CSE 154: Web Programming Summer 2019

Practice Exam 2A

Note that this practice exam is slightly longer (~5-10 minutes) than you should expect to provide more practice when studying.

Name:

UWNet ID: @uw.edu

TA (or section):

Rules:

- You have 60 minutes to complete this exam.
- You will receive a deduction if you keep working after the instructor calls for papers.
- You may not use any electronic or computing devices, including calculators, cell phones, smartwatches, and music players.
- Unless otherwise indicated, your code will be graded on proper behavior/output, not on style.
- Do not abbreviate code, such as writing ditto marks ("") or dot-dot-dot marks (...). You may not use JavaScript frameworks such as jQuery or Prototype when solving problems.
- If you enter the room, you must turn in an exam and will not be permitted to leave without doing so.
- You must show your Student ID to a TA or instructor for your submitted exam to be accepted.

Question	**Score**	**Possible**
Short Answer		12
JS with AJAX		12
Node.js/Express Web Service		14
SQL		12
XC		
Total		50

1. Short Answers

1. Validation Methods. What is one example of validating user input on the client-side?

2. GET vs. POST. Provide and justify a specific example when it is more appropriate to use a POST request instead of a GET request (2-3 sentences).

3. Regex. For each of the two regular expressions, circle all the string(s) below that match it:

i. `/^[A-Z]+4\d\d$/`

- `CSE431`
- `CSE154`
- `http400`
- `ABC4dd`
- `404`

ii. `/^pup.*y.jpg$/`

- `puppy.jpg`
- `^puppypuppy.jpg$`
- `pupy.jpg`
- `puppyparty.jpg`
- `puppykitty.JPG`

Regex reference:

[abc]	A single character of: a, b, or c	.	Any single character	(...)	Capture everything enclosed
[^abc]	Any single character except: a, b, or c	\s	Any whitespace character	(a|b)	a or b
[a-z]	Any single character in the range a-z	\S	Any non-whitespace character	a?	Zero or one of a
[a-zA-Z]	Any single character in the range a-z or A-Z	\d	Any digit	a*	Zero or more of a
^	Start of line	\D	Any non-digit	a+	One or more of a
$	End of line	\w	Any word character (letter, number, underscore)	a{3}	Exactly 3 of a
\A	Start of string	\W	Any non-word character	a{3,}	3 or more of a
\z	End of string	\b	Any word boundary	a{3,6}	Between 3 and 6 of a

options: i case insensitive m make dot match newlines x ignore whitespace in regex o perform #{...} substitutions only once

4.fs, glob, and path modules. Suppose a directory has the following structure (with `package.json/node_modules` included as appropriate):

```
app.js
mydir/
    images/
        puppy1.jpg
        puppy1.png
        puppy2.gif
    puppy-facts.txt
    puppy-haz-pizza.jpg
```

Consider the following statement written in app.js, **assuming readdir and globPromise are promisified versions of fs.readdir and glob:**

```
let result = STATEMENT;
```

For each statement replacing STATEMENT, what would be the value of `result`? Use [ ] for any Arrays and "" for Strings.

Statement	**Value of `result`**
`await readdir("mydir");`	
`await readdir("mydir/images");`	
`await globPromise("mydir/puppy-facts.txt");`	
`await globPromise("mydir/*/*");`	
`await globPromises("mydir/puppy*");`	
`path.extname("mydir/images/puppy1.jpg");`	
`path.basename("mydir/images/puppy1.jpg");`	

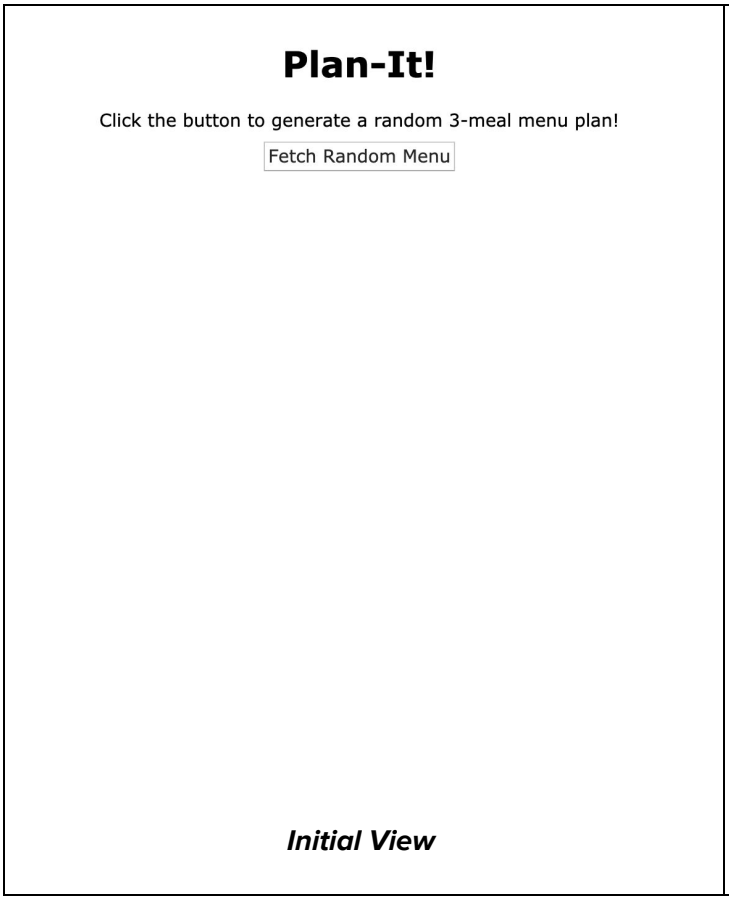
5. Data Storage Methods. For each data storage method, mark "X" in each column for the location it can be accessed during an HTTP request/response in Node.js/Express. Note that some methods may have X in both Client and Server columns.

	Client (Browser)	**Server (Node.js/Express)**
The website's DOM.		
Text Files		
SQL databases		
Cookies		

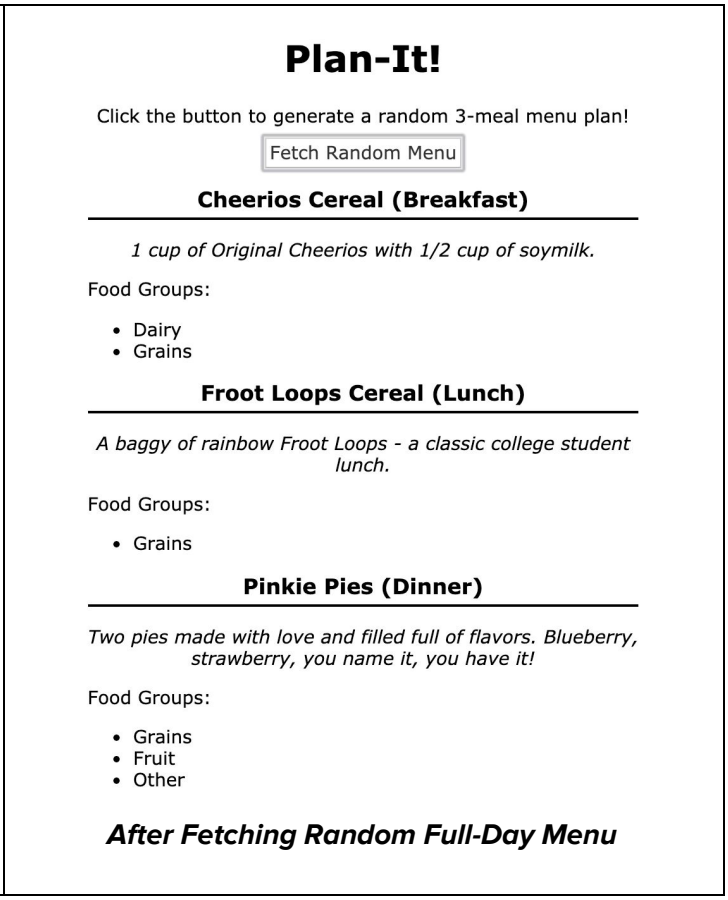
6. Node/mysql Connection. Why is it important to use ? placeholders in `db.query` when using promise-mysql in a Node.js app? (2-3 sentences).

2. (JS with Fetch): Plan-It! Fetching You Meals a Day at a Time

In this question, you will write JavaScript to implement a small Meal Planner web page called Plan-It!. For simplicity, we will consider a "full day meal plan" as a breakfast, lunch, and dinner (the 3 standard meal types). Below are two screenshots of the Plan-It! page:



Initial View



After Fetching Random Full-Day Menu

Directory Structure

You will fetch from a Node.js API you implement in Problem 2. Assume your all Problem 2 and Problem 3 files are inside a `planit` directory as follows, where app.js correctly serves the static `public/` directory (similar to the HW4 and the Final Project required directory structures).

```
planit/
 └── public/
    └── planit.html
       planit.js
       styles.css
   app.js
   package.json
   data/
     └── // meal directories specific to Problem 3, irrelevant in Problem 2
```

Implementation Requirements

In planit.js, you will fetch using **only one** of the two GET endpoints you will implement in `app.js` for Problem 3.

Clicking the `#day-btn` should make a fetch request to the API's `/dayplan` GET endpoint to fetch a random full day meal plan (one option for each of breakfast, lunch, and dinner). The response JSON format will be in the following format (example):

```
{ "breakfast" : {
    "name" : "Blueberry Oatmeal",
    "description" : "One cup of hot oatmeal with 1/4 cup of fresh blueberries.",
    "food-groups" : ["Grains", "Fruit"]
  },
  "lunch" : {
    "name" : "Froot Loops",
    "description" : "A baggy of rainbow Froot Loops - a classic college student lunch.",
    "food-groups" : ["Grains"]
  },
  "dinner" : {
   "name" : "Spaghetti with Tomato Sauce",
    "description" : "Two pies made with love and filled full of flavors. Blueberry,
                     strawberry, you name it, you have it!",
    "food-groups" : ["Grains", "Fruit", "Other"]
  }
}
```

Upon success, remove the `.hidden` class from the `#day-results` (you will not ever need to add it back) and use the response JSON to populate each of the `#breakfast`, `#lunch`, and `#dinner` aside elements on the page such that:

- The `.name` element is populated with the corresponding meal name.
- The `.description` is populated with the corresponding meal description
- The `.food-groups` is populated with a list of the food groups for that item (one or more food groups may be in the "food-groups" array in the JSON).

A client should be able to click the `#day-btn` multiple times to get a new randomly-generated 3-meal menu (previous menu results should be replaced as a result).

Note: You do not need to know any CSS used by the page other than the `.hidden` class which should be added/removed as previously specified.

Provided HTML:

```
<body>
  <h1>Plan-It!</h1>
  <p>
    Click the button to generate a random 3-meal menu plan!
    <button id="day-btn">Fetch Random Menu</button>
  </p>

  <section id="day-results" class="hidden">
    <article id="breakfast">
      <h2><span class="name"></span> (Breakfast)</h2>
      <p class="description"></p>
      <p>Food Groups:</p>
      <ul class="food-groups"></ul>
    </article>
    <article id="lunch">
      <h2><span class="name"></span> (Lunch)</h2>
      <p class="description"></p>
      <p>Food Groups:</p>
      <ul class="food-groups"></ul>
    </article>
    <article id="dinner">
      <h2><span class="name"></span> (Dinner)</h2>
      <p class="description"></p>
      <p>Food Groups:</p>
      <ul class="food-groups"></ul>
    </article>
  </section>
</body>
```

Write your client-side JS solution below. You may assume that the `checkStatus(resp)` function and the aliases `id(idName)`, `qs(el)`, `qsa(sel)`, and `gen(tagName)` are defined for you and are included as appropriate.

```
"use strict";
(function() {

})();
```

3. (Node.js Web Service) *Serving* Up Some Meal Ideas

In this question, you will *implement* the Plan-It API (`app.js`) you were asked to use in Problem 2.

Directory Structure and File Format Details

`app.js` will be located in the `data/` subdirectories corresponding to each of the standard meals in a day (breakfast, lunch, and dinner). You may assume these are the only subdirectories in `data`.

Project Directory Structure

```
app.js
public/
└── planit.html
  planit.js
  styles.css
data/
└── breakfast/
    └── <foodoption>.txt
      <foodoption>.txt
      ...
  lunch/
    └── <foodoption>.txt
      <foodoption>.txt
      ...
  dinner/
    └── <foodoption>.txt
       <foodoption>.txt
      ...
```

Example `data/` Directory Contents:

```
breakfast/
└── banana.txt
   blueberry-oatmeal.txt
   ...
lunch/
└── froot-loops.txt
   spinach-salad.txt
   ...
dinner/
└── brown-rice-with-veggies.txt
  pinkie-pies
```

Inside of data subdirectory (which you may assume is non-empty), are .txt files for different food options for that subdirectory meal type, each having three lines:

.txt File Content Format:

```
name
description
food groups
```

Example File Contents: (blueberry-oatmeal.txt)

```
Blueberry Oatmeal
One cup of hot oatmeal with 1/4 cup of fresh blueberries.
Grains Fruit
```

where `name` is the full name of the food option, the `description` is a short description, and `food groups` lists any major food groups associated with the food item (on a single line).

Web Service Implementation

You will implement 2 GET requests in app.js as Part A and Part B, described on the next page.

Part A: `GET /meal/:type`

This request should return a plain text response of all food options available in the corresponding `type` directory (ignoring letter-casing, so "breakfast" is treated the same as "BREAKfast"). Each food option should be output on a new line in the format of:

```
name: description
```

where `name` corresponds to the first line of the .txt file for that food item and `description` is the second line. The order of lines in the output not matter. For example, a GET request to `/meal/breakfast` might output the plain text:

```
Banana: A good source of potassium on the go.
Blueberry Oatmeal: One cup of hot oatmeal with 1/4 cup of fresh blueberries.
Cheerios Cereal: One cup of original Cheerios with 1/2 cup of soymilk.
```

Note that this response corresponds to three breakfast meal options in the breakfast directory, but your request should work for any number of txt files that may be found in the directory.

Part B: `GET /dayplan`

This request should return a JSON response containing a random option from each meal type. One example response may look like the following (identical to the example we gave in Problem 3):

```
{ "breakfast" : {
    "name" : "Blueberry Oatmeal",
    "description" : "One cup of hot oatmeal with 1/4 cup of fresh blueberries.",
    "food-groups" : ["Grains", "Fruit"]
  },
  "lunch" : {
    "name" : "Froot Loops",
    "description" : "A baggy of rainbow Froot Loops - a classic college student lunch.",
    "food-groups" : ["Grains"]
  },
  "dinner" : {
   "name" : "Spaghetti with Tomato Sauce",
    "description" : "Two pies made with love and filled full of flavors. Blueberry,
                     strawberry, you name it, you have it!",
    "food-groups" : ["Grains", "Fruit", "Other"]
  }
}
```

Error Handling

Your web service should handle the following errors (in order of highest priority to lowest priority). If any error occurs, only output the respective error message in plain text.

- If, when lower-cased, the `:type` parameter for `meal/:type` does not correspond to a directory in `data/`, send a 400 error with the message: "`<type> not a valid meal type.`", replacing `<type>` with the value of the `:type` path parameter (using the same letter-casing as the parameter passed by the client).
- If any file- or directory- processing error occurs, send a 500 error using the provided `SERVER_ERROR_MSG` constant, "Something went wrong on the server, please try again later.".

For full credit, your `app.js` must always send the endpoint's response, may not overwrite content headers, and may not modify the response after it has been sent.

You will write your solution to `app.js` below, completing the sections labeled Part A and Part B:

```
"use strict";
const express = require("express");
const util = require("util");
const fs = require("fs");
const glob = require("glob");
const readFile = util.promisify(fs.readFile);
const readdir = util.promisify(fs.readdir);
const globPromise = util.promisify(glob);
const app = express();
app.use(express.static("public/"));

const SERVER_ERROR_MSG = "Something went wrong on the server, please try again later.";

// Part A: Implement GET /meal/:type endpoint
```

```
// Part B on next page.
```

```
// Part B: Implement GET /dayplan endpoint. While not required, we suggest implementing a
// getRandOption(mealType) function below your endpoint to factor out building the data
// for each of the three data/ subdirectories.
```

```
const PORT = process.env.PORT || 8000;
app.listen(PORT);
```

4A. (SQL table creation/insertion): Preparing the Dining Table.

After some file recovery following a fatal ENOENT error, your team has decided to move their directory structures to a SQL database. At the moment, our new **Foods** setup.sql contains a single table called **`groupings`** to associate foods with their respective food groups - this allows us to associate one unique food element (e.g. "Blueberry oatmeal") with potentially multiple groups (e.g. "Grains" and "Dairy").

The **`CREATE TABLE`** statement for this table is shown below, with an example of the corresponding table with a few rows it may eventually have:

```
CREATE TABLE groupings (
  group_name VARCHAR(32) PRIMARY_KEY,
  food_id INT,
  FOREIGN KEY food_id REFERENCES foods(id)
);
```

food_id	*group_name*
1	Grains
1	Fruit
2	Grains
...	...

groupings table

In order for this **groupings** table to be created, we need to first have a CREATE TABLE statement for the referenced **foods** table. Your job is to implement the **foods** table creation, which will represent individual food items to replace the directory structure in the `data` directory from Problems 2/3. Below is an example of a single row in the table corresponding to the same date for the dinner item in the example `/dayplan` request of Problems 2/3.

id	*food_name*	*description*	*meal_type*
1	Blueberry Oatmeal	One cup of hot oatmeal with 1/4 cup of fresh blueberries.	breakfast

Note that the ***id*** value should be automatically incremented each time a new record is added to this table.

1. Write the CREATE TABLE statement which will initialize the **foods** table below, requiring:
   - `food_name` strings having no more than 100 characters, `description` strings having no more than 255 characters, and `meal_type` strings having no more than 20 characters
   - `food_name` and `meal_type` are required, but `description` defaults to NULL

2. Write the INSERT statement (assuming your CREATE TABLE is correct) to insert the example row data for Blueberry Oatmeal as the first row in the **foods** table:

4B. (SQL Queries): Around the World in SQL

This question uses the World database from the CSE 154 Query Tester:

world:

code	name	continent	independence_year	population	gnp	head_of_state	...
AFG	Afghanistan	Asia	1919	22720000	5976.0	Mohammad Omar	...
NLD	Netherlands	Europe	1581	15864000	371362.0	Beatrix	...
...							

countries

Other columns: region, surface_area, life_expectancy, gnp_old, local_name, government_form, capital, code2

country_code	language	official	percentage
AFG	Pashto	T	52.4
NLD	Dutch	T	95.6
...			

languages

id	name	country_code	district	population
3793	New York	USA	New York	8008278
1	Los Angeles	USA	California	3694820
...				

cities

a. Write a SQL query to list the **name** and **gnp** of all countries in Asia with a gnp greater than 123456.

Expected results (9 rows total):

name	gnp
India	447114.00
Iran	195746.00
Japan	3787042.00
...	

Write your SQL query below:

b. Write a SQL query to list the **name of all countries** having a population of fewer than 85,000 and which have languages spoken (but not necessarily official) by a percentage of at least 50% of the population. Include the **language name** and **language percentage** in your results. Order the results by language name (alphabetically increasing) breaking ties by percentage (decreasing). *Hint: The percentage column in the languages table represents the percentage of the population speaking that language in the corresponding country.*

Expected results (16 rows total):

language	name	percentage
Creole English	Dominica	100.0
Creole English	Saint Kitts and Nevis	100.0
Creole English	Antigua and Barbuda	95.7
English	Bermuda	100.0
English	Gibraltar	88.9
Faroese	Faroe Islands	100.0
...	...	...

Write your SQL query below:

c. Write a SQL query to list the **city name** and **continent name** of all cities containing the word "sea" which are cities in countries having English as an official language. Order the results by city name (increasing alphabetically).

Expected results (3 rows total):

name	continent
Seattle	North America
Southend-on-Sea	Europe
Swansea	Europe

Write your SQL query below:

X. Extra Credit (1pt)

For this question, you can get 1 point of extra credit (demonstrating at least 1 minute's worth of work and being appropriate in content).

Write a poem for your TA or draw a picture of them on their perfect summer vacation.