**Bookshop Dynamic Web Application**

You can access the running web application via the URL below:

<http://doc.gold.ac.uk/usr/474/>

**WHAT MARKERS SHOULD PAY ATTENTION TO?**

**Register Page**

I have implemented form validation specifically in this register form. When you register as a user, you must ensure that you have a valid email that contains an @ and ensure that your password is at least characters long. Also, you must ensure that all the required fields have been field in before submitting the register button.

**List Page**

I have implemented an advanced total macro calculator for the list page as part of the going beyond and above requirement. In order to use this properly, you must first select the checkbox of the food item you want to calculate, select a quantity that is a whole number. If you add a decimal quantity, it will lead to an overflowing floating-point result which is not ideal. Once you have checked the checkboxes and entered the quantities, you must then press “Calculate Sum” to be produced with the total macros.

**Add Food Page**

I have implemented several input restrictions for the add food page. This includes string input restrictions for fields that are meant for numbers. For example, if you try to enter any letters or unicode for the typical values, calories, carbs, fat, protein, salt or sugar fields, you will be restricted as they are specifically intended for entering numbers. The same applies with food name and unit, they are only intended for string values not numbers. Although numbers can be entered, it is advised not to enter numbers. Finally, once you add a food item into the database, you will not be able to add an item with the same food name again, since it will already exist in the database. The only way to avoid this is to enter a different food name with the numbers you want to enter. The reason I have added this restriction is, to avoid any clashes with names when updating or delete a record from the database.

**Search Food Page**

I have implemented a basic and an advanced search for the search food page, which means if you search for a generic term e.g. milk, you will be produced with all the results that contains the word milk. Once you search for the data, you will be produced with all the data for that record e.g. foodname, typical value, unit, calories, carbs, fat, protein, salt, sugar as well as the name of the user.

**Update Food Page**

I have implemented a specific search for the update page, as you would need to search for a specific record with a specific food name, before carrying out any update or delete operations on it. This also takes into consideration case sensitivity. Once you have searched for a specific record, you will be displayed with its results as well as an option to either search again, update the record, or delete the record. When you click update, you will be presented with a form that already contains existing data. This will make it easier for you to update a specific record knowing what it already contains. **If you are not displayed with the update form, then either you entered the wrong data or you are not the user that added this record.** Once you have updated it, you simply press the update button which will result to a successful update message (taking into consideration that all the form fields have been filled in properly).

When you are deleting a record, you simply just press the delete button. This will display a confirmation box that says, “Are you sure?”. If you are sure that you want to delete the record, you can press the OK button in the confirmation box and then it will delete, otherwise nothing will happen. If the record is not successful in being deleted, then you may not be the user who has the authority to delete this record. Only the user who added this record, can delete it.

**API**

If you click the API button on my web app, you will be redirected to a new page with json results of all the data in the Fooddatabase collection. This will be displayed in JSON format. This specific API route implements GET ALL as it is retrieving all data.

**All pages I have implemented**

**Home Page**

**About Page**

**Login Page**

**Add Food Page**

**List Food Page (with advanced macro calculator)**

**Search Food Page (with advanced search)**

**Update Food Page (including delete)**

**API (with GET ALL, GET, POST)**

**Log Out**

**Requirement List**

**R1: Home page:**

**R1A:** Display the name of the web application.

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

**R2: 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.

**R3: Register page:**

**R3A:** Display a form to users to add a new user to the database. The form should consist of the following items: first name, last name, email address, username, and password.  Display a link to the home page or a navigation bar that contains links to other pages.

**R3B:**  Collect form data to be passed to the back end (database) and store user data in the database. Each user data consists of the following fields: first name, last name, email address, username and password. To provide security of data in storage, a hashed password should only be saved in the database, not a plain password.

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

**R4: Login page:**

**R4A:** Display a form to users to log in to the dynamic web application. The form should consist of the following items: username and password.  Display a link to the home page or a navigation bar that contains links to other pages.

**R4B:** Collect form data to be checked against data stored for each registered user in the database. Users are logged in if and only if both username and password are correct.

**R4C:** Display a message indicating whether login is successful or not and why not successful

**R5: Logout**

There is a way to logout, a message is displayed upon successful logout.

**R6: Add food page** (only available to logged-in users):

R6A: 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.

**R6B:**  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.

**R6C:** Display a message indicating that add operation has been done.

**R7: Search food page**

**R7A:** 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.

**R7B:**  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.

**R7C:** 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.

**R8: Update food page** (only available to logged-in users)

**R8A:** Display search food form. Display a link to the home page or a navigation bar that contains links to other pages.

**R8B:** 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. You can go beyond this requirement by letting ONLY the user who created the same food item update it.

**R8C:** 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. You can go beyond this requirement by letting ONLY the user who created the same food item delete it.

**R9: List food page** (available to all users)

**R9A:** 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.

R8B: You can gain more marks for your list page is organised in a tabular format instead of a simple list.

**R9C:** 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.

**R10: API**  
There is a basic API displayed on '/api' route listing all foods stored in the database in JSON format. i.e., food content can also be accessed as JSON via HTTP method, it should be clear how to access the API (this could include comments in code). Additional credit will be given for an API that implements get, post, push and delete.  
  
**R11: form validation**  
All form data should have validations, examples include checking password length, email validation, integer data is integer and etc. 

**R12:** Your dynamic web application must be implemented in Node.js on your virtual server. The back end of the web application could be MongoDB or MySQL. Make sure you have included comments in your code explaining all sections of the code including database interactions.

**Requirement Appendix**

**R1: Home page:**

R1A: DONE

R1B: DONE

**Refer to index.html**

**Refer to main.css**

**Refer to script.js**

**Refer to main.js line 17-19**

**R2: About page:**

R2A: DONE

**Refer to about.html**

**Refer to main.css**

**Refer to about.css**

**Refer to script.js**

**Refer to main.js line 22-24**

**R3: Register page:**

R3A: DONE

R3B:  DONE

R3C: DONE

**Refer to register.ejs**

**Refer to registered.ejs**

**Refer to login.css**

**Refer to main.js line 83-87, line 202-263**

**R4: Login page:**

R4A: DONE

R4B: DONE

R4C: DONE

**Refer to login.ejs**

**Refer to loggedin.ejs**

**Refer to login.css**

**Refer to colour.js**

**Refer to main.js line 90-98, line 266-316**

**R5: Logout**

DONE

**Refer to logout.ejs**

**Refer to activesession.ejs**

**Refer to login.css**

**Refer to main.js line 101-108**

**R6: Add food page**:

R6A: DONE

R6B:   DONE

R6C: DONE

**Refer to addFood.html**

**Refer to login.css**

**Refer to list.css**

**Refer to food.css**

**Refer to foodadded.ejs**

**Refer to foodexists.ejs**

**Refer to failedInsert.ejs**

**Refer to main.js line 27-29, line 431-491**

**R7: Search food page**

R7A: DONE

R7B: DONE

R7C: DONE

**Refer to search.html**

**Refer to search-results.ejs**

**Refer to main.js line 32-34, line 145-166**

**R8: Update food page** **and Delete page**

R8A: DONE

R8B: DONE

R8C: DONE

**Refer to update.html**

**Refer to updatesearch.ejs**

**Refer to updateform.ejs**

**Refer to foodupdated.ejs**

**Refer to fooddeleted.ejs**

**Refer to failedupdate.ejs**

**Refer to faileddelete.ejs**

**Refer to adminonly.ejs**

**Refer to main.js line 37-39, line 42-80, line 166-189, line 334-387, line 390-425**

**R9: List food page**

R9A: DONE

R8B: DONE

R9C: DONE

**Refer to listfood.ejs**

**Refer to totalMacros.js**

**Refer to main.js line 319-334**

**R10: API**  
DONE

**Refer to main.js line 111-125**  
  
**R11: form validation**  
DONE

**Refer to main.js line 202-212, line 282-286, line 350-360, line 440-450**

**R12:**

DONE

Dynamic web application was implemented in Node.js on virtual server. The back end was developed using MongoDB

**MongoDB Database Models**

**Fooddatabase Database**

**Collection Name:** foods

**Collection Fields:** {

username: <string>,

foodname: <string>,

value: <number>,

unit: <string>,

calories: <number>,

carbs: <number>,

fat: <number>,

protein: <number>,

salt: <number>,

sugar: <number>

}

**Myfirstdatabase Database**

**Collection Name:** accounts

**Collection Fields:** {

firstname: <string>,

lastname: <string>,

email: <string>,

username: <string>,

password: <string>

}