2022

CAB230 REST API – Server Side

CAB230 Volcano API – The Server Side Application

<Lingdong Guo>
<n10840656>
6/8/2022

Contents

This template is adapted from one created for a more elaborate application. The original author spends most of his professional life talking to clients and producing architecture and services reports. You may find this a bit more elaborate than you are used to, but it is there to help you get a better mark

This report will probably be around 5 pages or so including screenshots

Introduction

Purpose & description

It is a server-side application that functions as an API. By using Express applications, React applications can request volcanoes and their detailed information. Swagger documentation is also used to make it easy for users to use the application for testing. The proper channels for requesting these resources will be explained later. The JSON string containing all the information will be sent by the application. The Express application is also secure since various technologies and resources have been implemented in order to provide both application and user security. It is possible for the application to handle errors, for instance, if someone types in and wants information on an entity that doesn't exist, they will be shown the appropriate information, as some examples. It is possible for the application to handle errors and respond with successful resources from the API. To provide services, this Express application utilizes a range of different technologies, packages, modules, libraries, and other tools, which are all discussed below.

Completeness and Limitations

All routes were successfully implemented, and all of the required functionality was completed. The application runs on a virtual machine provided by the CAB230 teaching team using Horizon VM. The application serves the swagger docs on the home page or at the root level. The application can implement all the requirements including node, express, mysql, swagger (html source was supplied), knex, helmet, morgan, jwt, and others. All of these resources helped in the development of this application. Every aspect of the login process, registration process, and any other requirements were met.

/countries

The endpoint is fully functional

/volcanoes

The endpoint is fully functional

/volcano/{id}

The endpoint is fully functional

/user/register

The endpoint is fully functional

/user/login

The endpoint is fully functional

/user/{email}/profile

The endpoint is fully functional

/me

The endpoint is fully functional

Modules used

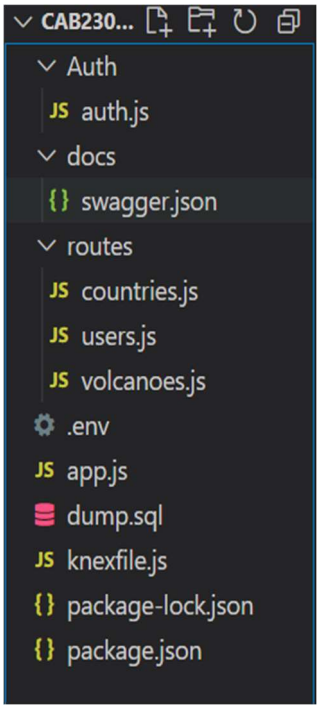
Module to provide valid date checking, especially for invalid leap years

https://momentjs.com/

Technical Description

Architecture

The architecture of the application is professionally organised and laid out. There is a folder called assign which contains an item called CAB230-Assignment 2. In this folder, there are folders and files for the application. All these files include: auth.js file inside the auth directory that helps to get authorize of endpoint /volcano/{id} and /user/{email}/profile. The knexfile.js, which uses knex to connect to the database. Npm installs node_modules when running npm install, these modules are crucial to the project's performance. The dump.sql has been added in the root level for easy access. The routes directory controls almost all of the endpoints. The docs directory contains the swagger documentation needed for this application. The .env file contains information about some of the variables used by the application. app.js is the main file, all the routing and middleware is installed in the app.js file and npm will run this documentation when typing sudo npm start. In addition, app.js controls the endpoint /me. The package-lock.json is also in the main directory and is used by the application. The the package.json file contains all the modules and its version is written here. As can be seen in the screenshot.



About the use of the application. As can be seen in the screenshot, app.js loads all the application middleware and route here. app.js files use the route directory and the files in it, which are users, countries and volcanoes. The application uses these files for all endpoints such as /countries, /volcanoes, /user/register, /user/login, /user/{email}/profile and /volcano/{id}.

```
const logger = require('morgan');
const cors = require('cors');
const helmet = require('helmet');
require('dotenv').config();
const express = require('express');

const swaggerUi = require('swagger-ui-express');
const cookieParser = require("cookie-parser")
const app = express();
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(logger('dev'));
app.use(cors());
app.use(helmet());
const knexOptions = require('./knexfile.js');
const knex = require('knex')(knexOptions);
app.use(cookieParser());
const swaggerDoc = require('./docs/swagger.json');

const countryRouter = require('./routes/countries');
const volcanoRouter = require('./routes/volcanoes');
const userRouter = require('./routes/users');

app.use("/", swaggerUi.serve);
app.get(
  "/",
  swaggerUi.setup(swaggerDoc, {
    swaggerOptions: { defaultModelsExpandDepth: -1 },
  })
```

Security

Middleware functions were incorporated into the app to enhance its security features and capabilities. Injection attacks cannot occur using App.js since raw SQL is not used for the building of SQL queries. Connecting the app to the SQL database has been done via a knex.js file. Also, App.js' HTTP headers are encrypted through helmet. Furthermore, App.js uses the Morgan logger in its development mode for logging. When a request is made, Morgan's logging system automatically generates logs. Passwords are handled safely and securely. By hashing the password using bcrypt and creating a token with Jason web token, a token for the user's password is not directly saved. Tokens with the email address, expiration date, and secret key are instead created from the password hash. In addition, salt rounds are used to ensure that users with the same password are not given the same hash value. To further avoid saving passwords directly at any point where they are used, bcrypt is also used to compare hashed passwords. Security is guaranteed through the use of https and SSL during deployment. Using self-signed keys and certificates, the app encrypts data sent to users and saves the SMS key securely on the server to encrypt that data, while the SMS certificate is available to all users, enabling them to decrypt the content received. Utilizes HTTPS rather than HTTP, a more secure and modern protocol. Security features are implemented in the app, but they are basic in scope and implementation. Despite the scope of attack types, more needs to be done to improve the app's security. The following paragraphs provide a glimpse of a few of the top threats that the Open Web Application Security Project has identified. A server can query a database, for instance, and injection attacks exploit these communications by inputting queries into the URL parameter to be able to use these values in an unexpected way. knex addresses the injection security threat as it is a SQL query builder that uses bindings to run SQL queries and queries which prevent injection attacks. This function could have made more use of knex and its capabilities. An impersonation attack can occur when a website's authentication process is compromised. It can also happen when session management weaknesses are exploited, such as when attackers have

access to a database with valid usernames and passwords. Validation tokens are all equipped with expiration timers, so if used for a period of time, the token will no longer be valid to achieve a successful attack. By reducing the time the token can be used, the chance of a successful attack is minimized. Despite the fact that the expiration times are currently 24 hours, the time could be shortened in order to limit attacks. Furthermore, authentication has been added to check the JWT tokens against the email address that the user provides for some routes. The storage of JWT tokens in conjunction with HTTP cookies can improve authentication. When a server passes sensitive data to an attacker, it creates a security risk. All passwords in the app are encrypted, meaning that even if an attacker got hold of one it would take an unreasonable amount of time to decrypt it. To make sure the password itself is not used or saved in its plain form in the future, password comparisons are also done using encryption, which ensures that two or more passwords with the same hash are not returned. By exploiting the access control of a server, attackers can obtain administrative accounts or privileges. A mitigation strategy in the app is the checking of parameters and queries. The type and length of parameter and query values for most endpoints with parameter options are carefully checked to prevent the input of unexpected values. The parameters and queries could be used only for their intended purposes if more is done to ensure this. A security misconfiguration occurs when assets are not monitored or used, and unchecked errors are present. By avoiding unused pages in the app and redirecting samples routes to the swagger documentation, the app prevents this threat from occurring. An attacker can use cross-site scripting to attack victims by sending a script-injected version of a real application. An unsuspecting victim may think these altered applications are genuine, since they typically have malicious scripts. In order to prevent this, helmet sets appropriate HTTP headers as well as HTTP response headers related to security. In spite of helmet being implemented, only its default modules are used, so many more modules could be activated and implemented. A common issue that plagues component-heavy applications is misunderstood and forgetting to protect the components. It can lead to security issues as most components are run on similar privileges as the app, so a crack in one component could lead to an entire app being compromised. It is possible to solve this problem by having the latest version of libraries in the app that implements them. NPM audit and NPM fix can be used to resolve dependency issues even though they are not used in this app. As stated in the article, inadequate logging and monitoring is a security risk. Morgan has been integrated into the app to allow for increased monitoring and reporting. When Morgan receives a request, it automatically generates a log. Only Morgan's development mode is used, which may not provide enough logging and monitoring as a result.

Testing

Difficulties / Exclusions / unresolved & persistent errors /

No bugs are present in the application. No errors are present in the application. All the application's required functionalities have been completed.

Installation guide

Step 1: Download the assign.zip folder and unzip it

Step 2: Open a terminal within the folder

1. Unzip the folder if required.
2. Right-click on the wallpaper and select "Open terminal", then type cd CAB230-Assignment2.

Step 3: Create the mySQL database

1. Type "sudo mysql -u root -p", then enter the Virtual Machine's password.
2. Use "CREATE DATABASE volcanoes_db; "to generate a new database, and USE "volcanoes_db;" to change to that database.
3. Type source "dump.sql" to generate and populate the database with necessary information.
4. Use "show tables;" to ensure this command worked.

Step 4: Then open the assign.zip folder in command line, and you need to type the following command: "npm install" to install all the packages requried for this project.

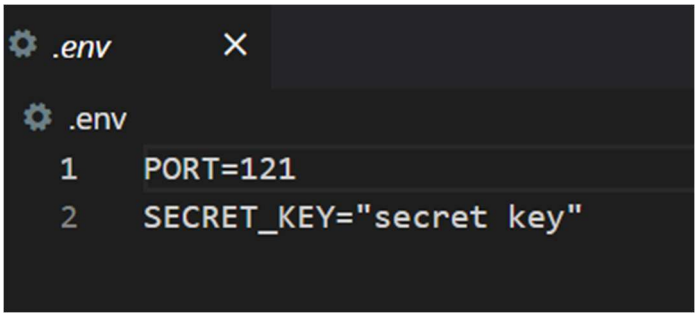
Step 5: Create self-signed keys**.** Run the following command:" sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/private/node-selfsigned.key -out /etc/ssl/certs/node-selfsigned.crt."

Step 6: Within the base directory of the app, create a file entitled .env as the screenshot below, and paste the following contents:

PORT=121

SECRET_KEY="secret key"

Sometimes the .env files disappear when zipping and transferring.



Step 9: Run the server

1. cd ...
2. sudo chown root CAB230-Assignment2
3. cd CAB230-Assignment2.

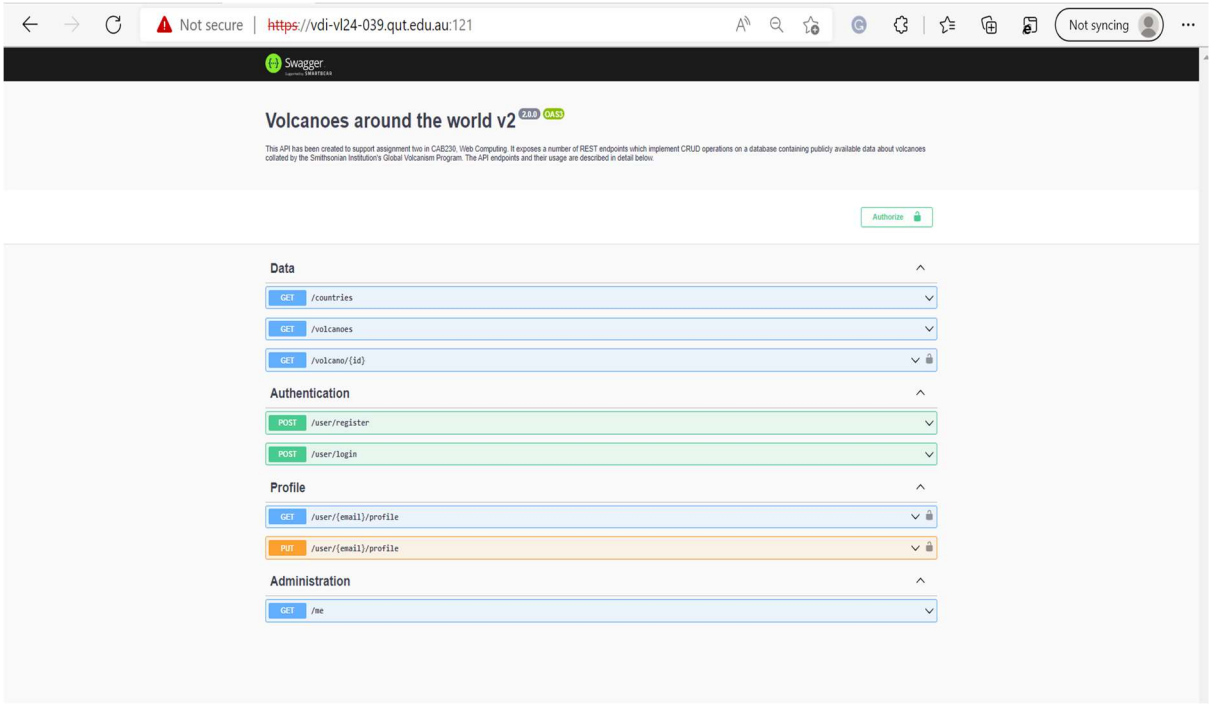
4. sudo npm start
5. enter the Virtual Machine's password

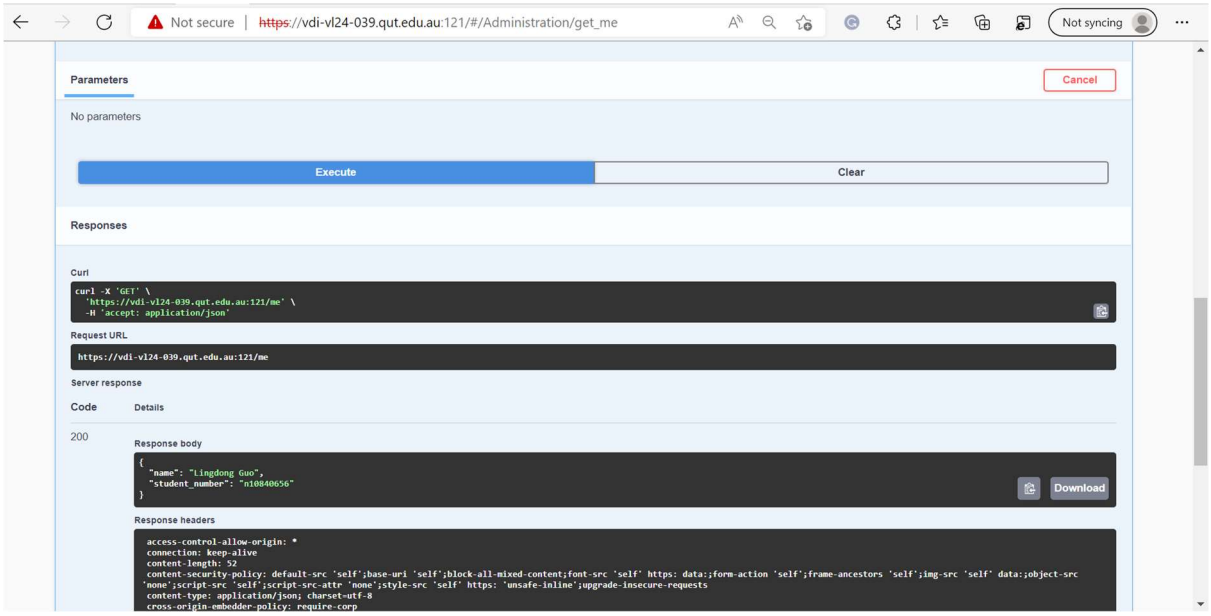
Extensions (Optional)

Extensions of this API include adding more endpoints to increase functionality, especially to include an account delete function, and to add additional security measures to ensure safe and secure traversal of the app.

Appendices as you require them

Anything you think should be included but is better left to the end.





{"error":true,"message":"Page not found!"}