1 Project 3

Due Date: 11/6 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 authentication web-service.
- To develop and apply authentication protocols.

1.2 Specifications

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

```
./index.js [ -t|--auth-time AUTH_TIME ] [ -d|--ssl-dir SSL_DIR ] PORT
```

and results in a web server listening for HTTPS requests on port `PORT`. The program takes the following options:

`-t | --auth-time` The time in seconds before an authentication token times out. If not specified, the value should default to 300 seconds.

`-d | --ssl-dir` The path to the directory containing SSL credential files `key.pem` and `cert.pem`. If not specified, the value should default to the directory from which the server was started.

It should use the mongo database at `mongodb://localhost:27017/users` as its database.

The program should provide an authentication web service. Specifically, 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***`?pw=`***PASSWORD*** 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 after specifying the password *PASSWORD*.

If there was a previously created user for *ID*, then the server should return a status 303 `SEE_OTHER` with the `Location` header set to the absolute URL for `/users/`*ID*. It should return a body:

```
        { "status": "EXISTS",
  "info": "user <ID> already exists"
}
```

with <ID> replaced by the *ID* in the request URL.

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 Location header set to the absolute URL for /users/*ID* and with a JSON body set to:

```
{ "status": "CREATED",
  "authToken":‘ "<authToken>",
}
```

where <authToken> is a random authentication token (see below).

PUT /users/***ID***/auth with a JSON body which must be of the form {"pw": *PASSWORD* }.

If there was a previously created user for *ID* with password *PASSWORD*, then the server should return a status 200 OK with a response body:

```
{ "status": "OK",
  "authToken":‘ "<authToken>",
}
```

where <authToken> is a random authentication token (see below).

If the user specified by *ID* is not found, then the server must return a 404 NOT FOUND status code with a response body:

```
        { "status": "ERROR_NOT_FOUND",
  "info": "user <ID> not found"
}
```

with <ID> replaced by the *ID* in the request URL.

If the pw parameter is not present in the request body, or the *PASSWORD* is not correct, then the server must return a 401 UNAUTHORIZED response with a response body:

```
        { "status": "ERROR_UNAUTHORIZED",
  "info": "/users/<ID>/auth requires a valid 'pw' password query parameter"
}
```

with <ID> replaced by the *ID* in the request URL.

GET /users/***ID*** The request must have a Authorization: Bearer <auth-Token> header (case insensitive) where <authToken> is a currently valid authentication token.

If everything is valid, then this request should return a JSON of the information previously stored under *ID* when created using the PUT request.

If the user specified by *ID* is not found, then the server must return a 404 NOT FOUND status code with a response body:

```
        { "status": "ERROR_NOT_FOUND",
  "info": "user <ID> not found"
}
```

with `<ID>` replaced by the *ID* in the request URL.

If the `Authorization` header is not present, or the `<authToken>` is not valid, then the server must return a 401 `UNAUTHORIZED` response with a response body:

```
        { "status": "ERROR_UNAUTHORIZED",
  "info": "/users/<ID> requires a bearer authorization header"
}
```

with `<ID>` replaced by the *ID* in the request URL.

The program is also required to meet the following additional requirements:

- Passwords should not be stored in plain text.
- All generated authentication tokens should timeout after the time specified in the `-t` or `--auth-time` option specified on the command-line.
- The credential files needed for running a HTTPS server should be found in the directory specified by the `-d | --ssl-dir` command-line options.
- If there is an error not covered by the above specification, then the server should return a suitable HTTP response status with a JSON body containing a suitable `status` and `error` fields.
- If there is an error during processing, then the server should return a 500 `INTERNAL SERVER ERROR` status code with a suitable JSON body. Additionally, the program should log a suitable message on standard error (using `console.error()`).

1.3 Rationale

The required functionality provides the basis for an authentication web service:

`PUT` `/users/`***ID***`?pw=`***PASSWORD*** This service allows registering a user with id *ID*.

`PUT` `/users/`***ID***`/auth` This service can be used to allow a user to login, exchanging a password for an authentication token.

`GET` `/users/`***ID*** This service is an example of accessing a resource which requires authentication.

The authentication token times out after the specified period to make it hard to replay.

1.4 Provided Files

You are being provided with a options.js module which takes care of processing command-line options. It requires the use of the minimist npm module. You may run this module directly to see usage messages and what the exported options look like:

```
$ ./options.js
usage: options.js [ -t|--auth-time AUTH_TIME ] [ -d|--ssl-dir SSL_DIR ] PORT
$ ./options.js 443
{ port: 443, authTimeout: 300, sslDir: '.' }
$ ./options.js --auth-time 10 443
{ port: 443, authTimeout: 10, sslDir: '.' }
$
```

1.5 HTTPS Server

To set up a HTTPS server, you will need to generate a certificate. You can do so using the openssl program as follows:

```
$ openssl req -nodes -x509 -newkey rsa:2048 \
          -keyout key.pem -out cert.pem -days 100
Generating a 2048 bit RSA private key
.....+++
...........+++
writing new private key to 'key.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:NY
Locality Name (eg, city) []:Binghamton
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Binghamton University
Organizational Unit Name (eg, section) []:CS
Common Name (e.g. server FQDN or YOUR name) []:<YOUR IP ADDRESS>
Email Address []:<YOUR EMAIL ADDRESS>
$
```

It is important to specify `<YOUR IP ADDRESS>` as the IP address for your gcloud vm. The other parameters are not terribly important for a development certificate and can be left blank (by entering a .),

Running the above command will generate two files in the current directory:

`key.pem` A file containing the key for the certificate. This file is not password protected (the no DES `-nodes` option above).

`cert.pem` The certificate file.

To set up express to run a HTTPS server, you will need code along the lines of:

```
const https = require('https');
const express = require('express');
const app = express();
...
https.createServer({
  key: fs.readFileSync(KEY_PATH),
  cert: fs.readFileSync(CERT_PATH),
}, app).listen(PORT);
```

where `PORT` is the port the server is required to listen to, `KEY_PATH` and `CERT_PATH` are the paths to the previously generated `key.pem` and `cert.pem` files.

1.6 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 HTTPS port (443), you will need to start your web server using `sudo`.]

The following steps are merely advisory:

1. Review material you will need for this project; the material for URLs, HTTP, web-services, authorization and authentication.
2. Decide how you will store the password. Based on the material discussed in class, minimally you should hash the password. Decide on which hash function to use and install (using `--save`) a suitable module. Some considerations when deciding on or choosing an external module:
   - Does the module provide functionality which is non-trivial and difficult to implement by yourself? This will definitely be true for a hashing module.
   - How popular / mainstream is the module?
   - Is the module currently maintained?
   - How many committers does the module have?
   - Is the module backed by a major organization?

- How technically attractive is the module?

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

4. Implement a model which provides the functionality necessary for this project.

5. Implement the server. Follow the above procedure to set up a HTTPS server. Try out a simple `hello world` test route to verify that you can access your server. Note that since your certificate is not signed by a known certificate authority you will need to tell the client you are using that it should accept the certificate.

6. Implement the `PUT` action to create a new user. You may choose not to implement the password functionality at this stage and simply return an authentication token.

7. Implement the `GET` action to get a previously created user using the returned authentication token.

8. If you did not implement the password functionality for the `PUT` action, implement it now.

9. Implement the `PUT` action to create a new authentication token.

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

1.7 Project Submission

Submit your project using gitlab:

- Submit your project files in a top-level `submit/auth` 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*.