1 Project 4

Due: Apr 21 by 11:59p

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 consuming web services.
- To give you some familiarity with using the axios client library.
- To give you experience with developing templates using mustache.
- To expose you to building a JavaScript project starting with a totally blank slate.

1.2 Requirements

Add a `submit/prj4-sol` directory to your gitlab repository such that typing `npm install` in that directory followed by

```
$ ./index.js WS_URL PORT
```

will start a web server running on port `PORT`. This web server should consume the URL-shortening web services running on `WS_URL` to provide a web site which displays the following pages:

Home page at URL / This page should display links to the following three pages.

Shortener page This page should display a form allows the user to type in arbitrary text which may be of any length. When the form is submitted it should redisplay the page with the user input retained. Additionally, it should display the user input elsewhere on the page with all URLs in the text replaced by HTML links to the corresponding short URLs returned by the shortener web service.

Info page This page should display a form which allows the user to specify a URL. When the form is successfully submitted, it should redisplay the page with information about the input url which includes:

- The short URL.
- The long URL.
- A count of the number of times that short URL has redirected to the long URL in the underlying web services.
- The activation status of the URL.

Deactivation page This page should display a form which allows the user to specify a URL may be either a short or long url. When the form is successfully submitted, it should redisplay the page with a suitable success message after deactivate the input URL.

Form submission should check for errors which include:

- Empty user input.
- Badly formed urls: they should always start with `http://` or `https¬://` and contain a valid domain. Note that the text input on the shortener page can be badly formed without resulting in any errors.
- Web service errors.

When errors are detected, the action for the form submission should not be taken. Instead the form should be redisplayed **with all user input retained** and suitable error messages displayed close to the offending widget.

The shortener, info and deactivation pages must display a footer which provides navigation links like those on the home page. The implementation should use only a single copy of this footer.

The solution should respect web semantics; i.e. appropriate form submission methods should be used and caching semantics should be correct.

The behavior of the browser back button should be "reasonable". Specifically, the browser should not display any warning messages when the user presses the back button.

Any styling of the content is acceptable as long as the content meets the functionality described above.

Your solution is also subject to the following implementation constraints:

- You must use mustache for all server-side templating.
- You must use axios for calling the web services as `WS_URL`.
- You must use expressjs for setting up your web server.
- The content served up by your server to the browser must not contain any JavaScript; i.e. all code should be run only on the server.

1.3 Example Operation

A running version of this project is available at *<http://zdu.binghamton.edu:2346>*. As usual, this site is accessible only from within the CS department network. Note that since the underlying web services run only on `http`, `https` redirects are not handled.

1.4 Underlying Web Services

You will need a server running the underlying web services at the URL specified by the `WS_URL` command-line argument. Alternatives:

1. Deploy your solution to *Project 3*.
2. Deploy the *provided solution* to *Project 3* (this will be made available within a few days once the late submission deadline expires).
3. Use the solution to *Project 3* running on *<http://zdu.binghamton.edu:2345>*. Note that the underlying database is cleared every hour. This site is accessible only from within the CS department network.

The solution provided for *Project 3* has the following changes made for the text shortening service running at `/x-text`:

- The ~ tilde character has been added to the list of characters which can continue a URL.
- The service recognizes an optional `isHtml` property in the input JSON. If the value of this property is truthy, then:
  - All shortened URL's in the returned text are returned as HTML `<a>` links.
  - Any HTML meta-characters are escaped using the usual HTML escape sequences.

  Hence by using `isHtml`, it is possible to plug the text returned by the `/x-text` directly into the shortener page.

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. Look at the sample *sample web site*. Understand the users xample covered in class.

   Note that unlike your previous projects, you have not been provided with any starting code and will be starting with a complete blank slate. Hence

it is imperative that you understand all the material thoroughly before you start the project.

2. Design the HTTP interactions for your project. These will include:
   - The URLs for the different pages (the requirements only specify the URL for the home page).
   - The HTTP methods used when submitting forms. The requirements specify that you must respect web semantics.
   - Any caching strategies.

   Some observations:
   - Submitting the shortener form is an unsafe action since doing so may create long to short URL mappings on the server.
   - Submitting the deactivation form is an unsafe action.
   - If the *Post-Redirect-Get pattern* (around pg. 4 of the *users example*) is used for the submission of the shortener form, then it is not possible to send the input text and transformed text to the `GET` page as URL parameters as there is no limit on their length.
   - If the response to a `GET` depends on something other than its URL, then it is incorrect to have that response cached.
3. Choose and `npm install` all your project dependencies.
4. It is possible to set up your solution to look similar to the *sample web site* if you structure your HTML in a similar way and use its stylesheet.
5. Implement your project.

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.