

Databases, Network and the Web Coursework for Midterm: CalorieBuddy

Documentation report

D3: List of requirements: for each sub-requirement (R1A → R6C) state how this was achieved or if it was not achieved. Explain where it can be found in the code. Use focused, short code extracts if they make your explanation clearer.

All requirements were achieved after the following initial process:

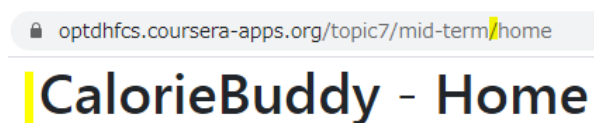
- `cd/topic7/mid-term`: Execute this command to change my working directly to the mid-term folder.
- `npm init`: Execute this command to set up a new npm package.
- `npm install express --save`: Execute this command to install the Express framework.
- `npm install ejs --save`: Execute this command to install the EJS framework
- `npm install body-parser --save`: Execute this command to install the body-parser module.
- `npm install mysql --save`: Execute this command to install the mysql module.

R1: Home page:

R1A: Display the name of the web application.

-> line 15 in index.html

-> line 6~8 in main.js

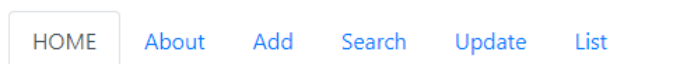


```
6 app.get("/home",function(req, res){
7   res.render("index.html")
8 });
```

Achieved by creating index.html file and adding its route in main.js file. In index.html, I indicated the name of the web application, "CalorieBuddy" as a title of the web site. Each page also include this name along with its page title. In main.js, I included app.get() method to add a route of "home" rendering the index.html.

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

-> line 18~37 in index.html



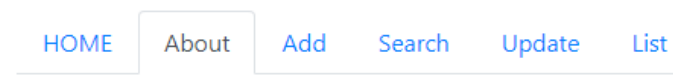
Achieved by including Bootstrap framework in the web site and using its navigation tab class. "Home" tab is activated and appeared as a primary tab while user is on Home page.

R2: About page:

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

-> line 19~58 in about.html

-> line 13~15 in main.js



```
13 app.get("/about",function(req, res) {
14   |   res.render("about.html");
15   | });
```

This is a web application that helps to manage your diet.
It displays nutritional facts including calories, carbs, fat, protei

Developer name: Mari Ashiga

Achieved by creating about.html file, adding its route in main.js file, and using Bootstrap navigation tab class to display links. In about.html, I included the information about the web application with my name. "About" tab is activated and appeared as a primary tab while user is on About page. In main.js, I included app.get() method to add a route of "about" rendering the about.html.

R3: Add food page:

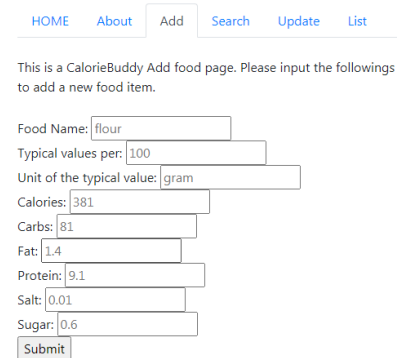
R3A: Display a form to users to add a new food item to the database. The form should consist of the following items: name, typical values, unit of the typical value, calories, carbs, fat, protein, salt, and sugar. Display a link to the home page or a navigation bar that contains links to other pages.

-> line 18~54 in add-food.html

-> line 20~22 in main.js

Achieved by creating add-food.html file, adding the "add-food" route in main.js file, and using Bootstrap navigation tab class to display the links. In add-food.html, I created a form consists of the required items showing the input examples using placeholders. "Add" tab is activated and appeared as a primary tab while user is on Add page. In main.js, I included app.get() method to add a route of "add-food" rendering the add-food.html.

```
20 app.get("/add-food",function(req, res) {
21   res.render("add-food.html");
22 });
```



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.

-> line 43~54 in add-food.html

-> line 27~48 in main.js

```
43 <form method="POST" action="/topic7/mid-term/food-added">

27 app.post("/food-added", function (req,res) {
28   // saving a food item data in database
29   let sqlquery = "INSERT INTO foods (name, typicalval, unit, calcs, c
```

Achieved by using POST method in the form to redirect to the "food-added" route and adding the "food-added" route in main.js file. In main.js, I included app.post() method to add a route of "food-added" to save the input data to database using the "INSERT INTO" query.

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

-> line 32~47 in main.js

```
32 db.query(sqlquery, newrecord, (err, result) => {
33   if (err) {
34     return console.error(err.message);
35   }else{ // send an HTTP response object displaying the added data
36     res.send(" This food is added to database, <br>Name: "+ req.body.name
```

Achieved by including db.query() method in app.post() method. In the app.post() method that handles a route of "food-added", I included the db.query() method. It outputs the error message on console if the query is failed, otherwise it displays the saved food items on Add food page as the responded message from the server.

R4: 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.

-> line 18~46 in search-food.html

-> line 53~55 main.js



This is a CalorieBuddy Search food page. Please input the name


```
53 app.get("/search-food",function(req, res) {
54   res.render("search-food.html");
55 });
```

Achieved by creating search-food.html file, adding the "search-food" route in main.js file, and using Bootstrap navigation tab class to display the links. In search-food.html, I created a form consists of one input field showing the input example using a placeholder. "Search" tab is activated and appeared as a primary tab while user is on Search food page. In main.js, I included app.get() method to add a route of "search-food" rendering the search-food.html.

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.

-> line 43~46 in search-food.html

-> line 48~79 in search-result.html

-> line 61~73 main.js

Name	Typical Value	Unit	Cals	Carbs	Fat	Protein	Salt	Sugar
flour	100	gram	383	82	1.5	9.2	0.02	0.8

Collecting and passing data was achieved by using GET method in the form to redirect to the "search-result-db" route and adding the "search-result-db" route in main.js file.

```
61 app.get("/search-result-db", function (req, res) {
62   //search a keyword that partially or fully matches in the
63   let word = [req.query.keyword];
64   let sqlquery = "SELECT * FROM `foods` WHERE name rlike ?";
```

In main.js, I included app.get() method to add a route of "search-result-db" to search in the database using the "SELECT * FROM" query which enables to search with the keyword. In the app.get() method, I included the db.query() method and it outputs the error message on console if the query is failed, otherwise it renders the list.html file passing a variable of "availableFoods" as the result.

Displaying data was achieved by creating list.html file, adding the "search-result" route in main.js file, and

using Bootstrap navigation tab class to display the links. In search-result.html, I created a table consists of the required data fields and displayed the result using EJS template tags.

```
65 <!-- go through the list of foods and print the contents -->
66 <% availableFoods.forEach(function(food){ %>
```

Specifically, I used the variable "availableFoods" passed by the sql query and used forEach loop to loop over the each food contained in the list of availableFoods. Displayed the following fields by accessing each variable that "food" instance contains: name, typical values, unit of the typical value, calories, carbs, fat, protein, salt, and sugar. "Search" tab is activated and appeared as a primary tab while user is viewing the search result.

```
70 res.render('search-result.html',{availableFoods: result});
```

In main.js, I included app.get() method to add a route of "search-result" rendering the search-result.html file as well as passing the variable "availableFoods" as the search result to frontend side .

```
62 <% if(availableFoods.length == 0){ %>
63 <p>No food item is found.<br></p>
64 <% } %>
```

Displaying a "not found" message was achieved by checking the length of the variable "availableFoods". If length is 0 it displays a message "No food item is found". For users' convenience, I included the search form above the search result table so they can continue searching a keyword.

R4C: Going beyond, search food items containing part of the food name as well as the whole food name.

-> line 64 in main.js

Name	Typical Value	Unit	Cals	Carbs	Fat	Protein	Salt	Sugar
flour	100	gram	383	82	1.5	9.2	0.02	0.8
miso soup	200	cc	50	40	1.5	20	1	1

Achieved by using "SELECT * FROM `foods` WHERE name rlike ?" query in main.js file. This enables pattern match of the keyword so the food items containing part of the food name and the whole food name are returned as the result.

R5: 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.

-> line 18~46 in update-food.html

-> line 77~79 main.js



```
77 app.get("/update-food",function(req, res) {
78   res.render("update-food.html");
79 });
```

This is a CalorieBuddy Update food page. Please input the na

Achieved by creating update-food.html file, adding the "update-food" route in main.js file, and using Bootstrap navigation tab class to display the links. In update-food.html, I created a form consists of one input field showing the input example using a placeholder. "Update" tab is activated and appeared as a primary tab while user is on Update food page. In main.js, I included app.get() method to add a route of "update-food" rendering the update-food.html.

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.

-> line 43~46 in update-food.html

-> line 50~92 in update.html

-> line 85~123 in main.js

```
43 <form action="/topic7/mid-term/food-displayed" method="GET">
```

Achieved by using same scheme I used for the Search food page and modifying it to add the update function. Specifically, collecting and passing data was achieved by using GET method in the form to redirect to the "food-displayed" route and adding the "food-displayed" route in main.js file.

```
85 app.get("/food-displayed", function (req, res) {
86   //search a keyword that partially or fully matches in the database
87   let word = [req.query.keyword];
88   let sqlquery = "SELECT * FROM `foods` WHERE name rlike ?";
```

In main.js, I included app.get() method to add a route of "food-displayed" to search in the database using the "SELECT * FROM `foods` WHERE name rlike ?" query which enables pattern match search. In the app.get() method, I included the db.query() method and it outputs the error message on console if the query is failed, otherwise it renders the update.html file passing a variable of "availableFoods" as the result.

Name	ID	Typical Value	Unit	Cals	Carbs	Fat	Protein	Salt	Sugar	
flour	1	100	gram	383	82	1.5	9.2	0.02	0.8	<input type="button" value="Update"/>

Displaying data was achieved by creating update.html file, adding the "update" route in main.js file, and using Bootstrap navigation tab class to display the links. In update.html, I created a table consists of the required data fields and displayed the result using EJS template tags. Usage of "availableFoods" variable is the same as I did for the requirement of R4B. "Update" tab is activated and appeared as a primary tab while user is viewing the search result.

```
// pass the variable, availableFoods to front-end
res.render('update.html', {availableFoods: result});
```

In main.js, I included app.get() method to add a route of "update" rendering the update.html file as well as passing the variable "availableFoods" as the search result to frontend side.

Displaying a "not found" message was achieved by checking the length of the variable "availableFoods". If length is 0 it displays a message "No food item is found". For users' convenience, I included the search form above the search result table so they can continue searching a keyword of the food item they wish to update.

```
<td><input id="sugar" type="text" size="5" name="sugar" value=<%= food.sugar %> /></td>
<td><input type="submit" formaction="/topic7/mid-term/food-updated" value="Update" /></td>
```

Updating data was achieved by providing input fields on the search result page, using POST method in the form to redirect to the "food-updated" route, and adding the "food-updated" route in main.js. In update.html file, I included the input form and Update button in which each field shows the current data so user can input new data and submit it.

```
103 app.post("/food-updated", function (req,res) {
104     // update data in database
105     let sqlquery = "UPDATE foods SET typicalval=" + req.body.typicalval +
```

In main.js file, I included app.post() method to add a route of "food-updated" to update the input data to database using "UPDATE foods SET" query. In the app.post() method, I included the db.query() method and it outputs the error message on console if the query is failed. Otherwise it displays a message indicating the update operation has been done along with the updated food items on Update food page as the responded message from the server.

```
115 res.send(" The update operation has been completed.
```

R5C: Going beyond by implementing 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.

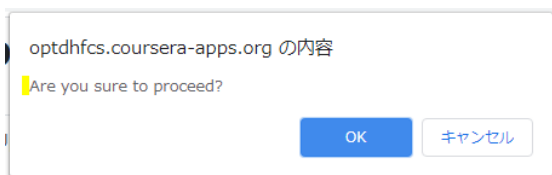
-> line 50~107 in update.html

-> line 128~142 in main.js

Name	ID	Typical Value	Unit	Cals	Carbs	Fat	Protein	Salt	Sugar	
flour	1	100	gram	383	82	1.5	9.2	0.02	0.8	<input type="button" value="Update"/> <input type="button" value="Delete"/>

```
76 <form method="POST" action="/topic7/mid-term/food-deleted" onSubmit="return check()">
```

Achieved by providing a Delete button next to the Update button in update.html file, using POST method in the form to redirect to the "food-deleted" route, and adding the "food-deleted" route in main.js file. In update.html file, I not only added the Delete button, but also implemented the check() method in Javascript that confirms the action with the user.



```
98 function check() {
99   if(window.confirm('Are you sure to proceed?')){ //
100     return true; // execute sql query if pressed OK
101   }
```

When user clicks on Delete or Update, the check() method is activated and waits for the returned value. If the form receives true it passes the data to the back-end and do nothing if receives false. In the check() method, I used window.confirm() method prompting a message "Are you sure to proceed?" to confirm with the user. If user clicks OK it returns true and returns false otherwise.

```
128 app.post("/food-deleted", function (req,res) {
129   // delete data from database
130   let sqlquery = "DELETE FROM foods WHERE id = " + req.body.id;
```

In main.js file, I included app.post() method to add a route of "food-deleted" to delete the selected food item from the database using "DELETE FROM" query. In the app.post() method, I included the db.query() method and it outputs the error message on console if the query is failed. Otherwise it displays a message indicating the delete has been done along with the name of food item on Update food page as the responded message from the server.

```
137 res.send(" This food has been deleted, <br>Name: " + req.body.name
```


R6: 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.

-> line 45~82 in list-foods.html

-> line 147~171 in main.js

Achieved by creating list-foods.html file and adding the "list-foods" route in main.js file. In list-foods.html, I created a table consists of the required data fields and displayed the result using EJS template tags. Usage of "availableFoods" variable is the same as I did for the requirement of R4B.

In main.js, I included app.get() method to add a route of "list-foods" to search all food items in the database using the "SELECT * FROM foods" query. In the app.get() method, I included the db.query() method and sorted the result by name before send it back to frontend. Let it output the error message on console if the query is failed. Otherwise it renders the list-foods.html file passing a variable of "availableFoods" as the result.

```
// pass the sorted result as availableFoods to frontend
res.render("list-foods.html", {availableFoods: result});
```

```
// sort the resulted food items by name
result.sort(function(a, b) {
    var nameA = a.name.toUpperCase(); //
    var nameB = b.name.toUpperCase(); //
    if (nameA < nameB) {
        return -1;
    }
    if (nameA > nameB) {
        return 1;
    }
    // names must be equal
    return 0;
});
```

R6B: Display a link to the home page or a navigation bar that contains links to other pages.

-> line 18~37 in list-foods.html

HOME About Add Search Update **List**

Achieved by using Bootstrap navigation tab class to display the links to all pages. "List" tab is activated and appeared as a primary tab while user is on List foods page.

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'.

-> line 45~100 in list-foods.html

-> line 43~104 in list-calc-nutrition.html

-> line 176~189 in main.js

Name	Amount	Typical Value	Unit	Cals	Carbs	Fat	Protein	Salt	Sugar
<input checked="" type="checkbox"/> flour	<input type="text" value="2"/>	100	gram	383	82	1.5	9.2	0.02	0.8

Calculate Nutrition

Achieved by modifying the table in list-foods.html, creating list-calc-nutrition.html file, and adding the "list-calc-nutrition" route in main.js file. Specifically, in list-foods.html, I added a checkbox and an Amount field next to each food item in addition to the hidden input field that stores all the IDs of checked food items.

```
66      <input type="hidden" id="ids" name="ids" >
80      <td><input type="submit" onclick="retrieveIds()" value="Calculate Nutrition"></td>
```

When user submits the form, retrieveIds() method is activated. It retrieves all the IDs of food items that user checked and stores them as a string in the input field which has the id = "ids". The collected IDs and the list of amounts are sent to back-end.

```
176 app.get("/list-calc", function(req, res) {
177     // query database to get the selected food items
178     let word = [req.query.ids]; // contains a list of food item IDs
179     let sqlquery = "SELECT * FROM `foods` WHERE `id` IN "+req.query.ids;
```

In main.js, I included app.get() method to add a route of "list-calc" to select all required food items in the database using the "SELECT * FROM `foods` WHERE `id` IN "+req.query.ids" query. In the app.get() method, I included the db.query() method and it outputs the error message on console if the query is failed. Otherwise it renders the list-calc-nutrition.html file passing variables of "availableFoods" and "amount" to front-end.

```
res.render("list-calc-nutrition.html", {availableFoods: result, amount: req.query.amount});
```

In list-calc-nutrition.html file, I looped over availableFoods, accessed each field (e.g. food.cals accesses calory of a food item in availableFoods), summed it up for all fields, respectively. Each field is multiplied by the amount that user input.

```
68     <!-- go through the list of foods, calculate nutritions -->
69     <% availableFoods.forEach(function(food){ %>
70     <% calories += food.cals * amount[i] %>
97     <td><b><%= Math.round(calories * 10) /10 %><b></b></td> <!-- round the number -->
```

For example, to calculate the calories field, I declared a variable "calories" initialized with 0, calculated food.cals * amount[i], and added it to calories. Looping this over N times where N as the number of food items in the list of availableFoods. Once the loop ends, display the rounded final value of calories at the last row of the table. Repeat this process for all fields: calories, carbs, fat, protein, salt, and sugar.

Name	Amount	Typical Value	Unit	Cals	Carbs	Fat	Protein	Salt	Sugar
flour	2	100	gram	383	82	1.5	9.2	0.02	0.8
green salad	3	100	gram	60	30	1.6	10	1	1
Nutritional Information				946	254	7.8	48.4	3	4.6

D4: Database structure: Structure and tables including purpose, field names, and data types for each table.

myFoodItems

```
|_ foods
    |_ id (PRIMARY KEY)
    |_ name
    |_ typicalval
    |_ unit
    |_ cals
    |_ carbs
    |_ fat
    |_ protein
    |_ salt
    |_ sugar
```

<p>DATABASE NAME: myFoodItems</p> <p>PURPOSE:</p> <p>To store and retrieve food items input by user. Also to be used for nutritional calculation.</p>	<pre>mysql> show databases; +-----+ Database +-----+ information_schema myBookshop myFoodItems myRestaurantMenu mysql performance_schema sys +-----+</pre>
<p>TABLE NAME: foods</p> <p>PURPOSE:</p> <p>To manage each food item with its fields.</p>	<pre>mysql> show tables; +-----+ Tables_in_myFoodItems +-----+ foods +-----+</pre>
<p>FIELD NAMES AND DATA TYPES:</p> <p>id -> the number of primary key</p> <p>name -> the name of food item</p> <p>typicalval -> the value of unit</p> <p>unit -> name of unit (e.g. gram or cc)</p> <p>cals -> the value of calories</p> <p>carbs -> the value of carbohydrate</p> <p>protein -> the value of protein</p> <p>salt -> the value of salt</p> <p>sugar -> the value of sugar</p> <p>PURPOSE:</p> <p>To manage the nutritional information that each food item contains. A unique (auto-incremented) ID is assigned to each food item when it's newly added to the database.</p>	

ABOUT DATA TYPES:

As you can see from the table below, the field of “id” uses integer, “name” and “unit” use varchar (can be used as a string), and rest of the fields use unsigned decimal (can be used as a double) as their data type. “id” is used as a primary key and is auto-incremented. Except the “id” field, all fields accept null value so when they are input with blank, null value is entered to the database by default.

```
mysql> describe foods;
```

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
name	varchar(50)	YES		NULL	
typicalval	decimal(5,2) unsigned	YES		NULL	
unit	varchar(50)	YES		NULL	
cals	decimal(5,2) unsigned	YES		NULL	
carbs	decimal(5,2) unsigned	YES		NULL	
fat	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	