CSE186 Summer 2023

Assignment 3

In this assignment you will write JavaScript classes to modify the content of HTML pages via the DOM. You will use knowledge of CSS, HTML, and JavaScript acquired in the previous two assignments and should gain new skills in the process.

This assignment is worth 10% of your final grade.

Installation

None needed if you followed the instructions in Assignment 2.

Setup

Download the starter code archive from Canvas and expand into an empty folder. I recommend, if you have not already done so, creating a folder for the class and individual folders beneath that for each assignment.

The starter code archive contains the following files that you will modify:

```
templater.html
Templater.js
picker.html
picker.css
Picker.js
picker.test.js
```

And the following files that you should ***not*** modify:

```
.eslintrc.js
package.json
templater.css
templater.test.js
```

To setup the node environment, ***navigate to the folder where you extracted the starter code*** and run the following command:

```
$ npm install
```

This will take some time to execute as `npm` downloads and installs all the packages required to build and test the assignment.

To execute all tests, run the following command:

```
$ npm run test
```

To check the code quality against Google standards by running `eslint`, run the following command:

```
$ npm run lint
```

Requirements

Basic:

- Implement a JavaScript class with the ability to set the text content of HTML table elements with values supplied as JSON in the following circumstances:
  - Where existing element text contents are `{{ }}` tagged
  - Where elements have an id attribute

  For example, in the first case, if an element is written as:

  ```
  <td>{{R11}}</td>
  ```

  And the JSON supplied is:

  ```
  {"R11":"Hello World"}
  ```

  The document should be modified such that it would have been written as:

  ```
  <td>Hello World</td>
  ```

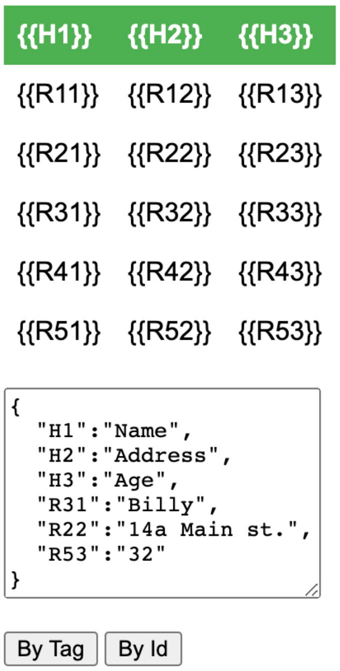
  Similarly, in the second case, if an element is written as:

  ```
  <td id="R11">Some Text</td>
  ```

  And the same JSON is supplied, the document should be modified in the same way.

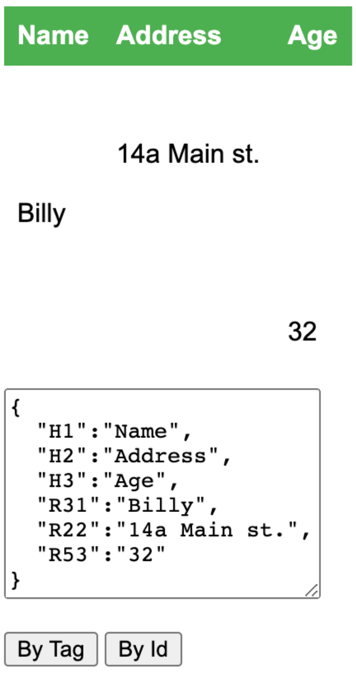
- To achieve this you will need to modify the `byTag()` and `byId()` methods on the `Templater` class in `Templater.js` and modify `template.html` to make use of your implementation ***taking great care not to modify the id's of existing HTML elements***.

  Regardless of which button is clicked in `templater.html`, when your implementation is complete the result should be as shown on the right assuming the illustrated JSON is entered in the text field:



{{H1}}	{{H2}}	{{H3}}
{{R11}}	{{R12}}	{{R13}}
{{R21}}	{{R22}}	{{R23}}
{{R31}}	{{R32}}	{{R33}}
{{R41}}	{{R42}}	{{R43}}
{{R51}}	{{R52}}	{{R53}}

```
{
  "H1":"Name",
  "H2":"Address",
  "H3":"Age",
  "R31":"Billy",
  "R22":"14a Main st.",
  "R53":"32"
}
```



Name	Address	Age
	14a Main st.	
Billy		
		32

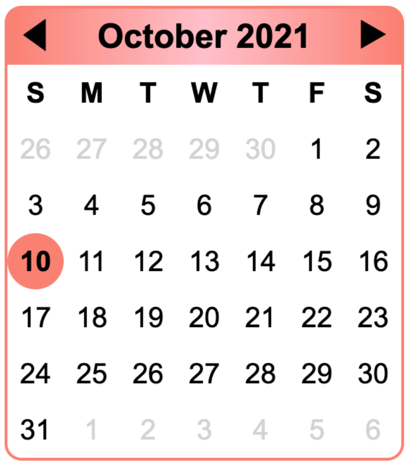
```
{
  "H1":"Name",
  "H2":"Address",
  "H3":"Age",
  "R31":"Billy",
  "R22":"14a Main st.",
  "R53":"32"
}
```

- You can test by hand using a browser, but make sure you also pass the automated tests by running:

  ```
  $ npm run test templater
  ```

Advanced:

- Implement a date Picker that looks something like this:



◀ October 2021 ▶

S	M	T	W	T	F	S
26	27	28	29	30	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

  Your Picker can look anyway you like, but the following features must be implemented:

  - Today's date should be shown initially
  - The month and year must be shown
  - The day of the month of the "current date" must be highlighted in some way
  - There must be "next" and "previous" navigation buttons to change the month
  - Days representing the end of the previous month and start of the next month must be shown "greyed out" to indicate they're not part of the displayed month

- In addition, you Picker must have the following features:

  - Clicking on a date sets it as the current date
  - Clicking on a date in the previous month or next month shows that month with the date clicked is show as the current date
  - Clicking in the date and year display takes the picker back to the month and year of the current date and highlights the appropriate cell

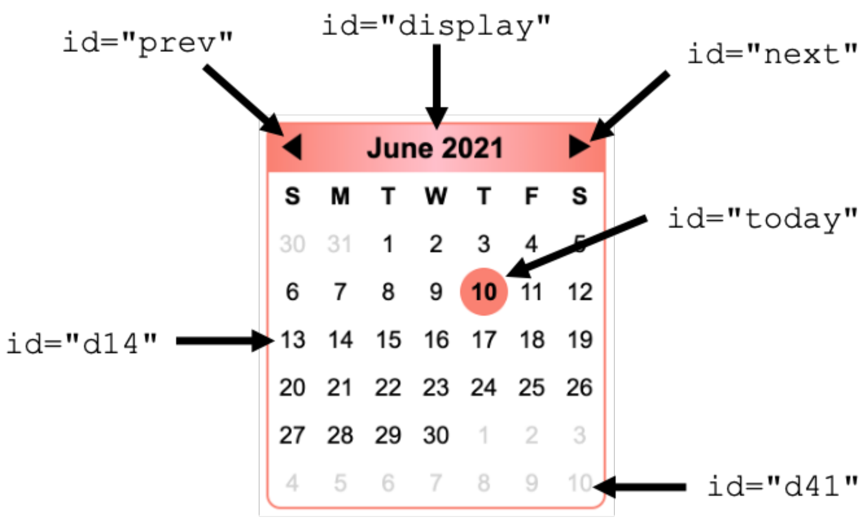
- Place an instance of your Picker inside a container `<div id="picker"/>` in `picker.html`. You will pass the id of this container to your Picker constructor.

- Regardless of how you implement your Picker, the generated document must have HTML elements with the following ids and behaviors:

  - `next` advances the date to the next month
  - `prev` retards the date to the previous month
  - `display` showing the current month and year
  - `today` showing the currently selected date if it is in in the displayed month
  - 42 x `dn` where `n` is in the range 0 to 41 each representing one day

  (continued on next page)

For example:



- You can test by hand using a browser, but make sure you also pass the supplied tests by running:

```
$ npm run test picker
```

- You will also want to write additional tests, in which case you should study the existing ones and research the Google Puppeteer framework which gives you the ability to manipulate a Chromium browser instance from your tests: https://developers.google.com/web/tools/puppeteer

What steps should I take to tackle this?

First, take a deep breath. These requirements are not as hard as they might initially appear.

The first part of Basic is like Assignment 2 with the addition on JSON parsing and the ability to iterate over DOM nodes of certain types - both are topics we covered in class. The second part of Basic merely requires that you get DOM nodes by their id.

For Advanced, work on the functionality first and worry about the appearance second. Make sure you rigorously ***separate content from style*** so you can change the look and layout simply by modifying your style sheet. A simple implementation for Advanced might look something like this:

<
June 2021
>
30 31 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3 4 5 6 7 8 9 10

And from there you can style the heck out of it to make it look nice AND professional.

How much code will I need to write?

A model solution that satisfies all requirements adds approximately 30 lines of code to `Templater.js`, and around 160 lines to `Picker.js`. You will also need to modify `templater.html`, `picker.html`, and `picker.css` plus related tests.

Grading scheme

The following aspects will be assessed:

1. (100%) **Does it work?**

   a. Basic Requirement (50%)
   b. Advanced Requirement (50%)

2. (-20%) **Is it reasonably well written?**

   a. Deduction of 20% for any linter errors or warnings (-20%)

3. (-100%) **Did you give credit where credit is due?**

   a. Your submission is found to contain code segments copied from on-line resources and you failed to give clear and unambiguous credit to the original author(s) in your source code (-100%). You will also be subject to the university academic misconduct procedure as stated in the class academic integrity policy.

   b. Your submission is determined to be a copy of a past or present student's submission (-100%)

   c. Your submission is found to contain code segments copied from on-line resources that you did give a clear an unambiguous credit to in your source code, but the copied code constitutes too significant a percentage of your submission:

      - < 25% copied code No deduction
      - 25% to 50% copied code (-50%)
      - > 50% copied code (-100%)

What to submit

Run the following command to create the submission archive:

```
$ npm run zip
```

**** UPLOAD** `CSE186.Assignment3.Submission.zip` **TO THE CANVAS ASSIGNMENT AND SUBMIT ****