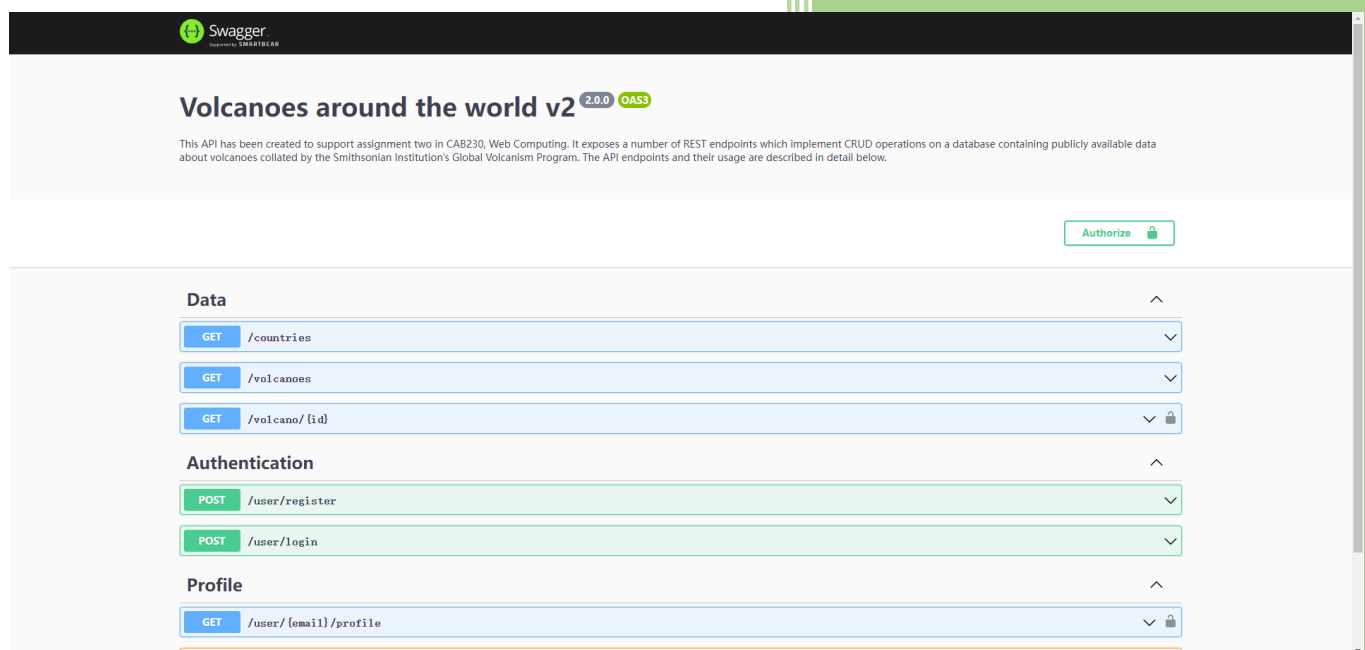


2022

CAB230 REST API – Server Side



The image shows the Swagger UI for the 'Volcanoes around the world v2' API. The header includes the Swagger logo and the API title 'Volcanoes around the world v2' with version '2.0.0' and 'OAS3' specification. A description states: 'This API has been created to support assignment two in CAB230, Web Computing. It exposes a number of REST endpoints which implement CRUD operations on a database containing publicly available data about volcanoes collated by the Smithsonian Institution's Global Volcanism Program. The API endpoints and their usage are described in detail below.' An 'Authorize' button is visible. The endpoints are categorized into three sections: 'Data' (GET /countries, GET /volcanoes, GET /volcano/{id}), 'Authentication' (POST /user/register, POST /user/login), and 'Profile' (GET /user/{email}/profile).

Swagger
Volcanoes around the world v2 2.0.0 OAS3

This API has been created to support assignment two in CAB230, Web Computing. It exposes a number of REST endpoints which implement CRUD operations on a database containing publicly available data about volcanoes collated by the Smithsonian Institution's Global Volcanism Program. The API endpoints and their usage are described in detail below.

Authorize

Data

- GET /countries
- GET /volcanoes
- GET /volcano/{id}

Authentication

- POST /user/register
- POST /user/login

Profile

- GET /user/{email}/profile

CAB230 Volcano API – The Server Side Application

<student name> Haobo Jin

<student number>n10642536

6/12/2022

Contents

Introduction	2
Purpose & description.....	2
Completeness and Limitations.....	2
/countries.....	2
/volcanoes.....	2
/volcano/{id}	2
/user/register	2
/user/login	2
/user/{email}/profile	2
/me.....	2
Modules used.....	3
Ag-grid-react	错误!未定义书签。
Module 1	3
Module n.....	错误!未定义书签。
Technical Description.....	3
Architecture	3
Security	3
Testing.....	4
Difficulties / Exclusions / unresolved & persistent errors.....	5
Extensions (Optional).....	错误!未定义书签。
Installation guide	5
References	5
Appendices as you require them	7

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

Applications are volcanic database application programming interface (API) that allow users to access database data through an interface. The volcano database contains information about each volcano. The API allows users to retrieve city names, Volcano lists, and specific volcano details from a database. It also allows users to register and log into accounts to access endpoints that require authorization. In some breakpoints, the data obtained by authorized and unauthorized users is different. The application also allows users to update and get information about a given user. The Swagger interface information visualization breakpoints.

Data		^
GET	/countries	▼
GET	/volcanoes	▼
GET	/volcano/{id}	▼ 🔒
Authentication		^
POST	/user/register	▼
POST	/user/login	▼
Profile		^
GET	/user/{email}/profile	▼ 🔒
PUT	/user/{email}/profile	▼ 🔒
Administration		^
GET	/me	▼

Completeness and Limitations

The result of the test is that no errors are found, proving that all breakpoints are eligible. All breakpoints return the correct response data. The application uses tokens for authentication and uses Morgan, Helmet, knex, and Cors Middleware to secure the API. Connect to the SQL database and return data using KNEX middleware. The Swagger file has been successfully deployed.

[/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

Moment

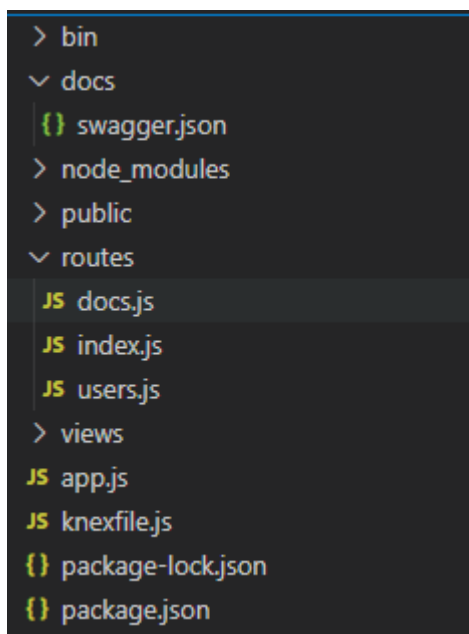
The module provides effective date format checking and transformation of the date field obtained from SQL into YYYY-MM-DD format

<https://momentjs.com>

Technical Description

Architecture

The structure of the application is based on the Express-Generator application, on which I added several files. First, I add a knexfile.js file which connects to the SQL database and enables the application to interact with the SQL database. Create a new docs folder for Swagger. json file, Added a docs. js file acts as the router, responsible for the swagger interface page display.



The routing folder contains routing index.js, users.js, docs.js. Index.js contains all the router related to volcano data, including /countries/volcanoes, and /volcano {id} breakpoints, allowing the API to interact with the data table. Users.js contains all routers related to user login registration and authentication, including user/register, user/login, user/{email }/profile, user/{ email }/profile breakpoints. It realizes the interaction between API and Users table, the generation and validation of Token, the generation and modification of user information. docs.js implements the deployment of the swagger page.

Security

To ensure the security of the application, app uses several middleware. Knex middleware is used in the SQL query build so that the original SQL does not have to interact directly with the database. This prevents SQL injection attacks by default and prevents others from inserting SQL commands into the Query string of a post get web form or page request, causing the web server to execute malicious commands.

Morgan middleware can be used to log HTTP requests and errors. Morgan logs are automatically generated when a client makes a request. Use helmet middleware to secure the HTTP headers returned by the application.

The application handles the passwords safely and securely. When app processes a user's password, it does not store the user's password directly in the SQL database. Instead, it hashes the password using the BCRYPT function and stores it in the database. Putting passwords directly into the database is highly insecure because anyone who can open the database will be able to see them directly. So use the password with hash function encryption and then stored into the database to save the password. The hash function is irreversible, and even if someone opens the database, they can't see what the user's password is. When the user logs in, app gets the password entered by the user and calls bcrypt again to compare the password to the hash value in the database. In addition, the app uses JWT to create tokens for logged-in users and encrypt user information. And use jwt.sign function signs e-mail, expire, and secret keys to protect the data from tampering.

Deploying applications with HTTPS and SSL ensures application security. HTTPS encrypts information so that sensitive information cannot be accessed by third parties. SSL adds a layer over HTTP to the module that handles encrypted information. Both server and client transmissions are encrypted by TLS, so the data is encrypted.

Although apps implement many security features, there are still many threats to consider. Here is a brief overview of some of the top-level threats to open Web application security projects.

1. Injection attacks

An injection attack is an attack in which code is added to an input parameter and passed to the server for parsing and execution. An attacker can use SQL injection to bypass the authentication and authorization mechanisms of a Web application and retrieve the contents of an entire database. *To avoid SQL injection flaws is simple. Developers need to either: a) stop writing dynamic queries; and/or B) prevent user-supplied input which contains malicious SQL from affecting the logic of the executed query.* in app, I used KNEX middleware to prevent SQL injection attacks (Lifars,2020)

2. Broken authentication

Broken authentication is typically caused by poorly implemented authentication and session management functions. Broken authentication attacks aim to take over one or more accounts giving the attacker the same privileges as the attacked user. Authentication is "broken" when attackers are able to compromise passwords, keys or session tokens, user account information, and other details to assume user identities. (Contrast.2017) In short, when a hacker exploits a vulnerability in an online platform to gain access to a system administrator account, it is called breaking authentication. Broken Authentication may result in the theft of critical business data and the leakage of personal information. Broken Authentication can be prevented by using SMS authentication to authenticate the user, forcing the user to use a combination of lowercase letters, majuscule, alphanumeric symbols, and special characters when creating a password, enhance password complexity, and so on.

In app, we use JWT authentication to encrypt the user's identity to avoid this.

3. Sensitive Data Exposure

The disclosure of sensitive data is a security risk that can occur when a server delivers information that is intended to be stolen by an attacker. Sensitive Data Exposure results in the theft of important information and the theft of credit card information. To avoid Sensitive Data Exposure, users can use encryption algorithms in transmission and storage, such as the hash algorithm used in apps, which can take an unreasonable amount of time for an attacker to decrypt Data.

In our application, this was avoided by HTTPS deployment and storing the password after hash processing.

Testing

Test Report

Started: 2022-06-11 19:20:47

Suites (1) 1 passed 0 failed 0 pending	Tests (276) 276 passed 0 failed 0 pending
D:\homework\cab230\volcanoapi-tests-master\volcanoapi-tests-master\integration.test.js	
2.609s	
countries > with no query parameter	return a list of all countries
passed	0.002s

Difficulties / Exclusions / unresolved & persistent errors /

Difficulties / Exclusions / unresolved & persistent errors /

Outstanding Bugs (solved): when querying a profile in SQL, we look forward to return personal information about a particular user, such as name and date. But the SQL library returns a date attribute like 2021-06-26T00.00.000Z, of type undefined. To fix the bug, I had to call the external moment library to turn the data property field into a string field of type“YYYY-MM-DD”.

Extensions (Optional)

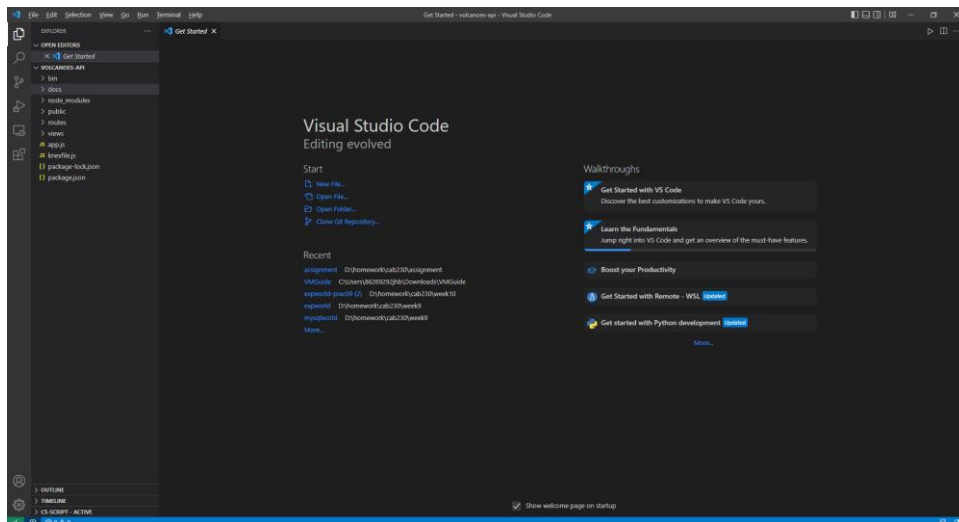
Where could you take this:

tell us about Potential future extensions / improvements for the API

Installation guide

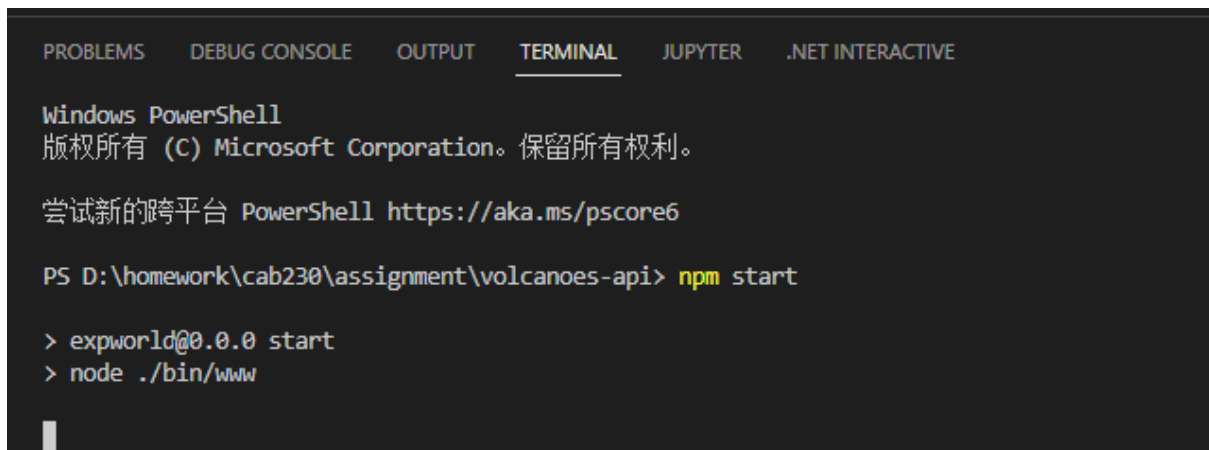
Step1: Download and unzip the assign zip file

Step2: use vscode to open the volcanoes-api file

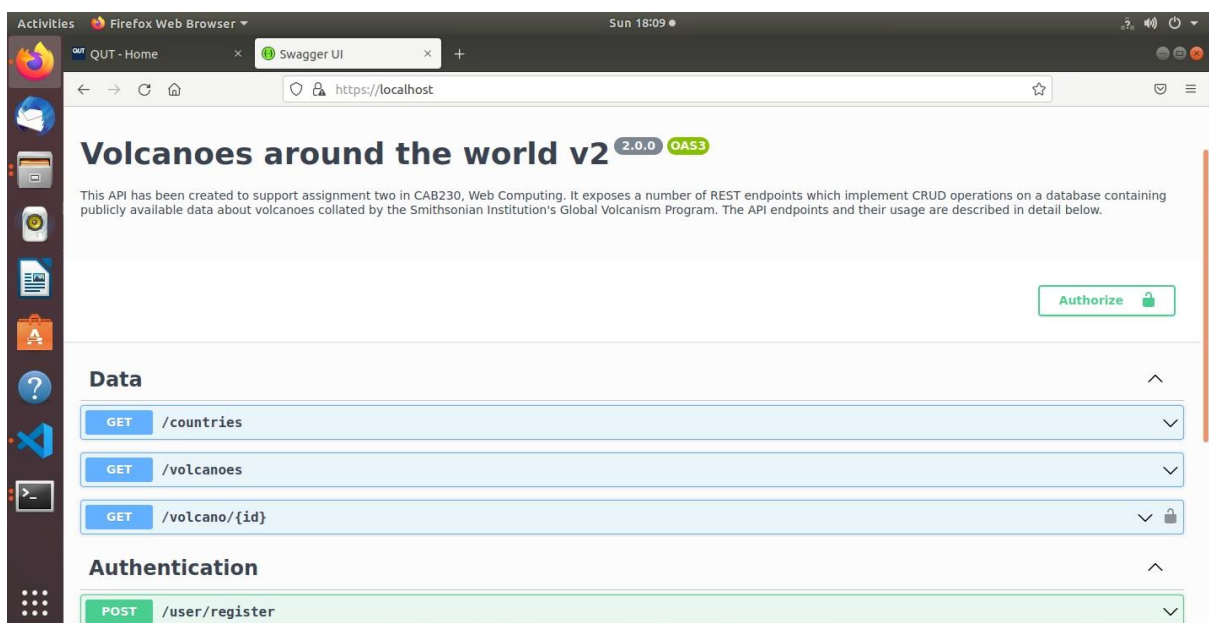


Step 3: Run 'npm install' command in the terminal

Step4: Run 'npm start' command to start the server



Enter the url <http://localhost> in website



References

Lifars.(2020) Injection Attacks Explained. <https://www.lifars.com/2020/04/injection-attacks-explained/>

Contrast.(2017). Broken authentication. <https://www.contrastsecurity.com/glossary/broken-authentication>

Appendices as you require them

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