COMP2001 Report

Web Server

GitHub Repo

`docker pull bobbymannino/comp2001-report`

Introduction

I have created an API written in python, it enabled all CRUD operations. It manipulates a set of trails that I have stored in a SQL database so they are persistent. I have also documented this API using OpenAPI standards.

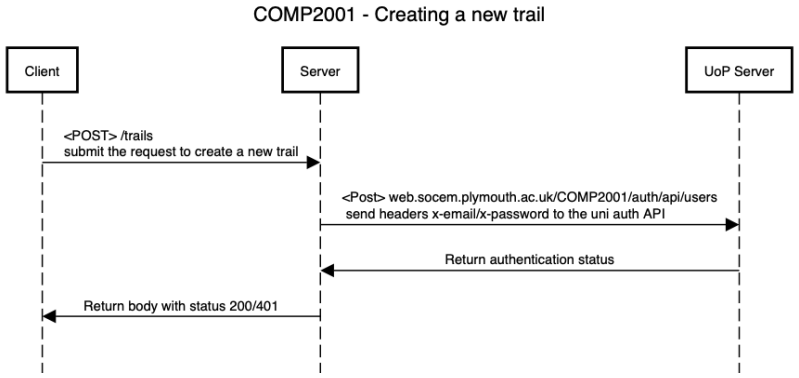
In this document you will find documentation for my implementation of the API interface. You will also find a UML diagram for the api, an OpenAPI YAML file, legal, social, ethical and professional concerns and approaches to fix those concerns as well as screenshots of everything working and at the very end an evaluation.

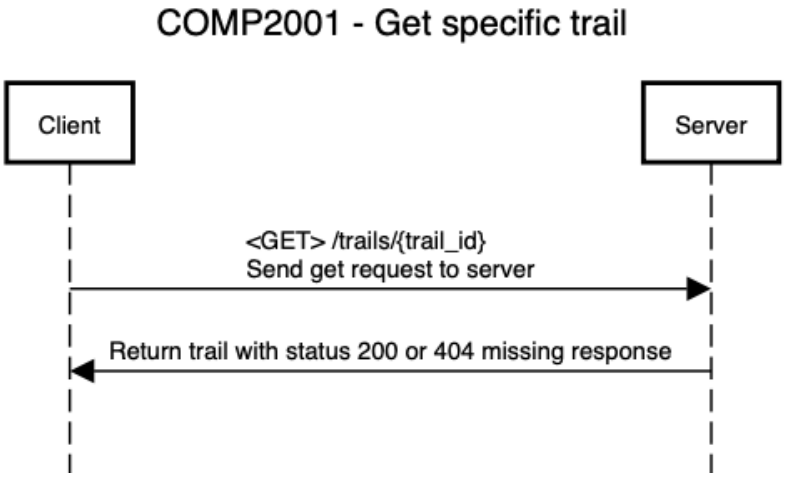
Background

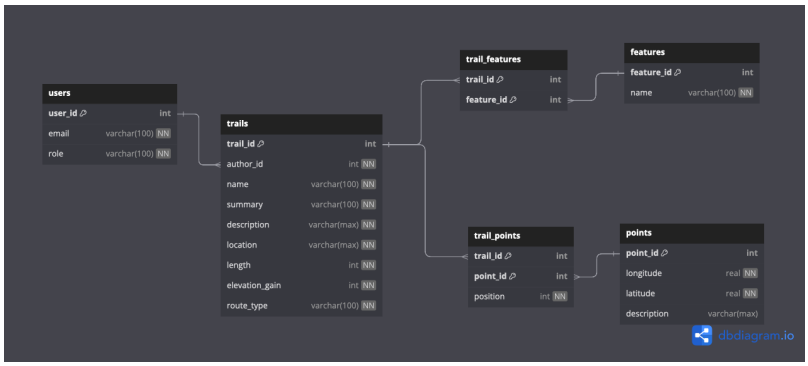
I had the database basis from the coursework before this one, but there were a couple things that i didnt like or would just like to change anyway. The things ive changed are;

- renamed “order” to “pos” as order is a keyword in SQL
- changed datatype from “text” to “varchar(max)” as text is depreciated
- added “email” property to users table

Design







I wrote the above 2 paragraphs about 2 weeks ago but things have changed since then, I spoke to some people and realised that i have made my database more complicated then it needs to be so i have decided to design it from the bottom up again. I have kept some of the tables as they were but some i have not kept at all or some i have made minor changes to. The new ERD is above, i have also designed a couple sequence diagrams to show how the api will work with the client, my api and the auth API provided by you. I have also done a couple sequence diagrams to create a vision of the flow of what will happen from when a user submits a request to reciving a response.

LSEP

Legal

Legally speaking in the Uk we have to abide by GDPR law. this essentially states that we have to store personal data securely, only access it when nessercary and only allow people who should have access to it to access it. For this i have made sure that to add/update/delete a trail you must be authorized as an admin, which means you must go through the auth API as well as being an admin in my database. The number one issue in OWSAP top 10 in 2021 was broken access controls. This means not setting the correct permissions on the database and not checking the permissions in the code. To combat this i have done what i said above and added a check to make sure you are admin before performing operations on trails. All trails are viewabble by anyone.

Social

The API application is designed to be used by anyone who has an internet connection and a web browser, it only includes what is needed to make the app work and nothing more, this means its not designed to get people addicted. Just a simple APi.

Ethical

Having 'blame' is important for responsability so each trail when update/created will have an author_id linked to a user in the database. We will also not collect any other information about a user, not their device details, IP addresses, location, etc. This would be unethical as they're simply no good reason to do this.

Professional

Professionally speaking the app is designed to be used by anyone but that includes developers. The APi is designed to use 4 HTTP methods. if the user enters a wrong trail id that doesnt exist it will return the correct error informing the user what the issue is. If the user enters wrong credentials for an action that requires them it will return the unauthorised response. The Api is designed to easily be implimented into the android app so it can have a user friendly interface to be used by a wider audience.

Implementation

trails.py

In the trails.py file there are 3 functions at the top;

- is_user_real checks if the user entered from the API is authorized on the auth API.
- get_user returns the user (if exists) in my database

- is_user_admin checks if the user is an admin in my database

I found that i was reusing the same logic so i decided to componentize it down and then i can reuse it with ease. It also makes it easier to read and understand.

To get the email/password of the user in each request the will use the x-email and x-password headers. While writing this i learnt about the Authorization header and that i could have used that like so `Authorization: Basic <email:password>` and it would have worked. I have already written the docs and the api using x-email and x-password so i decided to stick with that and not risk breaking everything. I realise this is less secure and less traditional but everything is done now. The auth api uses the email/password from the body so at least its better than that.

There is another function on line 39 that takes a trail_point as a parameter and returns null or a point if it exists. I do only use this function once but i still put it into its own function as it is reusable if i wanted to, makes it easier to read and test, and it keeps the function that is using it smaller and cleaner.

Evaluation

Further Possible Improvements

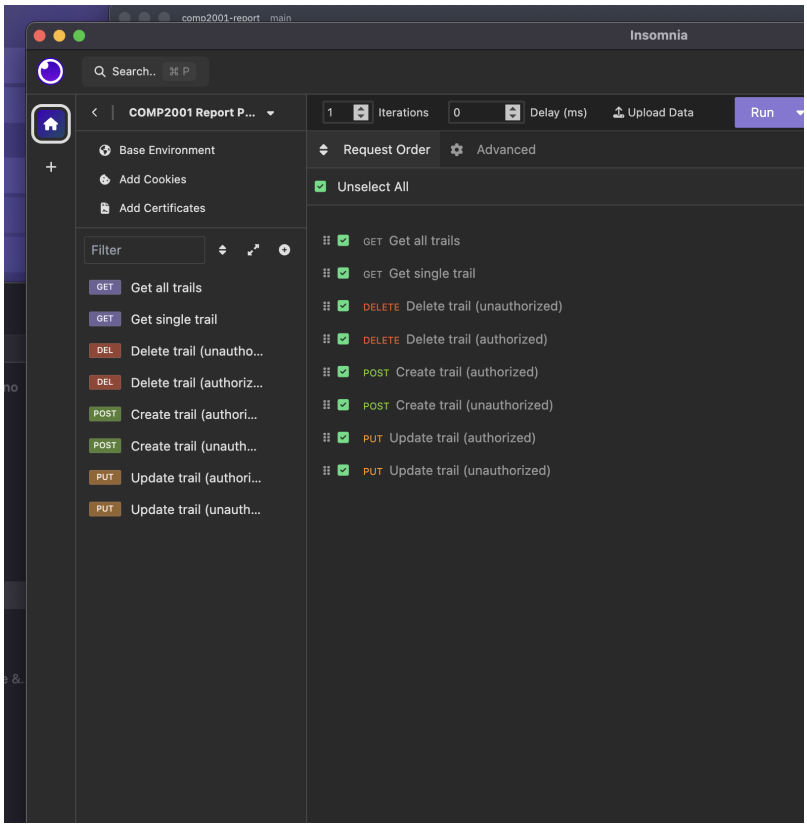
- Having endpoints for users so new users can be created/updated/deleted.
- Have a trail features endpoints
- could have implimented caching for the trails
- could have implimented pagination/ordering/filtering for the trails

Weak Areas

- i couldnt fully grasp my head around the marshmallow and sqlalchamy and how they work together, i understood flask and OpenAPI but i would of liked to have understood the inner workings of mashmallow more. before you say i didnt try hard enough or didnt ask the teachers for help, i did both and more, but i still could get my head around it.
- designing the database is also a huge weak area for me, maybe the biggest. Part one of the coursework i went way too overboard and that took me weeks to figure out, im a studious person, i enjoy to learn and be educated and to do, but databases are so difficult, you have to think of the bigger picture and how everything relates to each other, i do believe i made progress but still have a long way to go.

Testing

For testing i did personal tests, i used insomnia (beacuse its OS) and created a flow of requests, then if they all passed i assumed it was working, i of course actually was testing as i went along but this was just to be sure.



Another way i tested was once i had finished i created the docker image, went onto a different machine and pulled it, ran it and tested it again. All is working. Using docker ensures that it doesnt just work on my machine but it works anywhere. I did have trouble building the image on my machine but thats because i have an arm machine and i needed to specify platform amd as MSSQL doesnt support arm. Once i figured that out it was smooth sailing.