Lab 8

Re-submit Assignment

Due Saturday by 11:55pm **Points** 100 **Submitting** a file upload **File Types** zip

CS-546 Lab 8

Palindromes: Part 1

For this lab, you will be using HTML, CSS, and Handlebars to make your first simple form! Your form will make a **palindrome checker!**

A palindrome is a phrase that is spelled the same way, backwards and forwards (ignoring spacing and punctuation; only alphanumeric characters matter). For example, the following phrases are palindromes:

- Madam
- Was it a cat I saw?
- He did, eh?
- Go hang a salami, I'm a lasagna hog.
- Poor Dan is in a droop
- Taco cat? Taco cat.

You will create an express server with two pages: `/` and `/result`; you will also have one static asset, `/public/site.css`.

`http://localhost:3000/`

This page will respond with a valid HTML document. The title of the document should be "*The Best Palindrome Checker in the World!*". You should have the title set as the `<title>` element of the HTML document and as an `h1` in your document.

Your page should reference a CSS file, `/public/site.css`; this file should have *at least 5 rulesets* that apply to this page; these 5 rules can also apply to elements on `/result`, or be unique to this page.

You should have a `main` element, and inside of the `main` element have a `p` element with a brief (2-3 sentence description) of what a palindrome is, and what your website does.

Also inside the `main` element, you will have a `form`; this `form` will `POST` to `/result`. This `form` will have an `input` and a `label`; the `label` should properly reference the same `id` as the `input`. You should also have a `button` with a type of `submit` that submits the form. The `input` in your `form` should have a `name` of `text-to-test`. When your `POST` this string to `/result`, it will be checked to see if it is a palindrome or not.

`http://localhost:3000/result`

This page will respond with a valid HTML document. The title of the document should be "*The Palindrome Results!*". You should have the title set as the `<title>` element of the HTML document and as an `h1` in your document.

Your page should reference a CSS file, `/public/site.css`; this file should have *at least 5 rulesets*that apply to this page; these 5 rules can also apply to elements on `/`, or be unique to this page. **You must have 2 classes: `success` and `failure` that are used to color the message below either `#28a745` and `#dc3545` respectively.

You should have a `main` element, and inside of the `main` element have a `p` tag that has the `text-to-test` in it. **If the text is a palindrome, this `p` tag will have a class of `success`; if it is not, it will have a class of `failure`. After the `p` tag, you will have another `p` tag that states whether or not the string was a palindrome.

You must also provide an `a` tag that links back to your `/` route with the text `Try another palindrome`.

If the user does not input text into their form, make sure to give a response status code of 400 on the page, and render an HTML page with a paragraph class called `error`; this paragraph should describe the error.

`http://localhost:3000/public/site.css`

This file should have 5 rulesets that apply to the `/` route, and 5 rulesets that apply to `/result`. Rulesets may be shared across both pages; for example, if you styled a `p` tag, it would count as 1 of the 5 for both pages.

You may include more than 5 rulesets if you so desire.

References and Packages

Basic CSS info can easily be referenced in the **MDN CSS tutorial (https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Getting_started)**.

Requirements

1. You **must not submit** your node_modules folder
2. You **must remember** to save your dependencies to your package.json folder
3. You must do basic error checking in each function
    1. Check for arguments existing and of proper type.
    2. Throw if anything is out of bounds (ie, trying to perform an incalculable math operation or accessing data that does not exist)
    3. If a function should return a promise, instead of throwing you should return a rejected promise.
4. You **must remember** to update your package.json file to set `app.js` as your starting script!
5. **Your HTML must be valid (https://validator.w3.org/#validate_by_input)** or you will lose points on the assignment.
6. Your HTML must make semantical sense; usage of tags for the purpose of simply changing the style of elements (such as `i`, `b`, `font`, `center`, etc) will result in points being deducted; think in terms of content first, then style with your CSS.
7. **You can be as creative as you'd like to fulfill front-end requirements**; if an implementation is not explicitly stated, however you go about it is fine (provided the HTML is valid and semantical). Design is not a factor in this course.
8. All inputs must be properly labeled!
9. All previous requirements about the `package.json` author, start task, dependenices, etc. still apply