CAB230 Assignment 2, 2022
Server Side

1. Introduction

In assignment one for CAB230 you developed a React application that allowed users to view and analyse data about volcanoes that we exposed via a REST API. In assignment two, your task here is to write and deploy an Express application that replicates, and in some cases extends, the REST API.

The extended API for this assignment is published and documented at: **http://sefdb02.qut.edu.au:3002/** . There are several new endpoints that were not present in assignment one but are now required for this assignment – we will discuss these briefly below. However, we will not spend a lot of time in this specification on the API. You should rely on the Swagger documentation for the finer details and to try it out in practice.

This assignment builds upon the lecture and practical material covered in the second half of CAB230. In particular, you must use:

- Node
- Express
- MySQL
- Swagger (source code supplied alongside this specification)
- Knex, Morgan, and JSON Web Tokens

No alternatives to these core technologies are permitted. An essential part of this assignment is to also deploy your Express application to the QUT Linux Virtual Machine that we have allocated to you.

Although this assignment shares a lot in common with assignment one, insofar that we are using the same volcano dataset and working with a similar API, your grade level in assignment one will have no bearing on your grade level for assignment two – so treat it as a clean slate!

2. The Data

The volcano database will be provided alongside this specification as a standalone SQL dump file. You should import this database following a similar process to the "World Cities" database that we have been working with in the lectures and practicals. Once the dataset has been successfully imported, you should not touch the volcano data at all.

You will also need to create an additional table to handle the user accounts and profiles. For this step we suggest following the JWT Server Side worksheet and adapting it as needed.

3. The REST API

The REST API for this assignment is documented at: http://sefdb02.qut.edu.au:3002/ . You are required to implement all the documented endpoints and their associated error messages precisely. We will not rehash them here in any detail. If there is a conflict between what is written in this specification and the Swagger documentation – the Swagger docs have ***absolute precedence.*** The additional routes, which relate to user profile information, will be discussed below. However again, in the case of any conflict the Swagger docs are the authoritative source.

A reminder that our REST API is hosted on a QUT server, which is behind the QUT firewall. If you are off-campus, you will need to install and use the QUT VPN to access it:

https://qutvirtual4.qut.edu.au/group/student/it-and-printing/wi-fi-and-internet-access/accessing-resources-off-campus

4. Recommended development process

This assignment quite deliberately builds upon the World Database work that we have covered in the lectures and practicals. You should follow the conventions that we have used in respect of the application architecture and application security, including middleware and handling of private information such as hashing of passwords.

We strongly recommend developing your assignment in stages as outlined below:

Step 1: Create an Express application using Express Generator and establish the routes needed for the application without spending much time on the application functionality. You may organise these routes as you see fit, but we expect to see sensible application structure and use of routers to handle related tasks. The basic indexRouter vs userRouter split imposed by Express will get you started. Applications in which all the routes are handled from a single file will not be viewed favourably. Use simple logging to ensure that the routes operate successfully.

Step 2: Having created an application and ensured that the routing operates as intended, you should now replace the temporary logging statements with the requirements for each of the API endpoints. Most of the Data endpoints correspond to relatively straightforward SELECT queries using Knex, and these are a good place to begin. We suggest proceeding as follows:

- Implement the Data endpoints in their basic form, ignoring authentication
- Implement the User endpoints, ignoring authentication
- Implement the new Profile endpoints, again ignoring authentication
    - If you find the profiles difficult to implement, continue and return to them later
- Implement 'authentication' using a single dummy token to get the logic right
- Add proper JWT token generation, handling and protect the authenticated routes

Step 3: Go through and ensure that your endpoints have all the responses covered. The Swagger docs show all the appropriate HTTP codes that come back, and we have tried hard to show all the appropriate error messages. If you have the occasional incorrect error message, that is OK, but your response codes ***must*** match ours. We will test these extensively.

Step 4: At this stage, you should test your application using the test suite that we have made available via a git repository: https://github.com/chadggay/volcanoapi-tests/ . If you are not comfortable using GIT, you are able to download a zip folder containing the tests. Read the instructions accompanying the tests. In short: these tests are all based on the outward facing endpoints of your API. They are not unit tests. When run against our API, all the tests pass. To achieve a high grade in this assignment, ideally all the tests should pass for your Express application as well.

The testing section of your final report may consist solely of the automated report that is generated when running the test suite. Having run the tests, you now likely need to make some improvements to your application.

Step 5: Having created a working application locally, you should now deploy your application to the Linux Virtual Machine that we have provided to you. The practical material from week 10 will be a very useful resource for this task. Your application should serve HTTPS on localhost using a self-signed certificate.

Note that deploying your application to the Linux VM is a very important step in this assignment. We will access the running instance on the VM to assess your application. As such, a failure to deploy will have a significant impact on your overall grade. Please don't leave this step until the eleventh hour.

5. Security and Logging

We require that you undertake an appropriate security review of your application…which is a very overblown way of saying that we expect some basic level of security as covered in the lectures and practicals.

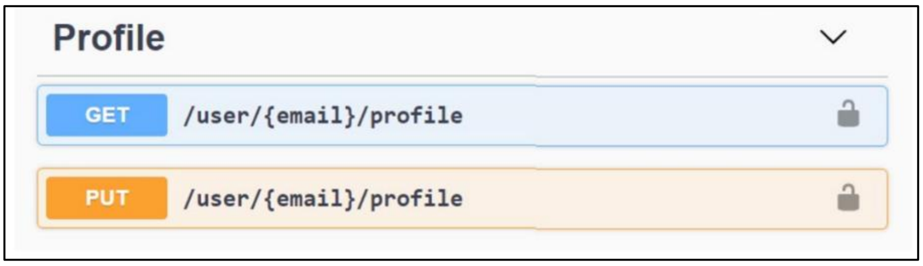
The key security requirements that you must implement are as follows:

- Use of knex – there should be no raw SQL
- Use of morgan with a logging level similar to that used in the practicals
- Appropriate handling of user passwords as described in the JWT Server-Side worksheet – inappropriate storage and handling of passwords will be heavily penalised
- TLS/HTTPS deployment with a self-signed certificate

In the report we will ask you to comment more extensively on the security of your application, but you do not need to actually implement anything beyond what is listed above.

6. The additional profile routes

This assignment includes two new routes based on user profile information that were not present during assignment one. A basic description follows below, although once again your primary source is the Swagger documentation. These routes are intended to be challenging and are a step up from the work covered in the practicals.



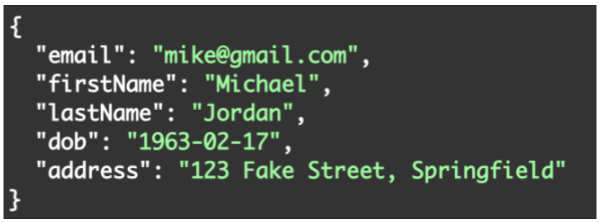
Profile

GET `/user/{email}/profile`

PUT `/user/{email}/profile`

The essential idea is that each registered user can provide some profile information about themselves. Some of this is information is publicly available and some of it is not. In a real application this profile data would include all sorts of information about the user. To keep this assignment simpler, we have limited the profile fields to only a few.

The profile fields are as shown in the Swagger example below:



```
{
  "email": "mike@gmail.com",
  "firstName": "Michael",
  "lastName": "Jordan",
  "dob": "1963-02-17",
  "address": "123 Fake Street, Springfield"
}
```

The key points of how the profiles work are as follows:

- User profile information is comprised of two components: public and authenticated. Public fields are email, firstName and lastName. The authenticated fields of dob and address can only be accessed by the profile's owner.

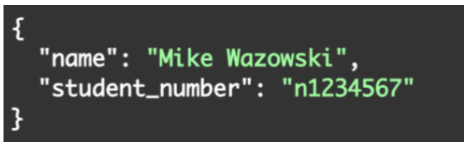
- As an authenticated user, if you attempt to access the profile of another user, the GET request will work but you will only see their public profile information. If no profile information has yet been entered for a user, all returned values except email are expected to be null.

- Updating a user profile (via the PUT request) is restricted to the owner of that profile. Attempting to edit the profile of another user should result in an error 403 Forbidden. If no authorization header is sent or there are issues with the token, a 401 unauthorized error is expected.

Finally, there is one technical issue that you will need to tackle – be careful of the date format, which follows the ISO 8061 format. The tests that we have provided for the profile routes are intended to assist you with this.

7. The /me route

To assist us in automating some of the marking process, we would like you to create a simple */me* route. This route should return an object containing your name and student number. The example from the Swagger docs is as follows:



```
{
  "name": "Mike Wazowski",
  "student_number": "n1234567"
}
```

You should replace the name with your name, and the student number with your student number (include the "n"). The test suite checks for the existence of this route but does not test the values as everyone's personal details will obviously differ.

8. Grade standards

We expect that you will follow a professional approach in the architecture and construction of the server. In particular, you should demonstrate that:

- The routes and the overall architecture are professional – logical and uncluttered, with appropriate use of specific routers mounted on the application
- There is appropriate use of middleware for managing components such as database connectivity and security
- There is appropriate error handling and the responses match those listed in the Swagger documentation
- There is an appropriate attention to application security
- The application successfully serves the Swagger docs on the home page route
- The application is successfully deployed to a QUT VM using HTTPS (self-signed certificate)

All these requirements must be met to achieve the highest grade levels.

Broadly speaking, the grade standards for this assignment will correspond to the feature levels laid out below. There is also reasonable alignment with the development steps that we outlined above. Similarly to assignment one, the grade levels below assume that the features have been implemented competently. If the implementation is substandard then the marks awarded will be reduced, sometimes substantially.

[Grade of 4 level]: Successful deployment to a QUT VM of an Express based REST API which supports some of the endpoints and interacts successfully with a MySQL database. In most cases the completed routes are likely to be Data routes. At this level there is likely significant issues with authentication, the Swagger docs may not be served and there may be significant gaps in the security requirements.

[Grade of 5 level]: Successful implementation of ***all the Data endpoints*** at a basic level. The volcanoes route should be filterable via the populatedWithin query parameter. Registration, login and JWT token handling must be attempted – although there may still be some minor issues at this level. A reasonable attempt must have also been made at the profile routes, although we would expect significant limitations. The test suite may report many incorrect responses and status codes, but the fundamentals of the API should work.

[Grade of 6 or 7 level]: A grade of 6 or 7 requires successful completion of ***all the routes***. In particular, the profile routes must work properly in both authorized and unauthorized modes at the 7 level. The distinction between the 6 and 7 level then turns on the test report and how well the API implements error responses, successful use of middleware security, database connectivity and serving of the Swagger docs. There is no "killer" requirement here that makes the difference between a 6 and a 7. These requirements are best seen as a set that go together, and done very well, give you a 7-standard mark. If you miss some of them, but still do a good job of the others, then you will likely get a 6-standard mark.

9. Submission

We will provide instructions for submission closer to the deadline. We intend to connect to your QUT VM to assess your deployed application and so it will be necessary to ensure that your server remains running during the marking period. If you have followed the process as outlined in the week 10 worksheet and practical, then we don't anticipate any difficulties.

As with assignment one, we will expect a short report, generally running to around 10 pages or so, including screenshots. We will again provide a template. Your report must include the following sections:

1. Introduction – telling us what was implemented and what wasn't.
2. Technical description of the application. This section is to allow you to talk about the application architecture and middleware choices, and to discuss technical issues that caused you problems. This is especially important if something doesn't work.
3. Security – a brief discussion of the security features of your application and a longer reflection on its limitations. We will provide more guidance on this in the report template, but this will attract a significant fraction of the report marks.
4. Testing and limitations – test result report as discussed above.
5. References
6. Appendix: a brief installation guide.

10. Acknowledgements & Further Reading

The volcanoes dataset is collated by the Smithsonian Institution's Global Volcanism Program and is publicly available at: https://volcano.si.edu/ . We are once again providing a slightly modified version to better suit the learning objectives of this assignment. The relevant citation is:

Global Volcanism Program, 2013. Volcanoes of the World, v. 4.10.5 (27 Jan 2022). Venzke, E (ed.). Smithsonian Institution.

Please now take the time to read the CRA and report template released alongside this specification.