1 Project 2

Due Date: 10/11 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.

1.1 Aims of This Project

The aims of this project are as follows:

- To build a simple web-service.
- To get more experience in writing a non-trivial JavaScript program.
- To get some experience using the express.js framework.

1.2 Specifications

Write a command-line nodejs program which is invoked as:

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

This invocation should start a web server listening on port `PORT`. It should use the mongo database at `mongodb://localhost:27017/users` as its database.

The web server should respond to the following relative URL's (relative to the base url of scheme, hostname and port) and HTTP methods:

`PUT` `/users/`***ID*** The request must have a body which must be a JSON object. If the request is successful then it must store the entire JSON object so that it can be retrieved by the *ID* part of the URL.

If there was a previously created user for *ID*, then the server should update it to the JSON object provided in the request and return a status code of 204 `NO CONTENT`.

If there was no previously created user for *ID*, then the server must create a user for *ID* set to the JSON object and return a 201 `CREATED` status code with a `Location` header set to the request URL.

`GET` `/users/`***ID*** This request should return a JSON of the information previously stored under *ID*. If the user specified by *ID* is not found, then the server must return a 404 `NOT FOUND` status code.

`DELETE` `/users/`***ID*** This request should delete the object previously stored under *ID*. If the user specified by *ID* is not found, then the server must return a 404 `NOT FOUND` status code.

`POST /users/`***ID*** This request is used to update the user stored for *ID*. The body of this request must be a JSON object. If the *ID* field of the URL does not reference an existing user, then the server must return a 404 `NOT_FOUND` status code. Otherwise it should replace the user stored under *ID* with the contents of the JSON object in the body of the request and return a 303 `SEE OTHER` status code with `Location` header set to the request URL.

The program must report all errors to the client using a suitable *HTTP error code* as documented above. If there is an error during processing, then the server should return a 500 `INTERNAL SERVER ERROR` status code. Additionally, the program should log a suitable message on standard error (using `console.error()`).

1.3 Hints

You may do the project on any machine which has compatible nodejs and mongodb installations. However, it is entirely your responsibility to make sure that the code works on your gcloud vm before submission.

[Note that if you set up your gcloud vm as instructed, the only ports which are open are the HTTP (80) and HTTPS (443) ports; to run your server on the HTTP port (80), you will need to start your web server using `sudo`.]

Note that all the requests share the same URL. It may be a good idea to set up a *middleware handler* which runs before any route handlers to read the user specified by the *ID* field of the URL from the database (it will be ok for the user object not to be found, specifically for the `PUT` method which is used to create it).

The following steps are merely advisory:

1. Review material you will need for this project; the material for URLs, HTTP and web-services. This project can be done largely via informed reuse of code in the store example covered in class. Review the documentation on *express.js routes*.

2. Get started by using `npm init` to create a skeleton project. Install necessary modules like mocha, mongodb and express using the `--save` option. Ensure that downloaded modules are never checked into git by creating and updating a `.gitignore` file.

3. Use the mongo shell to prepopulate a suitable collection like `users` in a mongo database with suitable dummy data.

4. Write create, read, update and delete actions along the lines of your previous project for this collection.

5. Implement the `GET` action. This should merely entail setting up a suitable route and calling the read action you implemented in the previous step. Make sure you return a 404 if the id specified in the URL does not correspond to a saved user.

6. Implement the `DELETE` action. This should merely entail setting up a suitable route and calling the delete action you implemented earlier. Make sure you return a 404 if the id specified in the URL does not correspond to a saved user.

7. Implement the `PUT` action to create a new user. This should merely entail setting up a suitable route and calling the create action you implemented earlier.

8. Implement the `POST` action to update a user. This should merely entail setting up a suitable route and calling the update action you implemented earlier.

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

1.4 Debugging Your Code

Besides the debugging tools mentioned for the previous project, you may find Chrome's *restlet plugin* useful for testing your web services.

1.5 Project Submission

Submit your project using gitlab:

- Submit your project files in a top-level `submit/users` directory of your `cs480w` or `cs580w` project in gitlab.
- 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.

If your project is incomplete on the due date, please add a file called `.LATE` in your directory so that it will not be graded. Once the project has been completed late, please remote that file and email the grader who has been assigned to your course: *CS 480W*, *CS 580W*.