1 Project 3

Due: Nov 3 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 REST web services.
- To expose you to the potential of HATEOS in REST.
- To give you some familiarity with using the express.js web framework.

1.2 Requirements

You must check in a `submit/prj3-sol` directory in your gitlab project such that typing `npm install` within that directory is sufficient to run the project using `./index.js` with usage as follows:

```
$ ./index.js MONGO_DB_URL PORT NOISE_FILE [CONTENT_FILE...]
```

where

`MONGO_DB_URL` Specifies the URL of the mongo database to be used for storing document information.

`PORT` Specifies the port at which your program will listen for HTTP requests.

`NOISE_FILE` Specifies a path to a file containing words which should be ignored within document content and searches.

`CONTENT_FILE...` Specifies zero-or-more paths to files containing content which should be used for initializing the document collection.

When started, the program will write its process id *PID* into a file `.pid` in the current directory.

Many of the web services described below are required to return a list of `links` to resources. Each link will have two properties:

`href` The URL of the linked resource.

`rel` The relationship the resource identified with the link has with the resource returned by the web service.

The project requires three types of `rel` relationships:

`self` The `href` links back to the returned resource.

`next` The `href` links back to the *next* resource(s) in a collection of resources.

`previous` The `href` links back to the *previous* resource(s) in a collection of resources.

Note that `next` and `previous` links can be used by a client for scrolling through a collection of resources.

The requirements below only provide partial specifications for error conditions. They specify the HTTP status code for only some errors. Suitable HTTP status codes should be returned for other errors. Specifically, any server errors should result in a 500 `SERVER_ERROR` status code. Besides the HTTP status code, all error response bodies should contain a JSON object giving details of the error with the following fields:

`code` A string specifying a code for the error.

`message` A human readable message giving the details of the error.

Your server should support the following web services:

Get Content: `GET /docs/`***name*** Return the contents of the document identified by *name*. The returned JSON object should have a `content` field giving the document's contents and a `links` property specifying a `self` link.

If the document *name* does not exist, the server must return a HTTP status of 404 `NOT FOUND` along with a JSON object giving details of the error.

Search Content: `GET /docs?` ***QUERY_PARAMS*** Search for documents which satisfy query parameters *QUERY_PARAMS*. The supported parameters must include the following:

`q` This **required** parameter specify the terms to be search for. Successive search terms will be separated by space (usually encoded as a `+` or `%20`).

`count` This optional parameter should specify the maximum number of returned results. Default: 5.

`start` This optional parameter should specify the index of the first item in the returned results in the overall results. Default: 0.

The returned JSON object should have the following fields:

`results` A list of results which satisfy *QUERY_PARAMS*. Each item in the list should be a JSON object containing the following fields:

`name` The name of the matching document.

`score` A score which is a measure of how well the matching document matches the search terms.

`lines` A list of the matching lines from the document in source order.

`href` A URL which can be used to access the matching document.

The results must be ordered in descending order by score with ties broken by sorting in ascending lexicographical order by document name.

If there are no documents which satisfy the *QUERY_PARAMS*, then `results` should be returned as the empty list `[]`.

`totalCount` An integer giving the total number of matching results.

`links` A list of link objects providing links for the `self`, `next` and `previous` `rel` relations.

If the request has incorrect *QUERY_PARAMS*, then the server must return a HTTP status of 400 `BAD REQUEST` along with a JSON object giving details of the error.

Add Content: `POST /docs` This request will add a document to the collection. The request body must be a JSON object containing the following fields:

`name` A name for the document being added.

`content` The contents of the document being added.

The document should be added to the collection updating the contents if added earlier. Success should be indicated by a HTTP status code of 201 `CREATED` with a `Location` header giving a URL for the added document and a JSON body provided a `href` field with value specifying a URL for the added document.

If the request body is incorrect, then the server must return a HTTP status of 400 `BAD REQUEST` along with a JSON object giving details of the error.

Get Completions: `GET /completions?text=`***TEXT*** Return a JSON list containing all the completions of the last word in *TEXT* sorted in lexicographically ascending order.

1.3 Project Log

An *annotated log* of using the project using curl as a client is available.

A working version of the project is also available on <*http://zdu.binghamton.edu:1235*>. This server has been started with all the `_*.txt` files in <*../../data/corpus/snark*> preloaded. Note that to prevent database buildup, the database is reset every one hour. This service is only available from **within** the campus network.

1.4 Provided Files

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

docs-ws.js This skeleton file constitutes the guts of your project. You will need to flesh out the skeleton, adding code as per the documentation. You should feel free to add any auxiliary function or method definitions as required.

index.js This file provides the complete command-line behavior which is required by your program. It requires docs-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.

The course data directory containing test data has an additional corpus with documents containing the individual verses of Lewis Carroll's *The Hunting of the Snark* poem. The emphasis of this project is not on indexing or the database and this new corpus allows searching many small documents.

Additionally, the course lib directory contains a modified solution to your previous project so that it can be used as a module for your project. Changes from the solution include the following:

- The `find()` instance method of `DocFinder()` takes a single string parameter specifying the words to be searched for (previously it took a list of non-noise words).
- A single `create()` factory method is used to construct a `DocFinder` instance (previously, construction required a call to a constructor followed by a call to `init()`).
- The `lines` in the search result object has been changed from a string to a list.

1.5 Testing your Project

Since this project requires implementing web services, you will need a web client for testing your project. You could simply use a web browser but doing so will be inconvenient especially for `POST` requests.

One alternative is to use a command-line client like curl as in the provided *annotated log*. You can pretty-print the returned JSON using `json_pp` (already available on your VM) or `jq .`. If using the latter, you will need to install it on your VM:

```
$ sudo apt-get install -y jq
```

Another alternative is to use a GUI client like Restlet.

1.6 Hints

This project merely requires four handler functions for the four required web services plus any auxiliary functions you may choose to define.

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 and play with the solution available on *<http://zdu.binghamton.edu:1235>*.

2. Study the *User Web Services* code discussed in class. In particular, note the set up of the routes as well as the wrapping of handlers to ensure that any server errors are handled properly.

3. You can use the provided `doc-finder` project as a module in this project. Install its dependencies:

   ```
   $ cd ~/cs580w/lib/doc-finder
   $ npm install
   ```

   You should be able to run the project using `./docs-cli.js`, with the same command-line behavior as your previous project.

4. 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.

5. Copy the provided files into your project directory:

   ```
   $ cp -p $HOME/cs580w/projects/prj3/prj3-sol/* .
   ```

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

6. Install project dependencies. Minimally, you will need `cors`, `express.js` and the locally installed doc-finder module:

   ```
   $ npm install cors express ~/cs580w/lib/doc-finder
   ```

   Note that npm will install the local `doc-finder` module by creating symlinks in your `node_modules` directory. You should be able to run its command-line interface:

   ```
   $ ./node_modules/.bin/doc-finder
   ```

7. You should now be able to run the project:

   ```
   $ ./index.js
   usage: index.js MONGO_DB_URL PORT NOISE_FILE [CONTENT_FILE...]
   $
   ```

8. You will need to add routes to the provided `docs-ws.js` skeleton file. Start by adding a route for the **Get Content** service. You will need to use a simple pattern to match the document name. Within your handler, you can access what matched a pattern identifier by using that identifier as a property on `req.params`.

   First simply ensure that you can use the `docContent()` `DocFinder` method to return the requested content. Then return a JSON object as per the specs; you can use the provided `baseUrl()` function to help build the URL needed for the required `self` link in the response.

9. Implement the **Get Completions** service using the `complete()` `DocFinder` method. You can access query parameters as properties of the `req.query` object. Add verification to return a suitable error response when the `text` query parameter is missing.

10. Implement the **Search Content** service using the `find()` `DocFinder(¬)` method (note that the change in specs for `find()` from your previous project allows you to provide the `q` parameter directly as its argument). Note that you will need to add a suitable `href` property to each result returned by the `find()` method.

    You should probably split off validations for the query parameters and assembly of the response into auxiliary functions. You can use JavaScript's `Number()` function to convert from the incoming `String` parameters to numbers.

11. Implement the **Add Content** service using the `addContent()` `DocFinder` method.

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.