CSE183 Fall 2020 - Assignment 5

In this assignment you will write a simple React Web App using JSX classes and tests to demonstrate they work as expected. You will use skills gained in previous assignments and learn new ones, particularly test automation using Jest and Puppeteer.

This assignment is worth 7% of your final grade.

Submissions are due NO LATER than 23:59, Thursday November 5 2020 (1 week)

Installation

See instructions in Assignment 3 and ensure you are running the current LTS version of Node.js.

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:

```
src/App.js
```

The following files that you can modify if you like:

```
test/App.test.js
test/index.test.js
```

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

```
public/favicon.ico
public/index.html
src/index.js
package.json
.eslintrc.js
```

To setup the development 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 the tests, run the following command:

```
$ npm test
```

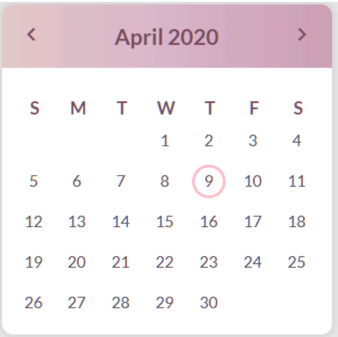
To check the code quality against Google and React standards by running `eslint`, run the following command:

```
$ npm run lint
```


Requirements

Basic:

- Implement a React JSX date Picker that might look something like this:



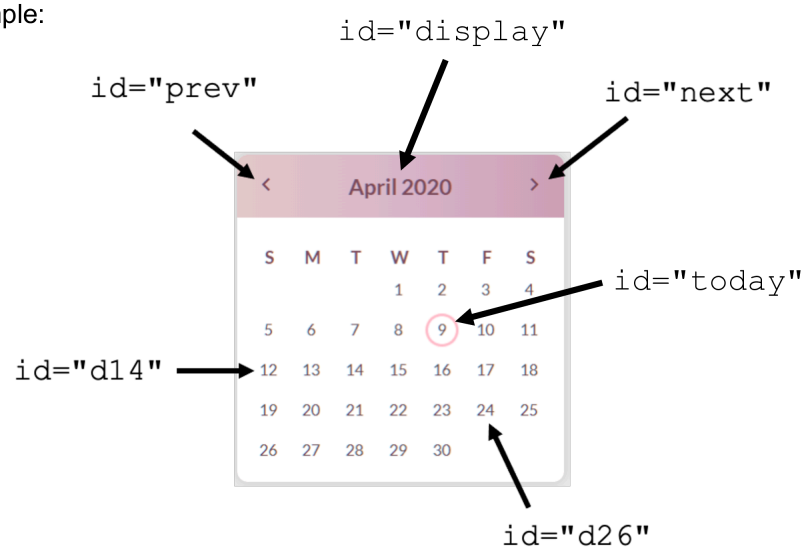
  Your JSX Picker can actually look anyway you like, but the following features must be implemented:

  - The current date should be shown initially
  - The month and year must be shown in text
  - The day of the month must be highlighted in some way
  - There must be “next” and “previous” navigation buttons to change the month
  - Days where the end of the previous and start of the next month would be must be blank

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

  - `display` the current month and year in long format, “October 2020” for example.
  - `today` the current day (if showing a month other than current, this id should not exist)
  - `next` advances the date to the next month
  - `prev` retards the date to the previous month
  - 42 x `dn` where `n` is in the range 0 to 41 each representing one day

  For example:



- Modify `src/App.js` to include an instance of your JSX Picker inside a with an id of `'picker'`.

- **Not that you MUST implement your new Picker in JSX and you MUST NOT use any third party libraries. You may only use built-in React and JSX Features.**

Advanced:

- Write Jest / Puppeteer tests in a new file `test/Advanced.test.js` that confirms your JSX Picker works as required by the Basic Requirement. The tests should be executed when the following command is issued:

  ```
  $ npm test Advanced
  ```

  Consult the Google Puppeteer documentation https://developers.google.com/web/tools/puppeteer and do your own research to discover how to use this powerful testing tool.

  Don't forget to give credit to on-line code resources you use in your solution.

- Tests you should consider writing for the default date include (with expected results based in the above example):

  - `display` is correctly set ("April 20")
  - `today` is correctly set ("16")
  - The first day of the month is in the correct element (`d3`)
  - Elements before the first day of the month are blank (`d0`, `d1`, `d2`)
  - Elements after the last day of the month are blank (`d33` through `d41`)

- Then consider tests where `prev` or `next` are clicked a number of times and repeat the above tests for what you expect the new date to be and add the following test:

  - After clicking `next` once, there should be no element with id `today`
  - After clicking `next` once, then `prev` once, there should be an element with id `today`
  - After clicking `next` once, then `prev` twice, there should be no element with id `today`

  And many others.

- The learning from form this requirement is that even for an apparently trivial component, there are many tests that have to be written to prove it works as expected. In any non-trivial web application there will be far more code in the tests than there is in the app itself.

Stretch:

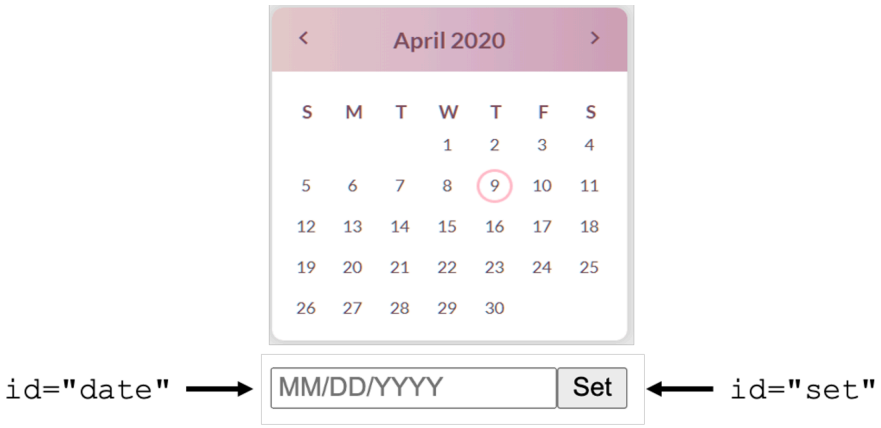
- Modify your JSX Picker class to include a `setDate(date)` method where `date` is a JavaScript `Date` object. When this new method is invoked, your JSX Picker should display the new date, regarding it as the current date until `setDate()` is invoked again.

- Modify `src/App.js` so that in addition to the JSX Picker from the Basic requirement there is a single line text input field into which a date in 'MM/DD/YYYY' format can be typed, and a button that when pressed will invoke `setDate()` on your JSX Picker passing a `Date` object created from the text in the input field.

  The following restrictions apply:

  - The text entry field should have an id of `'date'`
  - The text entry field should have an HTML placeholder of "MM/DD/YYYY"
  - The button should have an id of `'set'`
  - The button should only be clickable when a valid date is in the text entry field

  The resulting Web App might look something like this:



- Write Jest / Puppeteer tests in a new file `test/Stretch.test.js` that confirms your JSX Picker works as required by this Requirement. The tests should be executed when the following command is issued:

  ```
  $ npm test Stretch
  ```

  Take care to include the case where an invalid date has been entered in the text input field; the “Set” button should not be clickable.

What steps should I take to tackle this?

The JSX Picker should act in the same was as your Picker from Assignment 4. The logic is the same but the way you construct this new class will be somewhat different as you must use JSX.

If you use the power of JSX scripting, you should be able to achieve all the requirements with very few lines of code. If you find yourself writing page after page of JSX and JavaScript, you’ve probably taken a wrong turn.

Certainty you should do you own research but also consult the lecture handouts and the example React & JSX applications distributed after lectures 10 and 11.

Finally, don’t wait for Secret Sauce posts! For each requirement, write out a plan of action as code comments then go back and implement each part of the plan, testing as you go.

Lint-as-you-type

You’ll notice that the assignment has been configured so your React Web App will not run if your code fails any of the linter checks. Your code will therefore need to be super clean and tidy for you to able to manually test it ☺

How much code will I need to write?

A well-constructed solution satisfying all requirements would introduce up to 200 new lines of code and possibly as many as twice that for the additional Advanced and Stretch Requirement tests.

Grading scheme

The following aspects will be assessed:

1. (85%) **Does it work?**

    a. Basic Requirement (3 points)
    b. Advanced Requirement (2 points)
    c. Stretch Requirement (1 point)

2. (15%) **Does it look good?**

    a. Basic Requirement (1 point)

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:

        - < 33% copied code No deduction
        - 33% to 66% copied code (-50%)
        - > 66% copied code (-100%)

What to submit

Delete the `node_modules` folder then make a Zip archive of **ALL** your files and folders such that `package.json` is in the base directory of the Zip.

Your submission Zip should look something like this if you did every requirement:

```
src
      Picker.js          ( new )
      Picker.css         ( new )
      App.js             ( modified )
      index.js
test
      Advanced.test.js   ( new )
      Stretch.test.js    ( new )
      App.test.js
      index.test.js
public
      favicon.ico
      index.html
package.json
```

****** UPLOAD THIS ZIP ARCHIVE TO THE CANVAS ASSIGNMENT AND SUBMIT ******

