SER421 Fall 2021 Lab1 HTTP and Javascript Intro

__Objectives__:
1. Familiarize yourself with the HTTP request/response format
2. Gain proficiency in Javascript/JSON

Part 1 is an individual assignment. Part 2 you may choose to complete with a partner (optional). The intent of Part 2 is to gain Javascript proficiency if you do not have it, so the purpose of allowing a partner is so you have a "study buddy" to learn the language with – it is not to just divide up the work!

Part 1: Construct HTTP requests using Postman (35%)

In this part, you will construct HTTP requests using the Postman tool according to the specifications provided below. The requests should be header-complete, and if required URL and/or payload complete. You must capture the requests in a Postman collection, and you must screen-capture the full requests and full corresponding responses to each of the specifications below.

Preliminary:
Go to URL https://swent1linux.asu.edu/ser421lab1/index.html. This will show you a simple web form for inputting an item into a Grocery List. The form is submitted to https://swent1linux.asu.edu/ser421lab1/groceries, which displays a confirmation. You can also visit https://swent1linux.asu.edu/my_groceries to retrieve the list of all groceries. Further, the my_groceries endpoint may be filtered by a query string which accepts 1 or 2 parameters:

- Name: "aisle" Value: an integer – this will filter the results to products in the specified aisle
- Name: custom Value: a String – this will filter the results to products that have a substring matching the parameter value
- If both parameters are present then the filters are both applied ("AND" semantics)

NOTE: For you to be able to access swent1linux.asu.edu, you will either need to be on campus or VPN'd in to ASU if you are off campus

I encourage you to use this simple app, view the source of results, and use the developer tools to inspect the request and response header and payload traffic. Use this information to construct the required queries below:

1. Issue a proper request to retrieve the contents of the index.html page. Replicate what the browser does.
2. Issue a proper request to create a new grocery item (just like the results of submitting the form). You may create any item you like, but you should name the Product your asurite id. As you may try this more than once you may append a counter number at the end. For example, I would create products kgary1, kgary2, kgary3, etc.
3. Repeat #2 but this time indicate the payload is in JSON, and construct the proper JSON (Note: The JSON payload is simply name-value pairs using the same parameter names in the HTML form).
4. Issue a proper request to retrieve all grocery items in the grocery list as html
5. Issue a proper request to retrieve only those grocery items in aisle 3 and indicate you want the results as plain text.
6. Issue a proper request to retrieve only those grocery items in aisle 3 that are in a "veg" diet and indicate you want the results as JSON
7. Issue a syntactically *__incorrect request__* to retrieve grocery items. You determine the request and capture the results. Please delete entirely the header you used to indicate the desired results format in queries 4-6.

Submission: You should capture each query as part of a postman collection. You can then export that collection as a JSON file (please select a version 2.1 export!). Please provide screen captures of each Postman request with the Headers tab selected for each request, and the response body (in the bottom pane). It is OK if the response body gets cutoff a little as long as we see most of it. Put this cut-and-paste in a Word file and make sure you label each one 1-7.

Each of the 7 queries is worth 5 points.

Part 2: Javascript Golf Tournament Leaderboard (65%)

Golf is an individual competitive sport where a player plays multiple *rounds* (typically 4, but we will only have 1) in a *tournament*. In each round, a player plays 18 *holes*, where each hole has an expected score, called *par*. Score is tracked during the round with respect to par, so for example a score of -2 after 10 holes means the player has taken 2 less shots than expected (which is good). Likewise, a score of +3 would indicate taking 3 shots more than par (which is not so good). At the conclusion of the round the score is officially recorded in the aggregate sum of the holes. The player with the lowest score after all rounds (again for this task we will assume only 1 round is played) in the tournament are completed wins the tournament.

For this activity you will implement a collection of Javascript functions and manipulate Javascript objects to implement a scoreboard (or what is called in golf the *leaderboard*).

a. Implement a function to parse the JSON string below into objects of type *Tournament* and *Player*. It should be obvious that a Tournament will have many Players. The JSON string is to be the argument to the Tournament constructor.
b. Implement a function named *leaderboard* on Tournament that returns a JSON string back except with the player entries sorted first lowest score first. This function should be named *leaderboard*.
c. Implement a function named *projectScoreByIndividual* that accepts as parameters a Tournament object, plus the lastname and first initial of a Player and returns a projected score for the player when the round is completed. Project the player's final score based on his current score and hole (individual rate of progress). For example, if the player has a score of -2 after 12 holes, then project that he has a final score of -3 at the end of the round.
d. Implement a function named *projectScoreByHole* that accepts as parameters a Tournament object, plus the lastname and first initial of a Player and returns a projected score for the player at the end of the round by projecting the player's final score based on his current score and hole plus the collective rate of progress (the average score per hole for all Players in the Tournament). For example, if the player's score is -2 on he has finished the 9th hole, and on average golfers are scoring +0.11 per hole then add -2 plus 9 * +0.11 to get a projected score of -1. For the example above, the returned score would be -3 + (1 * 0.11) = -2.89, which rounded is -3.
e. Implement a function named *projectedLeaderboard* that does exactly what *leaderboard* does except it takes another argument representing a function (*projectScoreByXXX*) and uses this function to determine a leaderboard based on each player's projected finishing score. You should implement this by reusing the leaderboard function from part b.
f. Modify the prototype of the *Tournament* object by adding a function *printLeaderboard* that prints the leaderboard (sorted from best score to worst then by hole (later holes first)) to the console using console.log().
g. Add a function to *Player* named *postScore(s)* where *s* is a score with respect to par (e.g. -1, 0, 1, 2) that represents the Player's score on the next hole. Update the Player's score and hole properties based on *s*. If the Player has completed 18 holes set the hole to the special string "finished". If the Player has already finished then do not modify the Player score at all.
h. If the tournament is completed (all Players have hole == "finished"), create a new property on Tournament named *winner* that is assigned the player that is the winner of the tournament. Add a *getWinner* function on the Tournament object that returns the winner's last name and final score.
i. Except where stated, you are implementing objects and functions, not outputting things to the screen (console). Provided a unit test script that exercise requirements b-h by invoking the functions/methods and outputting results to the screen (the test code should have console.log statements). Call this file task2test.js. Use comments in the test case to indicate the rationale for each test step (Did you check edge cases? Why are these inputs appropriate? Are you negative and positive testing?).

Each function a-g is worth 7 points, with requirement h worth 9 points. The unit test script (requirement i) is worth 7 points (and is based on your rationale).

JSON example:
```
{ "tournament": {
        "name": "British Open",
        "year": "1998",
        "award": 840000,
        "yardage": 6905,
        "par": 71,
        "players": [
           {"lastname": "Montgomerie", "firstinitial": "C", "score": -3, "hole": 12 },
          {"lastname": "Fulke", "firstinitial": "P", "score": -5, "hole": "finished"}
   ]
}  }
```

<u>Extra Credit (20%): using *bind()*</u>

In task 2, the *projectScoreByIndividual* and *projectScoreByHole* functions are not on the Tournament object as this would make requirement e more difficult. Passing methods (functions associated with objects) around as parameters requires an extra binding step that ensures the method-as-function is bound to the proper object state. Research the use of the *bind* function that accomplishes this and modify your solution in task 2 to put these two functions as methods on Tournament (10 points). Second, research the use of events and event emitters to automatically determine when the last Player has completed the round, thereby completing the Tournament. Upon completion, an event should be fired indicating the tournament is complete, and an event listener should cause the winner to be automatically declared (requirement h) (5 ponts). Modify your test cases accordingly (5 points).

Name the files task2EC.js and task2testEC.js