1 Project 5

Due Date: 12/9 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.

1.1 Aims of This Project

The aims of this project are as follows:

- To build a single-page authentication web application which uses a authentication web service.
- To generate HTML on the client.

1.2 Specifications

Set up your gitlab project so that it contains a `web-client` subdirectory within the top level `submit` directory. This directory should contain a `index.html` file such that accessing that file using the `file://` protocol in a **chrome** browser should result in *single-page* user-authentication web application. This application should use the back-end `AUTH` *project 3* web service running on the URL given by the query parameter `ws-url` (which should default to `https://localhost:1236`).

The application should behave as follows depending on the login status of the user.

- If the user is not logged in, the application should display a login form (with a link to a registration form).
- If the user is logged in, the application should display account information for the user (with a logout button).

The different forms shown to the user will be very similar to the pages in your previous project:

Login Form The login form must provide the following suitably labelled input controls:

- An input field for an email address.
- An input field for a password.
- A *Login* submit button.

- A link to the *Registration* form.

Input fields should be validated as soon as possible, usually when the user leaves the field. If validation fails, a suitable error message should be displayed on the form.

When the form is submitted using the *Login* button, the program should again validate all the input fields:

- None of the input fields should be empty.
- The email address should look like a reasonable email address.

If the validation succeeds, the program should use the `AUTH` web service to set up a login session for the browser which submitted the request and replace the login form with the *Account* info.

If either the login web service or the validation fails, the program should continue displaying the login form with suitable error messages added and all user input retained.

Registration Form The registration page must provide the following suitably labelled input controls:

- An input field for a first name.
- An input field for a last name.
- An input field for an email address.
- An input field for a password.
- An input field for password confirmation.
- A *Register* submit button.
- A link to the *Login* page.

Input fields should be validated as soon as possible, usually when the user leaves the field. If validation fails, a suitable error message should be displayed on the form.

When the form is submitted using the *Register* button, the program should again validate the input fields:

- None of the input fields should be empty.
- The email address should look like a reasonable email address.
- The password should consist of at least 8 characters none of which is a whitespace character and at least one of which is a digit.
- The value for the password confirmation field must match the password field.

If the validation succeeds, the program should use the `AUTH` registration web service to create a registration corresponding to the submitted information and set up a login session for the browser which submitted the request and replace the registration form with the *Account* info.

If either the `AUTH` registration web service or the validation fails, the program should continue displaying the registration form with suitable error messages added and all user input retained.

Account Info The account info should display suitably labelled first name and last name from the registration corresponding to the current login session.

It should also contain a `Logout` button such that clicking that button terminates the current login session and displays the *Login* form.

Your application should run as a single-page web application with no requests made to the web server which served your `index.html` page after the initial page load.

Your web application must meet the following additional requirements:

- Leading and trailing whitespace should be ignored for all input fields.
- A reasonable email address must look like *user*`@`*domain* where *user* and *domain* can be arbitrary non-empty strings.

1.3 Provided Files

The files directory contains the following files:

setup.html A trivial HTML form which allows you to specify the backend `ws-url` query parameter for your `index.html` file.

index.html A start at the `index.html` file you are required to submit. It provides a function for getting the `ws-url` query parameter. It requires library scripts for reactjs and axios (replace appropriately if you use different libraries).

1.4 Hints

The following steps are merely advisory. They assume that you are using reactjs for building your UI within the browser and either jquery or axios for making ajax requests.

Note that chrome does not allow javascript to set cookies when using the `file¬ ://` protocol. A workaround is to use the browser localStorage to retain authentication information.

1. Review material you will need for this project; the material for react, jquery / axios and localStorage. Decide which JavaScript libraries you need to use for this project; in addition to `react` and `jquery`.

2. Start with the provided index.html file. Edit as necessary to ensure that all necessary JavaScript libraries are loaded into it; as far as possible, load them from external CDN's instead of within your project to avoid having to check-in those external libraries.

   The provided file has empty content. You can simply set up some `hello world` content in the file and test by using a appropriate `file` URL of the form `file://`*PATH* where *PATH* is the **absolute** path to your `index.html` file on your local machine. An example which might work for me is

   ```
   file:///home/umrigar/repos/cs580w/submit/client-auth/index.html
   ```

   Note the use of the third slash to ensure that the path is treated as absolute.

   You can also use the provided file to set up the `ws-url` query parameter used to invoke your `index.html`. By uncommenting out the `alert()` included in the provided `index.html`, file you can verify that the `ws-url` query parameter is being correctly provided.

3. Set up react components for login, registration and account-info. Since they share similar functionality, try to share code between them. Verify that you can display each component in the browser. For testing each component you can simply change your `index.html` to load that component into the top-level `root` div of your page.

4. Verify that you can display each component with error messages. This would be needed when errors are detected on user input.

5. Add necessary validations to each component and verify that errors are shown as expected. To ensure that errors are reported as soon as possible, you can do validation on a control whenever you detect a `onBlur` event on a control. You should do validation on all controls in a form whenever a `onSubmit` event is received for that form. At this point, there will be no action after a successful validation.

6. Set up the necessary infrastructure for the `AUTH` web service. You will need to ensure that the web service you are using will allow your browser to connect to it; the simplest solution is to `cors`-enable it by adding the `cors` module to it (`npm install --save cors`) and enabling `cors` for all your routes (`app.use(cors())` before setting up any route). You will also need to allow it to return `Location` headers by adding `Location` to the `Access-Control-Expose-Headers` header. Assuming `res` corresponds to a express.js response `res.set('Access-Control-Expose-Headers', 'Location')` will set things up for a response which needs to return a

`Location` header.

Note that the solution to Project 3 has been cors-enabled.

You will also need to ensure that your browser accepts the self-signed certificate from the `AUTH` web service you have set up. You can do so by manually accessing the URL *ws-url*`/users/xx` where *ws-url* is the URL where you have set up your `AUTH` web service. After going through the usual *accept insecure certificate* dialog, you should see a `UNAUTHORIZED` response from the web service.

It is imperative that you do use a cors-enabled web service and you enable your browser to accept the self-signed certificate. If you neglect to do so, any AJAX calls you make to the web service will fail.

7. Create an object which serves as a facade for the web services. Start implementing the web services facade with the registration service.

   - If you are using axios, the client-side code will be very similar to the server-side code you may have used in Project 4.
   - If you are using jquery, note that since `jquery` does not provide a shortcut method for `PUT`, you will need to use the jQuery.ajax API.

   Hook up the results to the react components you set up earlier.

   Set up localStorage to maintain your login session.

   Repeat for the other web services.

8. Test to ensure that you have met all the project specifications.

1.5 Project Submission

Submit your project using gitlab:

- Submit your project files in a top-level `submit/client-auth` directory of your `cs480w` or `cs580w` project in gitlab. To facilitate grading, please include the setup.html file, whether you modified it or not.
- You should **not** submit executables or object files.
- Make sure that you provide a `README` file containing minimally your Name and B-number. It may contain any other comments which you would like read by the grader.