Lab 4

Start Assignment

- Due Monday by 11:59pm
- Points 100
- Submitting a file upload
- File Types zip

CS-554 Lab 4

A Simple React Todo App

For this lab, you will create a simple Todo tracker using React. The purpose of this lab is to get familiar with using state and passing props into React components. It is a very basic application just so you can get some initial practice with React. A bigger more complex React lab is coming after our 2nd React lecture.

You will be creating a simple React application with just 3 components, `App` (your main component), `TodoList` (will display todos that have not been deleted or marked completed) and `CompletedTodos` (will display todos that have been completed). Our `App` component will hold the state off of the Todos, for this assignment, you will hard code 10 todo items (an array with 10 objects) in the initial state of the `App` component using the `useState` hook. Each todo will have the following fields:

```
{
id: 1,
title: 'Pay cable bill',
description: 'Pay the cable bill by the 15th of the month',
due: '3/15/2023',
completed: false
}
```

You will initially set the completed flag to be false for every item in the state. The id field must be unique for each todo item. You can use numbers like shown above, uuid, whichever you like as long as the id field is unique.

Components

App.jsx

Your app component will hold the state of the todos, they will then be passed to the other components (`TodoList` and `CompletedTodos`) as properties (props).

The initial state will be an array of todo objects with the fields shown above. You will set the state to

have at least 10 todo objects in the initial state. You will also need to create two functions in the `App` component

`deleteTodo (id)`: This function will remove a todo object from the array of todo objects in state for the id parameter passed into the function.

`toggleCompleted (todo)`: This function will toggle the `completed` boolean for the todo passed into it (pass the entire todo object into this function). Meaning, if the current value was `false`, this function will make it `true`, if it was `true`, then this function will make it `false`. It should update the `App` component's state for that todo that was marked complete/incomplete.

You `App` component will render both the `TodoList` and the `CompletedTodos` components. You will pass the todos from the `App` component's state into both the `TodoList` and `CompletedTodos` components as props. You will also pass in the `toggleCompleted` and `deleteTodo` functions into the `TodoList` component as a prop. You only have to pass in `toggleCompleted` as a prop to `CompletedTodos` (you do not have to pass `deleteTodo` into this component).

TodoList.jsx

This component will map through the todos that were passed in a prop and will render the following JSX for each todo item: (**Note**: you should ONLY render todos that are not flagged as completed)

```
<div>
  <h1>Pay cable bill</h1>
  <p>Pay the cable bill by the 15th of the month</p>
  <p>Due Date: 3/15/2023</p>
  <p>Completed: No</p>
  <button>Delete</button>
  <button>Complete</button>
</div>
```

When the user clicks the Delete button, it should call the `deleteTodo` function that was passed in from the App component as a prop and delete the todo from the `App` component's state.

When the user clicks the Complete button, it should call the `toggleCompleted` function that was passed in as a prop from the App component and then update the App component state, setting the completed field for that todo item to be `true`

If the todo is past due, display the title of the todo in the h1 element to be red, and also display the due date in red. (remember in React, we use the `className` attribute to set a class for an element instead of `class` like we do in normal HTML)

CompletedTodos.jsx

This component will map through the todos that were passed in a prop and will render the following JSX for each todo item: (**Note**: you should ONLY show todos that are marked completed)

```
<div>
  <h1>Pay cable bill</h1>
  <p>Pay the cable bill by the 15th of the month</p>
  <p>Due Date: 3/15/2023</p>
  <p>Completed: Yes</p>
  <button>Mark Incomplete</button>
</div>
```

When the user clicks the `Mark Incomplete` button, it should call the `toggleCompleted` function that was passed in as a prop from the `App` component and then update the `App` component state, setting the completed field for that todo item to be `false`

Extra Credit

This extra credit is worth 10%. I know we have not gone over form processing yet and that is why this functionality is extra credit and not required functionality. You will create another component called `AddTodo` that will also be rendered into the `App` component. This component will have a form with the following inputs: `title:` which is input type `text`, `description:` which is a `textarea`, and `due`: which is input with a type `date`. You must set the `min` attribute for the date input to be the current day's date(they shouldn't be able to select dates in the past). The completed boolean and ID will be handled by the function you pass into this component from the `App` component.
In your `App` component, you will create a new function. `addTodo`, this function will be passed to the `AddTodo` component as a prop. When the user fills out the form and submits it (You must make sure the form has values and is not empty!!!! and validate the form fields), you will then call the `addTodo` function passed in from the `App` component as a prop, passing in the values from the form that was filled out.

In the addTodo function in the `App` component, you will construct an object from the form values passed to it, you will increment the ID and also set `completed` property to be `false`. You will then add this object to your `App` component state.

Also, do not forget to reset the form after a successful form submission!

Form Field Constraints:
Title: Should be at least 5 characters long, and should not be an empty string or string with just spaces. Also, do not forget to trim your input!
Description: Should be at least 25 characters long and should not be an empty string or string with just spaces. Also, do not forget to trim your input!
Due: Must be a valid date in mm/dd/yyyy format (the default display format for the input type of date, but the value will be in yyyy-mm-dd format. Convert the value to mm/dd/yyyy before saving it in state) and should not allow the user to select dates before today's (the current) date.

General Requirements

1. Remember to submit your `package.json` file but **not** your `node_modules` folder.
2. Remember to have fun with the content.