Introduction (approximately 2 paragraphs)

```
docker pull batlordmagnus/comp2001image
```

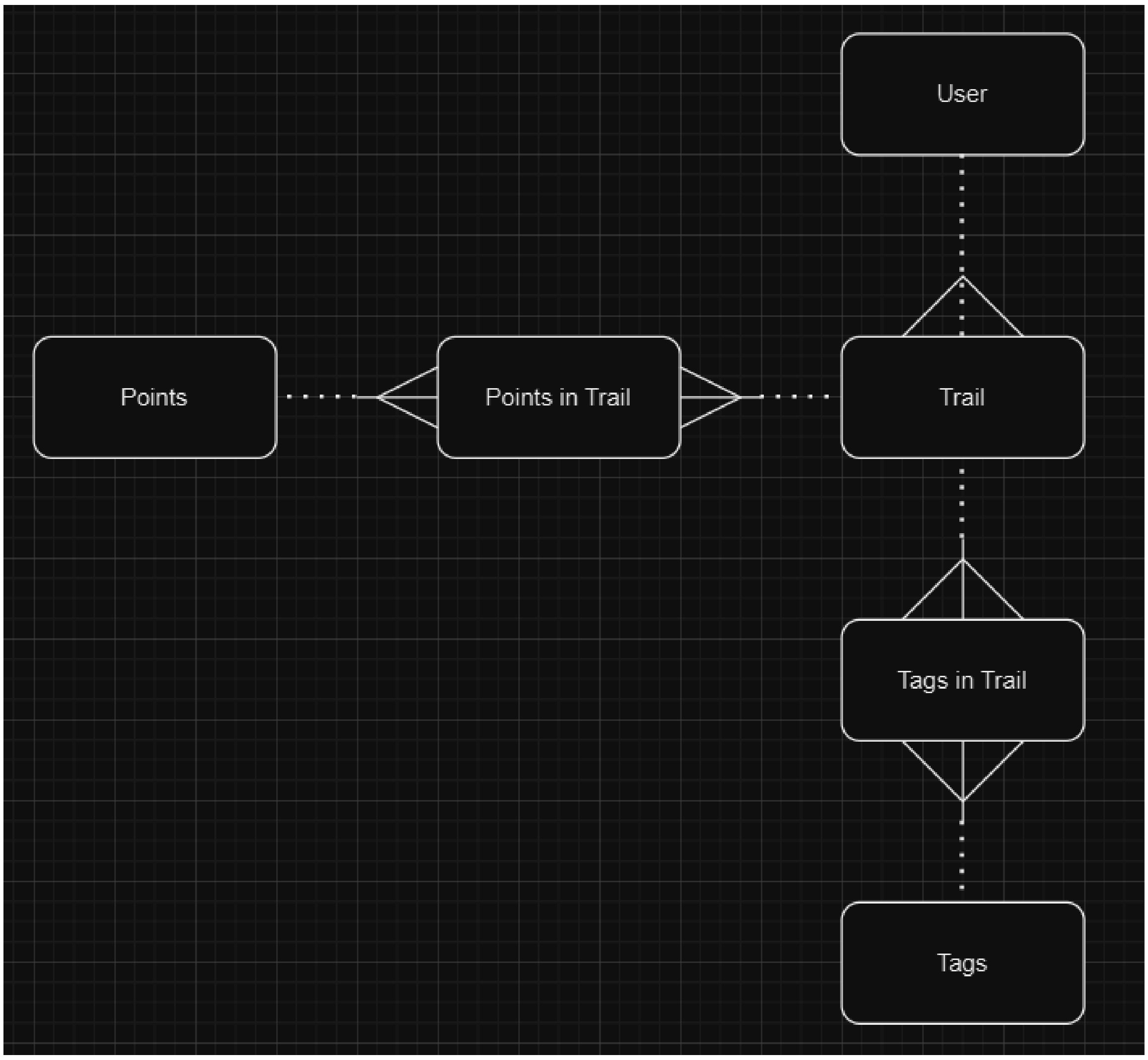
This document is a report on the process of creating a trail management microservice using SQL and python. The link to the GitHub where the code is stored is https://github.com/batgaythebatlord/comp2001-report.

This API will allow users to interact with a trail service. This service provides information about the trails that a user will be able to look at. In this report, the process of testing this database and API, alongside any legal, social, or ethical concerns that may arise, will be outlined.

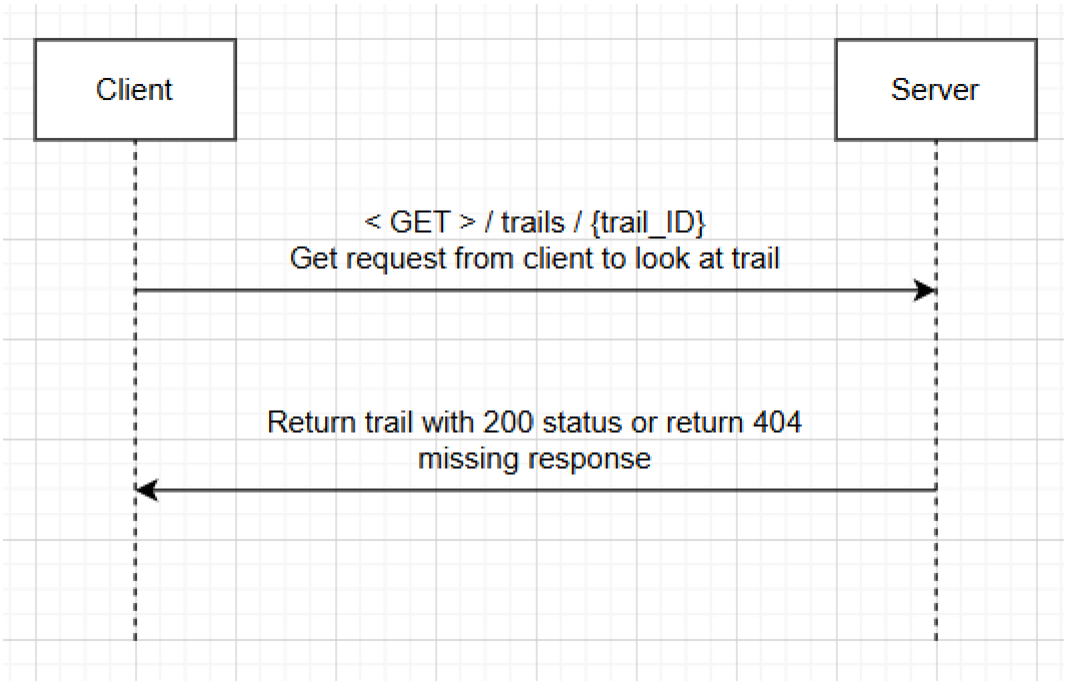
Background

The microservice allows users to login and view trails. If a user is an administrator they can create, edit, and delete trails. The users can view trails, but do not have the authority to create or delete them.

Design



This is the ERD for the database that the microservice uses.


Client
Server
< GET > / trails / {trail_ID}
Get request from client to look at trail
Return trail with 200 status or return 404
missing response

This is the sequence diagram for getting the trail for a client to look at.

Legal, Social, Ethical and Professional (LSEP)

As this database stores people’s information, GDPR had to be considered. Under GDPR, a person’s data has to be able to be easily accessed, updated, and deleted by the person whose data is being stored. For this reason, it was important that proper CRUD procedures be implemented for the user table, to allow a person to properly manage their information, and to fall inside GDPR guidelines. This software does not collect any personal data beyond what is strictly necessary. The only data stored on a user is their email address, password, whether or not they are an admin for the service, and the trails they have created. Additionally, there is no social media aspect to this service, as users cannot interact with each other or see each other's details.

Implementation

The micro service takes a person’s login information and authenticates it using the authenticator API. If the user is an admin they will be able to create, edit, and delete trails. If they are not they will only be able to look at trails.

The SQL has CRUD procedures for all of the tables, while currently the swagger only contains the information to perform them for the trail table.

Evaluation

The implementation was tested by running it through docker. While I have a very strong understanding of SQL, I greatly struggled to understand how to work with the python code or fix errors in it. Additionally I found it extremely difficult to complete the writeup and explain my work in words.

Potential improvements

- Adding the ability to look at other parts of the database (such as tags)
- Ability to add or delete users via the endpoints