1 Project 3

Due: Apr 5 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 building web services.
- To give you some familiarity with using the expressjs web framework.
- To give your more experience with text processing in JavaScript.

1.2 Requirements

Add a `submit/prj3-sol` directory to your gitlab repository such that typing `npm install` in that directory followed by

```
$ ./index.js MONGO_DB_URL PORT [LONG_URL...]
```

will start a web server running on port `PORT` using mongo db specified by `MONGO_DB_URL`. This web server should provide url-shortening web services as specified below. If one or more `LONG_URL`'s are specified, then the db will be cleared and initialized to contain the shortening information for all the `LONG_URL...`.

The web server should accept the following requests:

`GET /`***shortUrl*** If *shortUrl* is a previously created short url which is currently active, return a 302 redirect to the corresponding long url.

`POST /x-url?url=`***LONG_ URL*** Shorten *LONG_ URL* and return a JSON object having a property `value` which gives the short url. If *LONG_ URL* was added previously, the short url returned should be the same as that returned earlier. If *LONG_ URL* was added previously and is currently inactive, then is is activated.

`DELETE /x-url?url=`***URL*** Deactivate *URL* where *URL* is a previously added long or short url. It is not an error if *URL* is already deactivated. Return success if everything ok.

`GET /x-url?url=` ***URL*** Return information for the previously added long or short *URL* specified by the `url` query parameter. The information should be returned aas a JSON object containing the following properties:

`longUrl` the associated long url.

`shortUrl` the associated short url.

`count` a count of the total number of times `shortUrl` was successfully looked up.

`isActive` a boolean denoting whether or not the mapping is active.

Note that the info should be returned even if url is currently deactivated.

`POST /x-text` Expect a JSON body with property `text`. Replace any url's in the text with their short url's, adding/activating them in if not already present. If there is something wrong with a text url, then don't replace it. A url is case-insensitive and identified by starting with either `http://` ir `https://` and continues on with one-or-more of the following characters:

- An alphanumeric character.
- One of the following 11 characters:

  `_ - / . ? = & % # @ +`

It is terminated by the end of the string or the presence of some character which is not in the above set.

Returns a JSON having a property `value` giving the input text with all delimited URLs replaced by short url's.

[Note the list of characters given above is not the correct list of characters for url's. The precise rules for what consitutes legal characters in an url are quite complicated, as the set of allowed characters depends on the context within the url. Instead, the characters specified above were chosen based on pragmatic criteria].

1.2.1 Errors

If errors occur, then a suitable HTTP status should be returned with the response body consisting of a JSON object giving a `code` and `message` describing the error. Some suggestions for suitable HTTP statuses:

Not Found 404 When a short or long url which should exist is not found.

Bad Request 400 A url is badly formatted.

Server Error 500 An internal server error occurred.

Besides reporting errors to the client, all errors should always be logged on standard error along with a stack trace indicating the location of the error.

It is permissible for an error during start up or initialization to terminate the entire web server, but termination after the web server has started processing requests should be avoided as far as possible.

1.3 Example Operation

The required behavior of the project is illustrated by the following:

- A sample log.
- A server running on *<http://zdu.binghamton.edu:2345>*. The database used by this server is cleared every hour. Note that this server can only be accessed from within the CS network; for example, from the machines within the CS lab classrooms.

1.4 Provided Files

The prj3-sol directory contains a start for your project. It contains the following files:

shortener-ws.js This file is a skeleton file. It is the file where you will be writing the bulk of your code.

url-shortener.js This file is largely a solution to your previous project with the following changes:

- Errors are returned using exceptions rather than return values.
- The `count` service has been replaced by a `info` service, where instead of a simple scalar count, the returned value is a object containing the short and long urls, the activation status in addition to the count.
- Code for checking whether a domain is valid has been pulled out into a library lib544.

You should not need to modify this file.

index.js This file provides the complete command-line behavior which is required by your program. It requires url-shortener.js. and shortener-ws.js. You **must not** modify this file.

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.

lib544 Utility routines for checking the validity of a domain have been split out into a library. This may come in useful in future projects.

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 log to make sure you understand the necessary behavior. Understand the user-ws example covered in class.

2. Look into how you will drive your project. The provided log uses curl which lends itself well to scripting, but you may find a GUI client like restlet easier to use.

3. Start your project by creating a `work/prj3-sol` directory. 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.

4. Install necessary packages. Minimally, you will need the mongodb client library, cors, expressjs and body-parser, and the `lib544` library. Install using:

   ```
   $ npm install mongodb cors express body-parser file:~/cs544/lib544
   ```

   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 gitlab. If you have not done so, please add a line containing simply `node_modules` to a `.gitignore` file at the top-level of your `i444` or `i544` gitlab project.

5. Copy the provided files into your project directory:

   ```
   $ cp -pr $HOME/cs544/projects/prj3/prj3-sol/* .
   ```

   This should copy in the `README` template, the `index.js` command-line driver program, the db interface `url-shortener.js`, and the `shortener-ws.js` skeleton file into your project directory.

6. You should be able to run the project:

   ```
   $ ./index.js
   usage: index.js MONGODB_URL PORT [LONG_URL...]
   $
   ```

7. Set up a route for a `GET` to a short url. Set up the action routine to send a 302 redirect back to the client.

   This is as simple as:

   (a) Get the entire request url (use the provided `requestUrl()` utility); this is the short url being referenced.

   (b) Use the `query()` service from the `url-shortener` model to get the long url.

   (c) Send a redirect (using `res.redirect()`) to the `value` returned for the long url.

   Wrap all of the above within a `try-catch`, mapping any errors from the application level to the web level. Test using the short urls set up for the long urls provided on the command-line.

8. Set up a route for adding a long url using a `POST` to `/x-url` and hook it up to a corresponding handler which extracts the long url from the query parameter `url`. Test.

9. Set up a route for fetching information about a url using a `GET` to `/x-url` and hook it up to a corresponding handler which extracts the url from the query parameter `url`. Test.

10. Set up a route for deactivating a url using a `DELETE` to `/x-url` and hook it up to a corresponding handler which extracts the url from the query parameter `url`. Test.

11. Set up a route for a `POST` to `/x-text` to allow replacing any long urls in arbitary text with short urls. Extract the input text from the body parameter `text` (should be available via `req.body.text` since the body-parser is set up to use JSON in the earlier global routes). Matching for occurrences of url's within the text is probably best handled using a regex.

    Note that it is tempting to use JavaScript's `String` replace() with a function used to specify the replacement. Unfortunately the provided function needs to be asynchronous and `replace()` does not support calling asynchronous functions. Hence you will need to provide some kind of loop using either a conventional index-based loop or one based on `while` and exec().

12. 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.