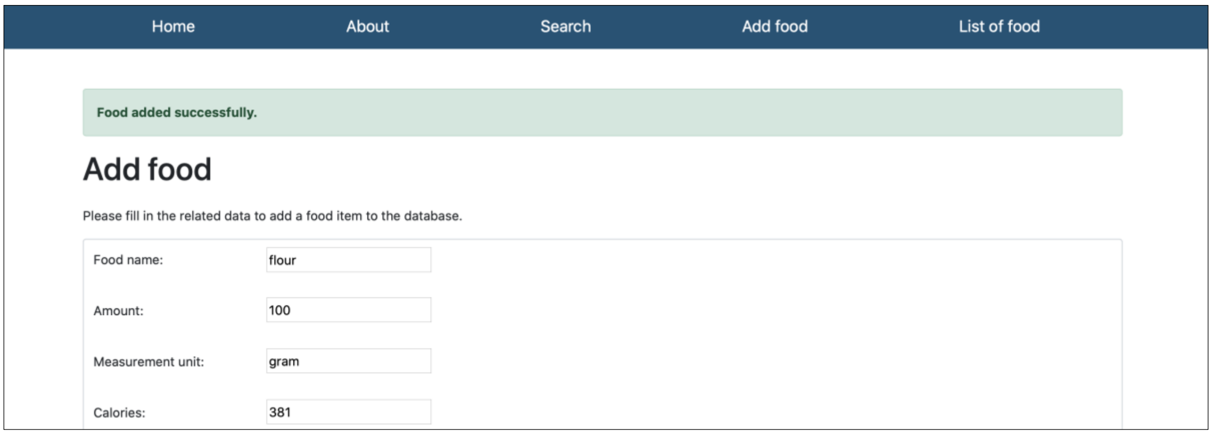
Food name:	flour
Amount:	100
Measurement unit:	gram
Calories:	381
Carbs(g):	81
Fats(g):	1.4
Protein(g):	9.1
Salt(g):	0.01
Sugar(g):	0.6

The fields in html all contain the ‘required’ attribute, so the user are forced to enter a value in every text field before being able to add a new food item. The text fields contain placeholder values to indicate the types of value that will be stored.
An example of the html code for one of the fields are as follows (lines 50-51 of ‘index.html’) :

```
<label for="name" class="col-sm-2 col-form-label">Food name:</label>
<input type="text" name="name" placeholder="flour" required> <br></br>
```

At the end of the page, there are two buttons: cancel and submit. The cancel button is a link which brings the user back to the home page, whereby the submit button sends a POST request to the middleware, which then adds the food to the database. Upon successful addition, the page refreshes itself with a successful message at the top, whereas if there is an error, for example, invalid data types being entered (string into decimal fields), an error message will be displayed at the top, telling the user to enter the valid data types as indicated by the placeholder.

The following pictures depict the scenarios just mentioned:



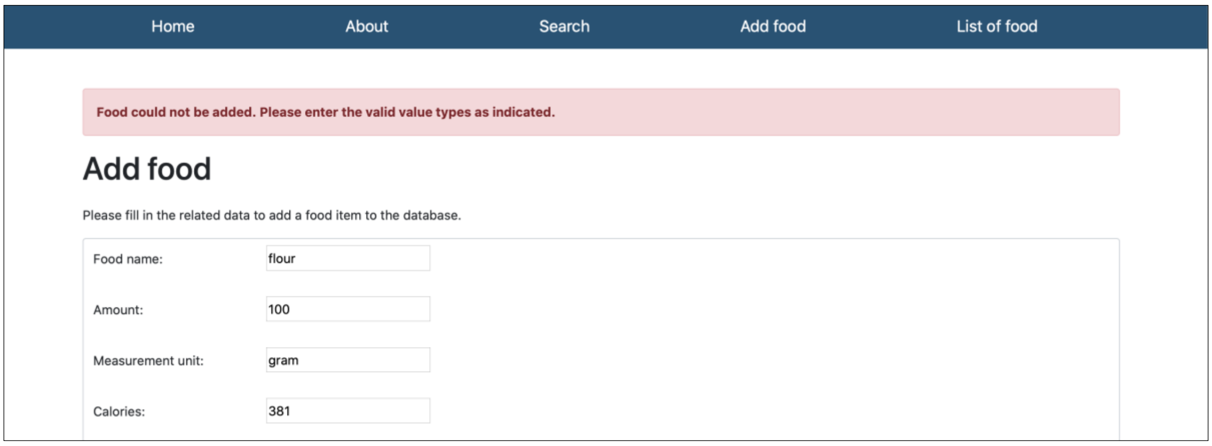
Food added successfully.

Add food

Please fill in the related data to add a food item to the database.

Food name:	flour
Amount:	100
Measurement unit:	gram
Calories:	381

^Successful entry



Food could not be added. Please enter the valid value types as indicated.

Add food

Please fill in the related data to add a food item to the database.

Food name:	flour
Amount:	100
Measurement unit:	gram
Calories:	381

^Invalid entry

In the middleware, POST request handler can be found in line **26** of '**main.js**'.

```
26      app.post("/foodadded", function (req,res) {
```

Upon receiving the POST request from the submit button, the values stored in the req.body will be the text input values entered by the user. These values will be passed into the db.query function to add these values into a new food item. If the query is successful, it will render the 'addfood.html' page again (refreshing it), allowing users to add more food items. At the same time, it will send a JSON object containing a single key and value as shown below:

```
db.query(sqlquery, newrecord, (err, result) => {
    if (err)
      res.render('addfood.html', {success: false});
    else
      res.render('addfood.html', {success: true});
});
```

This value is to determine whether the operation is successful or not by a script inside of 'addfood.html'. If the success variable contains true, it will then print the successful message at the top of the page, and vice versa. Otherwise, by default, this success value will be = null. The script that decides whether to display the success/error message is located in lines 27-39 of 'addfood.html':

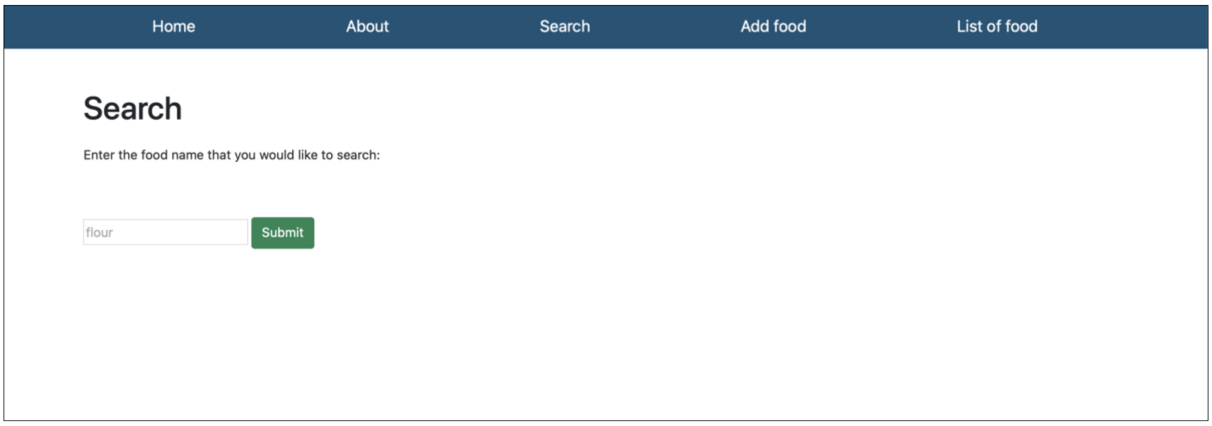
```
<!-- prints out message if food has been added successfully -->
      <% if(success == true){ %>
        <div class="alert alert-success">
          <strong>Food added successfully.</strong>
        </div>
      <% } %>

      <!-- prints out message if food cannot be added -->
      <% if(success == false){ %>
        <div class="alert alert-danger">
          <strong>Food could not be added. Please enter the valid value types as indicated.</strong>
        </div>
      <% } %>
```

R4 – Search Food Page

The rendered page view will be in **'search.html'**. This page displays a textbox form for users to enter a keyword to search for any food items in the database that contains the substring keyword.

The page looks like this:



If the user submits a valid keyword such that it is available in the database, it will send a GET request to the middleware, which will then query for all food items with a substring of the entered keyword, and render the list of found food items using 'list.html', displaying all of the available food items found. From there, the user is able to proceed with whatever functions that are available in 'list.html', which will be covered later on in **R6**.

The database is able to search for items based on the substring keyword due to the '%' syntax. By surrounding the keyword entered by the user with '%', it enables the db to query for similar words.

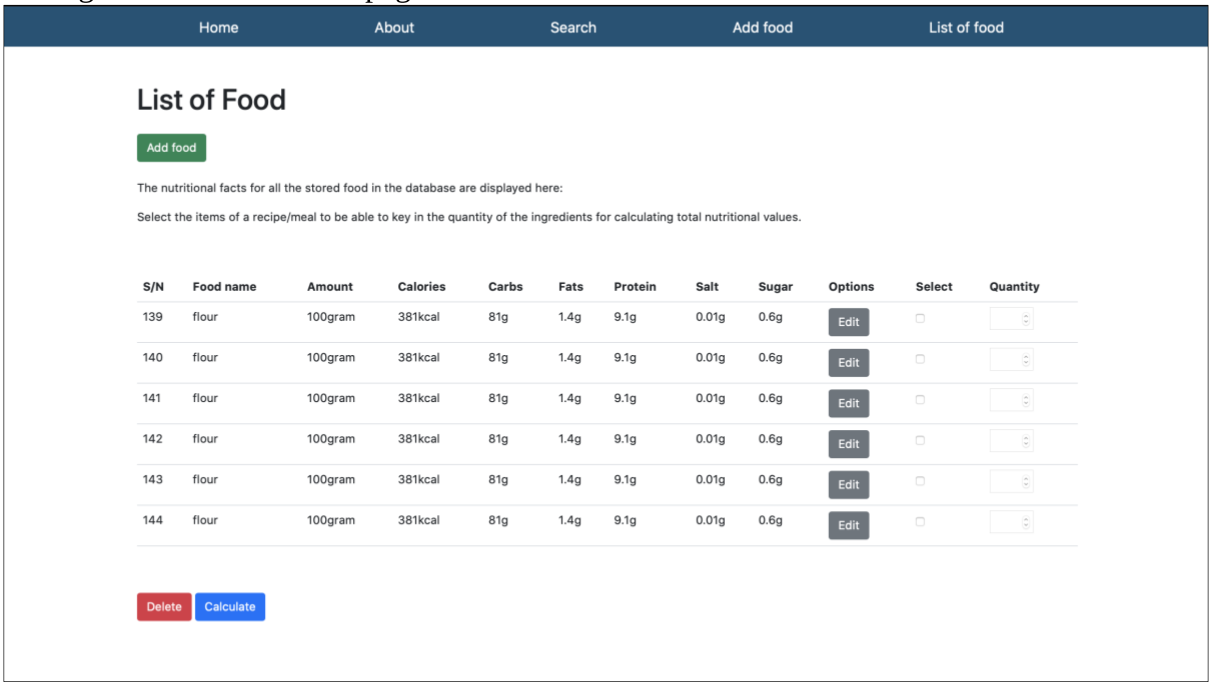
```
let searchInput = ['%' + req.query.name + '%'];
```

However, if the database is unable to return any items found, there is a local script in 'list.html' that checks if there are any results (food objects) returned by the db, and if not, it will display an alert message, telling the user that there are no food items found with that keyword. The middleware response will still render the page 'list.html', but there will be an empty table shown.

The local html script, that determines whether to show the error message can be found in lines 24-28 of 'list.html':

```
<% if(availableFood == ''){ %>
        <script>
            alert("No food found.")
        </script>
    <% } %>
```

The figures below show the page views of a successful and failed search:



List of Food

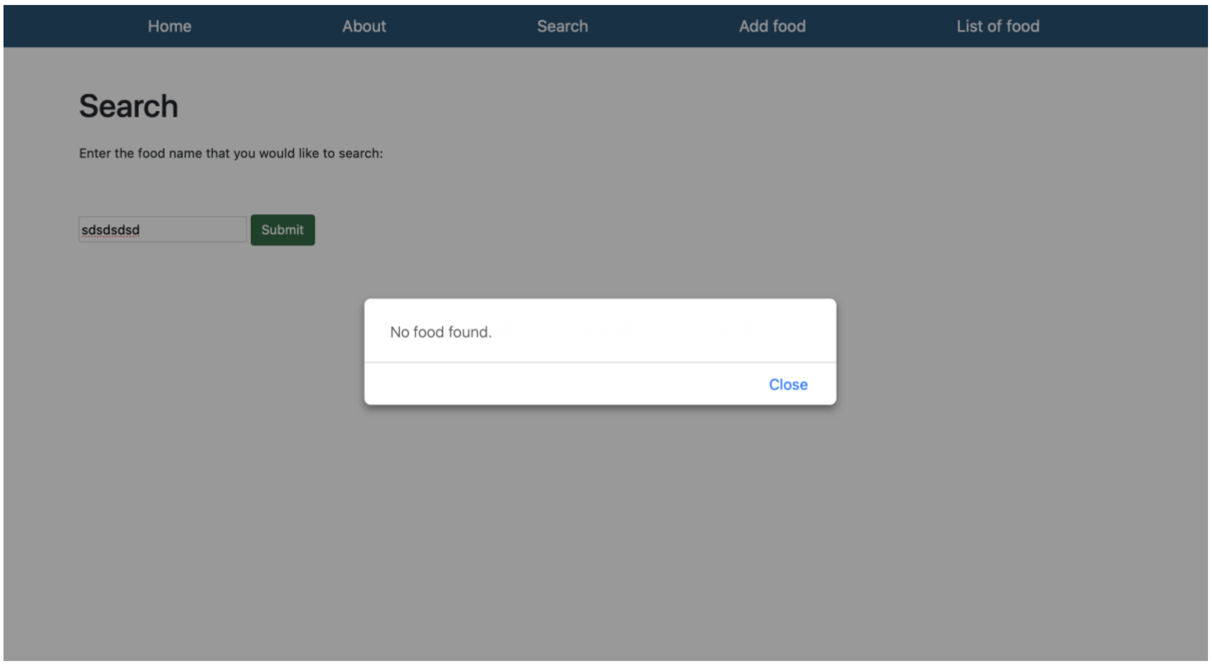
Add food

The nutritional facts for all the stored food in the database are displayed here:

Select the items of a recipe/meal to be able to key in the quantity of the ingredients for calculating total nutritional values.

S/N	Food name	Amount	Calories	Carbs	Fats	Protein	Salt	Sugar	Options	Select	Quantity
139	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit		
140	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit		
141	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit		
142	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit		
143	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit		
144	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit		

^Successful search



Search

Enter the food name that you would like to search:

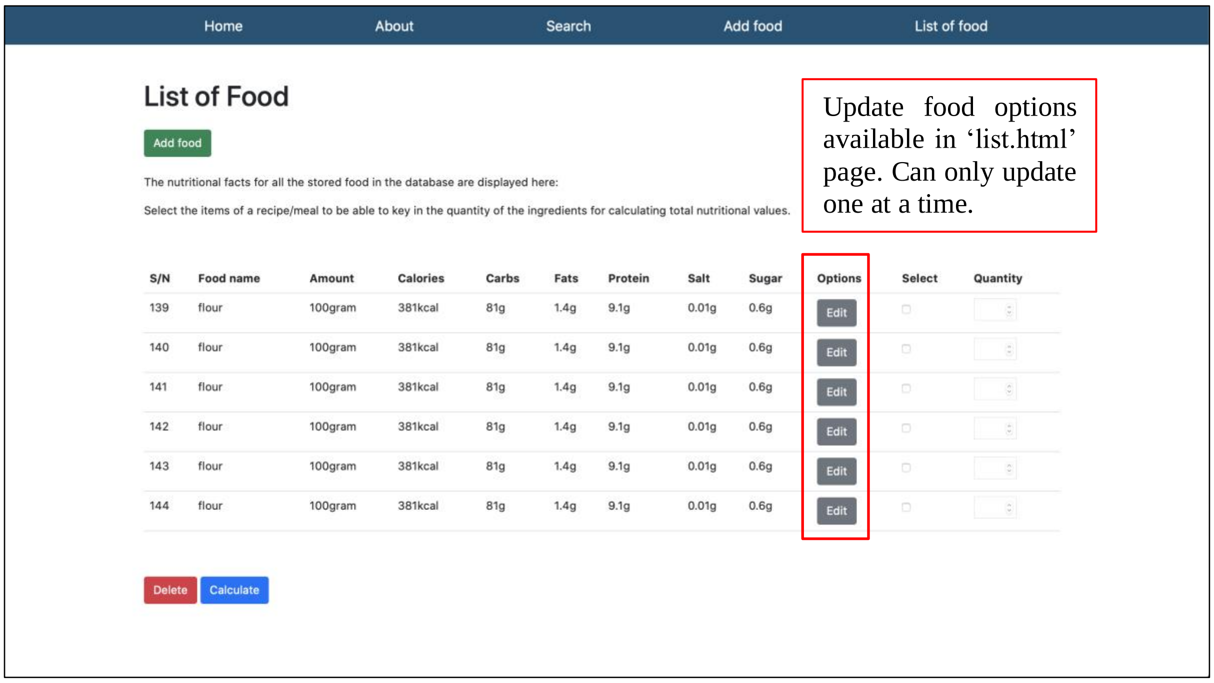
sdsdsdsd

No food found.

^Unsuccessful search

R5 – Update Food Page

The way I have approached adding the /updatefood function is through the table of food items rendered in 'list.html'. From the list of food, there is an edit button for every single row of food showed in the table. The user is then able to click on the button, which will then direct to the /updatefood page with textboxes for every single fields of the food. The current existing field values of the food will be entered into the textbox fields, which will allow for easier and selective editing by the user. Similarly, the user can also choose to update an item from search → list → updatefood.



List of Food

The nutritional facts for all the stored food in the database are displayed here:

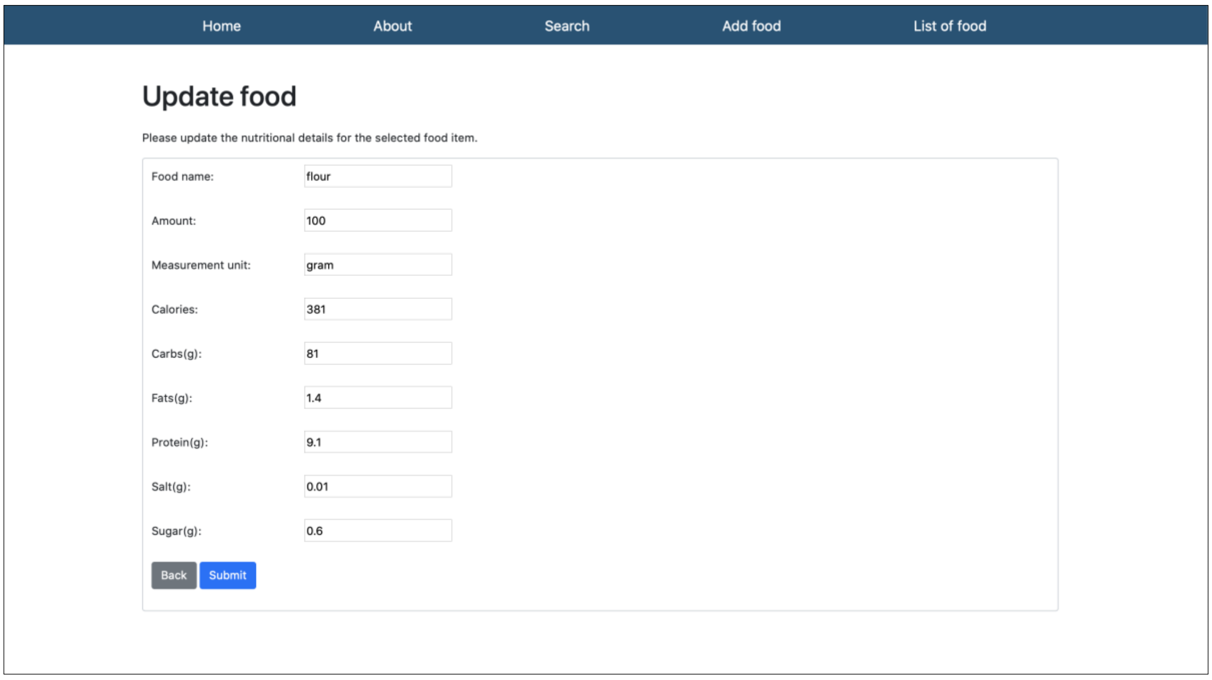
Select the items of a recipe/meal to be able to key in the quantity of the ingredients for calculating total nutritional values.

Update food options available in 'list.html' page. Can only update one at a time.

S/N	Food name	Amount	Calories	Carbs	Fats	Protein	Salt	Sugar	Options	Select	Quantity
139	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit	☐	
140	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit	☐	
141	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit	☐	
142	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit	☐	
143	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit	☐	
144	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit	☐	

The rendered page view will be in **'updatefood.html'**. This page is almost identical to that of 'addfood.html', with the only difference being in the header text and values present in the textboxes. There is also a cancel and submit button at the bottom.

The page looks like this:



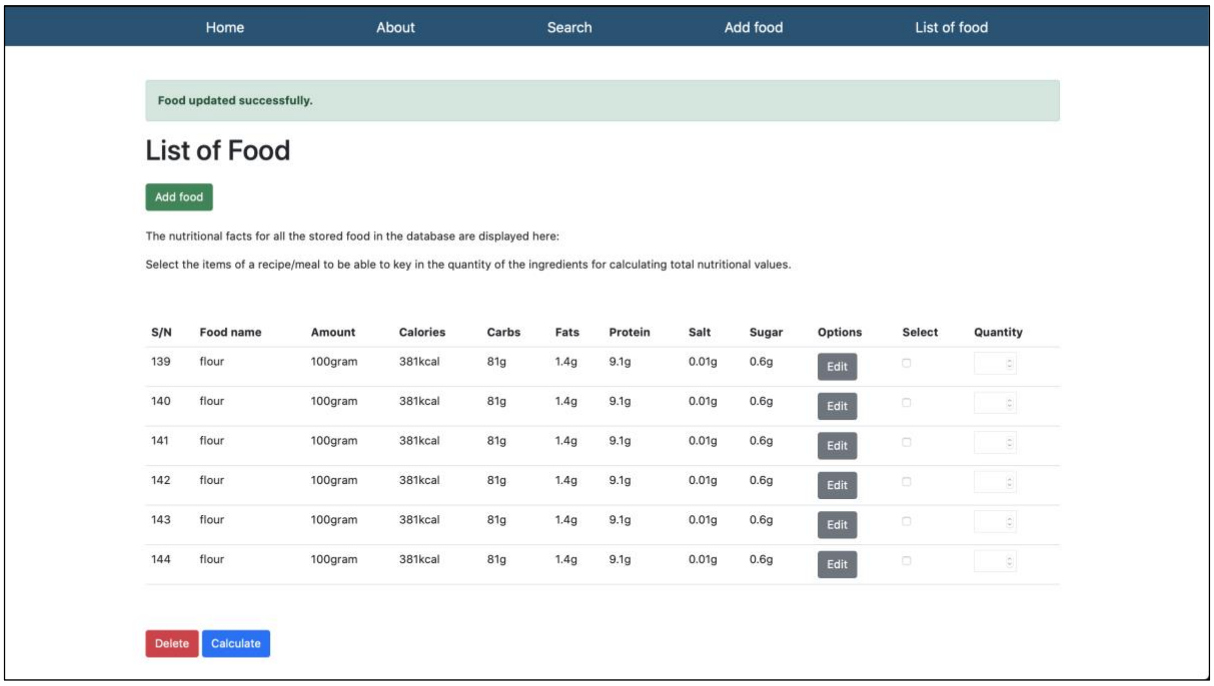
Update food

Please update the nutritional details for the selected food item.

Food name:	flour
Amount:	100
Measurement unit:	gram
Calories:	381
Carbs(g):	81
Fats(g):	1.4
Protein(g):	9.1
Salt(g):	0.01
Sugar(g):	0.6

Upon submission of the form data, a POST request will be submitted to the middleware, where the necessary values and fields will be updated in the database. If the update in the database is successful, the page will be redirected to the list page with a message at the top indicating that the update has been done successfully. However, if there is an error, for example, invalid data types being entered (string into decimal fields), an error message will be displayed at the top, telling the user to enter the valid data types as indicated by the placeholder.

The figures below show the page views of a successful and failed search:



Food updated successfully.

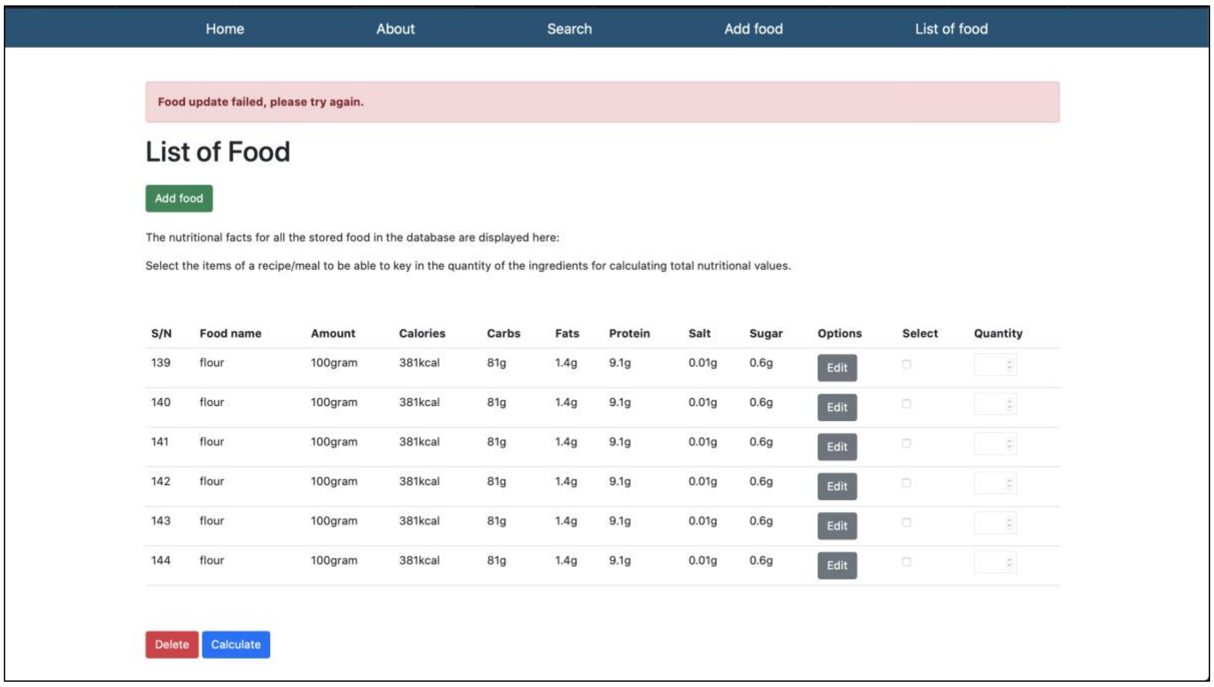
List of Food

The nutritional facts for all the stored food in the database are displayed here:

Select the items of a recipe/meal to be able to key in the quantity of the ingredients for calculating total nutritional values.

S/N	Food name	Amount	Calories	Carbs	Fats	Protein	Salt	Sugar	Options	Select	Quantity
139	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit	☐	
140	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit	☐	
141	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit	☐	
142	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit	☐	
143	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit	☐	
144	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit	☐	

^Successful update



Food update failed, please try again.

List of Food

The nutritional facts for all the stored food in the database are displayed here:

Select the items of a recipe/meal to be able to key in the quantity of the ingredients for calculating total nutritional values.

S/N	Food name	Amount	Calories	Carbs	Fats	Protein	Salt	Sugar	Options	Select	Quantity
139	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit	☐	
140	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit	☐	
141	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit	☐	
142	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit	☐	
143	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit	☐	
144	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit	☐	

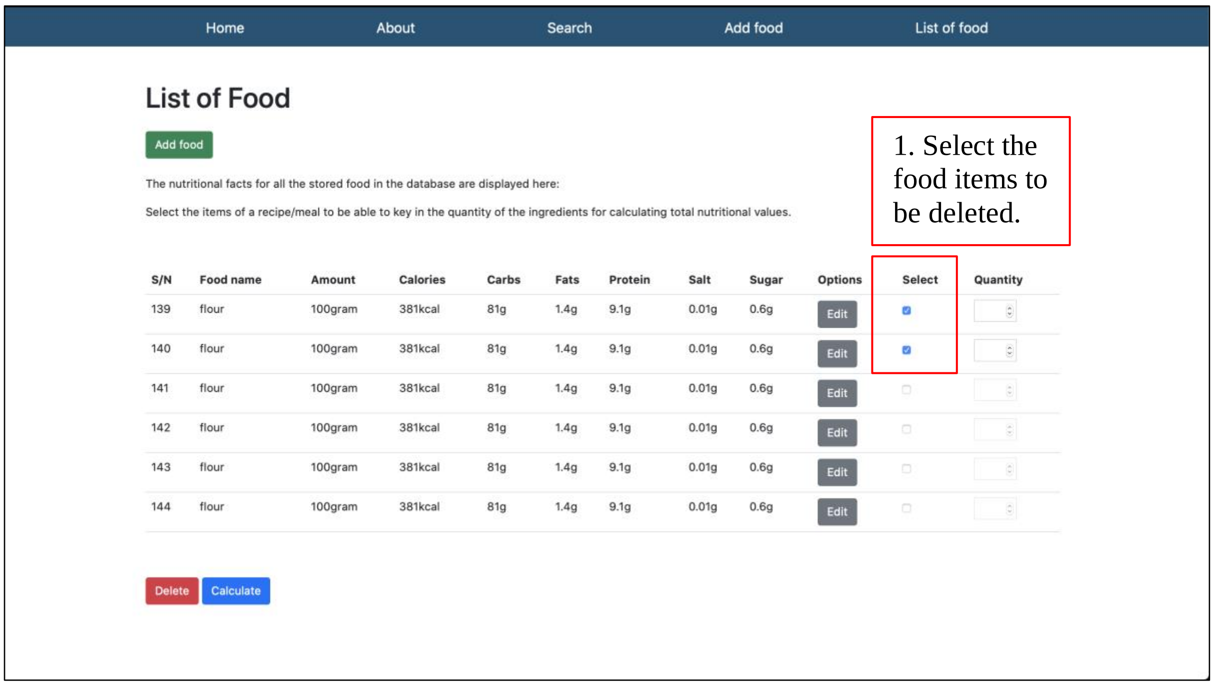
^Unsuccessful update

Notice that are two buttons at the bottom of the page: delete and calculate. I will cover the delete button here, whilst calculate will be explained in R6 later on.

To delete items from the database and from the table, the user will have to select the preferred items to be deleted using the checkboxes present in every row of food item. After selecting the food items, upon clicking the delete button, a confirmation message will pop up, ensuring that the user would like to delete the selected items.

Upon confirmation by the user, it will submit the <%= food.id %> data that is stored in these checkboxes to the POST request to the middleware. The middleware then handles this request by deleting the selected food items in the database based on the food.id in req.body. After querying in the database, the response given back will be to go back to the list.html page, this time with the updated food items shown (less the deleted ones).

The figures below show the process of deleting:



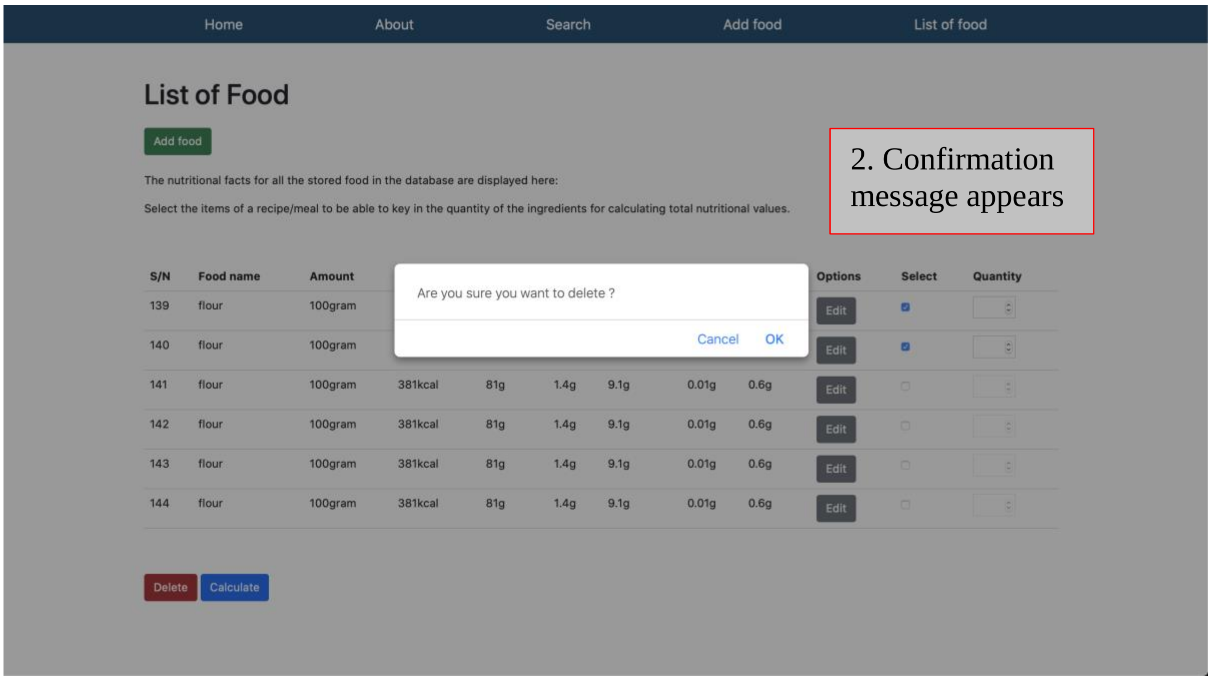
List of Food

The nutritional facts for all the stored food in the database are displayed here:

Select the items of a recipe/meal to be able to key in the quantity of the ingredients for calculating total nutritional values.

1. Select the food items to be deleted.

S/N	Food name	Amount	Calories	Carbs	Fats	Protein	Salt	Sugar	Options	Select	Quantity
139	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit	✓	
140	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit	✓	
141	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit		
142	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit		
143	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit		
144	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit		



List of Food

The nutritional facts for all the stored food in the database are displayed here:

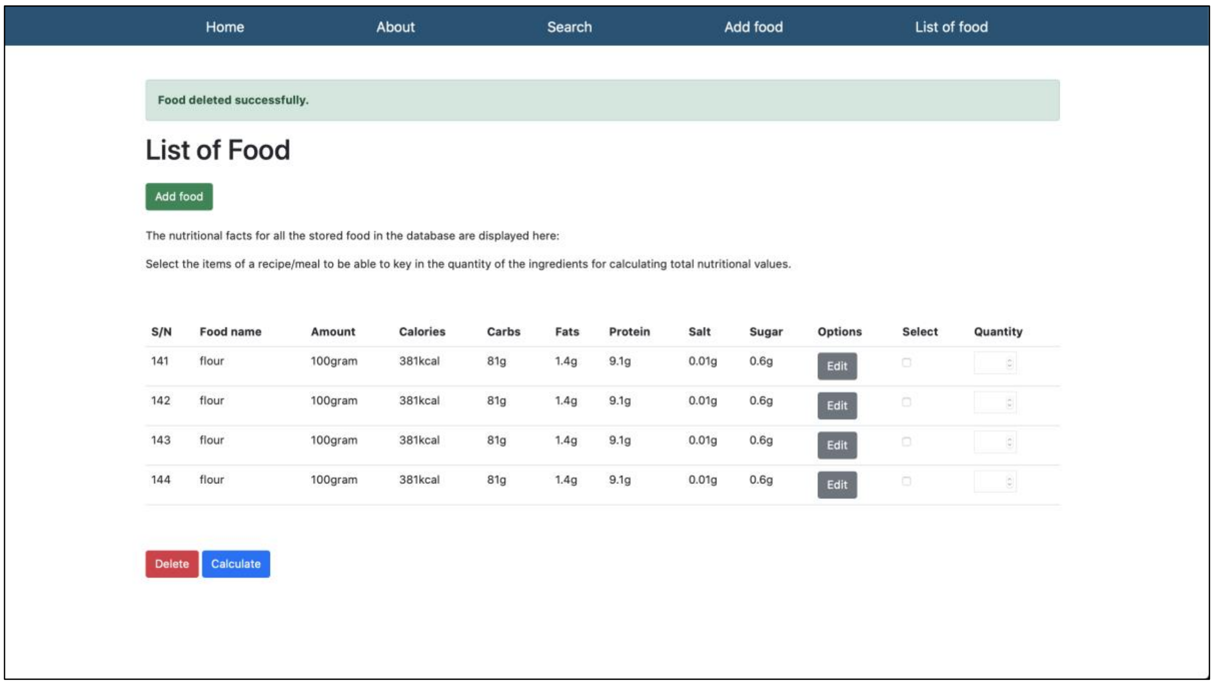
Select the items of a recipe/meal to be able to key in the quantity of the ingredients for calculating total nutritional values.

2. Confirmation message appears

Are you sure you want to delete ?

S/N	Food name	Amount							Options	Select	Quantity
139	flour	100gram							Edit	✓	
140	flour	100gram							Edit	✓	
141	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit		
142	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit		
143	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit		
144	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit		

If the user clicks 'OK', a message indicating the successful operation will be displayed on top of the page. But if the user clicks on 'cancel', the page simply refreshes itself:



Food deleted successfully.

List of Food

The nutritional facts for all the stored food in the database are displayed here:

Select the items of a recipe/meal to be able to key in the quantity of the ingredients for calculating total nutritional values.

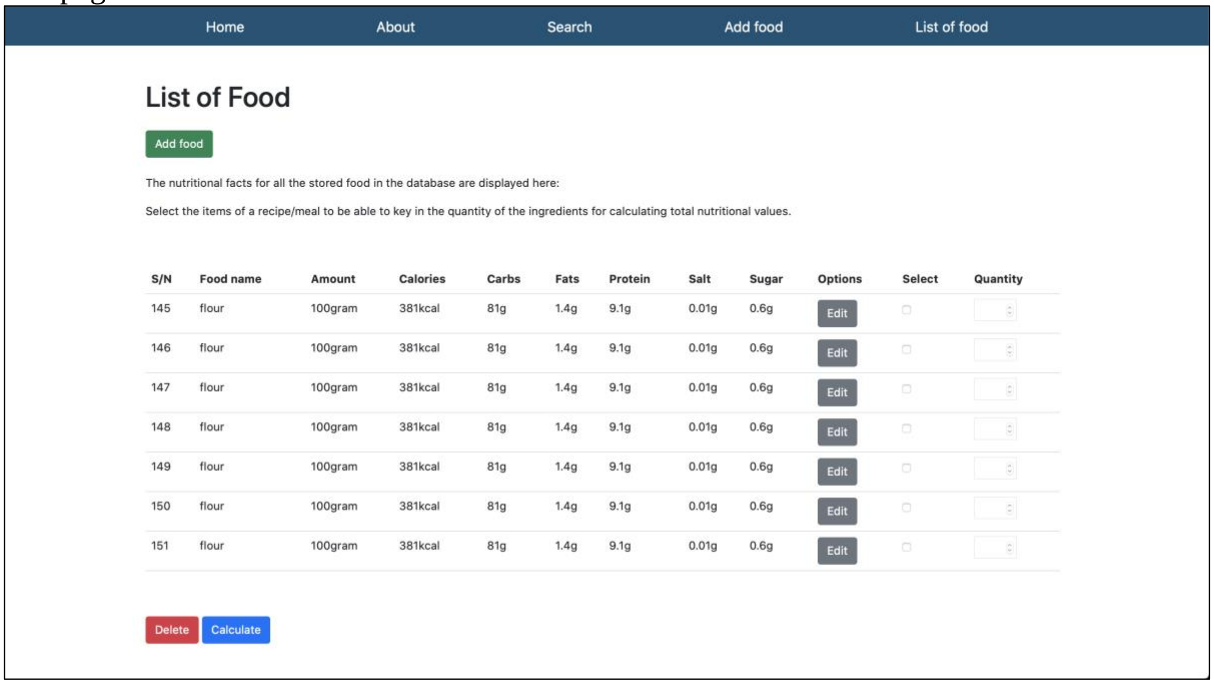
S/N	Food name	Amount	Calories	Carbs	Fats	Protein	Salt	Sugar	Options	Select	Quantity
141	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit		
142	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit		
143	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit		
144	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit		

^Successful delete

R6 – List Food Page

The rendered page view will be in **'list.html'**. The food data will be displayed in a tabular format in the page view. By using the *ejs forEach* function and html table tags, a row in the table will be created based on the number of food items there are in the database. The table header fields will be all based on the fields of the food items in the database. If there are no items, an alert message will show up, telling the user that there are no food items in the database. Additionally, in every row of the table, there is a checkbox and textbox field, which I will go into further detail later on.

The page view looks like this:



List of Food

The nutritional facts for all the stored food in the database are displayed here:

Select the items of a recipe/meal to be able to key in the quantity of the ingredients for calculating total nutritional values.

S/N	Food name	Amount	Calories	Carbs	Fats	Protein	Salt	Sugar	Options	Select	Quantity
145	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit	☐	
146	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit	☐	
147	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit	☐	
148	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit	☐	
149	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit	☐	
150	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit	☐	
151	flour	100gram	381kcal	81g	1.4g	9.1g	0.01g	0.6g	Edit	☐	

At the bottom of the table, there are two buttons like I mentioned earlier: delete and calculate. The delete button as mentioned in **R5** allows users to delete the selected food items. As for the calculate button, it allows users to be able to select the food items (recipe/meal) that they would like to calculate the total nutritional values of. Upon clicking the calculate button with the selected food items, it will submit the checkboxes' and textboxes values. Every checkbox has a unique id, which will be the food.id (<%= food.id %>) in the database, accessed using *ejs* tags. The values in the textboxes will contain a number input entered by the user (quantity).

The checkboxes and textboxes contain the following attributes:

```
<!-- DESC: Checkbox to select food item for deleting/calculating -->
<!-- The value of the checkbox will be the food.id, such that when the middleware handles this post request, this
food.id value will be able to be used to query the database and process delete/calculating -->
<td><input class="checkme"  type="checkbox" id="check<%= food.id %>" value="<%= food.id %>"></td>


<!-- DESC: Textbox to allow user to enter no. of food items/ingredients required for calculation -->
```

```
<!-- This textbox will only be enabled when the user has checked the checkbox in the same table row. The
entered value will then be pass into the middleware and then processed accordingly for calculations. -->
<td><input style="width: 60px" type="number" min = "1" max = "999" id="text<%= food.id %>" name = "quantity"
disabled></td>
```

The interaction between the checkboxes and textboxes are such that only when a checkbox of a certain table row is selected, will the textbox be enabled for input. By default, all of the textboxes are set to the *disabled* attribute.

The following script contains the logic for this operation, and they can be found in lines 111-118 in ‘list.html’:

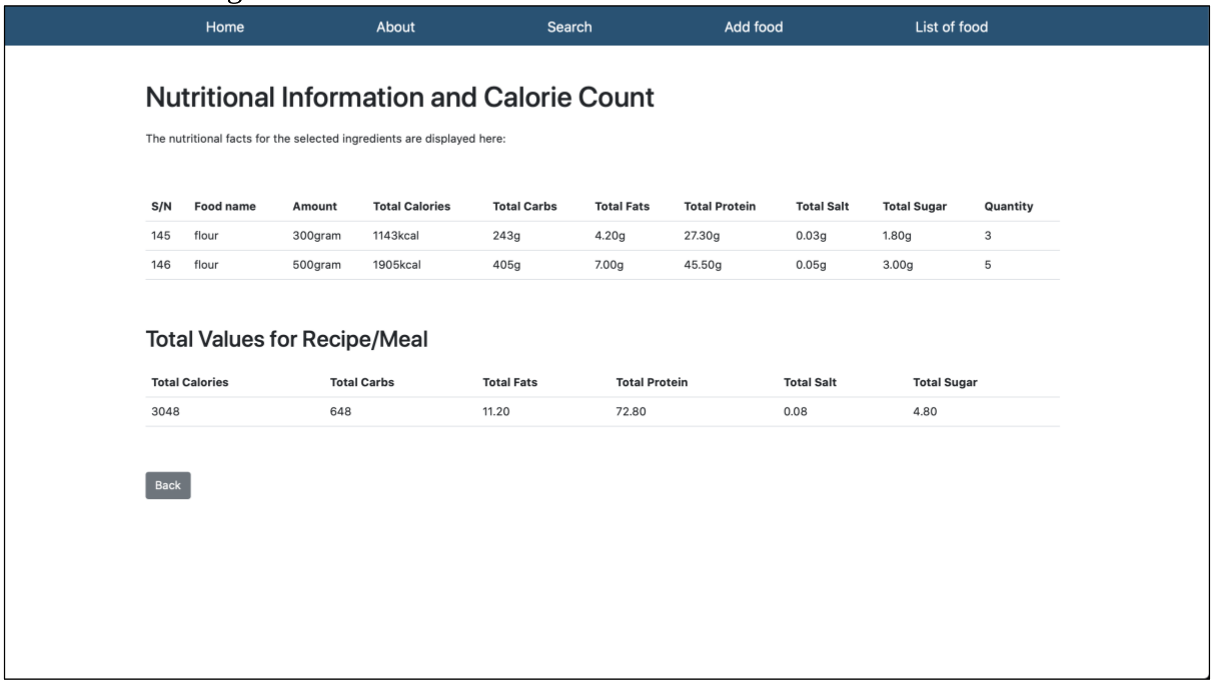
```
// DESC: Enable textbox after checking checkbox
// First, looks for every (input name = quantity) with disabled attr. Next, whenever the input with class name
"checkme" has been clicked, it looks for the closest (input name = quantity) in the same table row and enables it
if the input class "checkme" has been checked.
    $("input[name='quantity']").prop('disabled', 'true');
    $(".checkme").change(function(){
        $next = $(this).closest('tr').find('[name=quantity');
        $next.prop('disabled', !this.checked);
     });
```

The food.id and quantity will be passed into the middleware through a POST request, which will then query the database to retrieve the values of the other fields of the food items. Then, it adds the quantity values to the JSON food objects.

In the following code snippet, var food has been assigned with the result of the db.query. Currently, the food var holds all the JSON objects of the selected food items. Using a for loop to loop through the total number of food items, an addition {key:value} of {quantity: quantities} (Note that quantities is an array of the quantity values passed in through req.body). Afterwards, the response will be to render the **‘calculaterecipe.html’** page, which will collect the food JSON details (including quantity) through *ejs* tags and display the total values.

```
            food = result;
      for(var i=0; i<food.length; i++){
      // add the quantity input by the user to the food JSON for easier looping in html
            food[i]["quantity"] = quantities[i];
      }
 res.render("calculaterecipe.html", {selectedFood : food})
```

The rendered page view of **'calculaterecipe.html'** presents the tabular data of the total values of the selected ingredients *respectively* based on their quantity, followed by the total values of *all* the selected ingredients:



Nutritional Information and Calorie Count

The nutritional facts for the selected ingredients are displayed here:

S/N	Food name	Amount	Total Calories	Total Carbs	Total Fats	Total Protein	Total Salt	Total Sugar	Quantity
145	flour	300gram	1143kcal	243g	4.20g	27.30g	0.03g	1.80g	3
146	flour	500gram	1905kcal	405g	7.00g	45.50g	0.05g	3.00g	5

Total Values for Recipe/Meal

Total Calories	Total Carbs	Total Fats	Total Protein	Total Salt	Total Sugar
3048	648	11.20	72.80	0.08	4.80

By declaring a few global variables at the start of the **'calculaterecipe.html'** page, I am able to use that as a total counter to calculate the total values. Using the *ejs forEach* loop, I calculate the total values of the respective ingredients/food selected first; (*ingredients.values(carbs/salt/sugar..etc) * quantity*) and display them in the table data. This is done by multiplaying and parsing the result to a float within the *ejs* tags that display the values in each table row (lines 74-75 of 'calculaterecipe.html'):

```
<td><%= parseFloat(food.carbs  * food.quantity).toFixed(0) %>g</td>
<td><%= parseFloat(food.fats  * food.quantity).toFixed(2) %>g</td>
```

Likewise, these values will also be added to the total counter variables declared at the start (lines 61-62 of 'calculaterecipe.html'):

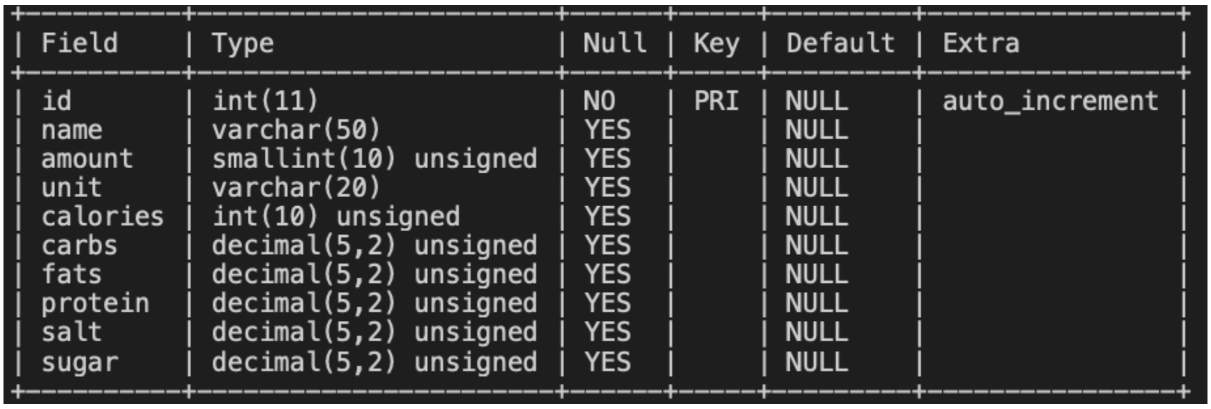
```
totalCalories += "<%= food.calories %>"  * "<%= food.quantity %>";
totalCarbs +=   "<%= food.carbs %>"    * "<%= food.quantity %>";
```

The total values for *all* of the ingredients; (*add all ingredients' values(carbs/salt/sugar…etc*) will then be presented in a tabular data below assigning the innerHTML values through a script (112-114 of 'calculaterecipe.html'):

```
document.getElementById("totalCalories").innerHTML = totalCalories;
document.getElementById("totalCarbs").innerHTML =parseFloat(totalCarbs).toFixed(0);
document.getElementById("totalFats").innerHTML = parseFloat(totalFats).toFixed(2);
```

At the bottom of the page, there will be a back button to go back to the '/list' page.

SQL Table



```
+----------+----------------------+------+-----+---------+----------------+
| Field    | Type                 | Null | Key | Default | Extra          |
+----------+----------------------+------+-----+---------+----------------+
| id       | int(11)              | NO   | PRI | NULL    | auto_increment |
| name     | varchar(50)          | YES  |     | NULL    |                |
| amount   | smallint(10) unsigned | YES  |     | NULL    |                |
| unit     | varchar(20)          | YES  |     | NULL    |                |
| calories | int(10) unsigned     | YES  |     | NULL    |                |
| carbs    | decimal(5,2) unsigned | YES  |     | NULL    |                |
| fats     | decimal(5,2) unsigned | YES  |     | NULL    |                |
| protein  | decimal(5,2) unsigned | YES  |     | NULL    |                |
| salt     | decimal(5,2) unsigned | YES  |     | NULL    |                |
| sugar    | decimal(5,2) unsigned | YES  |     | NULL    |                |
+----------+----------------------+------+-----+---------+----------------+
```

The purpose of the table is to store food objects with various field names and types according to their nutritional values. The field names corresponds to the types of nutritional attributes *e.g calories, carbs, fats etc.*

For the data types,

- id = auto_increment int to serve as the primary key which I will be using to identify different food items
- name = varchar(50) to contain a string of up to 50 characters
- amount = smallint(10) to contain the amount of food in terms of their unit measurement, and unsigned to only be able to take in positive integers.
- Unit = varchar(20) to contain a string with the type of measurement *e.g grams, milimetres*
- Calories = int(10) as it is usually a whole number, and unsigned to only take in positive integers
- For the rest of the data types namely: carbs, fats, protein, salt and sugar = decimal(5,2) to take in double values of up to 2 decimal places, and unsigned to only take in positive integers.

References

- Rahul, answer to a forum question in 'How to get multiple checkbox value using jQuery', Aug 2014, https://stackoverflow.com/questions/11945802/how-to-get-multiple-checkbox-value-using-jquery

- Milind Anantwar, answer to a forum question in 'jQuery – Checkbox checked enables field next to it table', Sep 2016, https://stackoverflow.com/questions/39662734/jquery-checkbox-checked-enables-field-next-to-it-in-table

- Jmtalarn, answer to a forum question in 'How to minimize the output value in ejs', Oct 2019, https://stackoverflow.com/questions/58269113/how-to-minimize-the-output-value-in-ejs

- Elad, answer to a forum question in 'NodeJS popup alert for POST method', Nov 2017, https://stackoverflow.com/questions/47529580/nodejs-popup-alert-for-post-method

- WalkThroughCode, Intro to EJS – injecting values into the view – 3, Jul 2017, https://www.youtube.com/watch?v=oalhTCW_5D8&list=PLtjob7NnSme96T9ajRv1OiB5eR1wpdG3H&index=16

- Tutorials Website, Display success alert message after CURD operation using express js and Mongoose js, May 2019, https://www.youtube.com/watch?v=DgY7eDkKXd0&list=PLtjob7NnSme96T9ajRv1OiB5eR1wpdG3H&index=17

- Web Dev Simplified, How to build a Markdown Blog using Node.js, Express, and MongoDB, Mar 2020, https://www.youtube.com/watch?v=1NrHkjlWVhM&list=PLtjob7NnSme96T9ajRv1OiB5eR1wpdG3H&index=13&t=2323s