1 Project 4

Due: May 5, by 11:59p. **No late submissions**

Important Reminder: As per the course *Academic Honesty Statement*, cheating of any kind will minimally result in your letter grade for the entire course being reduced by one level.

This document first provides the aims of this project. It then lists the requirements as explicitly as possible. This is followed by a log which should help you understand the requirements. 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 use web services from a program.
- To familiarize you to the mustache templating system.
- To give you some exposure to programming within a browser.

1.2 Requirements

You must push a `submit/prj4-sol` directory to your github repository such that typing `npm ci` within that directory is sufficient to run a web server using `index.mjs`.

```
$ node index.mjs PORT WS_URL
```

where

`PORT` Specifies the port number at which the server should listen for incoming HTTP requests.

`WS_URL` Specifies a URL where web services for *Project 3* are running.

You are being provided with an `index.mjs` which provides the required command-line behavior. What you specifically need to do is add code to the provided blog544-ss.mjs source file to have it produce HTML for the following pages:

Home page URL `/`. A page which should provide links (with descriptive text) to link to the **list users** and **search users** pages.

List Users URL not specified. This page should list users. It should be possible to filter the displayed users using query parameters. The information displayed for each user should include the user id, user's name (formed by

separating the first and last names by a single space), email, roles, creation time and update time. The latter two dates must be displayed in `mm/dd/yyyy` format.

The page should provide links to allow scrolling back-and-forth within the users.

Search Users URL not specified. This page should display a form to allow searching for a user by user id, email, first name, last name and creation time. The creation time must be specified as an ISO-8601 date-time.

- As soon as the user id widget is filled in, a suitable error should be displayed if there is no user for that user id. This should happen immediately once focus leaves the user id widget without requiring a form submission.
- If the form is submitted without any fields specified, then a suitable error should be displayed.
- If the value of a form widget is incorrect when the form is submitted, then the form should be redisplayed with a suitable error displayed, preferably next to the form widget. The widget values **must** be retained across the form submission.
- If there are no results for the specified search parameters, then a suitable error message should be displayed with widget values retained across the form submission.

If there are no errors, then the list users page should be displayed showing the users which meet the search criteria. Subsequent scrolling of the list users page should only scroll through the results specified by the search parameters.

Both the list users and search users pages should be displayed with a footer containing the same links as the home page.

If there are any internal errors, then a suitable error message should be shown.

A working version of the project is available at *<http://zdu.binghamton.edu:2346>* (as usual, this can only be reached from within the CS network).

1.3 Provided Files

The prj4-sol directory contains a start for your project. It contains the following files:

blog544-ss.mjs This skeleton file constitutes the guts of your project. You will need to flesh out the skeleton, adding code to meet the project requirements. You should feel free to add any auxiliary function, method definitions or even auxiliary files as required.

blog544-ws.mjs A wrapper which calls Project 3 web services. You should need to change this file.

index.mjs This file provides the complete command-line behavior which is required by your program. It requires blog544-ss.mjs and blog544-ws.mjs.

style.css A style sheet which can be used to make your project look somewhat reasonable.

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.

1.4 Project 3 Solution

The solution to *Project 3* has been modified slightly to facilitate this project:

- List results contain a `next` field only if there are subsequent results.
- Validation errors are reported as an `errors` list within the JSON of an error response. Each object within the `errors` list has `name`, `code` and `message` properties. The `name` gives the name of the widget causing the error, `code` is the application error code and `message` is the error message.

The project 3 solution is running at *<http://zdu.binghamton.edu:2345>* (available only from within the CS network).

You can view the behavior of the `next` field using the following queries:

- The query at URL users which allows scrolling back-and-forth among all users. Note the `next` and `prev` properties.
- The query at URL users?firstName=Adrian which does not produce any `next` and `prev` properties since there are only 3 results.
- The query at URL users?firstName=Harry which does produce a limited number of scrolled results (scroll through the results using `links[*].url`. Note the absence of the `next` property within the last result and the absence of the `prev` property within the first result.

You can view the behavior of the `errors` property using the following queries:

- The query at users?creationTime=2020 should produce a bad ISO 8601 datetime error.
- The query at users?creationTime=2020&email=xx should produce a bad ISO 8601 datetime and a bad email error.

1.5 Hints

When working on this project, use **view source** on the pages of the example web site to get some idea of how to construct your HTML. Though the JavaScript on the example web site has been obfuscated, you can try viewing it to get some 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 *example website* (reachable only from within the CS network) to make sure you understand the necessary behavior. Review the material covered in class including the users-ss example (note that the `users-ss` example displays several pages and is consequently more general than what is needed for this project which merely displays 2 pages).

2. The project requirements specifies only the top-level URL `/`. The choice of all other URLs is entirely up to you; decide on what those URLs should be.

3. Start your project by creating a `submit/prj4-sol` directory in a new `prj4-sol` branch of your `i444` or `i544` directory corresponding to your github repository. Change into the newly created `prj4-sol` directory and initialize your project by running `npm init -y`.

   ```
   $ cd ~/i?44
   $ git checkout -b prj4-sol #create new branch
   $ mkdir -p submit/prj4-sol #create new dir
   $ cd submit/prj4-sol       #enter project dir
   $ npm init -y              #initialize project
   ```

   This will create a `package.json` file; this file will be committed to your repository in the next step.

   Note that unlike your previous project, the `"type": "json"` addition for `package.json` is no longer necessary as we are using the `.mjs` extension for ES6 modules.

4. Commit into git:

   ```
   $ git add package.json #add package.json to git staging area
   $ git commit -m 'started prj4' #commit locally
   $ git push -u origin prj4-sol #push branch with changes
                                 #to github
   ```

5. Install necessary dependencies using

   ```
   $ npm install axios body-parser express mustache
   ```

This will install the library and its dependencies into a `node_modules` directory created within your current directory. It will also create a `package-lock.json` which must be committed into your git repository.

The created `node_modules` directory should **not** be committed to git. You can ensure that it will not be committed by simply mentioning it in a `.gitignore` file. You should have already taken care of this if you followed the directions provided when setting up github. If you have not already done so, please add a line containing simply `node_modules` to a `.gitignore` file at the top-level of your `i444` or `i544` github project.

6. Commit your changes:

```
$ git add package-lock.json
$ git commit -a -m 'added package-lock'
$ git push
```

7. Copy the provided files into your project directory:

```
$ cp -pr $HOME/cs544/projects/prj4/prj4-sol/* .
```

This should copy in the `README` template, the `index.mjs`, the `blog544-ss.mjs` skeleton file, and the `blog544-ws.mjs` web-service wrapper into your project directory.

8. You should now be able to get a usage message and start the server:

```
$ node index.mjs
usage: index.mjs PORT WS_URL
$ node index.mjs 2346 http://zdu.binghamton.edu:2345
listening on port 2346
```

The server will be listening for incoming requests; you can stop it by using ^C.

9. Replace the `XXX` entries in the `README` template.

10. Commit your project to github:

```
$ git add .
$ git commit -a -m 'set up prj4 files'
$ git push
```

11. Add a `index.html` file in the `statics` directory. This should contain the HTML for the home page. Test by starting up your server as above, and pointing a browser to /.

12. Look briefly at the code for the `blog544-ss.mjs` web services wrapper file to understand the single `list()` method which can be used for listing users by specifying `category` as `users`.

13. Open the copy of the `blog544-ss.mjs` skeleton file in your project directory. It contains code to start the web server, the start of a routing function, error handling code and some utility code. It does not contain code for any handlers.

14. Set up routes for listing users and searching users. Note that search is a safe and idempotent operation.

15. Create dummy handlers for all the handlers you specified when setting up your routes. Add `console.log()` to each handler and verify that your handlers are getting run when you access the corresponding URL.

16. Add code to the list users handler. For now, simply call the web service wrapper `list()` method passing across any query parameters. Simply console.log() the output of the web service.

17. The returned user information cannot be directly rendered by mustache. Build a view model better suited for rendering and `console.log()` it. Do not bother with the next and previous scrolling controls.

18. Create a mustache template for the list users page in a `templates` directory. If you are using the provided `style.css` stylesheet, you should simply stuff the results into a HTML `<table>`. Use the provided `doMustache()` utility to render the template. Test.

    It is probably a bad idea to advance to the next step until you have this step working.

19. Update the view model for the scroll controls. To produce a scroll control, the view model will need to contain something which can be tested by mustache using a `{{#...}} ... {{/...}}` section. A `next` control should be generated only if there is a value for `next` in the web service results; similarly for `prev`.

    You will find node's querystring invaluable to build up a query string for the scroll links. Simply merge a `_index` into the query parameters for the request, and then call `querystring.stringify()` on the merged object.

    The provided stylesheet depends on the scroll controls being under an element having class `scroll`. Set up each individual control as an `<a>` anchor element having a `rel` attribute of `next` and `prev` respectively.

20. Set up a template for the search users page. The provided style sheet uses grid layout for the form controls. It assumes that each wiget consists of a pair containing a label for the widget and the control for the widget. The elements of this pair must be **direct descendants** of the `form` element.

    Modify your search handler to render this template when the search URL is hit.

21. Add code for a search submission. Note that a search submission will have a value for the widget being used to submit the form.

(a) Get the query parameters and verify that there is at least one parameter other than the submitting control which has a non-empty value. If that is not the case, redisplay the page with a suitable error message.

(b) Call the web service wrapper `list()` method passing on the query parameters.

(c) If the results are empty, display a suitable error message.

(d) If the results are non-empty, then redirect to the users list page passing on the query parameters.

(e) If the web service throws an error, catch that error. If the error contains validation errors, then redisplay the page with the validation errors. If the error is not recognizable as containing validation errors, then redisplay the page with some kind of generic *server error* message and log the full error on the console.

22. The requirement that the user id be checked before submitting the search form implies the use of JavaScript within the browser.

    Modify your user list handler to return the JSON produced by the web services when a particular parameter is set.

    Add a `<script>` element to the search template to pull in a JavaScript file from the `statics` directory. Since the required JavaScript is rather trivial, it is sufficient to directly use the browser's DOM without any additional libraries.

    (a) Use an IIFE to enclose all your code.

    (b) Get hold of the user id widget `element` using `document.getElementById()`.

    (c) Add a listener to that element using `element.addListener(handler)`.

    (d) The `handler()` function will be called with an `event` as its first argument. Use `event.target.value` to get the value of the user id widget.

    (e) Call the modified user list handler passing in the extra parameter to ensure that the results are returned as JSON. You can do that using fetch() to fetch a JSON `response` from your server and then extract the JSON using response.json().

    (f) If the number of users contained in the extracted JSON is zero, output a suitable error.

23. 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 *git setup* document to submit your project to the TA via github.