CSE 154: Web Programming — Summer 2019

Practice Exam 2A | Key

1. Short Answers

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

Possible answers include:
- Using built-in HTML5 validation attributes, such as required and pattern.
- Using client-side JS to programmatically check input values (e.g. by length, checking equality with repeated inputs, regex patterns, etc.)

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).

Possible answers include:
- Sending sensitive user data with a create account form. POST requests are more secure than GET requests since the data is sent in the body instead of the URL.
- Requests where the server will be modified, such as a request to add a user. POST requests are more appropriate than GET requests to signify the data will change something on the server.

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`

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");`	`["images", "puppy-facts.txt", "puppy-haz-pizza.jpg"]`
`await readdir("mydir/images");`	`["puppy1.jpg", "puppy1.png", "puppy2.gif"]`
`await globPromise("mydir/puppy-facts.txt");`	`["mydir/puppy-facts.txt"]`

`await globPromise("mydir/*/*");`	`["mydir/images/puppy1.jpg",`
	`"mydir/images/puppy1.png",`
	`"mydir/images/puppy2.gif"]`

`await globPromise("mydir/puppy*");`	`["mydir/puppy-facts.txt",` `"mydir/puppy-haz-pizza.jpg"]`
`path.extname("mydir/images/puppy1.jpg");`	`".jpg"`
`path.basename("mydir/images/puppy1.jpg");`	`"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.	X	
Text Files		X
SQL databases		X
Cookies	X	X

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).

? escape parameter strings in SQL queries that may evaluate to SQL statements which leak sensitive information or modify the database. They also make using db.query much cleaner and less prone to programmer error compared to String concatenation.

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

One possible solution is provided below

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

  window.addEventListener("load", () => {
    id("day-btn").addEventListener("click", fetchFullMenu);
  });

  function fetchFullMenu() {
    fetch("/dayplan")
      .then(checkStatus)
      .then(resp => resp.json())
      .then(populateFullMenu)
      .catch(console.log);
  }

  function populateFullMenu(responseData) {
    id("day-results").classList.remove("hidden");
    for (let key in responseData) {
      let item = responseData[key];
      qs("#" + key + " .name").textContent = item.name;
      qs("#" + key + " .description").textContent = item.description;
      let ul = qs("#" + key + " .food-groups");
      ul.innerHTML = "";
      for (let i = 0; i < item["food-groups"].length; i++) {
        let li = gen("li");
        li.textContent = item["food-groups"][i];
        ul.appendChild(li);
      }
    }
  }

  // alias functions

})();
```

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

One possible solution is provided below:

```
// Part A Solution
app.get("/meal/:type", async (req, res) => {
  let type = req.params.type;
  res.type("text");
  try {
    let txts = await globPromise("data/" + type.toLowerCase() + "/*txt");
    if (txts.length === 0) {
      res.status(400).send(type + " not a valid meal type.");
    } else {
      let results = "";
      for (let i = 0; i < txts.length; i++) {
        let contents = await readFile(txts[i], "utf8");
        let lines = contents.split("\n");
        results += lines[0] + ": " + lines[1] + "\n";
      }
      res.send(results);
    }
  } catch (err) {
    res.status(500).send(SERVER_ERROR_MSG);
  }
});

// Part B Solution
app.get("/dayplan", async (req, res) => {
  try {
    let randBreakfast = await getRandOption("breakfast");
    let randLunch = await getRandOption("lunch");
    let randDinner = await getRandOption("dinner");
    let results = {
      "breakfast" : randBreakfast,
      "lunch" : randLunch,
      "dinner" : randDinner
    };
    res.json(results);
  } catch (err) {
    res.status(500).send(SERVER_ERROR_MSG);
  }
});

async function getRandOption(mealType) {
  let txts = await globPromise("data/" + mealType + "/*txt");
  let randOption = txts[Math.floor(Math.random() * txts.length)];
  let contents = await readFile(randOption, "utf8");
  let lines = contents.split("\n");
  return {
    "name" : lines[0], "description" : lines[1], "food-groups" : lines[2].split(" ")
  };
}
```

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

1. Write the CREATE TABLE statement which will initialize the **foods** table below.

```
CREATE TABLE foods(
  id INT PRIMARY KEY AUTO_INCREMENT,
  food_name VARCHAR(100),
  description VARCHAR(255) DEFAULT NULL,
  meal_type VARCHAR(20)
);
```

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:

```
INSERT INTO foods (food_name, description, meal_type)
VALUES ("Blueberry Oatmeal", "One cup of hot oatmeal with 1/4 cup of fresh blueberries.", "breakfast");
```

4B. (SQL) Around the World in SQL

a. Solution:

```
SELECT name, gnp
FROM countries
WHERE gnp > 123456
AND continent = "Asia";
```

b. Possible solutions:

```
-- WHERE solution
SELECT l.language, c.name, l.percentage
FROM languages l, countries c
WHERE l.country_code = c.code
AND c.population < 85000
AND l.percentage >= 50
ORDER BY l.language ASC, l.percentage DESC;
```

```
-- JOIN solution
SELECT l.language, c.name, l.percentage
FROM languages l
JOIN countries c ON l.country_code = c.code
WHERE c.population < 85000
AND l.percentage >= 50
ORDER BY l.language ASC, l.percentage DESC;
```

c. Possible solutions:

```
-- WHERE solution
SELECT ci.name, c.continent
FROM cities ci, countries c, languages l
WHERE ci.country_code = c.code
AND l.country_code = countries.code
AND ci.name LIKE "%sea%"
AND l.official = 'T'
AND l.language = "English"
ORDER BY ci.name;
```

```
-- JOIN solution
SELECT ci.name, c.continent
FROM cities ci
JOIN countries c ON ci.country_code = c.code
JOIN languages l ON l.country_code = c.code
WHERE ci.name LIKE "%sea%"
AND l.official = 'T'
AND l.language = "English"
ORDER BY ci.name;
```