CSE186 Summer 2024

Assignment 3

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.

This assignment is worth 10% of your final grade.

Installation

See instructions in Assignment 1 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 you will modify:

```
src/App.css
src/Entry.jsx
src/Entry.css
src/Picker.jsx
src/Picker.css
src/__tests__/Advanced.test.js
src/__tests__/Stretch.test.js
```

Do not modify any other file.

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. You can ignore any errors and warnings.

To execute all tests, run the following command:

```
$ npm test
```

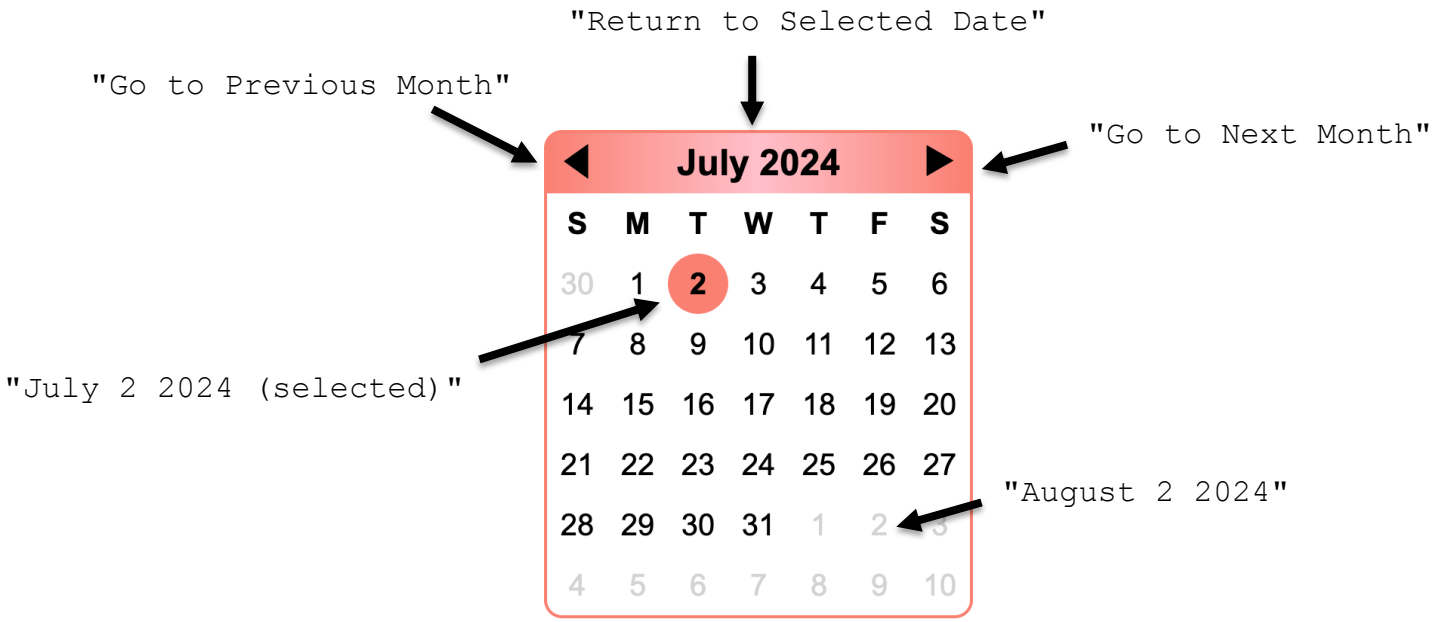
To run the Web App and reload every time your code changes, run the following command:

```
$ npm run dev
```


Requirements

Basic:

- Implement a React JSX date Picker in `src/Picker.jsx` with identical functionality to the pure JS version from Assignment 2.
- **Note 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.**
- Do not use IDs, visual elements should have aria-labels as below:



Advanced:

- Write React tests in `src/__tests__/Advanced.test.js` that confirm 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 React Testing Framework documentation https://github.com/testing-library/react-testing-library 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 current date (assuming Tuesday October 18, 2022) include:
  - "October 2022" appears on the screen
  - Element with `aria-label "October 18 2022 (selected)"` has value "18"
  - Elements before the first day of the month are set to the last days of the previous month
  - Elements after the last day of the month set to the first days of the next month

  And many others.

- Then consider tests where the previous and next buttons are clicked a few 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 selected element
    - After clicking next once, then previous once, there should be a selected element
    - After clicking next once, then previous twice there should be no selected element

    And many others.

- The learning from this requirement is that even for an apparently trivial component, there are many tests that must be written to prove it works as expected. In any non-trivial web application, there will be significantly more lines of code in the tests than 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/Entry.jsx` render 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.

- Under ***no circumstances*** add new code to `src/Picker.jsx` to satisfy this requirement.

- The following restrictions apply:

    - The text entry field should have an HTML placeholder of "`MM/DD/YYYY`"
    - The text entry field should have an aria-label of "`Enter date as MM/DD/YYYY`"
    - The button should have text 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 React tests in `src/__tests__/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 2. The logic is the same but the way you construct this new class will be somewhat different as you must use JSX and aria-labels rather than ids.

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 7 and 8.

For each requirement, write out a plan of action as code comments then go back and implement each part of the plan, writing tests *before* the implementation.

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 ☺

Test-as-you-type

To execute all tests every time your code changes, run the following command in a separate terminal to the one in which you ran `npm run dev`:

```
$ npm run watch
```

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. (80%) **Does it work?**

Basic Requirement	(40%)
Advanced Requirement	(20%)
Stretch Requirement	(20%)

2. (20%) **Is it right?**

Advanced and Stretch:	
No Lint Errors / Warnings	(10%)
Perfect Code Coverage	(10%)

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
      - 26% 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 ****

