Please ensure to go through lectures 9 and 10 before starting on this assignment.

Task: You have learned how to construct front-end web pages that communicate with back-end web servers using JavaScript. You also learned how to manage the mapping of data from back-end servers to HTML elements on a website's front-end. You additionally learned how to dynamically modify and control the appearance of HTML elements using CSS. This assignment requires you to use your knowledge of JavaScript, HTML, CSS, and Node.js to **design a complete REST API using Node.js that contains four or more microservices**, each associated with a specific route representing a specific functionality. The route identifies the pages (or functionality) that an application must serve in response to an HTTP request. Listed below is an illustration:

Assume that your Node.js application is accessible via http://127.0.0.1:3000 (or http://localhost:3000) and that the primary functionality we wish to implement is a simple fitness and health calculator that allows end users to calculate (a) Body Mass Index (BMI), (b) Body Fat, (c) Ideal Weight, and (d) Calories Burned. The focus of this application is fitness and health, and its four functionalities are BMI, body fat, ideal weight, and calories burned. Each function is associated with a unique route:

- BMI: http://localhost:3000/bmi
- body fat: http://localhost:3000/bodyfat
- ideal weight: http://localhost:3000/idealweight
- calories burned: http://localhost:3000/caloriesburned

When designing a REST API, you can specify any route name you choose. However, the routes should be meaningful and provide context for understanding the desired functionality. The following route, for instance, is valid but provides no meaningful representation:

- BMI: http://localhost:3000/abc or http://localhost:3000/1234
  vs.
- BMI: http://localhost:3000/bmi or http://localhost:3000/calcBMI

After designing and implementing the REST API with Node.js, you must **create an HTML web page that interacts with and consumes (or uses) the four microservices**. You may design the web page such that all functionalities are activated by a single form or by multiple forms (e.g., one form that points the action to a specific route). Your REST API can be comprised of four or more related microservices or a single major feature that requires four microservices to operate in sequence (e.g., service composition). A major functionality is composed of smaller functionalities in a service composition, where the output of microservice A becomes the input of microservice B, and so on. In service composition, the output of one service becomes the input for another. In this manner, you can compose major functionality from two or more microservices that run sequentially (microservice A runs, output is then sent to microservice B, microservice B runs, output is then sent to microservice C, etc.).

After completing the entire assignment, publish your work to GitHub. You can use this assignment to demonstrate your knowledge of Node.js and microservices/web services development when applying for jobs.

Please read below more specific details and requirements about the REST API assignment:

- Front-end Node.js application must be accessible via http://127.0.0.1:3000 (or http://localhost:3000)
- You do not have to use a cloud platform to deploy your REST API (i.e., must run locally on your machine)
- The functionalities that you implement in your REST API must be related. Think of a major functionality (or theme) that you wish to implement and break it down into a set of features, where each route represents a feature.
- All routes or functionalities must be local. That is, you cannot connect to an external API.
- Your web page should display the results of invoking the REST API asynchronously (you may use AJAX to update parts of your web page).
- At least one request within the web page must be generated through a form element (e.g. textbox, a drop-down menu, etc.)
- Your web page need to use CSS and JavaScript (or jQuery) for presentation.

- You are highly encouraged to use **Bootstrap or a CSS framework** to integrate into your web page.
- All of the operations need to be performed by the back-end server and can only be requested via your own REST API (i.e. no need for external operations).
- Node.js application does NOT require a database.
- You are not allowed to use any external node modules (except Express).
- Be creative: Your REST API will be graded based on functionality, complexity and creativity.
- Your REST API must generate responses in JSON format (for the response body).
- You must document your REST API very thoroughly.

SUBMIT What to submit:

1. **userid-a3.zip** containing (where userid is your UW userid):
   - **index.html** file (web page) along with any CSS or JavaScript files
   - **Node.js** application (REST API) files (*package.json, node-modules folder, etc*.)
   - **readme.txt** that contains
     - instructions on how to run the frontend and the Node.js backend
     - **GitHub address** where the code is deployed (must be publically accessible for grading)
2. All deliverables should be neatly formatted, readable and well-documented.

Bonus: 1 additional mark if application is deploy on GCP or Heroku (please include the IP address and ensure VM is running for at least one week after the due date to be able to grade)

Grading Rubric: **20 points**:

- 10 points for REST API design and development
- 5 points for creativity of the functionalities (and how they are related to each other)
- 2 points web page updating DOM
- 2 points for presentation
- 1 point for source code documentation (e.g., comments to explain what your code does)
- (-1) if readme is missing or unclear
- (-3) if quality of submitted work is not creative
- (-3) if not published on GitHub