Lab 7

Due Nov 7, 2016 by 5pm **Points** 100 **Submitting** a file upload

CS-546 Lab 7

Text Manipulation, Two Ways

For this lab, you will be manipulating text two ways.

- Sending a form to the server to render the result on a new page
- Intercepting a form on the client to render the result without leaving the page.

You will need to implement the same algorithm on the server in a module, and on the client in a script file.

You will make a form with 4 inputs:

1. A textarea that you will put a moderate amount of text in.
2. An input that will take a string; this string will be inserted into the text from the textarea.
3. An input that will take a number; this will be the number of times the string will be inserted
4. A second input that take a number; this will be the number of characters between each insert.

For example, with the following input:

1. Lorem ipsum dolor sit amet, consectetur adipiscing elit. In luctus augue urna. Nam in turpis sapien. Pellentesque vehicula augue quis vehicula egestas. Phasellus non iaculis justo, eget cursus purus. Ut id ante vel elit maximus ullamcorper a pretium erat. Nullam pharetra rutrum velit, quis commodo felis gravida a. Aliquam justo dolor, blandit sed turpis ultrices, tempus aliquam eros. Nulla sollicitudin, lorem a mattis tincidunt, ligula mi cursus nisi, a laoreet metus erat non libero.
2. HELLOHELLO
3. 5
4. 7

You will get the output:

> Lorem iHELLOHELLOpsum doHELLOHELLOlor sitHELLOHELLO amet, HELLOHELLOconsectHELLOHELLOetur adipiscing elit. In luctus augue urna. Nam in turpis sapien. Pellentesque vehicula augue quis vehicula egestas. Phasellus non iaculis justo, eget cursus purus. Ut id ante vel elit maximus ullamcorper a pretium erat. Nullam pharetra rutrum velit, quis commodo felis gravida a. Aliquam justo dolor, blandit sed turpis ultrices, tempus aliquam eros. Nulla sollicitudin, lorem a mattis tincidunt, ligula mi cursus nisi, a laoreet metus erat non libero.

Objectives

1. You will make three routes:
    1. `GET /clientform`
    2. `GET /serverform`
    3. `POST /serverform`

2. Both `GET` routes will provide a valid HTML document; each will have the form detailed above
3. `GET /clientform` will compute and render the results on the client
4. `GET /serverform` will `POST` the form to `POST /serverform`
5. Each will perform error checking, show errors, and show results

`GET /clientform`

Using client side JavaScript (which will be included via a script file, not on the HTML page itself), you will:

Validate:

- That the textarea has content
- That the insert string has content
- That both numbers are greater than or equal to 1 and less than or equal to 25.

When the user submits the form, you will capture the submission event and perform those validations.

- If there is an error, you will display an error message on an HTML element on the page; **this element will only be visible when an error occurs; it will be hidden via a CSS class when there is no error, and that class will be removed when there is an error**
- If there is a successful computation, you will display the output in an HTML element on the page; **this element will only be visible on successful computation; it will be hidden via a CSS class when there is no result, and that class will be removed when there is a result**

You will **not** be submitting this form to the server.

`GET /serverform`

This route will also show the form. It will `POST` the data `/serverform` where the error-checking will occur.

The `POST /serverform` route will check for errors. If there are any errors, it will create a model and pass them to the same template that shows the form originally. **Any data they filled out correctly will also populate the textarea or inputs**.

If there are no errors, `POST /serverform` will create a model with all the inputs that were filled out, as well as the computed result in a new div. Visually, it should act the same way as `/clientform` with the exception that rather than using classes to show elements, you will only conditionally render elements for errors / successful results in each case.

Your computations should occur in a module, not in your route file. You should follow the organization patterns from previous lectures and labs.

You must save all dependencies to your package.json file

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)**. If you need a quick CSS referfence,

You will use the **express-handlebars** package as your templating engine.

You can reference the **express-handlebars repository** (https://github.com/ericf/express-handlebars/) for details on adding the module; you may also want to check out the **handlebars website** (http://handlebarsjs.com/) for details.

You will use the **express** package as your server.

You can read up on **express** (http://expressjs.com/) on its home page. Specifically, you may find the **API Guide section on requests** (http://expressjs.com/en/4x/api.html#req) useful.

You may use the **lecture 4 code** (https://github.com/Stevens-CS546/CS-546-WS-Summer-1/tree/master/Lecture%20Code/lecture_4) as a guide.

You may use the **lecture 5 code** (https://github.com/Stevens-CS546/CS-546-WS-Summer-1/tree/master/Lecture%20Code/lecture_5) as a guide.

You may use the **lecture 6 code** (https://github.com/Stevens-CS546/CS-546-WS-Summer-1/tree/master/Lecture%20Code/lecture_6) as a guide.

You may use the **lecture 8 code** (https://github.com/Stevens-CS546/CS-546-WS-Summer-1/tree/master/Lecture%20Code/lecture_8) as a guide.

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. **Your client side JavaScript must be in its own file and referenced from the HTML accordingly**.