Coding Exercise - React/Redux

Introduction

In this project, you will find a React/Redux project set up using Webpack and ES6, along with skeleton code components with tasks for you to complete. You do not need to worry about setting anything up, all the configurations, including wiring up the Redux store and actions have already been done for you.

To get started, from the main project directory, execute *npm install*, followed by *npm start*.

The Webpack development server will be started on http://localhost:9090.

Note: If you're having trouble installing dependencies or starting up the server, then you can make switch to use the following NodeJS and NPM versions to ensure it's not a library compatibility issue:

NodeJS v9.11.0, **NPM** 5.6.0

Requirements

- All the functional requirements for the tasks outlined below must be complete and correct.
- Only use native JavaScript and packages/library listed in the package.json (i.e. do not import new dependencies).
- Pay attention to the comments in the code. There are sections/files of code that prohibits you from making changes to them.
- There cannot be any console errors or warnings at any time while your code is running.
- Any assumptions you have to make about functionality must be commented in the code. This is for your benefit!
- Return **only the src directory** in a zip file named "your-name-solution.zip" when you are done. Please make sure that your solution can be unzipped, compiled, and working as expected in the existing project structure.

What we are **NOT** looking for:

- A pretty website. The base project has very bare-bones CSS, and you do not have to worry about beautifying it or altering it in any way. This exercise is only to test the functionality of your code.

Task 1

The objective of Task 1 is to enable Task 2.

On the Home page, you will find a link for "Home" and "Task 1". Clicking on "Task 1" will show you a big number.

Functional Requirements

- When a user clicks on "Task 1", then number displayed must immediately count down, 1 per second, until it reaches 0 (e.g. If 5 is displayed, in 5 seconds it much reach 0).

- When the count reaches 0, enable the "Task 2" link. The 0 can stay on the screen.
- However, the App is wired with a bomb that will blow up and render "BOOM!" if it is not *defused* before "Task 2" is enabled.
    - **Note**: Be very mindful of App.js's comments. There is a section reserved for you that you can edit freely. Everything else if off limits!

Once "Task 2" is enabled, you can proceed by clicking on the link.

Note: If you have trouble with this task and want to go forward, you can always access Task 2 directly by navigating to "http://localhost:9090/task2".

Task 2

On this page, you will see two tables: one for to-do tasks, and one for completed tasks. The objective of Task 2 is to build out the functionality on this page by working with server APIs.

In the project's **api.js** file, you can find the APIs you are allowed to use: "getTasks", "getTask", and "putTask". (Remember, you cannot modify this file in any way.)

A task is represented as a JSON object with the following attributes:

- **id**: (number) A unique number identifying the task
- **description**: (string) The task description
- **completed**: (boolean) Indicates if the task has been completed or not
- **addedDateTime**: (string) The datetime in ISO format of when the task was created
- **completedDateTime**: (string) The datetime in ISO format of when the task was completed. This will be set when "completed" is given as true, or deleted when "completed" is given as false.

Functional Requirements

- The tasks shown by default on the page are stubbed and incorrect. You must use the APIs to properly display the correct items in the appropriate list.
- The user can select tasks from the "To-Do" list and click "Mark Completed" to move them to "Completed" list.
- The user can select tasks from the "Completed" list and click "Mark To-Do" to move them to the "To-Do" list.
- The user must see the tasks move to the appropriate list immediately on button click; however, changes are not saved to the server until the user clicks on the "Submit" button.
- All tasks in the "To-Do" list must only show the checkbox and the task description.
- All **completed** tasks in the "Completed" list must be show the checkbox, the task description, and the completedDateTime, i.e:
    - [] Task description (2018-01-01T12:12:00.000Z)
    - **Note**: A task is not considered completed until the changes have been submitted to the server.
- Tasks must be sorted and displayed in alphabetical order in both lists based on their descriptions.
- If a user navigates to the "Home" or "Task 1" links without pressing the "Submit" button to save their changes, then all their actions must be cancelled.
- When the user navigates back to "Task 2", all tasks must in their proper list.

Task 3

You might have noticed that once in a while, the server call takes over a few seconds to return a response for the API call. This means that the user does not know if data is loading or something has gone wrong.

The objective of Task 3 is to build a component that will tell the user that data is loading. We want to be able to reuse this component anywhere in our app that might have slow API calls!

Functional Requirements

- While either the "To-Do" or "Completed" lists are waiting to load, the corresponding tables should show the text "Loading" inside the table.
- While the data is being submitted to the server, the "Submit" button must be disabled, and the text should be replaced with "Loading".
- You can either show the results right away as soon as the data is ready, or wait a minimum amount of time to avoid the "Loading" text from flickering.