

Final Exam

Name: _____

UWNet ID: _____@uw.edu

TA (or section): _____

Rules:

- You have 110 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. You may receive a deduction if we hear or see your electronic device during our exam time.
- Unless otherwise indicated, your code will be graded on proper behavior/output, not on style.
- This is a closed-note exam, but you may use the provided cheat sheet for reference. As noted on the cheat sheet, you may assume `id`, `qs`, `qsa`, and `checkStatus` are provided in JS as shorthand for `document.getElementById`, `document.querySelector`, and `document.querySelectorAll`, respectively.
- 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.
- If you are done early, please pack up quietly and turn your exam into a TA as you leave. You will not be allowed to return once you have exited the exam room.

Question	Score	Possible
1. Short Answer		20
2. CSS		13
3. JS/DOM		15
4. Node.js Web Service		20
5. JS with Fetch		20
6. SQL Queries and PDO		12
7. Extra Credit		1
Total		100

1. Short Answers (20pts total)

1. HTML: Happy Little DOM Trees! (2.5pts)

Consider the webpage layout below. For each of the 5 arrows (each pointing to a **single** element surrounded by a dashed border), select the most appropriate HTML tag from the list on the left and write the name of the tag to the right of the arrow's number.


<head>
<header>
<main>
<footer>
<nav>
<title>
<section>
<input>
<p>

<h1>
<hr>
<code>

<button>

What I Learned in 19sp CSE 154

Reflections on a class:



I know the difference between a client and a server. I also know how to search for web programming resources that are on-line. I can debug client- and server-side programs. And I learned the importance of citing sources in my code.

Technologies I learned:

1. Content and structure with HTML
2. Style with CSS
3. Behavior with JS, JSON, AJAX, and browser storage like Dexie
4. Server side programming with PHP and Python/Flask
5. Databases with SQL

Other fascinating things I learned:

- There's some kind of intense TA debate about peanut butter or something...
- Bagels are too expensive
- Lauren likes to bake cookies
- Melissa's dog Mowgli is big

Smiley face from [Wikipedia](#)

Diagram description: A webpage layout for 'What I Learned in 19sp CSE 154'. It features a header, a main content area with a smiley face and a paragraph, a list of technologies learned, another main content area with a list of interesting facts, and a footer with a link to a smiley face on Wikipedia. Five arrows point to specific elements: 1 points to the paragraph in the first main content area, 2 points to the technologies list, 3 points to the 'Flask' link in the technologies list, 4 points to the first bullet point in the second main content area, and 5 points to the smiley face link in the footer.

2. Web Accessibility (2pts) What are 2 different ways we have learned for making a website more accessible?

3. Event Listeners (2pts) Consider the following JS program:

```
(function() {  
  window.addEventListener("load", init);  
  
  function init() {  
    id("my-btn").addEventListener("click", cat);  
    id("my-other-btn").addEventListener("click", doggo);  
    id("my-btn").addEventListener("click", frog());  
  }  
  
  function doggo() {  
    console.log("woof!");  
    id("my-other-btn").addEventListener("click", function() {  
      console.log("quack!");  
    });  
  }  
  
  function cat() {  
    console.log("meow!");  
    doggo();  
  }  
  
  function frog() {  
    console.log("ribbit!");  
    id("my-other-btn").removeEventListener("click", doggo);  
  }  
})();
```

Assuming the #my-btn and #my-other-btn buttons exist in the corresponding HTML, **circle** which of the four options would be correct console output when the page is loaded and the following user events occur:

1. A user clicks the #my-btn button
2. A user clicks the #my-other-btn button
3. A user clicks the #my-btn button

a	b	c	e
ribbit! meow! woof! quack! meow! woof!	meow! woof! ribbit! quack! meow! woof! ribbit!	ribbit! woof!	meow! woof! woof! quack! meow! woof!

4. JS Timers (2pts) Consider the following JS program:

```

(function () {
  let t1 = null;
  let t2 = null;
  let doggoCount = 0;

  window.addEventListener("load", init);

  function init() {
    t1 = setInterval(doggo, 300);
    t2 = setTimeout(dubs, 800);
  }

  function doggo() {
    doggoCount += 1;
    console.log(doggoCount + " doggo");
  }

  function dubs() {
    console.log("DUBS!");
    t1 = null;
    clearInterval(t1);
    t2 = setTimeout(dubs, 800);
  }

})();

```

Circle which of the following options would be correct as the first 8 lines of console output when the page is loaded (fewer than 8 lines indicate no more console output is possible until the program is restarted).

a	b	c	d	e
1 doggo 2 doggo DUBS! 3 doggo 4 doggo DUBS! 5 doggo 6 doggo	1 doggo 2 doggo 3 doggo DUBS!	1 doggo 2 doggo DUBS! 1 doggo 2 doggo DUBS! 1 doggo 2 doggo	1 doggo 2 doggo DUBS! 3 doggo 4 doggo 5 doggo DUBS! 6 doggo	1 doggo 2 doggo DUBS! DUBS! DUBS! DUBS! DUBS! DUBS!

5. Node.js Error-Handling (2pts)

a. Provide a specific example where it would be more appropriate to return a 400 error instead of a 503 error in a web service:

b. Provide a specific example where it would be more appropriate to return a 503 error instead of a 400 error in a web service:

6. GET vs. POST (2pts)

a. Provide an example where a GET request would be more appropriate than a POST request for a web service.

b. Provide an example where a POST request would be more appropriate than a GET request for a web service.

7. Validation Methods (2pts)

a. What is one advantage of validating user input on the client (HTML5 or JS) rather than on the server (Node)?

b. What is one advantage of validating user input on the server as opposed to on the client?

8. Data Storage Trade-Offs (2pts)

a. Recall that `localStorage` can be used to store data on a user's browser. In 1-2 sentences, explain when it is more appropriate to store data with `localStorage` instead of with a SQL database.

b. It is possible to store/process data on a server using `.txt` or `.json` files. What is one advantage of using SQL databases to store data instead?

9. Node File I/O (1.5pts) Suppose a directory has the following structure:

```

app.js
birds/
  eagle/
    call11.mp3
    image.png
    info.txt
  heron/
    call11.mp3
    call12.mp3
    image.png
    info.txt
  merlin/
    call11.mp3
    image.png
    info.txt

```

What are the `globPromise/fs.readdir` statements that can be used in `app.js` to get each **Resolved Promise Value**?

Statement	Resolved Promise Value
	<code>["eagle", "heron", "merlin"]</code>
	<code>["birds/merlin/image.png", "birds/merlin/info.txt"]</code>
	<code>["birds/eagle/call11.mp3", "birds/heron/call11.mp3", "birds/heron/call12.mp3", "birds/merlin/call11.mp3"]</code>

10. Regex (2pts) For each of the two regular expressions, circle all the string(s) below that match it:

i. `/^[irA]{3}b*[n]+b?$/`

- A3bnnnb
- rAin
- iiiBnB
- AAibnbb
- Airbnb

ii. `/^[A-Z]L(3){1,}..$/`

- AL3rt
- GL33..\$
- XL33TX
- LLLL
- WILL321

Regex reference:

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

Problems 2 and 3: Overview of "Welcome to Drumheller Fountain!"

Spring Quarter is almost over, which means two things: summer is almost here, and the Drumheller Fountain is *almost* back in business! The green-headed mallards, having missed their UW swimming hole for months, are ecstatic, as are the CSE 154 staff who look forward to having some fun with their rubber duckies after the quarter is over.

The UW staff has heard about all of the amazing work you've all done in CSE 154, and wants to hire you for a "virtual" Drumheller Fountain website to celebrate the re-opening. You've agreed to help, but only if it features rubber duckies and a CSE 154 Staff Party!

For Problem 2, you will write CSS for 3 different parts of the page, matching provided screenshots and specifications. **You will not write CSS for the other parts of the page (e.g. the #fountain) but we have provided an example screenshot of the full page for context .**

For Problem 3, you will write JS to implement functionality on the page (assuming all CSS is implemented correctly).

```
<body>
  <h1>Welcome to Drumheller Fountain!</h1>
  <div id="fountain">

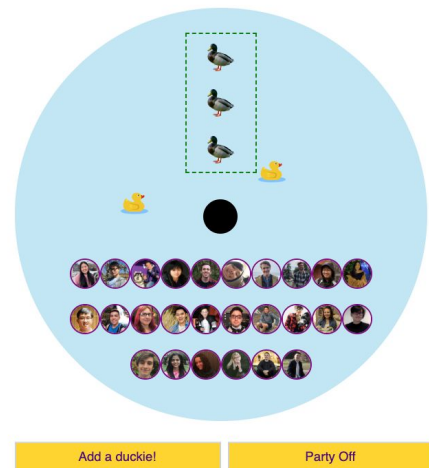
    <!-- Styled in Problem 2A -->
    <div id="mallard-container">
      
      
      
    </div>

    <div id="fountain-jet"></div>

    <!-- Styled in Problem 2B,
         Populated with images in Problem 3 -->
    <div id="party-container"></div>
  </div>

  <!-- Styled in Problem 2C -->
  <div id="button-container">
    <button id="add-duckie">Add a duckie!</button>
    <button id="party-mode">~Party On~</button>
  </div>
</body>
```

Welcome to Drumheller Fountain!



Full Drumheller Fountain Web Page with Party Mode "On" and two rubber duckies added

2. CSS Writing (13pts): Swimmin' in CSS

In this problem, you will implement the CSS for the following `fountain.html` page elements:

Part A: Container holding 3 mallard duck images

Part B: Container holding CSE 154 staff images

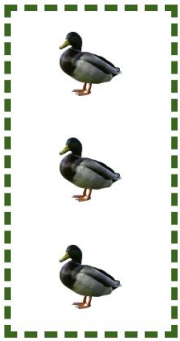
Part C: Container holding the `#add-duckie` and `#party-mode` buttons

You will **not** implement any CSS for the page `body`, the `#fountain`, or the `#fountain-jet`.

Part A: The Fountain Mallards

- The `#mallard-container` is 100px wide and 200px tall with a 4px dashed green border.
- The three mallard images are positioned in a column orientation within the container and are distributed using the `space-around` flex property on the appropriate element. These images are centered horizontally within the container.
- Each mallard image is 40px wide by 40px tall.
- For full credit, you must choose selectors in your answer to Part A so that the CSS would not affect other elements (e.g. other `div`s or `img`s) that will ever be on the page.

Expected output:



```
/* Write your CSS for Part A below */
```

Part B: Instructor Party!

In Problem 3 (JS), you will add a feature to the page to populate the `#party-container` with a `.party-pic` image for each instructor. Here, you will write the CSS assuming the `#party-container` holds images with the `.party-pic` class, as seen in the example screenshot below (omitting the blue `#fountain` background).

- The `#party-container` is 200px tall with a width 80% of its parent (the `#fountain`).
- This container's `.party-pic` images should be displayed in row orientation. Use flex layout out as appropriate to ensure elements wrap across multiple rows in order to fit the width of the container and also to center the images horizontally within the container as seen in the expected output.
- Each `.party-pic` element is 40px wide and 40px tall, is a circle, and has a 2px thick solid purple border.
- For full credit, you must choose selectors in your answer to Part B so that the CSS would not apply to any other elements (e.g. other `div`s or `img`s) that will ever be on the page.

Write your CSS below:

Expected output:




```
/* Write your CSS for Part B below */
```

Part C: Button Container

- Button text should be centered, with Verdana font (defaulting to sans-serif font if unavailable) and having indigo text color.
- The size of the button text is 14pt.
- Each button has a gold background, a width of 300px, a margin of 10px to the left, and have top and bottom padding of 10px.
- Use flex layout to center the two buttons horizontally within the #button-container (the #button-container spans the width of the page).
- For full credit, you must choose selectors in your answer to Part C so that the CSS would not apply to any other elements (e.g. divs) that will ever be on the page.

Write your CSS below:

Expected output (remember these are centered on the page in a #button-container that spans the width of the page):



```
/* Write your CSS for Part C below */
```

3. JS with DOM (15pts): Jump in with JS!

The UW staff is ecstatic at the progress you've made on the Drumheller Fountain website. Now it's time to add the interactivity for ducks and a fountain party!

In this problem, you will write JS to allow users to add rubber duckies randomly in the fountain and also to toggle a special "Party Mode" for the CSE 154 staff (because they've had a long quarter supporting the course and they deserve a break too!). You will not write any CSS in this Problem. As a reminder, the HTML and example screenshots for Problems 2 and 3 is provided in **Appendix A** at the back of this exam.

Behavior Details:

Adding Rubber Duckies

- Every time the `#add-duckie` button is clicked, you should create a new `div` that has the `.duckie` class and add it as a child element to the `#fountain`. Assume the provided CSS will give any `div` with the `.duckie` class a background image of `duckie.png` (a yellow duck) as shown in **Appendix A**.
- Use a provided `getDuckiePos()` function to help set the `top` and `left` properties to randomly position the new duckie element:
 - This function returns a 2-element array holding a random top coordinate and a random left coordinate for the duck (you do not need to do any calculation otherwise). The implementation details of this function are not important.
 - The fountain has a "relative" position style and each duckie has an "absolute" position style (you may refer to the partial CSS in **Appendix A**). These position properties will make it so the absolute position of the duckie is actually relative to the fountain. **You just need to set the `top/left` properties of each `.duckie` using this function's return values.**
 - For example, a returned array of `[0, 150]` would represent a duckie position 0 px from the top border of the fountain and 150 px from the left border of the fountain (remember to include "px" when setting top and left properties of a DOM element).

Instructor Party Mode!

- The `#party-mode` button toggles "party mode" on and off (initially there is no fountain party going on).
- Whenever this button is clicked and party mode is off (i.e. `#party-container` is empty):
 - You should use the provided `PICS` array of instructor image names to populate the `#party-container` with a single image for each instructor. Each of these images should have the class `.party-pic`. You may assume the pictures are in the same directory as `fountain.js`.
 - The `#party-mode` button text should be changed to "Party Off"
- Whenever this button is clicked and party mode is already on (i.e. `#party-container` contains images):
 - All images should be removed from the `#party-container` to "turn off" party mode.
 - The `#party-mode` button text should be changed to "~Party On~".

You will write your solution on the following page.

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

    window.addEventListener("load", init);

    // Images of all the CSE 154 instructors
    const PICS = ["ann.jpg", "chao.jpg", "conner.jpg", "daniel.jpg", "hawk.jpg", ... ];

    // Returns array with [randomTop, randomLeft] values, where each value is an
    // integer for a position for a .duckie to fit in the #fountain.
    function getDuckiePos() {
        // Details of function not provided here
    }

    // Implement the rest of Problem 3 on this page.
```

```
})();
```

4. Node.js Web Service (20pts): Club CSE 154!

The CSE 154 TAs are developing a brand new game called Club CSE 154 (not to be confused with Club Koala). It is filled with many notable references made throughout the Spring 2019 quarter. The team has compiled a bunch of folders filled with images and descriptions of each notable reference. In this question, you will *implement* the web service (`app.js`) that will eventually allow us to build a simple character showcase webpage with JS in Problem 5 (for that problem, you may assume `app.js` is implemented correctly).

API Documentation

The `app.js` API supports two GET requests, dependent on the value of a required `mode` parameter.

Query 1: Get all Characters

Request Format: `/club154?mode=all`

Request Type: GET

Returned Data Format: plain text

Description: This request outputs a plain text response with each character's directory on a new line.

Request: `/club154?mode=all`:

Output (complete): Below is the expected output when the data directory has the same 6 character directories as shown in Table 1 (Page 16).

```
corrin
debugduck
mowgli
piazza
pikachu
rainbowdash
```

Query 2: Get Data for a Single Character

Request Format: `/club154?mode=lookup&char=<charname>`

Request Type: GET

Returned Data Format: JSON

Description: If a `mode` of `lookup` is passed along with a second (required) GET parameter `char`, the service will output a JSON response with information about a specific character, where the `<charname>` value corresponds to a directory name that was returned from Query 1 (`mode=all`). For example, to get the information about the character "Debug Duck", you would use `debugduck` for the `<charname>` value.

The general format of the expected JSON response is as follows:

```
{
  name: <charname>,
  series: <seriesname>
  description: <description>
  appearances: [<appearance1>, <appearance2>, ...]
}
```

Where each `<string>` above is replaced with the corresponding information unique to the character. You may assume that the response will contain valid information and that there is at least one appearance. You may also assume that if the `char` parameter is passed, it corresponds to a valid character directory on the server.

An example request and response for Query 2 is provided on the next page.

Example Request: /club154?mode=lookup&char=debugduck

Example Response :

```
{
  "name": "Debug Duck",
  "series": "Programming Aid",
  "description": "Stuck on a programming project? Fear no more! A handy debug duck
    will be provided to aid to your most difficult problem.",
  "appearances": [
    "Course website homepage illustration",
    "4/19 Lecture, with everyone obtaining their own free Debug Ducks"
  ]
}
```

Error-handling

The `app.js` web service returns the following errors with 400 error codes (in order of highest priority to lowest priority). If any error occurs, only output the respective error message in plain text.

- If no `mode` is passed or has a value different than `all` or `lookup`, the error message output should be: "Please pass in a mode of all or lookup."
- If `mode=lookup` but no `char` parameter is passed, the error message output should be: "Please pass in a character name."
- if the `char` parameter is passed, but does not correspond to a valid character, the error message output should be "Please pass in a valid character name."

If anything else goes wrong, a 500 error code should be returned with the error message "Something went wrong on the server. Please try again later."

Web Service Implementation

Your web service will be implemented to support the two requests documented on Pages 14-15. You will write your solution to `app.js` in three parts using the directory structure described below.

Directory Structure and File Format Details

Your web service will be located in the same level as a `data` directory which contains subdirectories corresponding to each of the character names. Inside of each of these character directories are 3 files:

- An `avatar.png` (the image of the character).
- A text file `info.txt` with 3 lines of information about the character:


```
name
series
description
```
- An `appearances.txt` file that contains a line for each unique appearance of the character in CSE 154 19sp. For example, a file with 4 lines would correspond to 4 appearances during the quarter. This file is guaranteed to have at least one line.

In the example txt files below, we have included (in bold) the newline "`\n`" character (usually hidden) to make it clear where line breaks occur. For each character directory, you can assume that all files are present with the format specified.

<p>Directory Structure</p> <pre>app.js public/ club154.html club154.js data/ corrin/ appearances.txt avatar.png info.txt debugduck/ appearances.txt avatar.png info.txt mowgli/ appearances.txt avatar.png info.txt piazza/ appearances.txt avatar.png info.txt pikachu/ appearances.txt avatar.png info.txt rainbowdash/ appearances.txt avatar.png info.txt</pre>	<p>info.txt File Content Structure:</p> <pre><name info>\n <series info>\n <description>\n</pre> <p>Example File Contents: (data/debugduck/info.txt)</p> <pre>Debug Duck\n Programming Aid\n Stuck on a programming project? Fear no more! A handy debug duck will be provided to aid to your most difficult problem.\n</pre> <hr/> <p>appearances.txt Content Format</p> <pre><First appearance of the character in CSE 154 19sp>\n <Second appearance of the character in CSE 154 19sp>\n ... <Last appearance of the character in CSE 154 19sp>\n</pre> <p>Example File Contents: (data/debugduck/appearances.txt)</p> <pre>Course website homepage illustration\n 4/19 Lecture, with everyone obtaining their own free Debug Ducks\n</pre>
--	--

Table 1: Directory and File structure for Club CSE 154 website

Part A:

Implement a function `get_chars` to return a string concatenating each directory name in the `data/` directory. Each directory name should be appended with a newline ("`\n`") character at the end. For example, if the `data` directory contains the same 6 folders as shown in Table 1 on the previous page, the function should return the string: "`corrin\ndebugduck\nmowgli\npiazza\npikachu\nrainbowdash\n`".

NOTE: Do NOT set any headers in this function. Any headers and output will be implemented in Part C.

```
const fs = require("fs").promises;
```

```
async function getChars() {
```

```
}
```

Problem 4 Continued with Part B on the next page.

Part B: Building JSON for Query 2

Implement a function `getCharInfo(chardir)` to take a character directory name (e.g. "debugduck") as a parameter and return a **JS object** having the form:

```
{
  "name": <name of the character>,
  "series": <character series>,
  "description": <description of character>,
  "appearances": <array of appearances>
}
```

containing the character information described in the API Documentation for Query 2 (Page 14).

For example, this function would return the following array for the character corrin:

```
{
  "name": "Corrin",
  "series": "Super Smash Bros. & Fire Emblem",
  "description": "Corrin carries a Yato sword wherever she goes. If any danger is
                 present, she can transform herself into a dragon to attack!",
  "appearances": [ "April Fools Course Website", "Week 6 Exploration Session" ]
}
```

Remember that you may assume `chardir` corresponds to a directory in the `data` directory.

```
const fs = require("fs").promises;
```

```
async function getCharInfo(chardir) {
```

```
}
```

Part C: Completing the app.js Web Service

In this part, you will write the rest of the `app.js` web service to handle Query 1, Query 2, and the invalid request error handling specified in the API Documentation on pages 14-15. Assuming Part A and Part B functions are provided, your solution for Part C should be a complete `app.js` web service following the API Documentation. We have summarized the specifications below for convenience:

Query 1: `/club154?mode=all` - output a plain text response with a line for each character directory available on the `app.js` web service. Use your function from Part A to help handle this request.

Query 2: `/club154?mode=lookup&char=<charname>` - when a character directory name (one of those retrieved from Query 1) is passed, output a JSON-encoded response as specified in the Appendix C for Query 2. Use your function from Part B to help handle this request. While you may not assume the `char` parameter is passed, you may assume that if it is passed, the `<charname>` value will correspond to an existing directory name in the `data/` directory.

400 Invalid Request Error-Handling:

- If no `mode` is passed or has a value different than `all` or `lookup`, the error message output should be: "Please pass in a mode of all or lookup."
- If `mode=lookup` but no `char` parameter is passed, the error message output should be: "Please pass in a character name."
- if the `char` parameter is passed, but does not correspond to a valid character, the error message output should be "Please pass in a valid character name."

If anything else goes wrong, a 500 error code should be returned with the error message "Something went wrong on the server. Please try again later."

```
const express = require("express");
const fs = require("fs").promises;
const app = express();

app.get("/club154", async function (req, res) {
  # Write your code here
```

Extra space provided on the next page

You may continue your answer for any of Part of Problem 4 on this page (clearly labeled)

});

5. JS with Fetch (15pts): Gotta Fetch 'em All

Now that the `app.js` API is finished (and you may assume it is implemented correctly), the CSE 154 TAs are looking for someone to implement the “Choose your Character” webpage to showcase each featured CSE 154 character in a “character spotlight”. You are hired to do just that. Welcome to the team!

The TAs have designed two sample screenshots and a specification that you are to work from, detailed below.

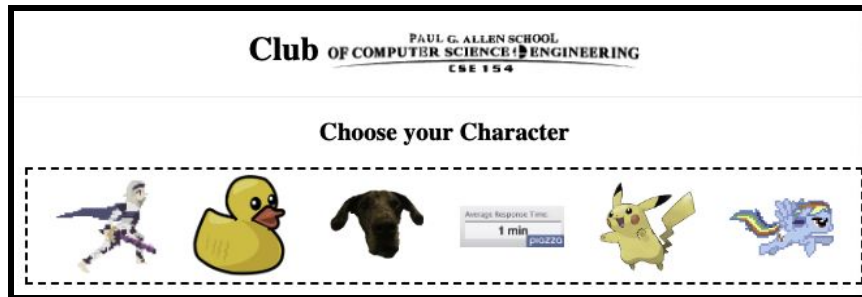


Figure 1: When page is loaded (after the character images have been added to the page)

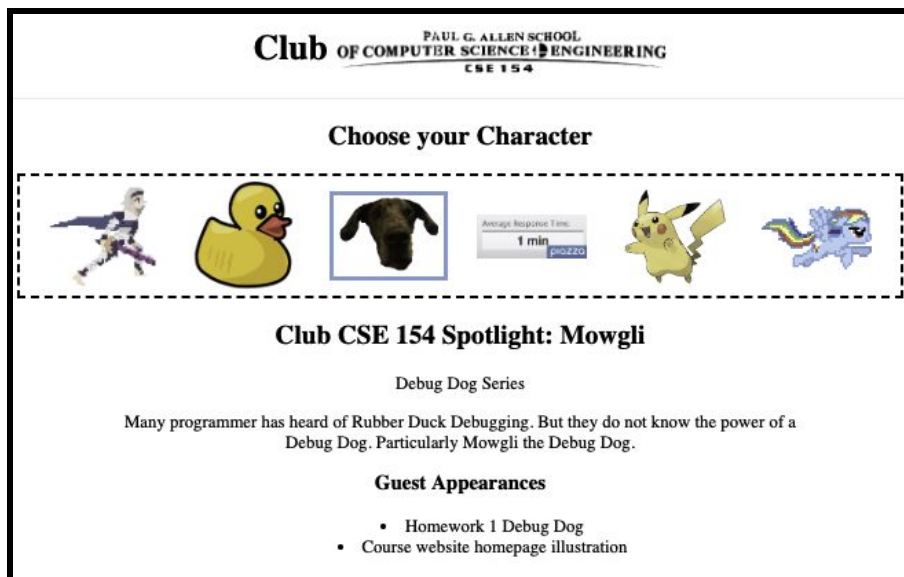


Figure 2: After a user selects a character image (e.g. the dog Mowgli)

Behavior Details

The endpoint your JS will fetch data from is `"/club154"` (the same web service you implemented in Problem 4). Assume that `app.js` is correctly configured to serve static files from the public directory

Details about the `"/club154"` endpoint are provided in Problem 4 on pages 14 - 15.

The body of the provided `club154.html` file is provided below for reference:

```
<body>
  <h1>Club </h1>
  <hr>
  <h2>Choose your Character</h2>

  <div id="playground"></div>

  <section id="spotlight" class="hidden">
    <h2>Club CSE 154 Spotlight: <span id="name"></span></h2>
    <p><span id="series"></span> Series</p>
    <p id="description"></p>
    <h3>Guest Appearances</h3>
    <ul id="appearances"></ul>
  </section>
  <p id="error" class="hidden">Club CSE 154 is currently out of service :(</p>
</body>
```

Initial View

When the page loads:

- A request should be made to the `"/club154"` endpoint of `app.js` using Query 1 (`mode=all`). Recall that this request returns a plain text response listing all character directory names, each as a single line.
- For each line returned in a successful request, an image element should be added to the `#playground` with the path `data/<chardir>/avatar.png`, replacing `<chardir>` with the line text (e.g. `debugduck`). The resulting page will appear as seen in Figure 1.
- Each image should be given an id corresponding to the directory name for use in Query 2 requests later.

Character Spotlight

When one of the added character images is clicked:

- Data about that character should be requested from the `"/club154"` endpoint of `app.js` (using Query 2, `mode=lookup&char=<charname>`). The response JSON should be used to populate the `#spotlight` view:
 - `#name` should be populated with the returned name
 - `#series` should be populated with the returned series name
 - `#description` should be populated with the returned character description
 - `#appearances` should be populated with a list of appearances, with one list item for each appearance in the returned appearances array.
- The clicked image should be given the `.selected` class (assume provided in CSS), which includes the styling for the blue border. Figure 2 shows an example of what the page would look like after Mowgli's (the dog) image is clicked.
- `#spotlight` should be made visible if not visible already. The `.hidden` class, which you may assume is provided in the CSS to hide/show elements, is described in more detail in the **Error-Handling** section and will have the same behavior as in HW3/HW4.
- At most one character image should have the `.selected` at any given time. If an image with `.selected` is clicked, nothing should change on the page (until a different image is clicked).
- Note that `#appearances` should only ever contain appearances for the current spotlight character.

Error-Handling

If an error occurs in any fetch request:

- The `#error` element should be displayed and the `#spotlight` view should be hidden.
- No element should have the `.selected` class.

- If an image is clicked again and the request is successful, `#error` should be hidden and `#spotlight` should display again, resuming the behavior as described in **Character Spotlight** section above.

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

```
(function() {  
  "use strict";  
  const URL = "/club154";  
  
  window.addEventListener("load", init);  
  
}) ();
```


6. SQL and Node.js (12pts): Message in a Bottle

In this problem, you will work with a table called `posts` in the database `messageboarddb`. This table contains information about questions students have posted to a message board and the category for each question. Each row has a unique `id` that is an INTEGER - the rest of the columns are VARCHAR types. Below are some example rows in the `posts` table (where ... indicates more rows not shown):

id	name	question	category
404	John Doe	I can't find the exams page on the course website??	website
405	Terra Billpun	my code	creative project
406	Terra Billpun	Sorry the last post was wrong, here's my code.	creative project
407	Steve W.	How do you pronounce "gif"?	other
408	Sharon Tumuch	This is my code for HW5. Why doesn't it work?	homework
...

Part A: Basic SQL Queries/Statements

i. Write a SQL query that lists the names of all students, with no duplicates, who have posted questions containing the phrase "my code". The names of the students should be sorted alphabetically in ascending order.

Expected results:

name
Sharon Tumuch
Terra Billpun
...

Write your SQL query below:

ii. Write a SQL statement that inserts a new row into the table so that the row would look like the following:

id	name	question	category
541	Piaz Za Rocks	Will this be tested?	other

Write your SQL Statement here:

iii. Write a SQL statement that would delete all rows with questions containing the phrase "code does not work".

Write your SQL Statement here:

Part B: Node.js with SQL using mysql

For the following `/update` endpoint:

i. Briefly explain why this endpoint is vulnerable to malicious clients.

ii. Identify what change(s) you would need to make it more secure using what we've covered in lecture, crossing out and clearly modifying the endpoint where necessary to make the changes while still correctly updating the row in the `posts` table.

```
const db = mysql.createPool(cnctInfoObj);

app.post("/update", async function (req, res) {
  // assume id and newQuestion exist
  let id = req.body.id;
  let newQuestion = req.body.newQuestion;

  let str = "UPDATE posts SET question = " + newQuestion + "WHERE id = " + id;

  try {

    await db.query(str);

    res.type("text").send("Updated the database!");

  } catch (err) {

    res.type("text");
    res.status(500).send("Something went wrong on the server. Please try later.");

  }
});
```

This page intentionally left blank

7. 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.

Appendix A: Drumheller Fountain HTML <body> and Screenshots (for Problems 2 & 3)

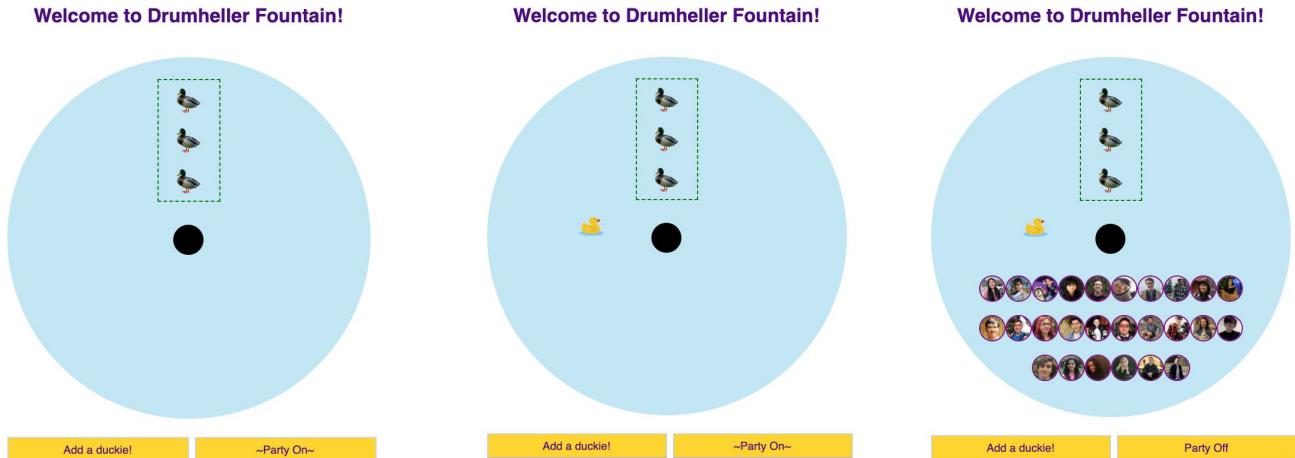


Figure 1: Starting the simulation

Figure 2: After clicking the "Add a duckie!" button

Figure 3: After clicking the "~Party On~" button

```
<body>
  <h1>Welcome to Drumheller Fountain!</h1>
  <div id="fountain">
    <div id="mallard-container">
      
      
      
    </div>
    <div id="fountain-jet"></div>
    <div id="party-container"></div>
  </div>
  <div id="button-container">
    <button id="add-duckie">Add a duckie!</button>
    <button id="party-mode">~Party On~</button>
  </div>
</body>
```

Reference CSS for .duckie Positioning (assume provided):

```
#fountain {
  position: relative;
  /* Other CSS unimportant */
}

.duckie {
  position: absolute;
  background-image: url("duckie.png");
  /* Other CSS unimportant */
}
```

