2022

CAB230 REST API – Server Side

CAB230 Volcano API – The Server Side Application

Harrison Leach

N11039639

6/6/2022

Contents

Introduction

Purpose & description

This assignment involves itself with the development and deployment of a server-side application using a NodeJS framework, Express. This application is the server that houses the Volcano REST API. It has been enhanced from the API the React application used in Assignment One. The database containing information on volcanoes and users was stored in MySQL and was manipulated with Knex in the program. A Swagger page was included for future developers to make use of the created endpoints. The JSON Web Tokens were used to create authenticated tokens for users and logging was performed via Morgan. Upon completion, the server was deployed to a QUT Linux Virtual Machine.

Completeness and Limitations

Comparing against the standards given by the assignment specification, this application is within the 6-7 grade as all endpoints perform as intended. The list of endpoints in the API are below and simply labelled '*Fully functional*'.

/countries

Fully functional

/volcanoes

Fully functional

/volcano/{id}

Fully functional

/user/register

Fully functional

/user/login

Fully functional

/user/{email}/profile

Fully functional

/me

Fully functional

Modules used

Helmet

This module protects the Express app from web vulnerabilities by setting HTTP headers.

https://www.npmjs.com/package/helmet

Luxon

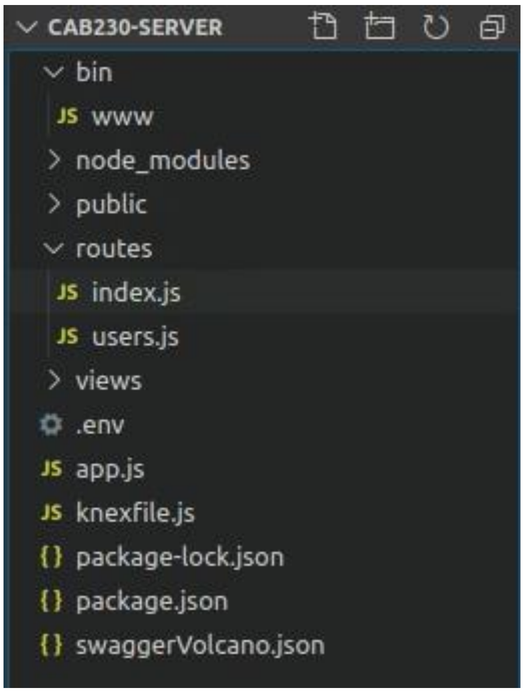
Module to provide great assistance with date and times in JavaScript.

https://github.com/moment/luxon/

Technical Description

Architecture

The architecture of this server is based upon the initial layout of a generic express application. Regarding the endpoints, the importance of this application occurs in the index.js and users.js files, found in the routes folder. The index file uses '/' which allows the index to house the /countries, /volcanoes, /volcano/{id} and /me endpoints. In users.js, '/user' is used, this is where /user/login, /user/register and user/{email}/profile routes are handled. Connection to the MySQL database is done via Knex, this is found in the knexfile.js. The JSON for the Swagger page is in the file called swaggerVolcano.json. Both the Knex and Swagger connections are done in the app.js



```
CAB230-SERVER
  bin
    www
  node_modules
  public
  routes
    index.js
    users.js
  views
  .env
  app.js
  knexfile.js
  package-lock.json
  package.json
  swaggerVolcano.json
```

Figure 1: Source Code Architecture

Security

The layers of security for the application are as followed. The application makes use of Knex which one of the advantages of this is it avoids the use of raw SQL, which is prone to SQL injections. Morgan was applied as a means of logging the applications status. The terminal on the server's system is what receives these logs. Passwords were handled appropriately as they were encrypted and stored as a JSON Web Token (JWT). Finally, the API was deployed with a self-signed TLS certificate which can prevent data breaches and other hacks. The helmet module – a module with powerful security features – was also included. The default status of the module and what features it implements is included in the appendix.

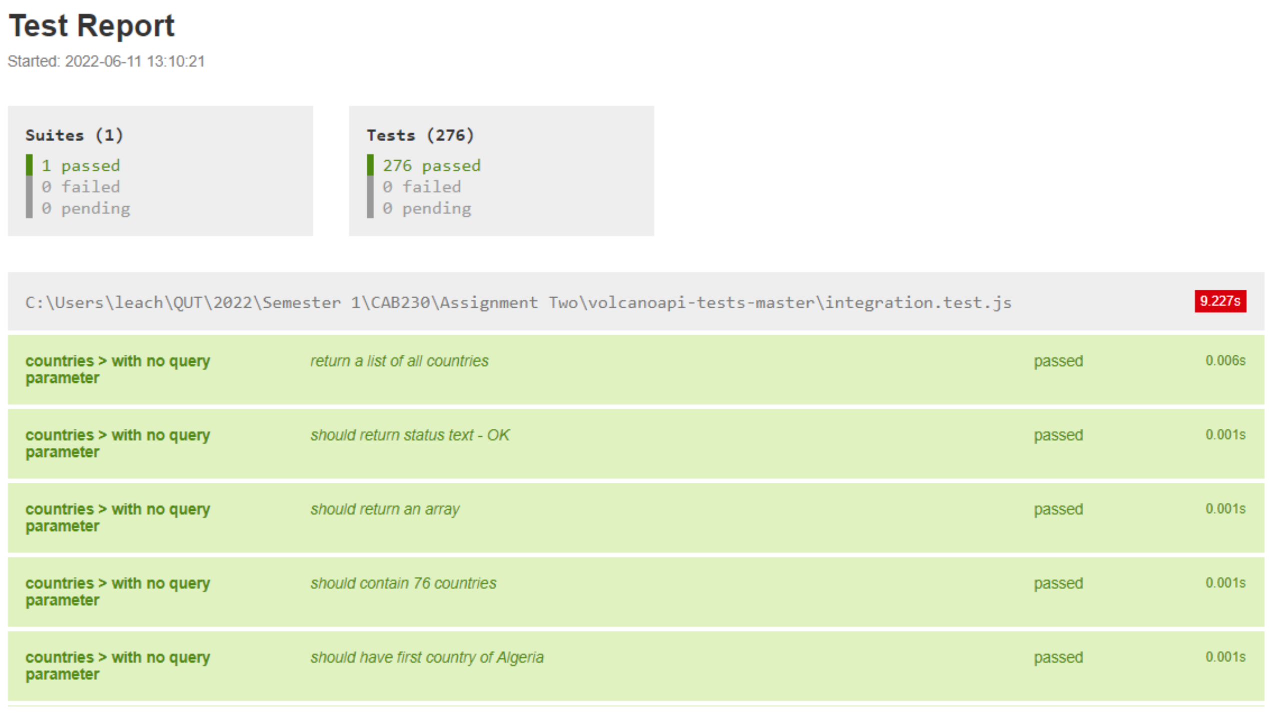
Below is an evaluation of the application's security strengths and weaknesses in comparison to OWASP's (Open Web Application Security Project) Top Ten list.

- Injection – Occurs almost every time data is involved within an application. It can result in data loss, corruption, denial of access, etc. As stated above Knex's query builder does a sufficient job of preventing this.
- Broken Authentication/Access Control – This occurs when an attacker obtains a session token that does not belong to them. By checking the token with JWT, the application can confirm

the user is valid or not. This has been done manually and is likely vulnerable to unforeseen methods of getting past this security.

- Sensitive Data Exposure – This issue is when an attacker retrieves important data such as a password or credit card information. Commonly due to this data not being encrypted when stored to the database. This application does hash passwords when being stored, however industry standards are bound to be updated and the application will comply to inevitable changes.
- XML External Entities – An attacker can upload malicious XML, which can be used to extract data, scan internal systems, an execute other fatal attacks. This is an issue for the application as there hasn't been any implementations to avoid this.
- Security Misconfiguration – Essentially if example/test applications are still running on a deployed server, malicious users can gain the ability to access system functionality without authorization. This should not be an issue as there're no access points at this level of vulnerability that the application is aware of.
- Cross-Site Scripting – This can happen when an attacker attempts to give a user a link to a seemingly identical application that is in fact dangerous. The way to get around this is by obtaining a TLS certificate to use HTTPS which proves that our product is the true one and not trying to steal data. This has been done with the use of a self-signed certificate which doesn't appear as trustworthy on most browsers, so an upgrade for the future would be to get a CA (Certificate Authority) approved certificate instead.
- Using Components with Known Vulnerabilities – this issue is straight forward, and it is important that the most recent version of well-known libraries is used for development.
- Insufficient Logging & Monitoring – It is vital that the logging of the server is prevalent after the deployment of the API, as attackers depend on poor monitoring to execute vulnerability probing. If probing is successful, a full breach is almost guaranteed to work.

Testing



Test Report

Started: 2022-06-11 13:10:21

```
Suites (1)
1 passed
0 failed
0 pending
```

```
Tests (276)
276 passed
0 failed
0 pending
```

```
C:\Users\leach\QUT\2022\Semester 1\CAB230\Assignment Two\volcanoapi-tests-master\integration.test.js    9.227s
```

countries > with no query parameter	*return a list of all countries*	passed	0.006s
countries > with no query parameter	*should return status text - OK*	passed	0.001s
countries > with no query parameter	*should return an array*	passed	0.001s
countries > with no query parameter	*should contain 76 countries*	passed	0.001s
countries > with no query parameter	*should have first country of Algeria*	passed	0.001s

Figure 2: Testing Report

Difficulties / Exclusions / unresolved & persistent errors /

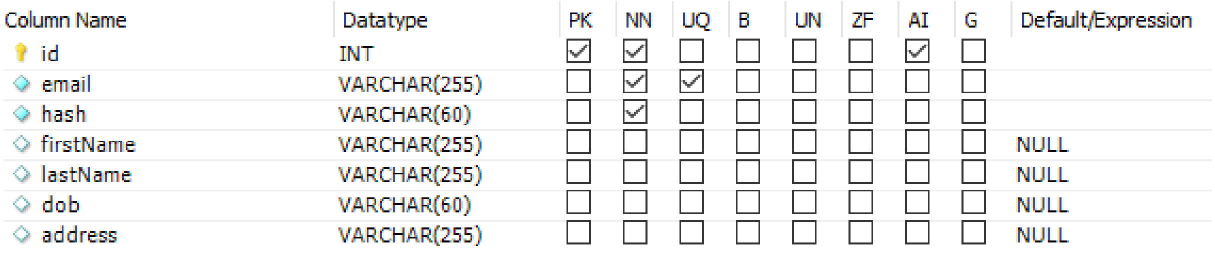
The main difficulty that occurred during development of this application was consistent challenges at each endpoint. These obstacles are at first difficult to solve but after some researching and persistence they were no longer a nuisance.

Extensions

A possible extension could be to add an *admin* user type. These users could potentially have the ability to update a user's profile information without having to be the user themselves. They could also see all information of any profile once again without having to be the user in question.

Installation guide

To begin, obtain a dump SQL file containing volcano data. Using MySQL, create the *volcanoes* database on the desired server. Create another table named *users* with the following columns:



Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id	INT	✓	✓					✓		
email	VARCHAR(255)		✓	✓						
hash	VARCHAR(60)		✓							
firstName	VARCHAR(255)									NULL
lastName	VARCHAR(255)									NULL
dob	VARCHAR(60)									NULL
address	VARCHAR(255)									NULL

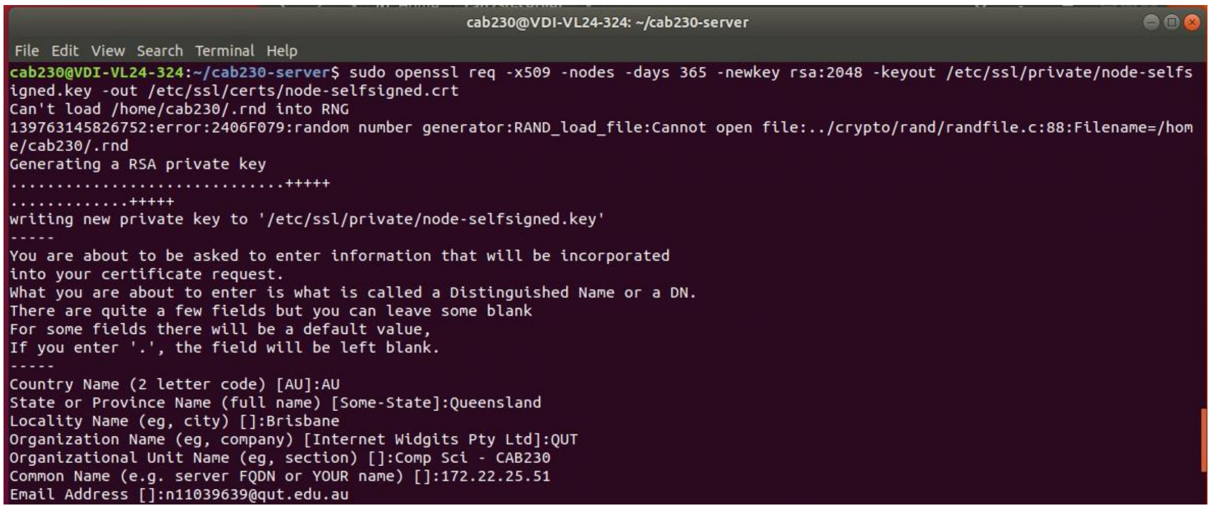
Figure 3: Users Table Configuration

Place the code in a file location of your choice. In the server's terminal, change your directory to this file location and enter the following:

npm install --save

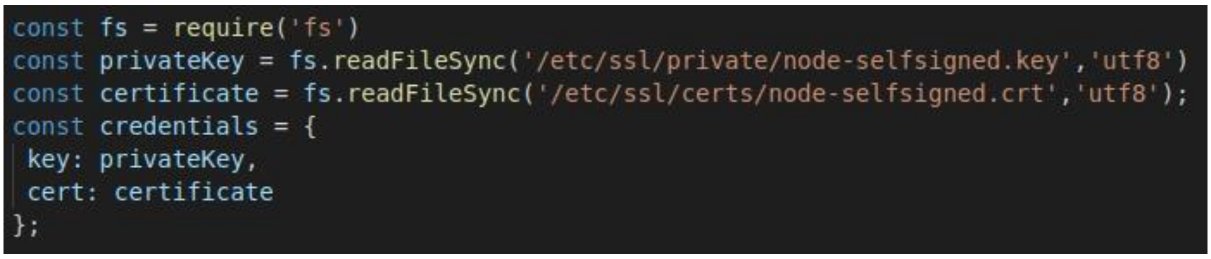
This retrieves the *node_modules* that are necessary for the API to work.

To get a self-signed HTTPS certificate a key and certificate pair must be given. Keys and certificates are stored in defined locations in filesystems, so depending on the OS that your server is using you will need to store your keys and certificates there. An example for a Linux Ubuntu server is shown below:


cab230@VDI-VL24-324: ~/cab230-server
File Edit View Search Terminal Help

```
cab230@VDI-VL24-324:~/cab230-server$ sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/private/node-selfs
igned.key -out /etc/ssl/certs/node-selfsigned.crt
Can't load /home/cab230/.rnd into RNG
139763145826752:error:2406F079:random number generator:RAND_load_file:Cannot open file:../crypto/rand/randfile.c:88:Filename=/hom
e/cab230/.rnd
Generating a RSA private key
.............................+++++
.............+++++
writing new private key to '/etc/ssl/private/node-selfsigned.key'
-----
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]:AU
State or Province Name (full name) [Some-State]:Queensland
Locality Name (eg, city) []:Brisbane
Organization Name (eg, company) [Internet Widgits Pty Ltd]:QUT
Organizational Unit Name (eg, section) []:Comp Sci - CAB230
Common Name (e.g. server FQDN or YOUR name) []:172.22.25.51
Email Address []:n11039639@qut.edu.au
```

Figure 4: Certificate & Key Terminal Command



```
const fs = require('fs')
const privateKey = fs.readFileSync('/etc/ssl/private/node-selfsigned.key','utf8')
const certificate = fs.readFileSync('/etc/ssl/certs/node-selfsigned.crt','utf8');
const credentials = {
  key: privateKey,
  cert: certificate
};
```

Figure 5: Code Handling Keys & Certificates

NOTE: The file path for keys and certificates will need to be changed within the code, depending on the filesystem as stated above. (Found in bin/www.js)

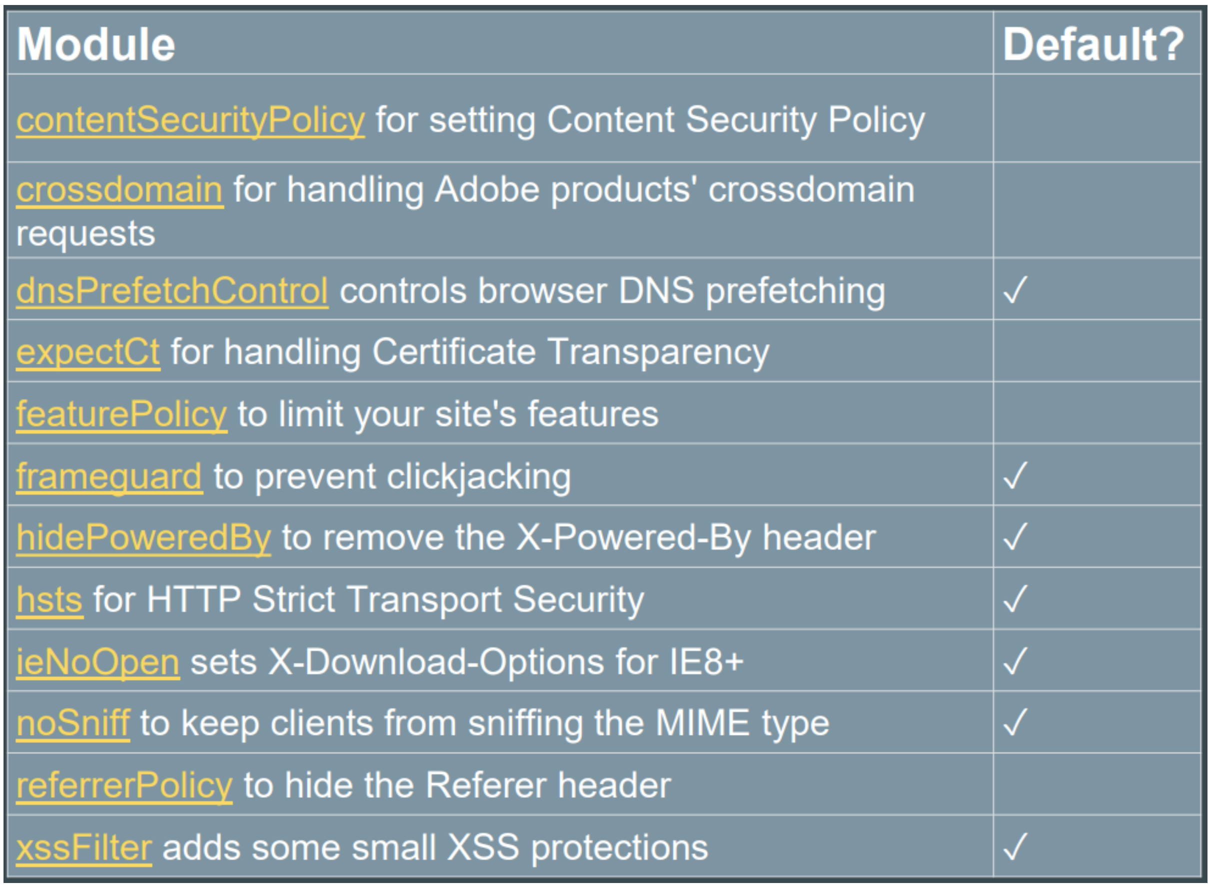
When the above has been completed, the server is ready to start. In the terminal enter:

npm start

To access the application, open the following URL in your browser of choice:

https://[NAME_OF_YOUR_SERVER]:443

Appendix



Module	Default?
contentSecurityPolicy for setting Content Security Policy	
crossdomain for handling Adobe products' crossdomain requests	
dnsPrefetchControl controls browser DNS prefetching	✓
expectCt for handling Certificate Transparency	
featurePolicy to limit your site's features	
frameguard to prevent clickjacking	✓
hidePoweredBy to remove the X-Powered-By header	✓
hsts for HTTP Strict Transport Security	✓
ieNoOpen sets X-Download-Options for IE8+	✓
noSniff to keep clients from sniffing the MIME type	✓
referrerPolicy to hide the Referer header	
xssFilter adds some small XSS protections	✓

Table 1: Helmet Module Default Performance