1 Project 5

Due: Dec 10 by 11:59p
No late submissions

Important Reminder: As per the course *Academic Honesty Statement*, cheating of any kind will minimally result in receiving an F letter grade for the entire course.

This document first provides the aims of this project. It then lists the requirements as explicitly as possible. It then provides some background information. Finally, it provides some hints as to how those requirements can be met.

1.1 Aims

The aims of this project are as follows:

- To introduce you to building a dynamic web site using only client-side programming.
- To expose you to the use of the popular reactjs library.
- To enable you to build a single-page web app which accesses remote web services directly from a browser.

1.2 Requirements

You must check in a `submit/prj5-sol` directory in your gitlab project such that typing `npm install` within that directory followed by `npm start` is sufficient to start up a server running on port 1239.

Accessing that port using a browser should display a **setup page** which contains a form which allows the user to specify the URL at which the web services are running (this should default to `http://zdu.binghamton.edu:1235`).

Submitting the form on the setup page should display an **app page**. This app page should constitute a *single-page app* which will allow the user to search, add-to and display documents in a documents collection. Specifically, the app page should display 3 tabs:

Search Tab This tab should display a input widget. The user should be able to type in search terms into this input widget. When the input widget looses focus or a return is typed into it, the application should use the underlying web services to search the document collection for the search terms.

- A suitable error message should be displayed if there are no matching results.

- If there are matching results, then up to 5 results should be displayed:
  - Each result should be rendered in a suitable HTML element having a `class` attribute which includes `result`.
  - This result element should contain the name of the matching document rendered in a suitable HTML element having a `class` attribute which includes `result-name`.
  - The document name element should be followed by the text of the lines from the document which contain the search terms. Each occurrence of a search term within a line should be rendered within a suitable HTML element having a `class` attribute which includes `search-term`.
  - Clicking on a document name should switch the app over to the **Content Tab** displaying the contents of the document corresponding to the clicked-on document name.

Add Tab This tab should display a file selection widget. When the user selects a file from her local computer, the contents of that file should be uploaded to the document collection with the name of the newly added document set to the basename of the local file with any `.txt` extension removed.

A successful upload should result in the app switching over to the **Content Tab** displaying the contents of the uploaded document.

Content Tab This tab should display the name and contents of a document. It's contents should be rendered using a suitable HTML element which contains:

- A `<h1>` HTML element containing the document name.
- A `<pre>` HTML element containing the document contents.

The application must meet the following additional requirements:

- Any web service errors should be displayed suitably.
- All error messages should be reported with a `class` attribute containing `error`.
- The label for any form control should be rendered within a suitable HTML element having a class attribute which includes `label`.
- All form controls should be rendered with a class attribute which includes `control`.
- Tabs should retain their content unless explicitly changed by a user action.
- The content tab should be disabled when the application is first uploaded. Once some document has been loaded into it, the name of the tab should change over to the possibly abbreviated name of the document.

- Using the browser back button on the app page should return to the setup page.

[It is important to meet the specifications regarding `class` attributes, as meeting those specifications will allow reasonable styling using the provided stylesheets.]

1.3 Existing Servers

A slightly modified solution to *Project 3* is running on `zdu.binghamton.edu:1235` with the `snark` data files preloaded. It is restarted automatically every one hour.

The modifications to the project 3 web services includes the following:

- A `start` index to each of the links returned from the search results. This makes it possible to build links for this project without parsing the URLs returned in the links.
- The web services have been configured to allow larger uploads.

Additionally, a solution to this project is available via this *setup page*. (link only works from HTML version of this project; we are linking to a `http` web site on `cs.binghamton.edu`, as CORS restrictions do not allow calling `http` web services from a `https` page).

1.4 Provided Files

The prj5-sol directory contains a start for your project. It contains the following files:

HTML setup and app files Contains the HTML code for the setup and app pages. You should not need to change these files.

Main javascript driver Renders the top level component into the app page DOM.

Web services wrapper Uses axios to call web services. The web services are available as the `ws` property on the `app` component. You should not need to change this file.

Components directory This directory contains reactjs components:

app.jsx An `App` component for the overall application which displays the individual tabs. You should not need to change this file.

tab.jsx Provides component instances for each individual tab. You should not need to change this file.

search.jsx A skeleton file which provides the component contained within the search tab. You will need to edit this file.

add.jsx A skeleton file which provides the component contained within the add tab. You will need to edit this file.

content.jsx A skeleton file which provides the component contained within the content tab. You will need to edit this file.

Styles Directory A directory for CSS files. It contains a style.css stylesheet which provides styles for the specified `class` attributes. It also contains a tabs.css which is used for styling the tabs on the app page. You should not need to change these files.

***Babel configuration* file** Used for configuring babel. You should not need to change this file.

***Git Ignore* file** Set up to ignore artifacts created by npm and parcel.js which should not be committed to the repository.

README A README file which must be submitted along with your project. It contains an initial header which you must complete (replace the dummy entries with your name, B-number and email address at which you would like to receive project-related email). After the header you may include any content which you would like read during the grading of your project.

The course data directory has not changed since the previous project.

1.5 Hints

The following steps are not prescriptive in that you may choose to ignore them as long as you meet all project requirements.

1. Read the project requirements thoroughly. Play with the solution available via this *setup page*.
2. Study the *users app* code discussed in class. In particular, note the set up for the different reactjs components.
3. Ensure that your copy of the `cs580w` repository is up-to-date by pulling from the course repository.

   ```
   $ cd ~/cs580w
   $ git pull
   ```

4. Start your project by creating a `work/prj5-sol` directory in your gitlab project. Change into that directory and initialize your project by running `npm init -y`. This will create a `package.json` file; this file should be committed to your repository.
5. Copy the provided files into your project directory:

   ```
   $ cp -pr $HOME/cs580w/projects/prj5/prj5-sol/* .
   # copy .babelrc and .gitignore too
   ```

```
$ cp -pr $HOME/cs580w/projects/prj5/prj5-sol/.[bg]* .
```

6. Install runtime project dependencies. Minimally, you will need `axios` (a HTTP client used for accessing remote web services) and `react.js`.

   ```
   $ npm install axios react
   ```

7. Install development project dependencies. Minimally, you will need `parcel-bundler` (a no configuration bundler) and `@babel/plugin-transform-runtime`.

   ```
   $ npm install --save-dev parcel-bundler \
                            '@babel/plugin-transform-runtime'
   ```

8. Add the following `start` script to the `scripts` section of your `package.json` file:

   ```
   "start": "parcel docs-setup.html  --port 1239"
   ```

9. You should now be able to run the project sufficiently to display the setup and app pages:

   ```
   $ npm start
   ```

   Point your browser to *<http://localhost:1239>*; when you submit the setup page, you should see the app page with empty tabs.

   Besides `console.log()`, your browser's debugger available via the `F12` key may prove useful. Note that you will see both the packaged up code which is what is actually run by the browser, as well as the source code written by you. The browser maps the run code into your source code using source maps.

   One advantage of running parcel.js the above way is that it provides *hot module replacement* HMR such that code changes are automatically reflected in the application without needing to restart the server. Unfortunately, this does not currently appear to be 100% reliable; so if changes do not appear to be reflected, use control-C to break out of the server and restart after deleting the `dist` directory:

   ```
   $ rm -rf dist; npm start
   ```

10. Start working on the `search.jsx` component. Add code to its `render()` method to merely display a widget for the search terms (make sure you meet the specifications regarding the `class` attributes). Verify that you see the expected widget.

11. Add an `onChange()` handler to have your search component track the text within the HTML widget.

12. Add an `onBlur()` handler to the widget and an `onSubmit()` handler to its containing form. This handler should call the search web service (accessed via `this.props.app.ws`); for now, merely log the results on the console.

13. Format the results as per the specifications. It may be useful to define an auxiliary reactjs component to render each result.

    Set up the document name for each result as a link with an `onClick()` handler set up to change the content tab using `this.props.app.setContentName()` (don't forget to prevent the default action using `event.preventDefault()`). On clicking a result name link, you should at least see the name of the content tab changing.

    Note that reactjs does not have any way of correctly displaying a JavaScript string which contains HTML markup as the HTML markup will get escaped. This makes the formatting of the matching lines for each result inconvenient, since each occurrence of a search-term within a line must be rendered within a HTML element with a `class` attribute containing `search-term`. A solution is to split a line into suitable segments and render the list of segments.

14. Work on the content tab. The document name is provided as a `name` property. Note that the content should be a constituent of the component's state. Use the web services to get the content for the named document. Note that you will need to get the content when the component is first mounted (using *componentDidMount()*), as well as when the component `props` are changed (using *componentDidUpdate()*). For the latter, be sure to make the web service call only when the `props` have actually changed.

15. Work on the add tab. Set up a file upload widget with an `onChange()` handler which uses the appropriate web service to add the file contents to the document collection. Comments within the provided file give more details.

16. Iterate until you meet all requirements.

It is a good idea to commit and push your project periodically whenever you have made significant changes. When it is complete, please follow the procedure given in the *gitlab setup directions* to submit your project using the `submit` directory.