Final app and repo, assessment criteria and submission Instructions

Final App & Repo Assessment

This submission is worth 30% of the term 2 coursework mark i.e. 7.5% of your final mark overall.

- You will be assessed on the submission of a dynamic web application called "Calorie Buddy" (students who are suffering from eating disorders or similar and this project topic is of their concern, email me individually and I provide an alternative project topic for you)
- You should submit two URLs, the URL of your web application and the URL of the GitLab repo, as well as a copy of your main.js file
- You should include a clear statement in the README submitted with the app that describes what you have done and your database design.

Repo

The assessors will check your code, your README file as well as testing your web application.

Calorie Buddy Dynamic Web Application

In this module, you developed a small dynamic web application including routes, forms, and database access. In this assignment, you are tasked with creating another dynamic web application that will function as a digital calorie counter to help users manage their diet. Essentially the dynamic web application interacts with users to calculate and display nutritional facts including calories for their recipes or meals based on food ingredients in the recipe or the meal.

The user should be given the choice to add nutritional facts for food ingredients and to store them in the database. The web application should provide users with the sum of the calories for a recipe or a meal based on the food ingredients selected by the user to be included in the recipe or the meal. As an example, the user should be able to select the food ingredients for the recipe of 'apple pie' by choosing 'flour', 'egg', 'butter', 'brown sugar', and 'apple' and then the web application retrieves the nutritional facts for each food ingredient and calculates and displays the nutritional facts including calorie count of the 'apple pie' recipe. If a food ingredient is not in the database the user should be given the choice to add it to the database. A full list of the functionalities of the web application is explained in the next sections of this document. Some requirements are 'base' requirements to pass the assignment and some requirements are 'stretch goals' indicated as 'going beyond' and are designed for students who would like to achieve the full mark.

Requirement list

The purpose of the web application is to help people manage their diet by displaying nutritional facts including calories, carbs, fat, protein, salt, and sugar in a recipe based on food ingredients in the recipe. The web application provides several functionalities that should meet the following requirements:

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. 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, liter, tablespoon, cup, etc. Going beyond by saving the username of the user who has added this food item to the database.

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.

As you can see above, you need your own model (backend data structure), your own operations on that model, and the ability to access those operations through the web and (to some extent) through an API. Your dynamic web application has a database backend that implements CRUD operations (the database can be MySQL or MongoDB)

Detailed criteria for your GitLab repo:

1. The name of the project on GitLab must be "myapp"
2. There is a readme file in the root of your GitLab repo including the information listed below
3. There are comments inserted in the code, separating different parts of the code with a brief explanation of the functionality of the code

README

Include a *README.txt* file in your repo in the root directory. The README is your chance to tell us what you have worked on and show us what you have learned. Submission with no readme file can not be marked.

Use the README to:

- Highlight what parts of the app the markers should pay attention to i.e. **are your work**
- You should include a clear statement in the README submitted that describes what **YOU** have done. **Make sure you include the above requirements as well as "filename" and "line number" you have addressed each requirement**
- You should include your data model or database schema (Entity Relationship diagram) including names of tables or collections and their fields including primary and foreign keys. If you choose to use MongoDB not SQL make sure you explain your data model in details (including the name of the database, collection names, fields in each collection, references from one collection to the other, if there is any)

Submission Instructions:

Run your web application with 'forever' and submit two URLs as well as a copy of your main.js file.

Submit the URL for your app and the URL of your repo in the submission link that will be provided for you on VLE.

Your app should be running on:

`http://doc.gold.ac.uk/usr/YOUR_VIRTUAL_SERVER_ID`

- You should give master (developer) access to your repo to the following GitLab accounts: ehoma001 and TAs (ealja002, rberr004, and lrobi010).
- Refer to lecture notes for more information on final submission and also check this page before submission not missing any updates on this page
- Your app should be at the top level of your virtual server in "myapp" subdirectory and also at the top level of your repo

e.g.

|-\lab1

|-\lab2

|-\lab3

|- ...

|-/myapp

  |-/views

  |-/routes

Last modified: Friday, 19 March 2021, 1:31 PM

◄ Lecture 19 and 20 (final)

Jump to...

Introduction to the module, web application ►