Comp 2001 Report

Introduction

I have written a Trail micro-service application that exposes API endpoints that allow for CRUD operations. The API is using OpenAPI standards.

In this document you will find the background, design and implementation of the app. You will also see a discussion about Legal, Social, Ethical and Professional (LSEP) issues that might arise. At the end of the document there will be an evaluation.

GitHub repository link = https://github.com/PandoPl/COMP2001-REPORT-Project

Background

The scenario for the Trail micro-service is as follows:

"The team is creating a well-being trail application. The location of the trails you use/create is your choice. The product vision is:

For people who wish to enjoy the outdoors, to enhance their wellbeing and to have a reason to explore a particular area, the Trail App is a full trail management application providing a reason to explore a given area.

The application will meet the following user stories that have been placed on the product backlog"

Product Backlog   ...

As an Administrator I wish to create a new trail with information

As an administrator I wish to check that a new trail with information exists

As an administrator I wish to edit the information for an existing trail

As an administrator I wish to delete an existing trail

As a user of the trail app I wish to select and view a particular trail using my android mobile device

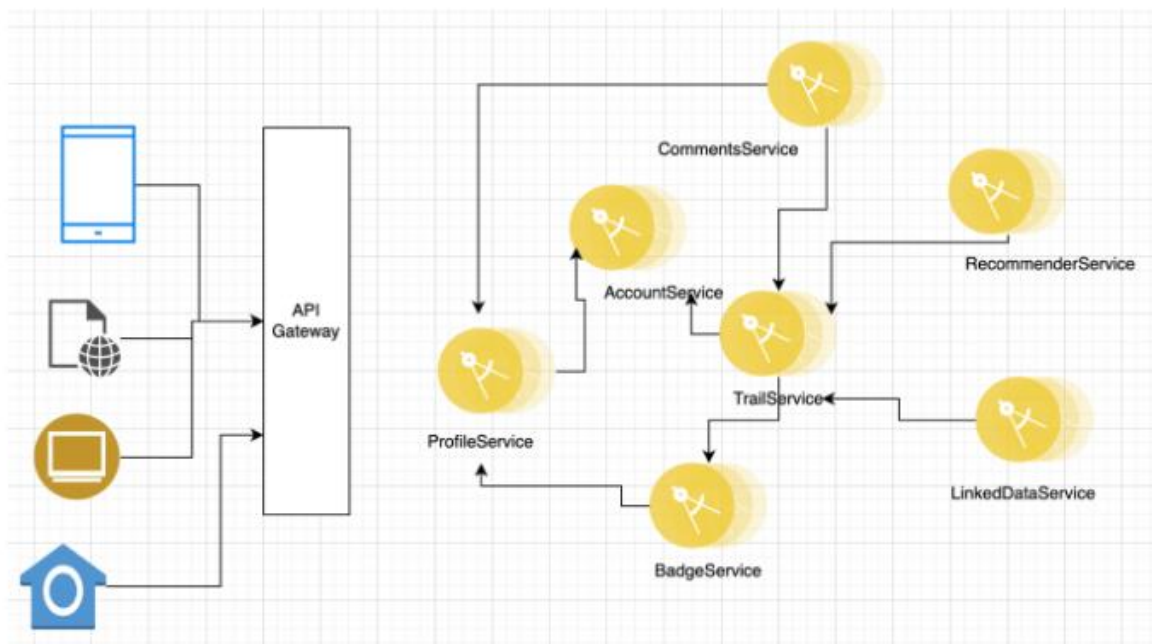An overview of the architecture is provided in the diagram below.

Figure 1: Architecture diagram for Trail Application

The trail application has many different micro-services that will be combined for the full product. One of those micro-services is the TrailService that can be seen in the diagram and that is the

one that I have created. It allows for seamless integration of Trail and User data with a SQL server database.

Design
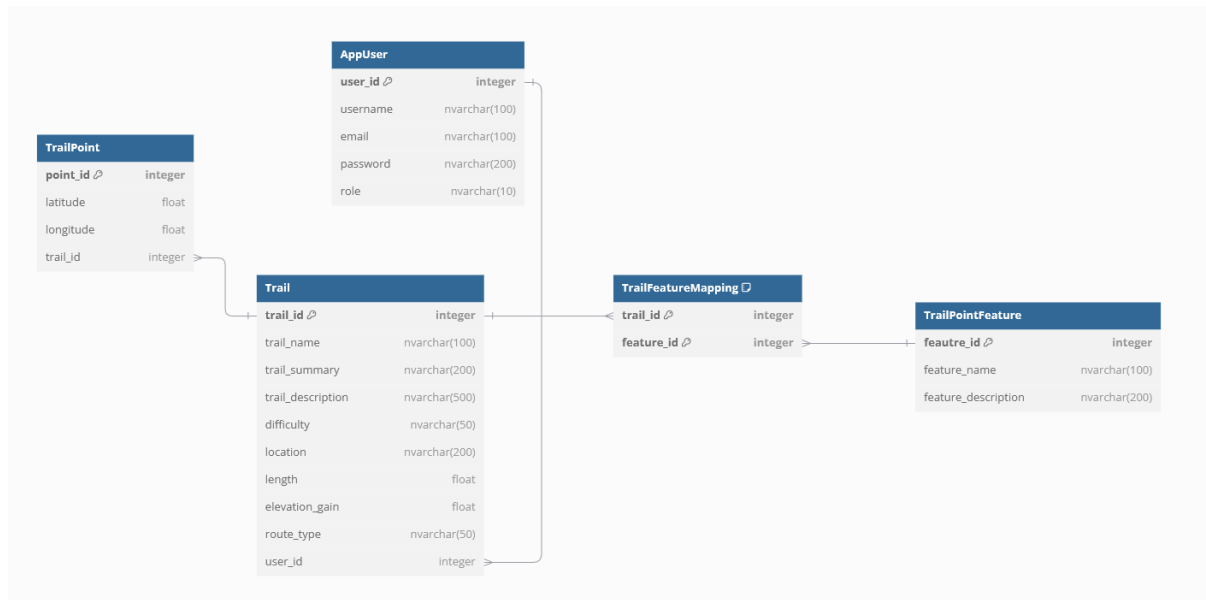
Here is an Entity Relationship Diagram of the application that I created. One user can own many trails but only one user owns a specific trail. One trail can have many trail points as trails are a series of location points but o. There is a "link entity" for trail and trail point feature.

Alternative Model Structure

TRAIL

TrailID	Sequence
Trail name	Alphabetic
Trail Summary	Alphanumeric
Trail Description	Alphanumeric
Difficulty	Alphabetic
Location	Alphabetic
Length	Numeric
Elevation gain	Numeric
Route type	Alphabetic
OwnerID*	Numeric
Pt1_Lat	Numeric
Pt1_Long	Numeric
Pt1_Desc	Alphanumeric
Pt2_Lat	Numeric
Pt2_Long	Numeric
Pt2_Desc	Alphanumeric
etc	

TRAIL-FEATURE

TrailID	Numeric
Trail FeatureID	Numeric

FEATURE

Trail FeatureID	Sequence
Trail Feature	Alphabetic

USER

UserID	Sequence
Email_address	Alphanumeric
Role	Alphabetic

This is the model that I decided to go through with in my project. I chose it as it has the least amount of tables so it is the most concise and efficient model.

COMP2001 - Get a specific trail
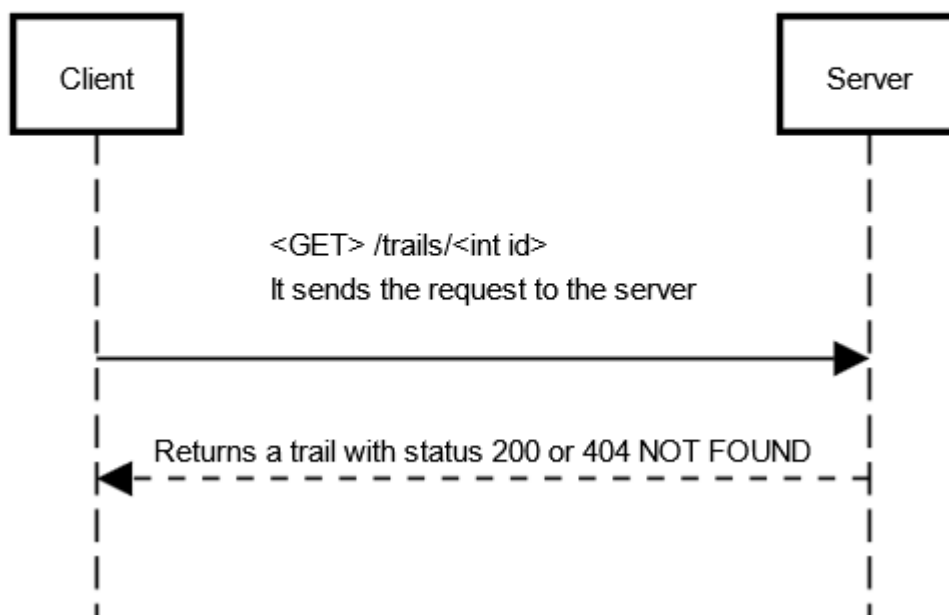
This is a sequence of the process in getting a specific trail.
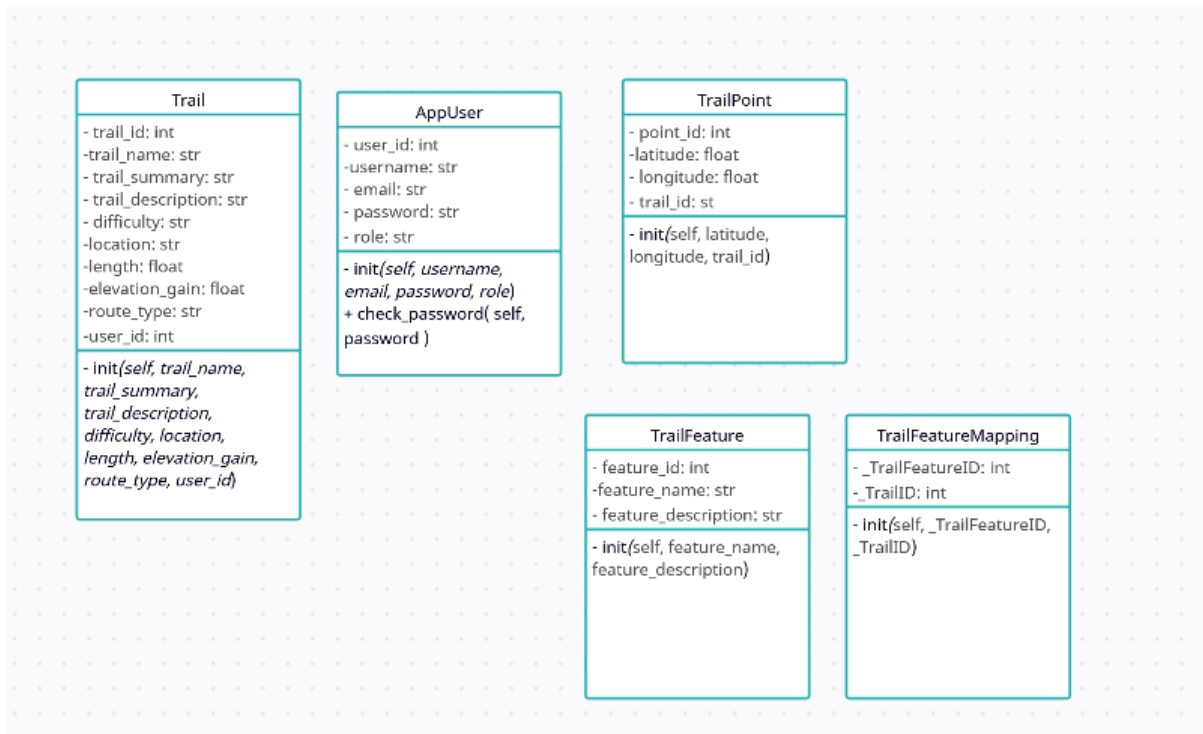
Here is a class diagram of my different models.

Legal, Social, Ethical and Professional (LSEP)

Legal:

Information privacy:

- Issues identified
  - Storing sensitive user data creates privacy risks.
  - Unauthorised access can lead to data misuse or breaches.
- Actions taken to resolve this:
  - Collected only necessary data.
  - Enforced role-based access control to ensure that only authorised users can access sensitive endpoints. OWASP

top ten vulnerabilities include "Broken Access Control" (OWASP (2021) at first place, this is to combat against this issue.

GDPR law states that we have to store personal data securely and access it only when it is absolutely necessary. It also states that only authorised individuals should be able to access it.

GDPR law also says that the data should be "accurate and, where necessary, kept up to date" (UK Government, n.d.). My micro-service has CRUD functions meaning that user information can be updated if necessary.

Social:

User trust and experience:

- Built the Trail micro-service to ensure the accuracy and reliability of trail information.
- Ensuring data privacy and security is essential for fostering trust between the application and its users. Users are more likely to engage with a service that respects and protects their personal information.
- If users feel their information is at risk, they may decide to not use the application anymore which will lead to a severe loss of business and reputation. There has been some research done that shows that "up to a third of customers in retail, finance and healthcare will stop doing business with organisations that have been breached" (Noonan, n.d.).

Ethical:

Transparency:

- Communication on how user data is handled and why it is collected.

Preventing Misuse:

- Restricted malicious activity, such as uploading false or harmful trail information, by requiring user authentication and role-based privileges.
- Allowed admins to remove inappropriate content, maintaining the integrity of the micro-service.

Inclusivity:

- Designed the micro-service to cater to a diverse audience by including features such as difficulty ratings, route types and trail summaries to help users of varying skill levels make informed decisions.

Professional:

Code quality and Documentation:

- Followed software engineering best practices, which includes modular design and the user of frameworks like Flask and SQLAlchemy for robust development.
- Provided thorough API documentation using Swagger to assist developers integrating with the micro-service.

Testing and Reliability:

- Conducted rigorous testing to ensure that the system operates reliably under different conditions.

Professional standards:

- Used version control to ensure that if anything goes wrong, I can always go back.

Implementation

- Flask: For building the REST API.

- Flask-RESTX: For structure, documentation, and Swagger UI support.

- Flask-JWT-Extended: For JWT-based authentication and authorisation.

- SQLAlchemy: For database interaction.

- Docker: For containerisation and deployment.

```sql
5     -- Create AppUser Table
6     CREATE TABLE CW2.AppUser (
7         user_id INT IDENTITY(1,1) PRIMARY KEY,
8         username NVARCHAR(100) NOT NULL UNIQUE,
9         email NVARCHAR(100) NOT NULL UNIQUE,
10        password NVARCHAR(200) NOT NULL,
11        role NVARCHAR(10) NOT NULL  -- Either 'admin' or 'user'
12    );
```

```sql
15    -- Create Trail Table
16    CREATE TABLE CW2.Trail (
17        trail_id INT IDENTITY(1,1) PRIMARY KEY,
18        trail_name NVARCHAR(100) NOT NULL,
19        trail_summary NVARCHAR(200) NOT NULL,
20        trail_description NVARCHAR(500),
21        difficulty NVARCHAR(50),
22        location NVARCHAR(200),
23        length FLOAT,
24        elevation_gain FLOAT,
25        route_type NVARCHAR(50),
26        user_id INT NOT NULL,
27        FOREIGN KEY (user_id) REFERENCES CW2.AppUser(user_id) ON DELETE CASCADE
28    );
```

```sql
31    -- Create TrailPoint Table
32    CREATE TABLE CW2.TrailPoint (
33        point_id INT IDENTITY(1,1) PRIMARY KEY,
34        latitude FLOAT NOT NULL,
35        longitude FLOAT NOT NULL,
36        trail_id INT NOT NULL,
37        FOREIGN KEY (trail_id) REFERENCES CW2.Trail(trail_id) ON DELETE CASCADE
38    );
```

```sql
41     -- Create TrailFeature Table
42   ∨ CREATE TABLE CW2.TrailFeature (
43         feature_id INT IDENTITY(1,1) PRIMARY KEY,
44         feature_name NVARCHAR(100) NOT NULL,
45         feature_description NVARCHAR(200)
46     );
```

```sql
49     -- Create TrailFeatureMapping Table
50     CREATE TABLE CW2.TrailFeatureMapping (
51         _TrailID INT NOT NULL,
52         _TrailFeatureID INT NOT NULL,
53         PRIMARY KEY (_TrailID, _TrailFeatureID),
54         FOREIGN KEY (_TrailID) REFERENCES CW2.Trail(trail_id) ON DELETE CASCADE,
55         FOREIGN KEY (_TrailFeatureID) REFERENCES CW2.TrailFeature(feature_id) ON DELETE CASCADE
56     );
```

This is the SQL based off the ERD in the Design section.

The micro-service was implemented using Flask with additional Python packages and tools, including Flask-RestX for API design, Flask-JWT-Extended for authentication, and Flask-SQLAlchemy for database interactions.

```python
1   from flask import Flask, request
2   from flask_restx import Resource, Api, fields
3   import requests
4
5   from models import db, Trail, TrailSchema, AppUser, TrailPoint, TrailFeatureMapping
6   from flask_jwt_extended import JWTManager, create_access_token, jwt_required, get_jwt
7   from functools import wraps
8
9   # Initialise Flask app
    ≪/
10  app = Flask(__name__)
11
12  # Set up database connection
13  app.config['SQLALCHEMY_DATABASE_URI'] = (
14      'mssql+pyodbc://PPandov:EwwC859*@dist-6-505.uopnet.plymouth.ac.uk:1433/COMP2001_PPandov'
15      '?driver=ODBC+Driver+17+for+SQL+Server'
16  )
17  app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
18
19  # Set up JWT configuration
20  app.config['JWT_SECRET_KEY'] = 'very_secret_key'
21  jwt = JWTManager(app)
```

There are a few commands that can be ran in Postman or just on the Swagger UI. They include:

GET /users/all

```python
# Get all users (Admin only)
@user_ns.route('/all')  ± PandoPI
class GetAllUsersResource(Resource):
    @api.doc(security='BearerAuth')  16 usages (16 dynamic)  ± PandoPI
    @jwt_required()
    @admin_required
    def get(self):
        """Get all users (Admin only)."""
        users = AppUser.query.all()
        return [{'id': user.user_id, 'username': user.username, 'email': user.email, "password": user.password,'role': user.role} for user in
                users], 200
```

GET /users/(int id)

```python
205     # Fetch or delete a specific user by ID (Admin functionality)
206     @user_ns.route('/<int:user_id>')  ± PandoPI
207   v class UserResource(Resource):
208         @api.doc(security='BearerAuth')  16 usages (16 dynamic)  ± PandoPI
209         @jwt_required()
210         @admin_required
211   v     def get(self, user_id):
212             """Get a user by ID (Admin only)."""
213             user = AppUser.query.get_or_404(user_id)
214             return {'id': user.user_id, 'username': user.username, 'email': user.email, 'role': user.role}, 200
215
```

POST /users/create

```python
# Create a new User (Admin only)
@user_ns.route('/create')  ± PandoPI
class CreateUserResource(Resource):
    @user_ns.expect(api.model('User', {  ± PandoPI
        'username': fields.String(required=True, description='Username'),
        'email': fields.String(required=True, description='Email'),
        'password': fields.String(required=True, description='Password'),
        'role': fields.String(required=True, description='User role', enum=['admin', 'user']),
    }))
    @api.doc(security='BearerAuth')
    @jwt_required()
    @admin_required
    def post(self):
        """Create a new user (Admin only)."""
        data = request.get_json()
        new_user = AppUser(
            username=data['username'],
            email=data['email'],
            password=data['password'],
            role=data['role']
        )
        db.session.add(new_user)
        db.session.commit()
        return {'id': new_user.user_id, 'username': new_user.username, 'email': new_user.email,
                'role': new_user.role}, 201
```

POST /users/login

```python
######################
# LOGIN ENDPOINT
######################

@user_ns.route('/login', methods=['POST'])   👤 PandoPI
class LoginResource(Resource):
    @user_ns.expect(api.model('Login', {   👤 PandoPI
        'email': fields.String(required=True, description='Email address'),
        'password': fields.String(required=True, description='Password'),
    }))
    def post(self):
        """Authenticate user and return a JWT token."""
        data = request.get_json()
        email = data.get('email')
        password = data.get('password')

        # Check if email and password fields are provided
        if not email or not password:
            return {"message": "Email and password are required"}, 400

        # Query database to authenticate user
        user = AppUser.query.filter_by(email=email).first()
        if user and user.check_password(password):
            # Generate a JWT token
            access_token = create_access_token(
                identity=str(user.user_id),
                additional_claims={
                    "email": user.email,
                    "role": user.role
                }
            )

            return {"access_token": access_token}, 200
        else:
            return {"message": "Invalid email or password"}, 401
```

PUT /users/(int id)

```python
223        @api.doc(security='BearerAuth')
224        @jwt_required()
225        @admin_required
226        def put(self, user_id):
227            """Update a user by ID (Admin only)."""
228            user = AppUser.query.get_or_404(user_id)
229            data = request.get_json()  # Parse the JSON body containing updated user data
230
231            # Update user fields based on provided data
232            if 'username' in data:
233                user.username = data['username']
234            if 'email' in data:
235                user.email = data['email']
236            if 'password' in data:
237                user.password = data['password']
238            if 'role' in data:
239                user.role = data['role']
240
241            # Commit the changes to the database
242            db.session.commit()
243
244            # Return the updated user information
245            return {'id': user.user_id, 'username': user.username, 'email': user.email, 'role': user.role}, 200
```

DELETE /users/(int id)

```python
247        @api.doc(security='BearerAuth')  2 usages (2 dynamic)  ⚲ PandoPI
248        @jwt_required()
249        @admin_required
250        def delete(self, user_id):
251            """Delete a user by ID (Admin only)."""
252            user = AppUser.query.get_or_404(user_id)
253            db.session.delete(user)
254            db.session.commit()
255            return {'message': 'User deleted successfully'}, 200
```

GET /trails

```python
#######################
# TRAIL CRUD ENDPOINTS
#######################
@trail_ns.route('/')  ± PandoPI
class TrailsResource(Resource):
    @jwt_required()  16 usages (16 dynamic)  ± PandoPI
    def get(self):
        """Get all trails (limited information if not an admin)."""
        trails = Trail.query.all()

        # Get the user's role from JWT claims
        claims = get_jwt()

        user_role = claims.get('role')

        if user_role == 'admin':
            #Admins get the full information
            return trails_schema.dump(trails), 200
        else:
            # Limited view for non-admin users
            limited_data = [
                {
                    "trail_id": trail.trail_id,
                    "trail_name": trail.trail_name,
                    "trail_summary": trail.trail_summary,
                    "difficulty": trail.difficulty,
                    "location": trail.location,
                    "length": trail.length,
                    "elevation_gain": trail.elevation_gain,
                    "route_type": trail.route_type,
                }
                for trail in trails
            ]
            return limited_data, 200
```

GET /trails/(int id)

```python
@trail_ns.route('/<int:id>')    👤 PandoPI
class TrailResource(Resource):
    @jwt_required()   16 usages (16 dynamic)   👤 PandoPI
    def get(self, id):
        """Get trail by ID (any user can view, Admin gets complete information)."""
        trail = Trail.query.get_or_404(id)

        claims = get_jwt()

        # Limited view for non-admin users
        limited_data = {
            "trail_id": trail.trail_id,
            "trail_name": trail.trail_name,
            "trail_summary": trail.trail_summary,
            "difficulty": trail.difficulty,
            "location": trail.location,
            "length": trail.length,
            "elevation_gain": trail.elevation_gain,
            "route_type": trail.route_type,
        }

        # Check if user is an admin, if they are then show them the rest of the info
        if claims.get('role') == 'admin':
            limited_data["trail_description"] = trail.trail_description
            limited_data["user_id"] = trail.user_id

        return limited_data, 200
```

POST /trails

```python
        @api.doc(security='BearerAuth')    PandoPI
        @trail_ns.expect(trail_model)
        @jwt_required()
        @admin_required
        def post(self):
            """Create a new trail (Admin only)."""
            data = request.get_json()
            new_trail = Trail(
                trail_name=data['trail_name'],
                trail_summary=data['trail_summary'],
                trail_description=data.get('trail_description'),
                difficulty=data.get('difficulty'),
                location=data.get('location'),
                length=data.get('length'),
                elevation_gain=data.get('elevation_gain'),
                route_type=data.get('route_type'),
                user_id=data['user_id'],
            )
            db.session.add(new_trail)
            db.session.commit()
            return trail_schema.dump(new_trail), 201
```

PUT /trails/(int id)

```python
        @api.doc(security='BearerAuth')    PandoPI
        @trail_ns.expect(trail_model)
        @jwt_required()
        @admin_required
        def put(self, id):
            """Update an existing trail (Admin only)."""
            trail = Trail.query.get_or_404(id)
            data = request.get_json()
            trail.trail_name = data.get('trail_name', trail.trail_name)
            trail.trail_summary = data.get('trail_summary', trail.trail_summary)
            trail.trail_description = data.get('trail_description', trail.trail_description)
            trail.difficulty = data.get('difficulty', trail.difficulty)
            trail.location = data.get('location', trail.location)
            trail.length = data.get('length', trail.length)
            trail.elevation_gain = data.get('elevation_gain', trail.elevation_gain)
            trail.route_type = data.get('route_type', trail.route_type)
            db.session.commit()
            return trail_schema.dump(trail), 200
```

DELETE /trails/(int id)

```
363        @api.doc(security='BearerAuth')  2 usages (2 dynamic)  👤 PandoPl
364        @jwt_required()
365        @admin_required
366        def delete(self, id):
367            """Delete an existing trail (Admin only)."""
368            trail = Trail.query.get_or_404(id)
369            db.session.delete(trail)
370            db.session.commit()
371            return {'message': 'Trail deleted successfully'}, 200
```

The app uses Flask-SQLAlchemy for object-relational mapping (ORM). The database models (e.g. Trail, AppUser) are initialised through db and used to perform CRUD operations.

I have both CRUD operations on the Users and the Trails. Users with the admin role are the only ones who can Create, Update and Delete other users and trails. Users with the user role can still look do GET /trails and GET /trail/(int id) but they will get less detailed information compared to an admin user.

The /users/login is used to get an authentication token that you enter in the top right of the Swagger UI where it says "Authorize", you must enter the token in the format "Bearer [token]" or in Postman in a Header with an Authorization Key and the Value of "Bearer [token]".

Evaluation

Testing:

Login:

| POST | /users/login | Authenticate user and return a JWT token | ∧ |

Parameters

Cancel

Name	Description

payload * required
object
(body)

Edit Value | Model

```
{
    "email": "admin@mail.com",
    "password": "pass"
}
```

Cancel

Parameter content type

application/json ∨

| Execute | Clear |

Responses

Response content type application/json ∨

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/users/login' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "email": "admin@mail.com",
  "password": "pass"
}'
```

Responses

Response content type application/json ∨

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/users/login' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "email": "admin@mail.com",
  "password": "pass"
}'
```

Request URL

```
http://127.0.0.1:8000/users/login
```

Server response

Code	Details
200	**Response body**

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczNjI8ODkwOSwianRpIjoiZTE3NzVhMGMtMWE2Zi00ZDY2LWFkYmYtNjBlZmJiYmMyNTc3IiwidHlwZSI6ImFjY2VzcyIsInN1YiI6IjQiLCJuYWYiOjE3NzYyNDg5MDksImNzcmYiOiIwMjViMTE5ZS04ZjI4LTRjYjMtOTQxYy1lMDIxYzJjY2ZjMjciLCJleHAiOjE3MzYyNDk4MDksImVtYWlsIjoiYWRtaW5AbWFppbC5jb20iLCJyb2xlIjoiYWRtaW4ifQ.KkReW4VI2fc1sGU-nPkn9_WRfAH
R7RUmRQGLa4fdVRo"
}
```

[copy] [Download]

Response headers

```
connection: close
content-length: 406
content-type: application/json
date: Tue,07 Jan 2025 11:21:49 GMT
server: Werkzeug/2.3.8 Python/3.9.21
```

Responses

Code	Description
200	Success

GET /users/all

Parameters

Cancel

No parameters

Execute | Clear

Responses

Response content type: application/json ▾

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/users/all' \
  -H 'accept: application/json' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczNjI0ODkwOSwianRpIjoiZTE3NzVhMGMtMwE2Zi00ZDY2LWFkYmYtNjBlZmJiYmMyNTc3IiwidHlwZSI6ImFjY2VzcyIsInN1YiI6IjQiLCJu
```

Request URL

```
http://127.0.0.1:8000/users/all
```

Server response

Code	Details

200

Response body

```
    "username": "biz",
    "email": "biz@mail.com",
    "password": "pass",
    "role": "admin"
  },
  {
    "id": 9,
    "username": "Grace Hopper",
    "email": "grace@plymouth.ac.uk",
    "password": "ISAD123!",
    "role": "admin"
  },
  {
    "id": 10,
    "username": "Tim Berners-Lee",
    "email": "tim@plymouth.ac.uk",
    "password": "COMP2001!",
    "role": "user"
  },
  {
    "id": 11,
    "username": "Ada Lovelace",
    "email": "ada@plymouth.ac.uk",
    "password": "insecurePassword",
    "role": "admin"
```

POST /users/create

POST `/users/create` Create a new user (Admin only) ∧ 🔒

Parameters [Cancel]

Name	Description

payload * required
object
(body)
	Edit Value | Model
	
	``
	{
	  "username": "TestUser",
	  "email": "test@mail.com",
	  "password": "tester",
	  "role": "user"
	}
	``

[Cancel]

Parameter content type

`application/json` ⌄

[Execute] [Clear]

Responses Response content type `application/json` ⌄

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/users/create' \
  -H 'accept: application/json' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczNjI0ODkwOSwianRpIjoiZTE3NzVhMGMtMWE2Zi00ZDY2LWFkYmYtNjBlZmJiYmMyNTc3IiwidHlwZSI6ImFjY2VzcyIsInN1YiI6IjQiLCJu
  -H 'Content-Type: application/json' \
  -d '{
  "username": "TestUser",
  "email": "test@mail.com",
  "password": "tester",
  "role": "user"
}'
```

Responses Response content type `application/json` ⌄

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/users/create' \
  -H 'accept: application/json' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczNjI0ODkwOSwianRpIjoiZTE3NzVhMGMtMWE2Zi00ZDY2LWFkYmYtNjBlZmJiYmMyNTc3IiwidHlwZSI6ImFjY2VzcyIsInN1YiI6IjQiLCJu
  -H 'Content-Type: application/json' \
  -d '{
  "username": "TestUser",
  "email": "test@mail.com",
  "password": "tester",
  "role": "user"
}'
```

Request URL

```
http://127.0.0.1:8000/users/create
```

Server response

Code	Details

201
Undocumented
	Response body
	
	``
	{
	  "id": 13,
	  "username": "TestUser",
	  "email": "test@mail.com",
	  "role": "user"
	}
	`` [Download]
	
	Response headers
	
	``
	connection: close
	content-length: 95
	content-type: application/json
	date: Tue,07 Jan 2025 11:23:37 GMT
	server: Werkzeug/2.3.8 Python/3.9.21
	``

Responses

Code	Description
200	Success

GET /users/(int id)

GET   /users/{user_id}   Get a user by ID (Admin only)

Parameters   Cancel

Name	Description
user_id * required integer (path)	13

Execute   Clear

Responses   Response content type   application/json

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/users/13' \
  -H 'accept: application/json' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczNjI0ODkwOSwianRpIjoiZTE3NzVhMGMtMWE2Zi00ZDY2LWFkYmYtNjBlZmJiYmMyNTc3IiwidHlwZSI6ImFjY2VzcyIsInN1YiI6IjQiLCJ...
```

Request URL

```
http://127.0.0.1:8000/users/13
```

Server response

Code	Details

200	**Response body**
	``{
	  "id": 13,
	  "username": "TestUser",
	  "email": "test@mail.com",
	  "role": "user"
	}``` Download
	
	Response headers
	``connection: close
	content-length: 95
	content-type: application/json
	date: Tue,07 Jan 2025 11:24:26 GMT
	server: Werkzeug/2.3.8 Python/3.9.21```

Responses

Code	Description

PUT /users/(int id)

GET   /users/{user_id}   Get a user by ID (Admin only)

Parameters   Cancel

Name	Description
payload * required object (body)	Edit Value | Model

```
{
    "username": "BestUser",
    "email": "email@email.com",
    "password": "bestUser",
    "role": "admin"
}
```

Cancel

Parameter content type

application/json   ∨

| user_id * required
integer
(path) | 14 |

Execute   Clear

Responses   Response content type application/json ∨

Curl

```
curl -X 'PUT' \
  'http://127.0.0.1:8000/users/14' \
  -H 'accept: application/json' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczNjI0ODkwOSwianRpIjoiZTE3NzVhNGMtNWE2Zi00ZDY2LwFkYmYtNjBlZmQiYmYyNTc3IiwidHlwZSI6ImFjY2VzcyIsInN1YiI6IjQiLCJuYmYiOjE3NzYyNDg
  -H 'Content-Type: application/json' \
  -d '{
  "username": "BestUser",
  "email": "email@email.com",
  "password": "bestUser",
  "role": "admin"
}'
```

Responses   Response content type application/json ∨

Curl

```
curl -X 'PUT' \
  'http://127.0.0.1:8000/users/14' \
  -H 'accept: application/json' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczNjI0OTQ3NSwianRpIjoiNjJiY2NjMmUtYWVkOS00MDEwLWE4M2QtZTMzZDU3MmZmYWQ5IiwidHlwZSI6ImFjY2VzcyIsInN1YiI6IjQiLCJuYmYiOjE3NzYyNDk
  -H 'Content-Type: application/json' \
  -d '{
  "username": "BestUser",
  "email": "email@mail.com",
  "password": "bestUser",
  "role": "admin"
}'
```

Request URL

```
http://127.0.0.1:8000/users/14
```

Server response

Code	Details
200	**Response body**

```
{
    "id": 14,
    "username": "BestUser",
    "email": "email@mail.com",
    "role": "admin"
}
```
Download

Response headers

```
connection: close
content-length: 97
content-type: application/json
date: Tue,07 Jan 2025 11:32:10 GMT
server: Werkzeug/2.3.8 Python/3.9.21
```

Responses

Code	Description
200	Success

DELETE /users/(int id)

DELETE `/users/{user_id}` Delete a user by ID (Admin only)

Parameters   Cancel

Name	Description

user_id * required
integer
(path)

```
13
```

| Execute | Clear |

Responses   Response content type application/json ▾

Curl

```
curl -X 'DELETE' \
  'http://127.0.0.1:8000/users/13' \
  -H 'accept: application/json' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczNjI0ODkwOSwianRpIjoiZTE3NzVhMGMtMWE2Zi00ZDY2LWFkYmYtNjBlZmJiYmMyNTc3IiwidHlwZSI6ImFjY2VzcyIsInN1YiI6IjQiLCJ1
```

Request URL

```
http://127.0.0.1:8000/users/13
```

Server response

Code	Details
200	**Response body**

```
{
  "message": "User deleted successfully"
}
```

Response headers

```
connection: close
content-length: 47
content-type: application/json
date: Tue,07 Jan 2025 11:25:06 GMT
server: Werkzeug/2.3.8 Python/3.9.21
```

Responses

GET /trails/

GET `/trails/` Get all trails (limited information if not an admin)

Parameters   Cancel

No parameters

| Execute | Clear |

Responses   Response content type application/json ▾

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/trails/' \
  -H 'accept: application/json' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczNjI0ODkwOSwianRpIjoiZTE3NzVhMGMtMWE2Zi00ZDY2LWFkYmYtNjBlZmJiYmMyNTc3IiwidHlwZSI6ImFjY2VzcyIsInN1YiI6IjQiLCJ1
```

Request URL

```
http://127.0.0.1:8000/trails/
```

Server response

Code	Details
200	**Response body**

```
[
  {
    "trail_points": [],
    "feature_mappings": [],
    "trail_id": 15,
    "trail_name": "Plymouth",
    "trail_summary": "Walking",
    "trail_description": "Good Description",
    "difficulty": "Hard",
    "location": "Plymouth",
    "length": 1,
    "elevation_gain": 12,
    "route_type": "Loop",
    "user_id": 9,
    "owner": 9
  },
  {
    "trail_points": [],
    "feature_mappings": [],
    "trail_id": 16,
    "trail_name": "Dartmoor",
    "trail_summary": "A very interesting place",
    "trail_description": "This is a really good description of the place",
    "difficulty": "Medium",
    "location": "Dartmoor",
```

POST /trails/

/trails/ Create a new trail (Admin only)

Parameters

Cancel

Name	Description
payload * required object *(body)*	Edit Value | Model

```
{
    "trail_name": "New Trail",
    "trail_summary": "Trail",
    "trail_description": "Good place to be",
    "difficulty": "Medium",
    "location": "TrailLand",
    "length": 12,
    "elevation_gain": 144,
    "route_type": "Loop",
    "user_id": 9
}
```

Cancel

Parameter content type

application/json ▾

Execute	Clear

Responses   Response content type application/json ▾

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/trails/' \
  -H 'accept: application/json' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczNjI0ODkwOSwianRpIjoiZTE3NzVhMGMtMWE2Zi00ZDY2LWFkYmYtNjBlZmJiYmMyNTc3IiwidHlwZSI6ImFjY2VzcyIsInN1YiI6IjQiLCJuYmYiOjE3MzYyND
  -H 'Content-Type: application/json' \
  -d '{
  "trail_name": "New Trail",
  "trail_summary": "Trail",
  "trail_description": "Good place to be",
  "difficulty": "Medium",
  "location": "TrailLand",
  "length": 12,
  "elevation_gain": 144,
  "route_type": "Loop",
  "user_id": 9
}'
```

Responses   Response content type application/json ▾

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/trails/' \
  -H 'accept: application/json' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczNjI0ODkwOSwianRpIjoiZTE3NzVhMGMtMWE2Zi00ZDY2LWFkYmYtNjBlZmJiYmMyNTc3IiwidHlwZSI6ImFjY2VzcyIsInN1YiI6IjQiLCJ
  -H 'Content-Type: application/json' \
  -d '{
  "trail_name": "New Trail",
  "trail_summary": "Trail",
  "trail_description": "Good place to be",
  "difficulty": "Medium",
  "location": "TrailLand",
  "length": 12,
  "elevation_gain": 144,
  "route_type": "Loop",
  "user_id": 9
}'
```

Request URL

```
http://127.0.0.1:8000/trails/
```

Server response

Code	Details
201 *Undocumented*	**Response body**

```
{
    "trail_points": [],
    "feature_mappings": [],
    "trail_id": 18,
    "trail_name": "New Trail",
    "trail_summary": "Trail",
    "trail_description": "Good place to be",
    "difficulty": "Medium",
    "location": "TrailLand",
    "length": 12,
    "elevation_gain": 144,
    "route_type": "Loop",
    "user_id": 9,
    "owner": 9
}
```

Download

Response headers

```
connection: close
content-length: 347
content-type: application/json
date: Tue,07 Jan 2025 11:26:21 GMT
server: Werkzeug/2.3.8 Python/3.9.21
```

Responses

Code	Description
200	Success

GET /trails/(int id)

PUT /trails/(int id)

PUT	/trails/{id} Update an existing trail (Admin only)	⌃ 🔒

Parameters

Cancel

Name	Description
payload * required	Edit Value | Model
object	
(body)	

```
{
    "trail_name": "Updated Name",
    "trail_summary": "Summary Updated",
    "trail_description": "Description Updated",
    "difficulty": "Hard",
    "location": "Somewhere",
    "length": 12,
    "elevation_gain": 55,
    "route_type": "Out-and-back",
    "user_id": 9
}
```

Cancel

Parameter content type

application/json ▾

id * required	18
integer	
(path)	

Execute	Clear

Responses

Response content type: application/json ▾

Curl

```
curl -X 'PUT' \
  'http://127.0.0.1:8000/trails/18' \
  -H 'accept: application/json' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczNjI0OTQ3NSwianRpIjoiNjJiY2NjMmUtYwVkOS00MDEwLwE4M2QtZTMzZDU3MmZmVwQ5IiwidHlwZSI6ImFjY2VzcyIsInN1YiI6IjQiLCJuYmYiOjE3MzYyNDk...
  -H 'Content-Type: application/json' \
  -d '{
    "trail_name": "Updated Name",
    "trail_summary": "Summary Updated",
    "trail_description": "Description Updated",
    "difficulty": "Hard",
    "location": "Somewhere",
    "length": 12,
    "elevation_gain": 55,
    "route_type": "Out-and-back",
    "user_id": 9
```

Responses

Response content type: application/json ▾

Curl

Request URL

```
http://127.0.0.1:8000/trails/18
```

Server response

Code	Details
200	**Response body**

```
{
    "trail_points": [],
    "feature_mappings": [],
    "trail_id": 18,
    "trail_name": "Updated Name",
    "trail_summary": "Summary Updated",
    "trail_description": "Description Updated",
    "difficulty": "Hard",
    "location": "Somewhere",
    "length": 12,
    "elevation_gain": 55,
    "route_type": "Out-and-back",
    "user_id": 9,
    "owner": 9
}
```

Download

Response headers

```
connection: close
content-length: 368
content-type: application/json
date: Tue,07 Jan 2025 11:34:45 GMT
server: Werkzeug/2.3.8 Python/3.9.21
```

Responses

Code	Description
200	Success

DELETE /trails/(int id)

Reflection

Areas that need further work is my python skills. I haven't used python in years before this project, it was quite confusing trying to understand it and I kept on making mistakes that cost me a lot of time. I spent too much time on this project fixing one error just for 3 other things that worked completely fine before to break too.

One of my strengths that helped a lot in this coursework project is my problem-solving nature, I was very invested into solving every issue that was popping up.

One of my weaknesses is that I didn't have enough knowledge of the technologies I was using so it was taking very long to get used to it.

Some improvements for the coursework I could've done is to try to implement some security measures like for example hashing the password instead of having it in plain text.

References

- **UK Government (n.d.)** Data protection. Available at: https://www.gov.uk/data-protection (Accessed: 7 January 2025).
- Noonan, L. (n.d.). **5 Damaging Consequences of a Data Breach**. *MetaCompliance*. Available at: https://www.metacompliance.com/blog/data-breaches/5-damaging-consequences-of-a-data-breach [Accessed 7 Jan. 2025].
- OWASP. (2021). **OWASP Top Ten - 2021**. [online] Available at: https://owasp.org/www-project-top-ten/ [Accessed 7 Jan. 2025].