DATABASES, NETWORK AND THE WEB
MIDTERM COURSEWORK: CALORIE BUDDY
STUDENT ID: 190488303

LIST OF REQUIREMENTS
HOMEPAGE

R1A: Display the name of the web application

```
<body>
  <h1>Calorie Buddy</h1>
```

A homepage.html file is created in the views folder. In the <body> section of the code of the html document, the name of the web application is displayed as “Calorie Buddy” in header 1 <h1>.

```
h1 {
  text-align: center;
}
```

This is styled in the <head> section of the code by horizontally aligning the text to the centre of the webpage.

```
app.get ('/', function (req, res)
{
    //set up and export homepage route to server
    res.render('homepage.html');
});
```

In the main.js file, the http method get is used to request data such that it can render the home page on the server.

R1B: Display links to other pages or a navigation bar that contains links to other pages.

```
<div class="navBar">
  <a href="/topic7/mid-term/">Home</a></li>
  <a href="/topic7/mid-term/listfoodpage">List food</a></li>
  <a href="/topic7/mid-term/searchfoodpage">Search food</a></li>
  <a href="/topic7/mid-term/addfoodpage">Add food</a></li>
  <a href="/topic7/mid-term/updatefoodpage">Update food</a></li>
  <a href="/topic7/mid-term/deletefoodpage">Delete food</a></li>
  <a href="/topic7/mid-term/aboutpage">About</a></li>
</div>
```

The above code can be found in the <body> section of the HTML document. The tag, which is named navBar, defines a division that contains hyperlinks to be styled in the <head> section. This is to create a navigation bar with links that provides easy access to other pages.

```
/* Add a black background color to the top navigation */
.navBar {
  overflow: hidden;
  background-color: #333;
}
```

The horizontal navigation bar is then styled within the <style> element under the <head> section of the HTML page. The code above styles the navigation bar by setting the background colour of the bar to black, and the overflow property ensures that overflow is clipped and anything outside the bar is not rendered.

```
/* Style the links inside the navigation bar */
.navBar a {
  font-size: 15px;
  text-align: center;
  padding: 18px 20px;
  float: left;
  color: #f2f2f2;
  text-decoration: none;
  border-right:1px solid #bbb;
}
```

The code above further styles the links inside the navigation bar. It sets font size of the link to 15px, aligns text to the centre of each box, defines the height and width of each box using 18px and 20x padding respectively, floats the links to the left of its container, sets the link colour to white, ensures that there is no text decoration hence removing the underline, and creates a border to separate each link from one another by using a line.

```
/* Change the color of links on hover */
.navBar a:hover {
  background-color: rgb(146, 158, 230);
  color: rgb(0, 0, 0);
}
```

This code above changes the colour of the links when the user hovers their mouse above it. It is done such that the bar changes to an indigo colour while the link itself changes to black for added visibility. The code explained in this section is added to every html page to ensure that each page can be accessed from any link of the application. Hence there is a navigation bar on every page.

ABOUT PAGE

R2A: Display information about the web application including your name as the developer. Display a link to the home page or a navigation bar that contains links to other pages.

```
<div class = "margin">
  <p>How to use Calorie Buddy:</p>

  <p>This web application is designed to give users the choice to add nutritional facts for food ingredients and
     store them into a food ingredients database. By allowing users to select the food ingredients to be included
     in a recipe or meal, it should then give users the total sum of calories in that recipe or meal.</p>

  <p>Web application developed by: Foo Rui Qi</p>

  <p>Student ID: 190488303</p>
</div>
</body>
```

An aboutpage.html file is created in the views folder. The above code can be found in the <body> section of the HTML document. The tag, which is named margin, defines a division that contains texts to be styled in the <head> section. These texts include information about the web application, and it includes my name as the developer.

```
  .margin {
    margin-right: 50px;
    margin-left: 50px;
    text-align: center;
  }
</style>
```

The aforementioned text is then styled within the <style> element under the <head> section of the HTML page. It is styled such that the text is aligned from the centre, and it leaves a margin of 50px on both the right and left side of the page such that those areas do not contain any text.

This file also contains the exact same code as R1B, to create a navigation bar that contains links to other pages.

```
app.get ('/aboutpage', function (req, res)
{
    //set up and export aboutpage route to server
    res.render('aboutpage.html');
});
```

In the main.js file, the http method get is used to request data such that it can render the about page on the server.

ADD FOOD PAGE

R3A: Display a form to users to add a new food item to the database. The form should consist of the following items: name, typical values, unit of the typical value, calories, carbs, fat, protein, salt, and sugar. Display a link to the home page or a navigation bar that contains links to other pages.

```
<div class = "margin">
  <h2>Add food data to database.</h2>

  <p>Input nutritional data about food item to database here.</p>

  <form method='POST' action="/topic7/mid-term/addfoodpage">
    <label for="name">Food name:</label><br>
    <input type="text" id="name" name="name" value=""><br><br>

    <label for="typicalValue">Typical Value:</label><br>
    <input type="number" id="typicalValue" name="typicalValue" value=""><br><br>

    <label for="unitOfTypicalValue">Unit of Typical Value:</label><br>
    <input type="text" id="unitOfTypicalValue" name="unitOfTypicalValue" value=""><br><br>

    <label for="calories">Calories:</label><br>
    <input type="number" id="calories" name="calories" value=""><br><br>

    <label for="carbs">Carbs:</label><br>
    <input type="number" id="carbs" name="carbs" value=""><br><br>

    <label for="fats">Fats:</label><br>
    <input type="number" id="fats" name="fats" value=""><br><br>

    <label for="protein">Protein:</label><br>
    <input type="number" id="protein" name="protein" value=""><br><br>

    <label for="salt">Salt:</label><br>
    <input type="number" id="salt" name="salt" value=""><br><br>

    <label for="sugar">Sugar:</label><br>
    <input type="number" id="sugar" name="sugar" value=""><br><br>

    <input type="submit" value="Submit">
  </form>
</div>
```

The code above can be found in a new file addfoodpage.html in the views folder and will first display some texts to briefly explain the contents of the webpage. It then uses the method post to send user input from the form on the webpage, to a server to be added as a record into the database. The label tag helps users to identify which field of data is being input while input type tag helps set the type of data that is being collected, for example, whether the form is collecting textual or numerical data. The last input type is a submit button which users can click to submit data into the form and database.

```
  .margin {
    margin-right: 100px;
    margin-left: 100px;
    text-align: center;
  }
</style>
```

The form as well as the text is then styled within the <style> element under the <head> section of the HTML page. It is styled such that the text is aligned from the centre, and it leaves a margin of 100px on both the right and left side of the page such that those areas do not contain any text.

This file also contains the exact same code as R1B, to create a navigation bar that contains links to other pages.

```
app.get ('/addfoodpage', function (req, res)
{
    //set up and export addfoodpage route to server
    res.render('addfoodpage.html');
});
```

Additionally, in the main.js file, the http method get is used to request data such that it can render the add food page on the server.

R3B: Collect form data to be passed to the back-end (database) and store food items in the database. Each food item consists of the following fields: name, typical values, unit of the typical value, calories, carbs, fat, protein, salt, and sugar.
Here is an example of a food item: name: flour, typical values per: 100, unit of the typical value: gram, calories: 381 kilocalories, carbs: 81 g, Fat: 1.4 g, Protein: 9.1 g, salt: 0.01 g, and sugar: 0.6 g. The unit of the typical value may have values such as gram, litre, tablespoon, cup, etc.

```
app.post("/addfoodpage", function (req,res) {
    //input data from form to database
    let sqlquery = "INSERT INTO food (name, typicalValue, unitOfTypicalValue, calories, carbs, fats, protein, salt, sugar) VALUES (?,?,?
    let newrecord = [req.body.name,
        req.body.typicalValue,
        req.body.unitOfTypicalValue,
        req.body.calories,
        req.body.carbs,
        req.body.fats,
        req.body.protein,
        req.body.salt,
        req.body.sugar];
    //execute sql query
    db.query(sqlquery, newrecord, (err, result) => {
```

In the main.js file, the http method post is used such that data sent from the server in the form can be collected and added to the database. The user would have input data into the form which would be stored into the variable newrecord, and later inserted into the database by using the INSERT INTO statement. This is also done using req.body.<name of field> such that each input data can be stored respectively into its following fields.

R3C: Display a message indicating that add operation has been done.

```
//execute sql query
db.query(sqlquery, newrecord, (err, result) => {
if (err) {
    //catch errors
    return console.error(err.message);
}else
   res.send('The food is added.');
});
```

In main.js, under app.post("/addfoodpage", function (req, res)), the code above is added to catch any errors in case user input cannot be added to the database. Otherwise, if add operation has been done successfully, it will redirect to a page that displays the message "The food is added".

SEARCH FOOD PAGE

R4A: Display a form to users to search for a food item in the database. 'The form should contain just one field - to input the name of the food item'. Display a link to the home page or a navigation bar that contains links to other pages.

```
<div class = "margin">
  <h2>This is the search page.</h1>

  <p>Input food name to find information about its nutritional value.</p>

  <form action="/topic7/mid-term/searchfoodpage-result" method='GET'>
    <input id="search-box" type="text" name="keyword">
    <input type="submit" value="Submit">
  </form>
</div>
```

A searchfoodpage.html file is created in the views folder. The code above will first display some texts to briefly explain the contents of the webpage. It then uses the method get to request data from the food database. The input id=”search-box” creates a search box which allows users to input textual data which is name of the food item to be searched. The input will be stored as a keyword, where type=”text” and name=”keyword”. The input type codes for a submit button which allows users to click and submit data into a file named /topic7/mid-term/searchfoodpage-result to process the input, as can be seen in the action attribute.

```
  .margin {
    margin-right: 100px;
    margin-left: 100px;
    text-align: center;
  }
</style>
```

The form as well as the text is then styled within the <style> element under the <head> section of the HTML page. It is styled such that the text is aligned from the centre, and it leaves a margin of 100px on both the right and left side of the page such that those areas do not contain any text.

This file also contains the exact same code as R1B, to create a navigation bar that contains links to other pages.

```
app.get ('/searchfoodpage', function (req, res)
{
    //set up and export searchfoodpage route to server
    res.render('searchfoodpage.html');
});
```

Additionally, in the main.js file, the http method get is used to request data such that it can render the search food page on the server.

R4B: Collect form data to be passed to the back-end (database) and search the database based on the food name collected from the form. If food found, display a template file (ejs, pug, etc) including data related to the food found in the database to users; name, typical values, unit of the typical value, calories, carbs, fat, protein, salt, and sugar. Display a message to the user, if not found.

```
app.get ('/searchfoodpage-result', function (req, res)
{
    //searching in the database using keyword
    let word = [req.query.keyword];
    let sqlquery = "SELECT * FROM food WHERE name like '%" +word+"%'";;
```

In the main.js file, the http method get is used to request data from the database. The user would have input data into the form which would be stored into the variable word, and later compared to the food names in the database to look for a match.

```
//execute sql query
db.query(sqlquery,word, (err, result) => {
    if (err || result == "") {
        //if keyword cannot be found in database
        res.send('The food cannot be found.');
        return console.error("No food found with the keyword you have entered" + req.query.keyword + "error: "+ err.message);
    }else{
        //show result of the search using an ejs template file, list.ejs can be used here
        res.render('listfoodpage.html',{available:result});
    }
});
```

The following code above matches the keyword to the food names in the database such that if there is no match, it will display a message to the user saying that “The food cannot be found.” If found, it displays a ejs template file including the rest of the data related to the food found in the database for that particular food item, by rendering results from the listfoodpage.html page.

R4C: Going beyond, search food items containing part of the food name as well as the whole food name. As an example, when searching for ‘bread’ display data related to ‘pitta bread’, ‘white bread’, ‘wholemeal bread’, and so on.

```
//searching in the database using keyword
let word = [req.query.keyword];
let sqlquery = "SELECT * FROM food WHERE name like '%" +word+"%'";;
```

The above code stores user input, keyword, into the variable word, and queries the database by using SELECT * FROM to select all fields from the database food. It then compares the food names stored in the database to the keyword such that any food name containing the keyword will be called and its data displayed.

UPDATE FOOD PAGE

R5A: Display search food form. Display a link to the home page or a navigation bar that contains links to other pages.

```
<div class = "margin">
  <h2>Search food to update.</h2>

  <form action="/topic7/mid-term/updatefoodpage-result" method='GET'>
    <input id="search-box" type="text" name="keyword">
    <input type="submit" value="Submit">
  </form>
</div>
```

An updatefoodpage.html file is created in the views folder. The code above will first display some texts to briefly explain the contents of the webpage. It then uses the method get to request data from the food database. The input id=”search-box” creates a search box which allows users to input textual data which is name of the food item to be searched. The input will be stored as a keyword, where type=”text” and name=”keyword”. The input type codes for a submit button which allows users to click and submit data into a file named /topic7/mid-term/updatefoodpage-result to process the input, as can be seen in the action attribute.

```
  .margin {
    margin-right: 100px;
    margin-left: 100px;
    text-align: center;
  }
</style>
```

The form as well as the text is then styled within the <style> element under the <head> section of the HTML page. It is styled such that the text is aligned from the centre, and it leaves a margin of 100px on both the right and left side of the page such that those areas do not contain any text.

This file also contains the exact same code as R1B, to create a navigation bar that contains links to other pages.

```
app.get ('/updatefoodpage', function (req, res)
{
    //set up and export updatefoodpage route to server
    res.render('updatefoodpage.html');
});
```

Additionally, in the main.js file, the http method get is used to request data such that it can render the update food page on the server.

R5B: If food found, display data related to the food found in the database to users including name, typical values, unit of the typical value, calories, carbs, fat, protein, salt, and sugar in forms so users can update each field. Display a message to the user if not found. Collect form data to be passed to the back-end (database) and store updated food items in the database. Display a message indicating the update operation has been done.

```
app.get ('/updatefoodpage-result', function (req, res)
{
    //searching in the database using keyword
    let word = [req.query.keyword];
    let sqlquery = "SELECT * FROM food WHERE name like '%" +word+"%'";;
```

In the main.js file, the http method get is used to request data from the database. The user would have input data into the form which would be stored into the variable word, and later compared to the food names in the database to look for a match.

```
//execute sql query
db.query(sqlquery,word, (err, result) => {
    if (err || result == "") {
        //if keyword cannot be found in database
        res.send('The food cannot be found.');
        return console.error("No food found with the keyword you have entered" + req.query.keyword + "error: "+ err.message);
    }else{
        //show result of the search using an ejs template file, list.ejs can be used here
        res.render ('updatefoodpage-update.html',{available:result});
    }
});
```

The following code above matches the keyword to the food names in the database such that if there is no match, it will display a message to the user saying that “The food cannot be found.” If found, it displays a ejs template file including the rest of the data related to the food found in the database for that particular food item, by rendering results to the updatefoodpage-update.html page.

```
<div class = "margin">
  <h2>This is the update food page.</h2>

  <p>This is data for the food item that you want to update:</p>

  <table style="width:100%">
    <tr>
      <th>Food ID</th>
      <th>Food name</th>
      <th>Typical Value</th>
      <th>Unit</th>
      <th>Calories</th>
      <th>Carbs</th>
      <th>Fat</th>
      <th>Protein</th>
      <th>Salt</th>
      <th>Sugar</th>
    </tr>
```

```
    <% available.forEach(function(food){ %>
    <tr>
      <td><%= food.id %></td>
      <td><%= food.name %></td>
      <td><%= food.typicalValue %></td>
      <td><%= food.unitOfTypicalValue %></td>
      <td><%= food.calories %></td>
      <td><%= food.carbs %></td>
      <td><%= food.fats %></td>
      <td><%= food.protein %></td>
      <td><%= food.salt %></td>
      <td><%= food.sugar %></td>
      <% }) %>
    </tr>
  </table>
```

A new file called updatefoodpage-update.html is created in the views folder. The two figures above show a table being created to store data from the food item that the user has searched for.

```
<h2>Update nutritional data of the food item here:</h2>

<form method='POST' action="/topic7/mid-term/updatefoodpage-update">
  <label for="id">Food id:</label><br>
  <input type="number" id="id" name="id" value = "" ><br><br>

  <label for="name">Food name:</label><br>
  <input type="text" id="name" name="name" value = "" ><br><br>

  <label for="typicalValue">Typical Value:</label><br>
  <input type="number" id="typicalValue" name="typicalValue" value=""><br><br>

  <label for="unitOfTypicalValue">Unit of Typical Value:</label><br>
  <input type="text" id="unitOfTypicalValue" name="unitOfTypicalValue" value=""><br><br>

  <label for="calories">Calories:</label><br>
  <input type="number" id="calories" name="calories" value=""><br><br>

  <label for="carbs">Carbs:</label><br>
  <input type="number" id="carbs" name="carbs" value=""><br><br>

  <label for="fats">Fats:</label><br>
  <input type="number" id="fats" name="fats" value=""><br><br>
  <label for="protein">Protein:</label><br>
  <input type="number" id="protein" name="protein" value=""><br><br>

  <label for="salt">Salt:</label><br>
  <input type="number" id="salt" name="salt" value=""><br><br>

  <label for="sugar">Sugar:</label><br>
  <input type="number" id="sugar" name="sugar" value=""><br><br>

  <input type="submit" value="Submit">
</form>
```

To collect form data to update the database, the code above will first display some texts to briefly explain the contents of the webpage. It then uses the method post to send user input from the form on the webpage, to a server to be added as a record into the database. The label tag helps users to identify which field of data is being input while input type tag helps set the type of data that is being collected, for example, whether the form is collecting textual or numerical data. The last input type is a submit button which users can click to submit data into the form and database.

```
  .margin {
    margin-right: 100px;
    margin-left: 100px;
    text-align: center;
  }
</style>
```

The form as well as the text is then styled within the <style> element under the <head> section of the HTML page. It is styled such that the text is aligned from the centre, and it leaves a margin of 100px on both the right and left side of the page such that those areas do not contain any text.

This file also contains the exact same code as R1B, to create a navigation bar that contains links to other pages.

```
app.post("/updatefoodpage-update", function (req,res) {
    //input data from form to database
    let sqlquery= "UPDATE food SET name = ?, typicalValue = ?, unitOfTypicalValue = ?, calories = ?, carbs = ?, fats = ?, protein = ?, sa
    let newrecord = [req.body.name,
        req.body.typicalValue,
        req.body.unitOfTypicalValue,
        req.body.calories,
        req.body.carbs,
        req.body.fats,
        req.body.protein,
        req.body.salt,
        req.body.sugar,
        req.body.id];
    //execute sql query
    db.query(sqlquery, newrecord, (err, result) => {
```

In the main.js file, the http method post is used such that data sent from the server in the form can be collected and used to update the database. The user would have input data into the form which would be stored into the variable newrecord, and later update the database by using the UPDATE statement. This is also done using req.body.<name of field> such that each input data can be update according to its respective fields.

```
//execute sql query
db.query(sqlquery, newrecord, (err, result) => {
    if (err) {
        //catch errors
        return console.error(err.message);
    }else{
        res.send('The food is updated.');
    }
});
```

In main.js, under app.post("/updatefoodpage-update", function (req, res)), the code above is added to catch any errors in case database cannot be updated with user input. Otherwise, if update operation has been done successfully, it will redirect to a page that displays the message "The food is updated".

R5C: Implement a delete button to delete the whole record, when the delete button is pressed, it is good practice to ask 'Are you sure?' and then delete the food item from the database, and display a message indicating the delete has been done.

To delete records, new files called deletefoodpage.html and deletefoodpage-delete.html are created in the views folder. They designed to be similar to the updatefoodpage.html and updatefoodpage-update.html pages, where user searches for food item by their name using a keyword in deletefoodpage.html, and if found, passes all data from that food record to be displayed in a table in deletefoodpage-delete.html.

```
<h3>Delete food</h3>

<p>Input first value from table above for food id.</p>

<form method='POST' action="/topic7/mid-term/deletefoodpage">
    <label for="id">Food id:</label><br>
    <input type="number" id="id" name="id" value = "" ><br><br>
    <input type="submit" onclick="return confirm('Are you sure?')"value="Delete">
</form>
```

In the html file deletefoodpage-delete.html, some text is used to explain the function of the webpage. It then uses the method post to send user input from the form on the webpage, to a server for the record to be deleted from the database. The label tag helps users to identify which field of data is being input while input type tag helps set the type of data that is being collected, which in this case, is the food id and numerical datatype. The last input type is a delete button which users can click to delete the record, in which case an “Are you sure?” prompt will appear. When confirmed, the record will be deleted.

```
db.query(sqlquery,id, (err, result) => {
    if (err || result == "") {
        //catch errors
        res.send('The food cannot be deleted.');
        return console.error("No food found with the keyword you have entered" + req.query.keyword + "error: "+ err.message);
    }else{
        res.send("The food is deleted");
    }
```

When deletion occurs, a message indicating the delete “The food is deleted” will be displayed.

LIST FOODS PAGE

R6A: Display all foods stored in the database including name, typical values, unit of the typical value, calories, carbs, fat, protein, salt, and sugar, sorted by name. Display a link to the home page or a navigation bar that contains links to other pages.

R6B: You can gain more mark for your list page, if it is organised in a tabular format instead of a simple list.

```
<div class = "margin">
  <h2>You can see nutritional facts for food ingredients here.</h2>

  <table style="width:100%">
    <tr>
      <th>Food ID</th>
      <th>Food name</th>
      <th>Typical Value</th>
      <th>Unit</th>
      <th>Calories</th>
      <th>Carbs</th>
      <th>Fat</th>
      <th>Protein</th>
      <th>Salt</th>
      <th>Sugar</th>
    </tr>
```

```
  <% available.forEach(function(food){ %>
  <tr>
    <td><%= food.id %></td>
    <td><%= food.name %></td>
    <td><%= food.typicalValue %></td>
    <td><%= food.unitOfTypicalValue %></td>
    <td><%= food.calories %></td>
    <td><%= food.carbs %></td>
    <td><%= food.fats %></td>
    <td><%= food.protein %></td>
    <td><%= food.salt %></td>
    <td><%= food.sugar %></td>
    <% }) %>
  </tr>
</table>
```

A listfoodpage.html file is created in the views folder. The above code can be found in the <body> section of the HTML document. The tag, which is named margin, defines a division that contains texts and data to be styled in the <head> section. These texts include introduction to the feature of the webpage, as well as a table containing all food data stored in the database. The tag <th> defines header cell elements, which are the fields, and the tag <td> defines data cell, which will contain the data from the database.

```
.margin {
  margin-right: 100px;
  margin-left: 100px;
  text-align: center;
}
```

The aforementioned text and table is then styled within the <style> element under the <head> section of the HTML page. It is styled such that the text is aligned from the centre, and it leaves a margin of 100px on both the right and left side of the page such that those areas do not contain any text.

```
table, th, td {
border: 1px solid black;
border-collapse: collapse;
}
```

The table is also styled by removing the space between borders and instead separating table cells with a 1px solid black line.

This file also contains the exact same code as R1B, to create a navigation bar that contains links to other pages.

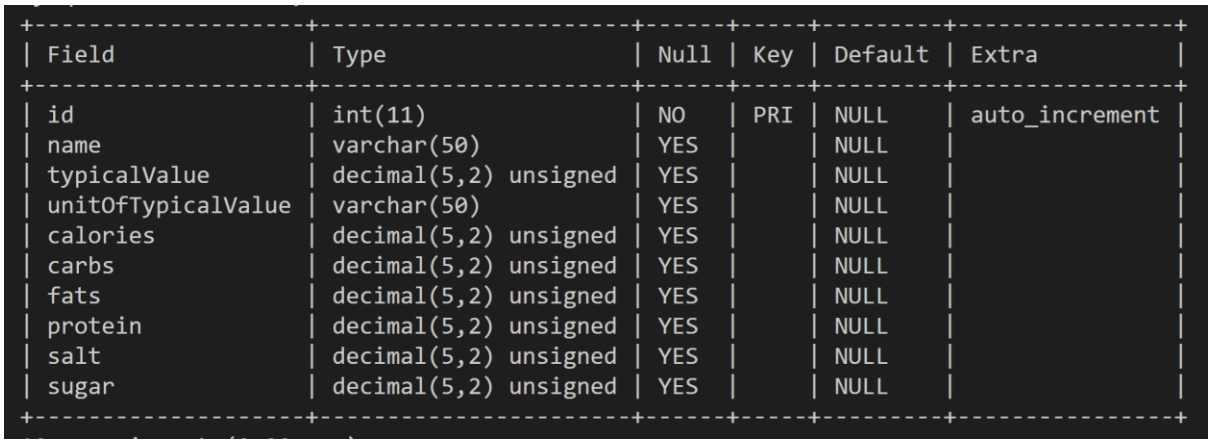
```
app.get("/listfoodpage", function(req, res) {
   //query database to get all the foods
   let sqlquery = "SELECT * FROM food";
   //execute sql query
   db.query(sqlquery, (err, result) => {
   if (err) {
       //catch errors
       res.redirect("/");
   }
   //set up and export listfoodpage while using results from database
   res.render("listfoodpage.html",{available:result});
   });
});
```

In the main.js file, the http method get is used to request data from the database. The code above queries the database by using SELECT * FROM to select all fields from the database food. It then uses the if loop to catch any errors. If there is none, it renders the data from the database to the page listfoodpage.html to be displayed in a tabular format.

R6C: going beyond by letting users select some food items (e.g. by displaying a checkbox next to each food item and letting the user input the amount of each food item in the recipe e.g. 2x100 g flour). Then collect the name of all selected foods and calculate the sum of the nutritional information (calories, carbs, fat, protein, salt, and sugar) related to all selected food items for a recipe or a meal and display them as 'nutritional information and calorie count of a recipe or a meal'. Please note, it is not necessary to store recipes or meals in the database.

Unable to complete, code written in listfoodpage.html

DATABASE STRUCTURE



Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
name	varchar(50)	YES		NULL	
typicalValue	decimal(5,2) unsigned	YES		NULL	
unitOfTypicalValue	varchar(50)	YES		NULL	
calories	decimal(5,2) 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	

For the assignment, I created a database called CalorieBuddy, and I have stored one table in it, named food. Its purpose is to store data or nutritional information of a food ingredient, into the respective fields of the table. From the SQL, the table food can be retrieved and displayed as shown in the screenshot above, by using the command DESCRIBE food;.

There are ten field names in the table, and they are called id, name, typicalValue, unitOfTypicalValue, calories, carbs, fats, protein, salt, sugar.

The datatypes for each field can be defined in the table below:

Field name	Datatype
id	int, it is set as the primary key of the table, and it auto increments
name	varchar(50)
typicalValue	unsigned decimal(5,2)
unitOfTypicalValue	varchar(50)
calories	unsigned decimal(5,2)
carbs	unsigned decimal(5,2)
fats	unsigned decimal(5,2)
protein	unsigned decimal(5,2)
salt	unsigned decimal(5,2)
sugar	unsigned decimal(5,2)