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.

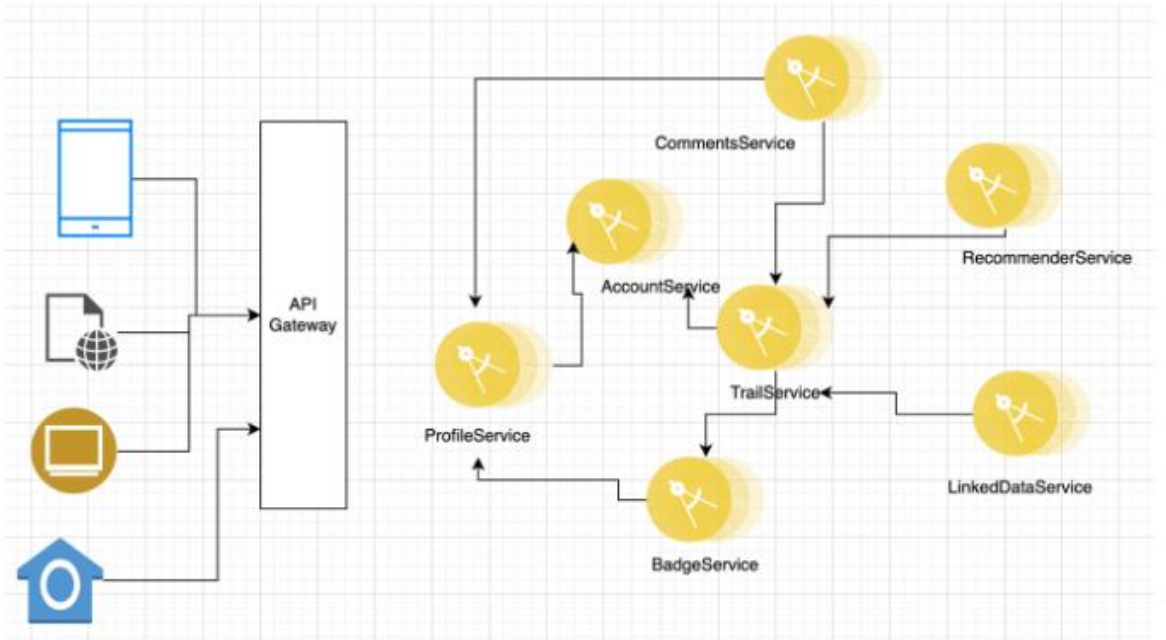


Figure 1: Architecture diagram for Trail Application

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. There is a “link entity” for trail and trail point feature.

Alternative Model Structure

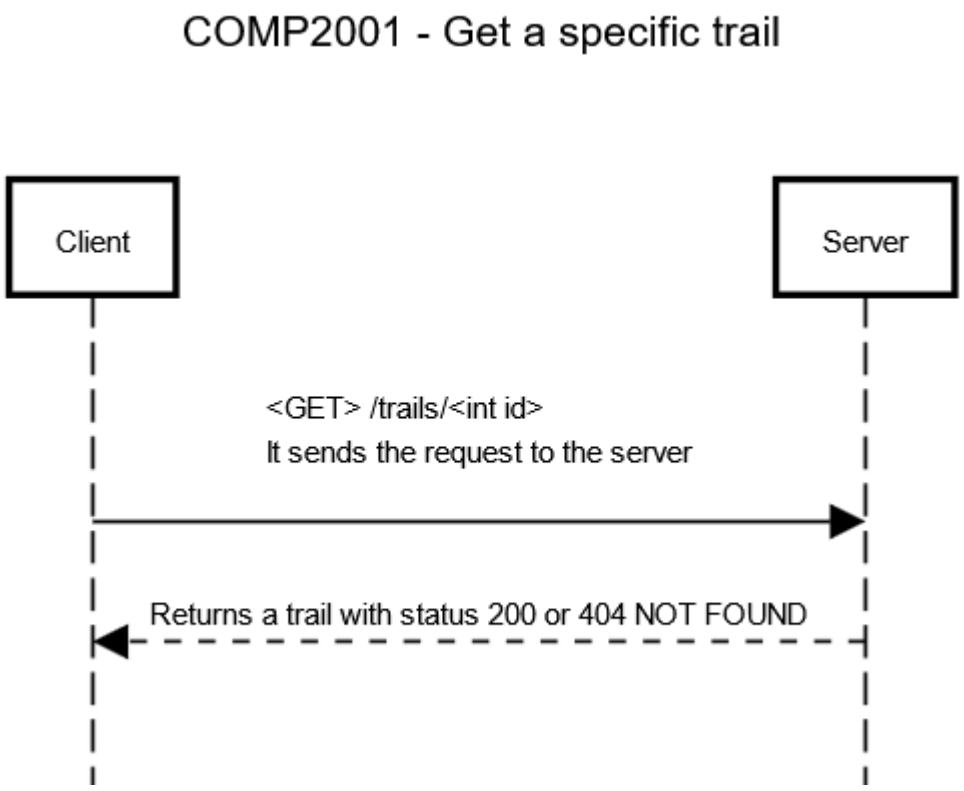
TRAIL	
TrailID	Sequence
Trail name	Alphabetic
Trail Summary	Alphanumeric
Trail Description	Alphanumeric
Difficulty	Alphabetic
Location	Alphabetic
Length	Numeric
Elevation gain	Numeric
Route type	Alphabetic
OwnerID*	Numeric
Pt1_Lat	Numeric
Pt1_Long	Numeric
Pt1_Desc	Alphanumeric
Pt2_Lat	Numeric
Pt2_Long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

TRAIL-FEATURE	
TrailID	Numeric
Trail FeatureID	Numeric

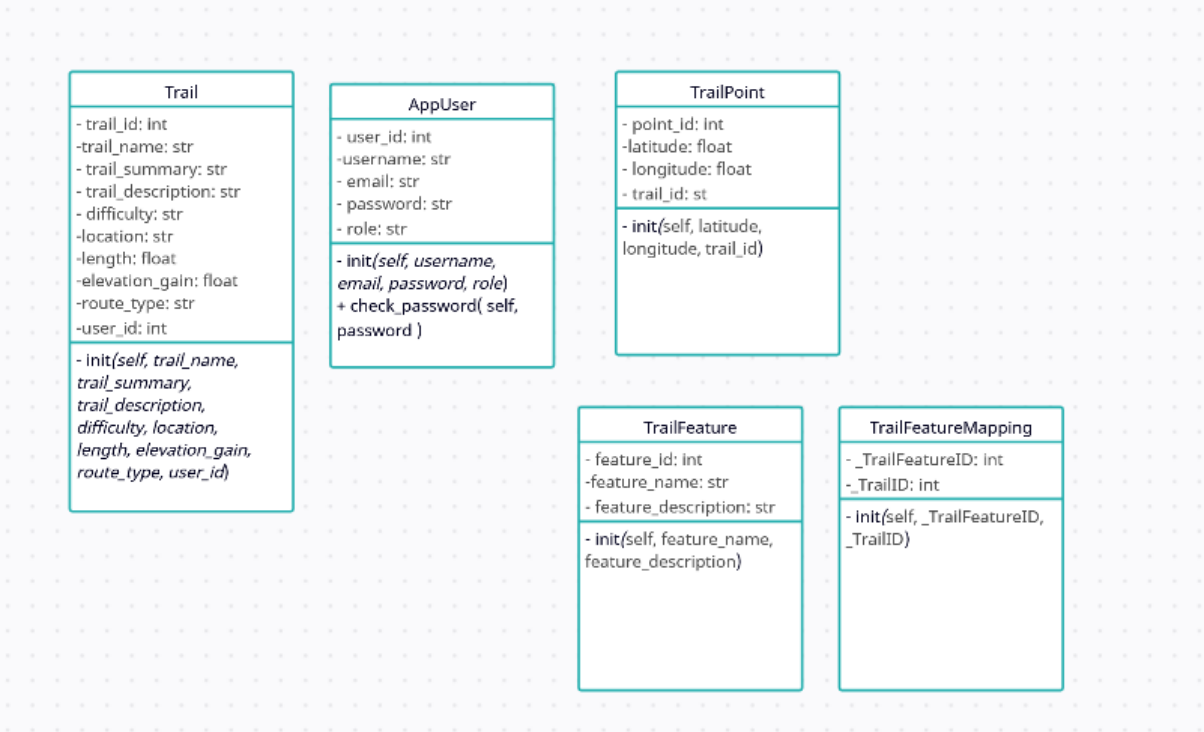
FEATURE	
Trail FeatureID	Sequence
Trail Feature	Alphabetic

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.



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

```sql
-- Create Trail Table
CREATE TABLE CW2.Trail (
    trail_id INT IDENTITY(1,1) PRIMARY KEY,
    trail_name NVARCHAR(100) NOT NULL,
    trail_summary NVARCHAR(200) NOT NULL,
    trail_description NVARCHAR(500),
    difficulty NVARCHAR(50),
    location NVARCHAR(200),
    length FLOAT,
    elevation_gain FLOAT,
    route_type NVARCHAR(50),
    user_id INT NOT NULL,
    FOREIGN KEY (user_id) REFERENCES CW2.AppUser(user_id) ON DELETE CASCADE
);
```

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

```sql
-- Create TrailFeature Table
CREATE TABLE CW2.TrailFeature (
    feature_id INT IDENTITY(1,1) PRIMARY KEY,
    feature_name NVARCHAR(100) NOT NULL,
    feature_description NVARCHAR(200)
);
```

```sql
-- Create TrailFeatureMapping Table
CREATE TABLE CW2.TrailFeatureMapping (
    _TrailID INT NOT NULL,
    _TrailFeatureID INT NOT NULL,
    PRIMARY KEY (_TrailID, _TrailFeatureID),
    FOREIGN KEY (_TrailID) REFERENCES CW2.Trail(trail_id) ON DELETE CASCADE,
    FOREIGN KEY (_TrailFeatureID) REFERENCES CW2.TrailFeature(feature_id) ON DELETE CASCADE
);
```

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
from flask import Flask, request
from flask_restx import Resource, Api, fields
import requests

from models import db, Trail, TrailSchema, AppUser, TrailPoint, TrailFeatureMapping
from flask_jwt_extended import JWTManager, create_access_token, jwt_required, get_jwt
from functools import wraps

# Initialise Flask app
app = Flask(__name__)

# Set up database connection
app.config['SQLALCHEMY_DATABASE_URI'] = (
    'mssql+pyodbc://PPandov:EwwC859*@dist-6-505.uopnet.plymouth.ac.uk:1433/COMP2001_PPandov'
    '?driver=ODBC+Driver+17+for+SQL+Server'
)
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

# Set up JWT configuration
app.config['JWT_SECRET_KEY'] = 'very_secret_key'
jwt = JWTManager(app)
```

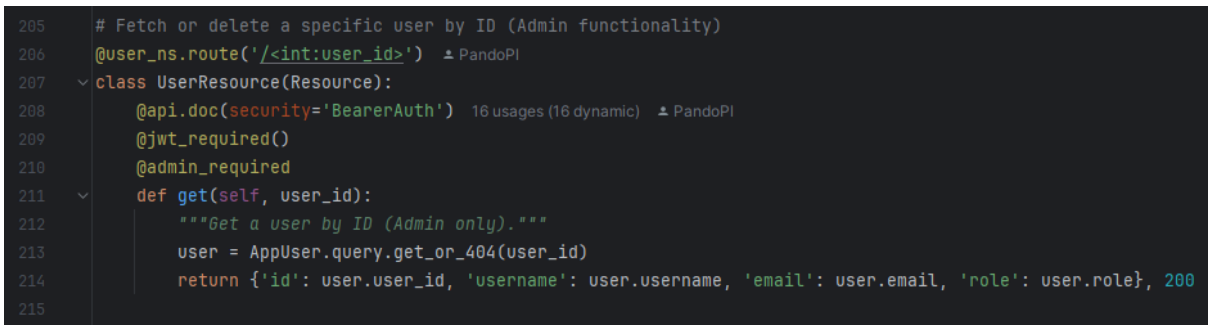
There are a few commands that can be ran in Postman or just on the Swagger UI. They include:

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

# Create a new user (Admin only)
```

GET /users/(int id)



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

```

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
@api.doc(security='BearerAuth')
@jwt_required()
@admin_required
def put(self, user_id):
    """Update a user by ID (Admin only)."""
    user = AppUser.query.get_or_404(user_id)
    data = request.get_json()  # Parse the JSON body containing updated user data

    # Update user fields based on provided data
    if 'username' in data:
        user.username = data['username']
    if 'email' in data:
        user.email = data['email']
    if 'password' in data:
        user.password = data['password']
    if 'role' in data:
        user.role = data['role']

    # Commit the changes to the database
    db.session.commit()

    # Return the updated user information
    return {'id': user.user_id, 'username': user.username, 'email': user.email, 'role': user.role}, 200
```

DELETE /users/(int id)

```python
@api.doc(security='BearerAuth')  2 usages (2 dynamic)  PandoPI
@jwt_required()
@admin_required
def delete(self, user_id):
    """Delete a user by ID (Admin only)."""
    user = AppUser.query.get_or_404(user_id)
    db.session.delete(user)
    db.session.commit()
    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)

```python
@api.doc(security='BearerAuth')  2 usages (2 dynamic)  PandoPI
@jwt_required()
@admin_required
def delete(self, id):
    """Delete an existing trail (Admin only)."""
    trail = Trail.query.get_or_404(id)
    db.session.delete(trail)
    db.session.commit()
    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.

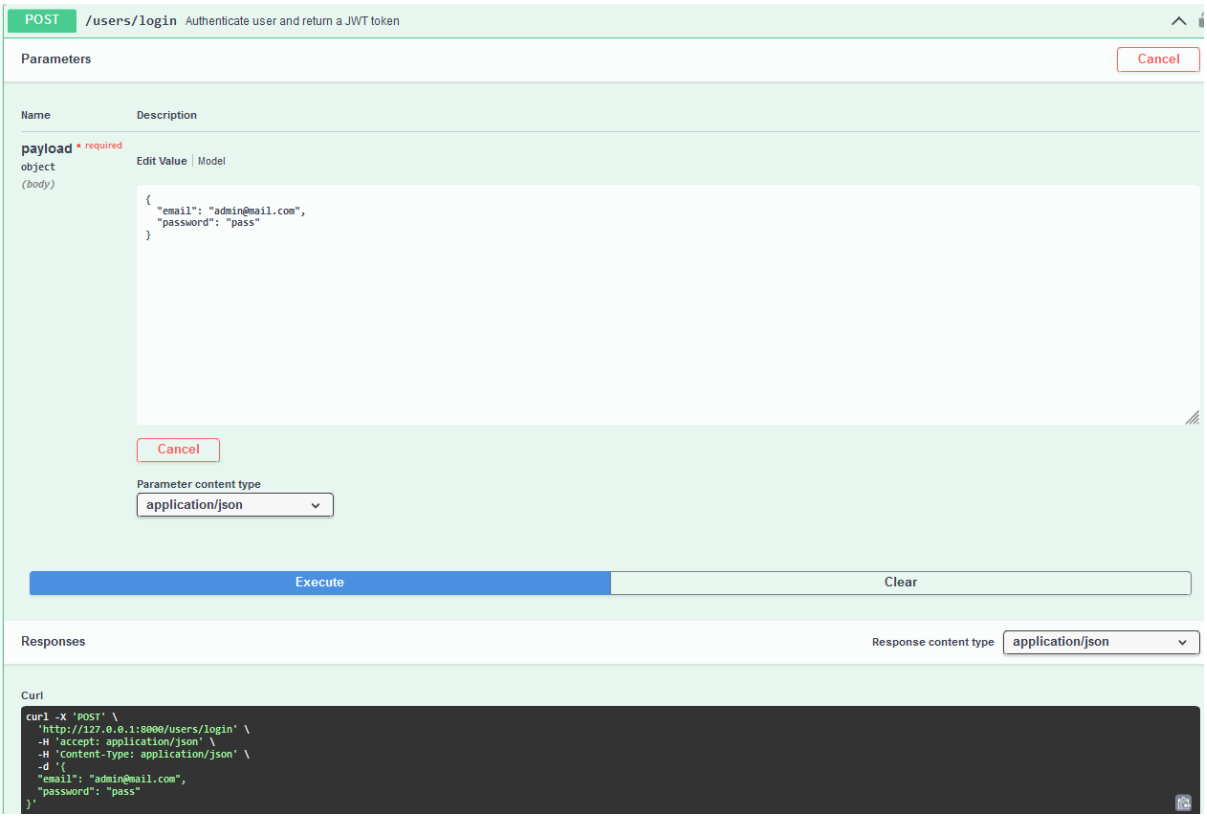
I have both CRUD operations on the Users and the Trails. Users with the admin role are the only ones who can Create, Update and Delete other users and trails. Users with the user role can still look do GET /trails and GET /trail/(int id) but they will get less detailed information compared to an admin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

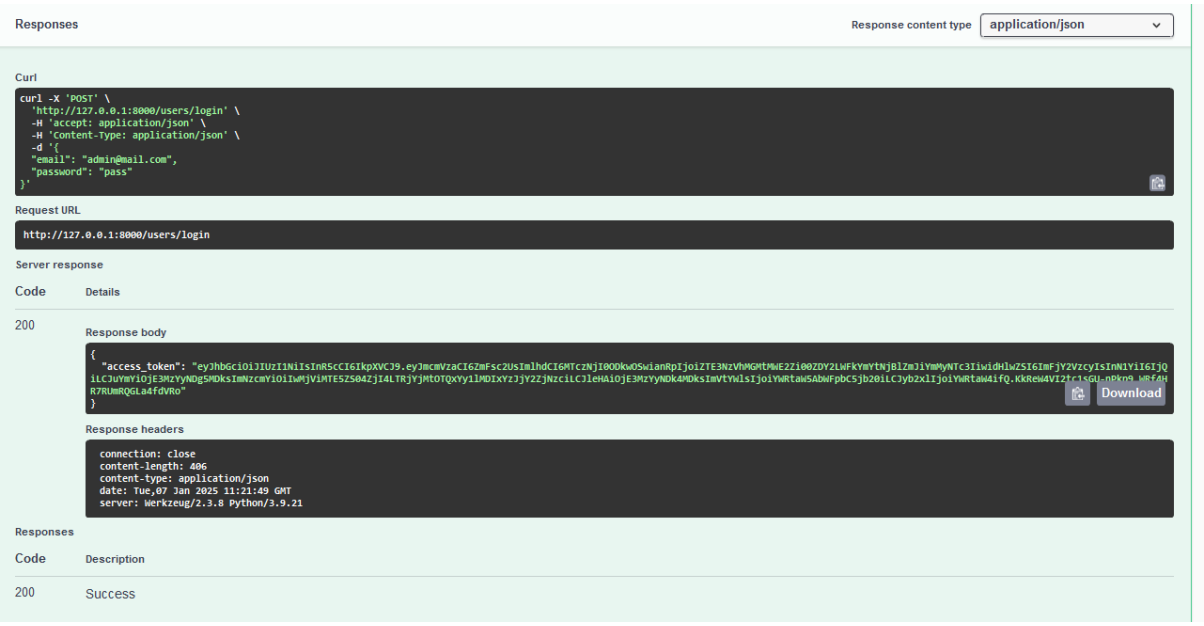
The /users/login is used to get an authentication token that you enter in the top right of the Swagger UI where it says “Authorize”, you must enter the token in the format “Bearer [token]” or in Postman in a Header with an Authorization Key and the Value of “Bearer [token]”.

Evaluation

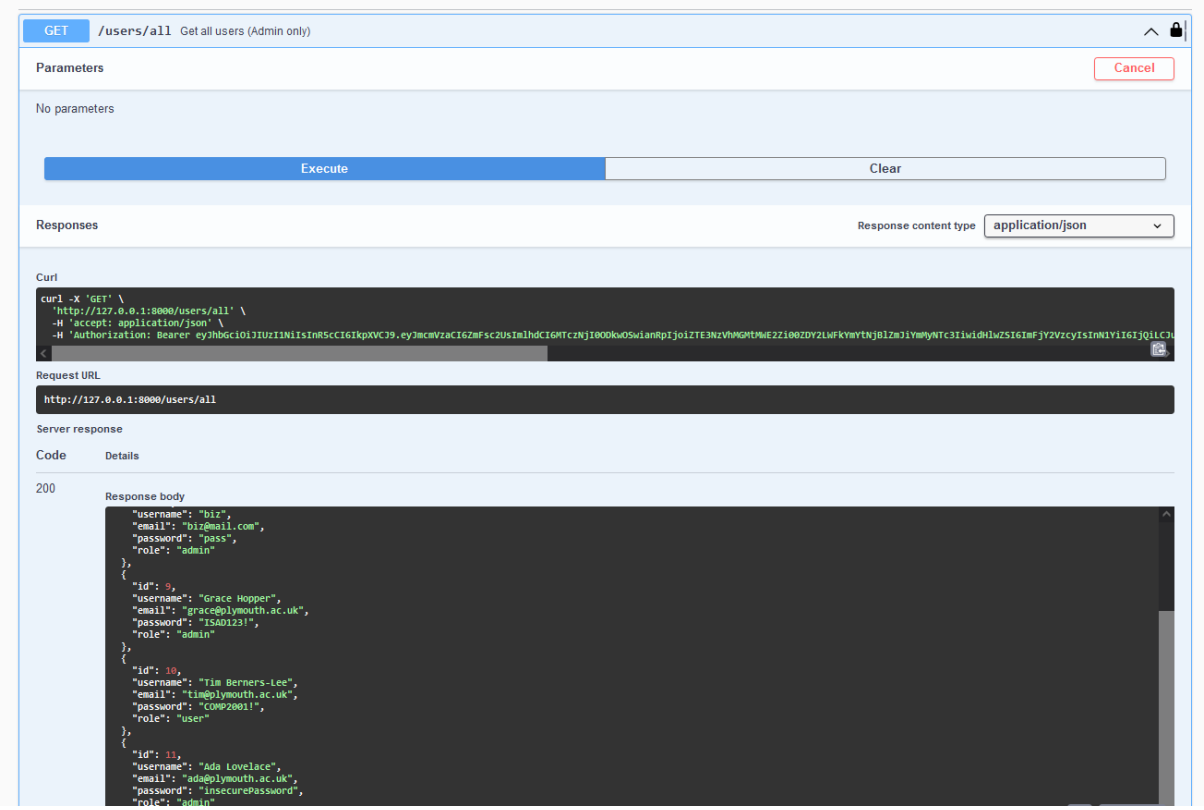
Testing:

Login:

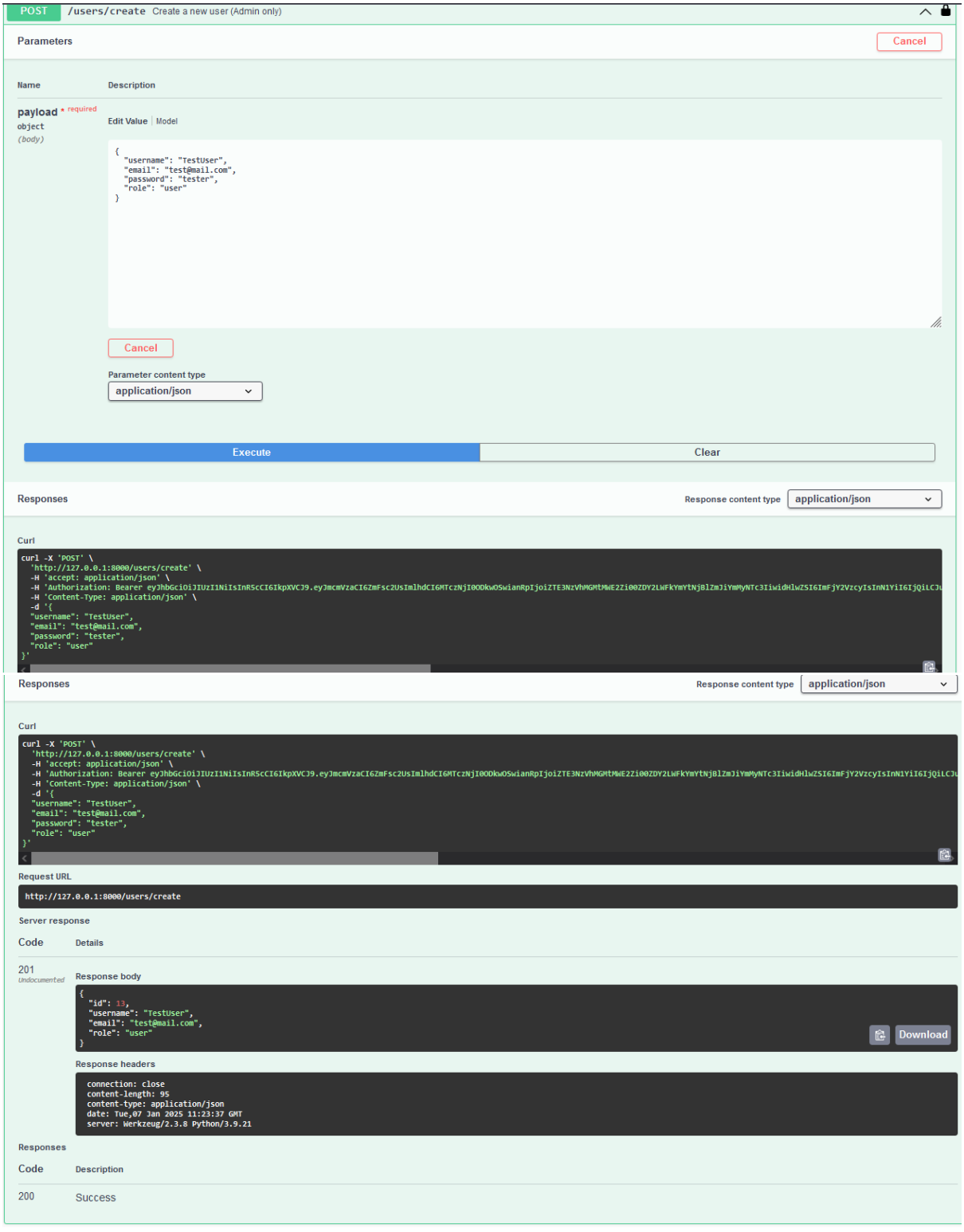




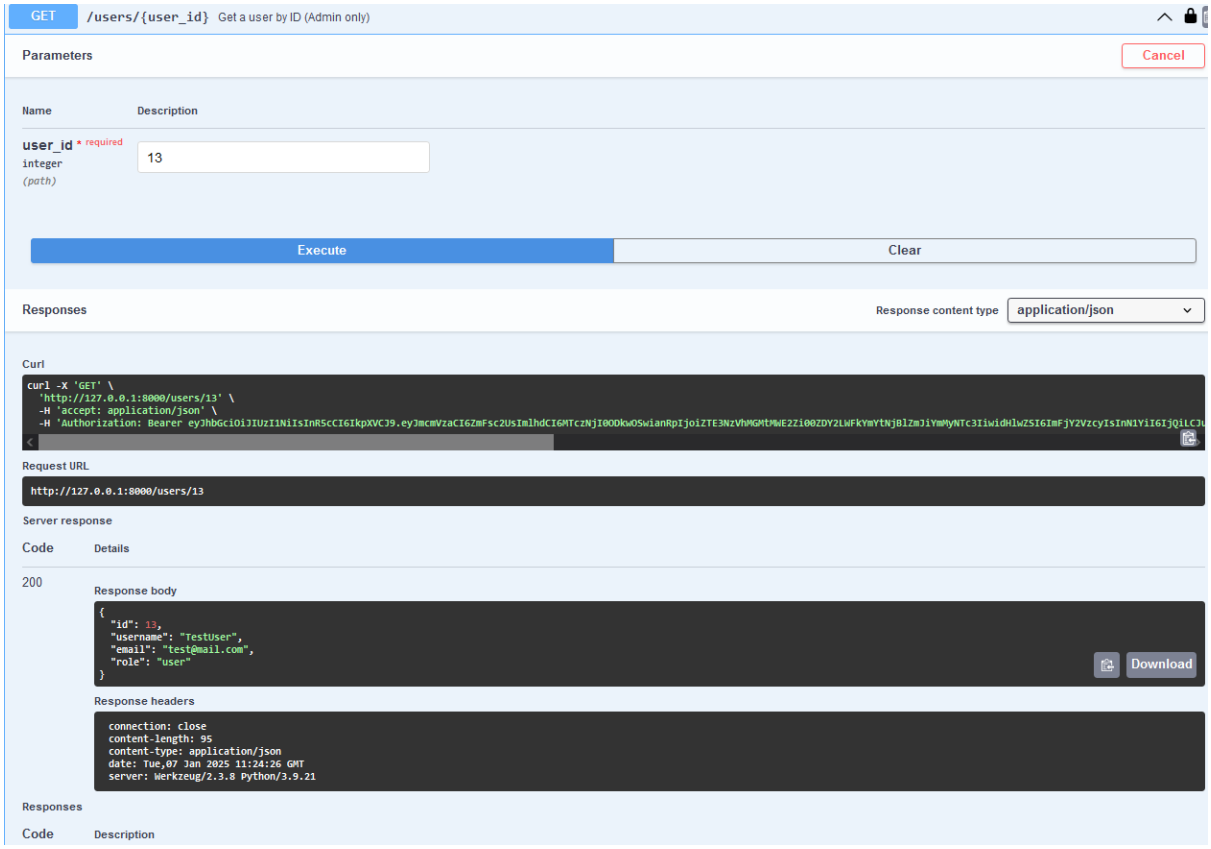
GET /users/all



POST /users/create

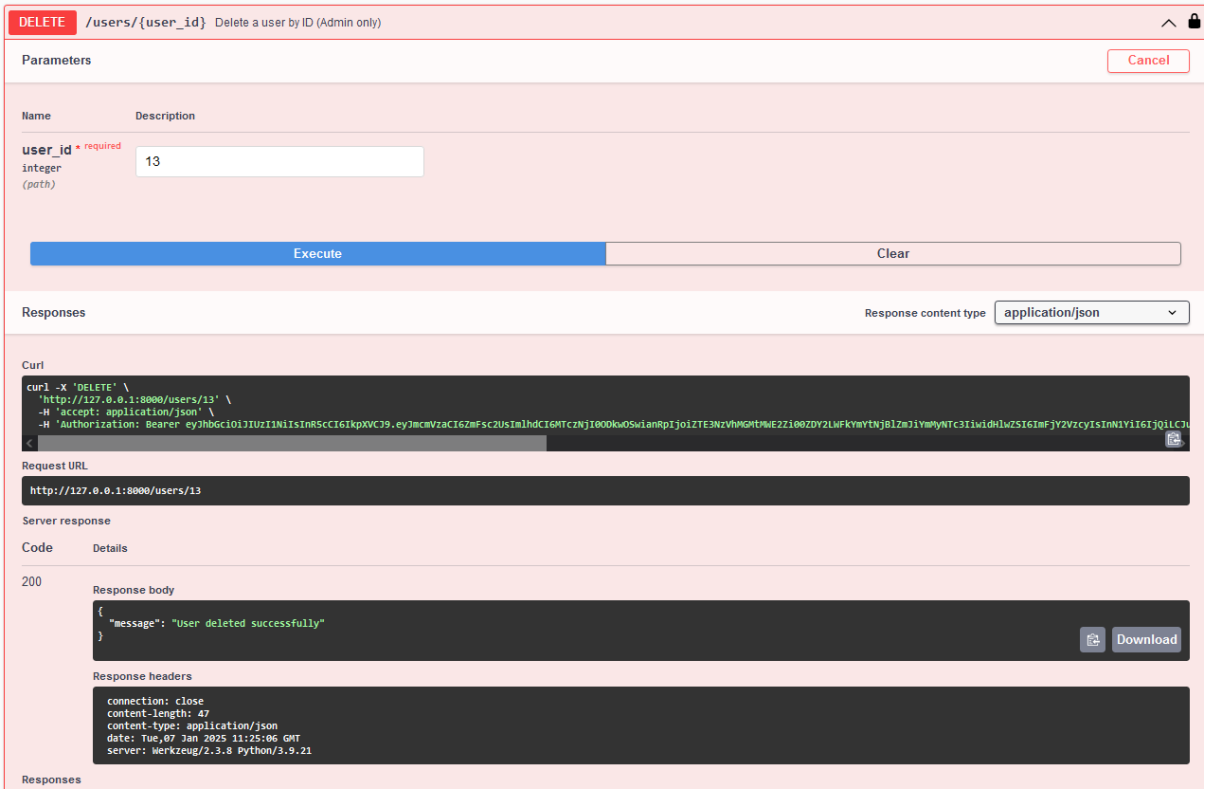


GET /users/(int id)

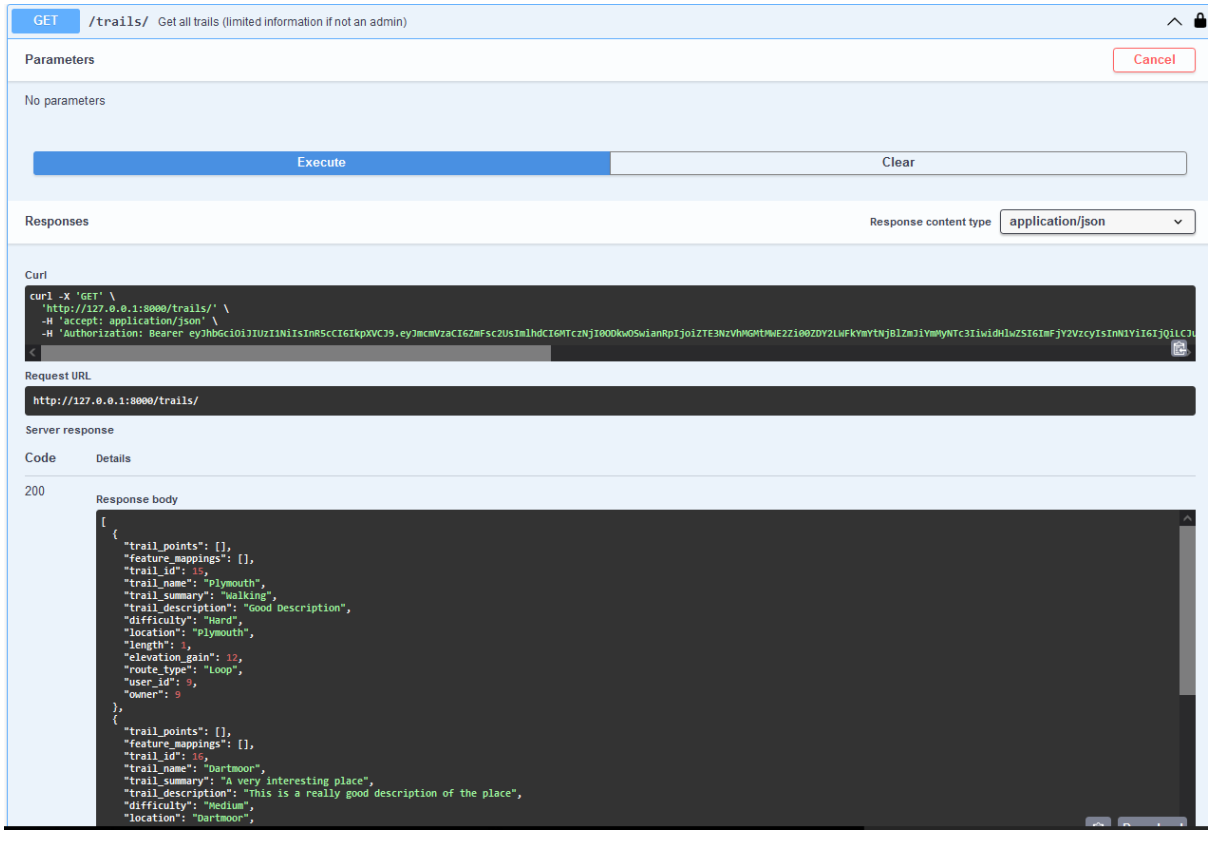


PUT /users/(int id)

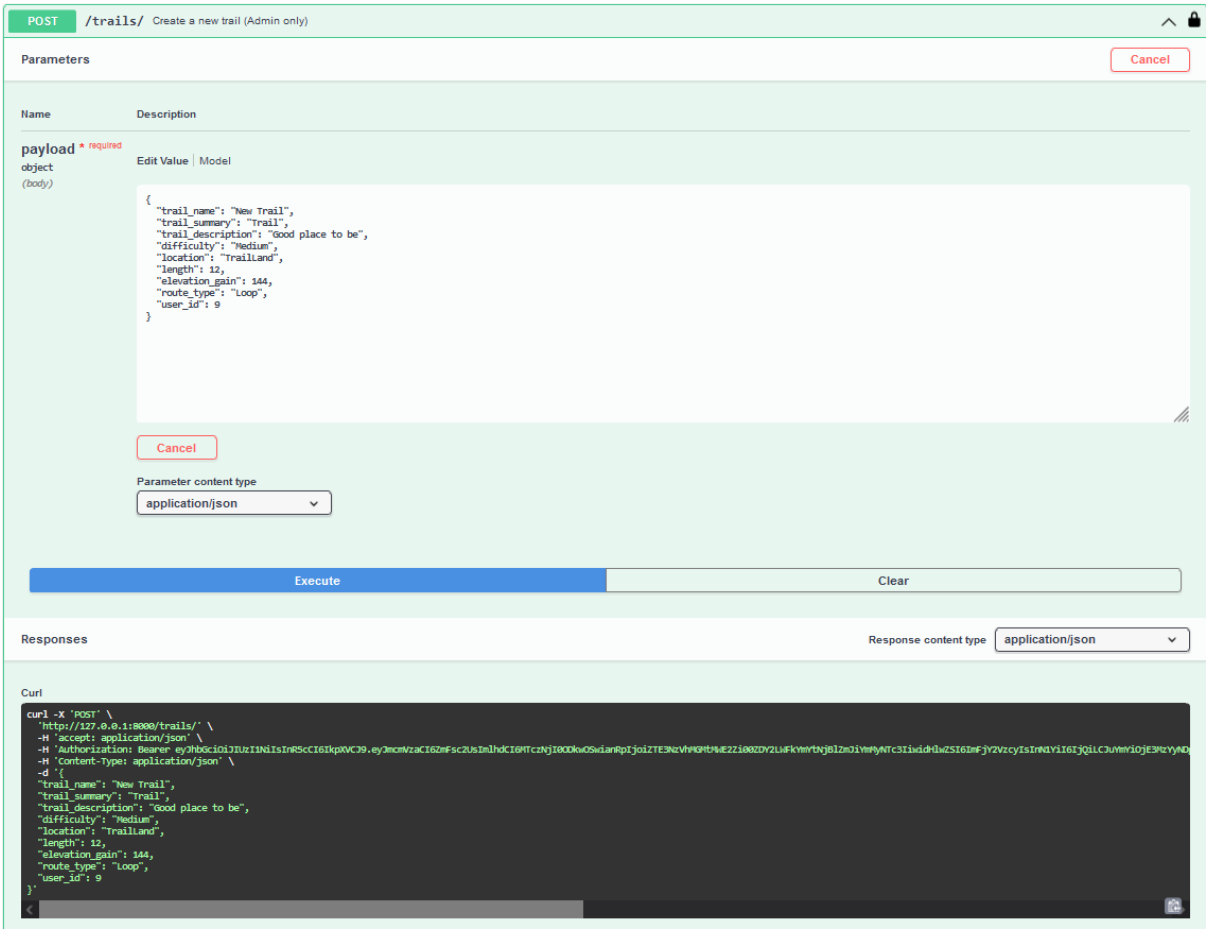
DELETE /users/(int id)

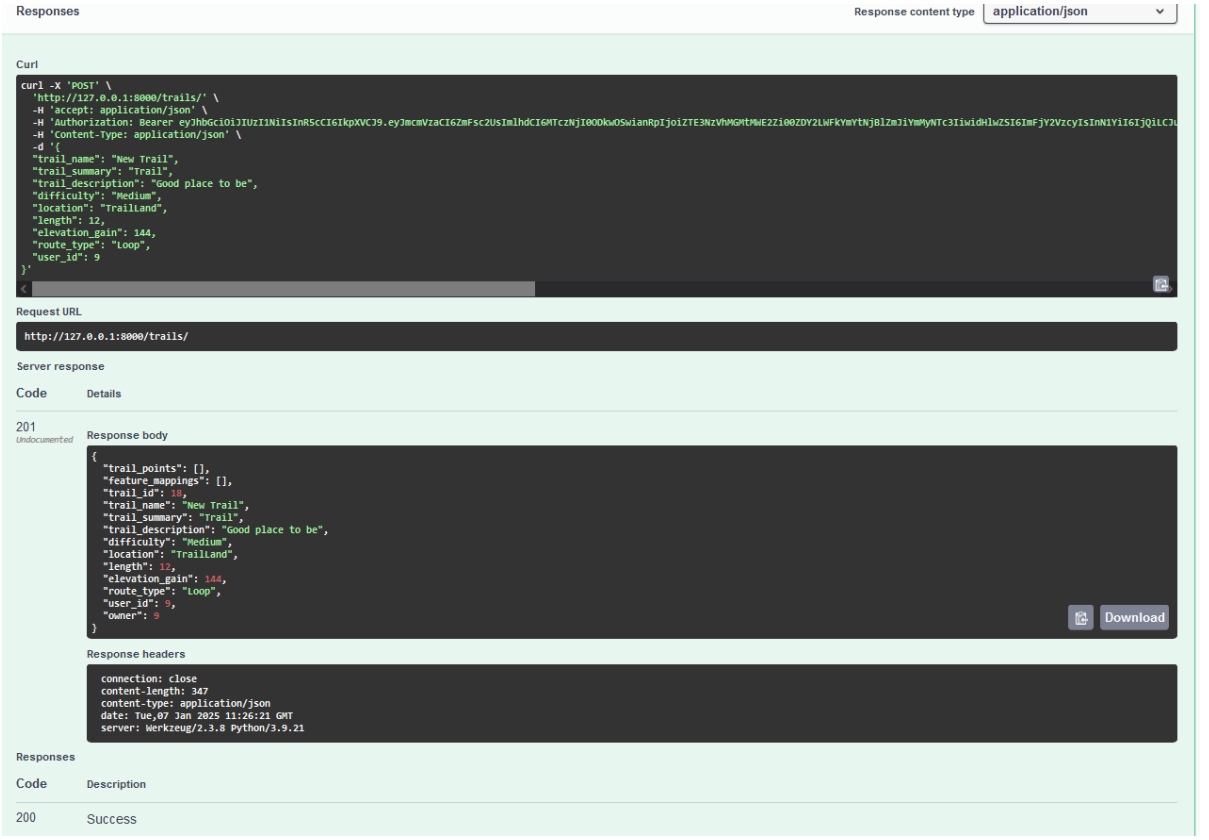

DELETE /users/{user_id} Delete a user by ID (Admin only)
Parameters
Cancel
Name
Description
user_id * required
integer
(path)
13
Execute
Clear
Responses
Response content type
application/json
Curl
curl -X 'DELETE' \
'http://127.0.0.1:8000/users/13' \
-H 'accept: application/json' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczNjI0ODkwOSwianRpIjoiZTE3NzVhMGMtMWE2Zi00ZDY2LWFkYmYtNjBlZmJiYmMyNTc3IiwidHlwZSI6ImFjY2VzcyIsInN1YiI6IjQiLCJu
Request URL
http://127.0.0.1:8000/users/13
Server response
Code
Details
200
Response body
{
"message": "User deleted successfully"
}
Download
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/

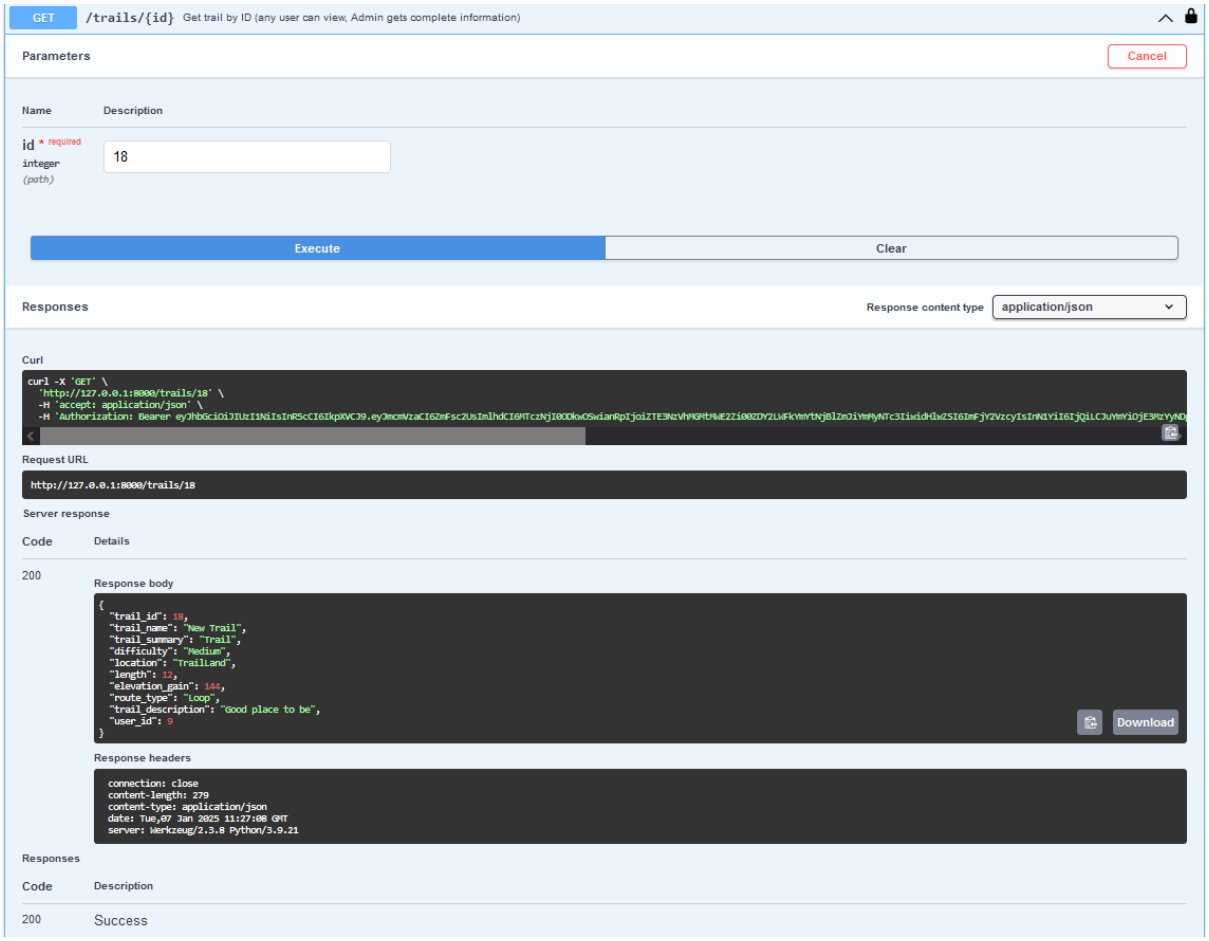

GET /trails/ Get all trails (limited information if not an admin)
Parameters
Cancel
No parameters
Execute
Clear
Responses
Response content type
application/json
Curl
curl -X 'GET' \
'http://127.0.0.1:8000/trails/' \
-H 'accept: application/json' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczNjI0ODkwOSwianRpIjoiZTE3NzVhMGMtMWE2Zi00ZDY2LWFkYmYtNjBlZmJiYmMyNTc3IiwidHlwZSI6ImFjY2VzcyIsInN1YiI6IjQiLCJu
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/

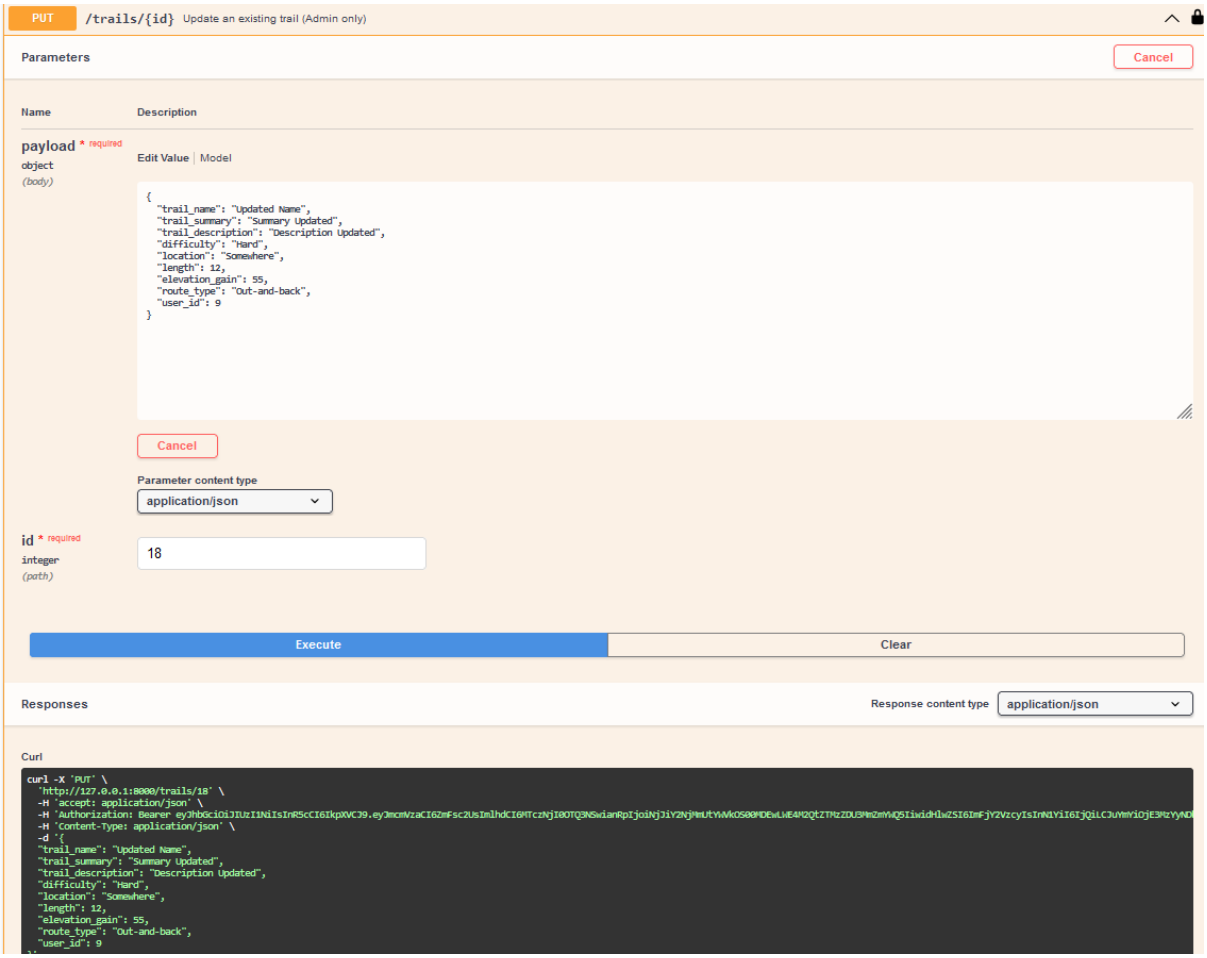

POST
/trails/
Create a new trail (Admin only)
Parameters
Cancel
Name
Description
payload * required
object
(body)
Edit Value | Model
{
"trail_name": "New Trail",
"trail_summary": "Trail",
"trail_description": "Good place to be",
"difficulty": "Medium",
"location": "TrailLand",
"length": 12,
"elevation_gain": 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.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczNjI0ODkwOSwianRpIjoiZTE3NzVhMGMtMWE2Zi00ZDY2LWFkYmYtNjBlZmJiYmMyNTc3IiwidHlwZSI6ImFjY2VzcyIsInN1YiI6IjQiLCJuYmYiOjE3MzYyNDg
-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
}'

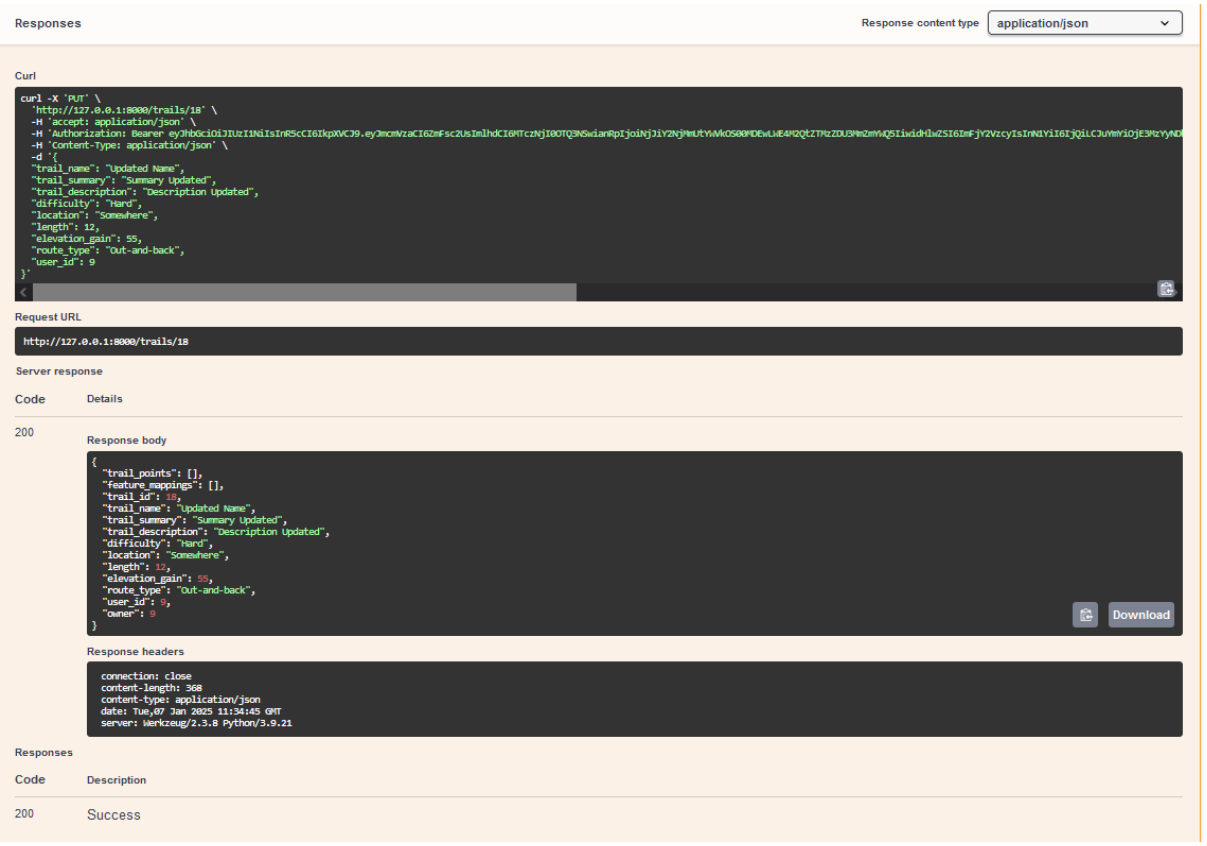

Responses
Response content type
application/json
Curl
curl -X 'POST' \
'http://127.0.0.1:8000/trails/' \
-H 'accept: application/json' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczNjI0ODkwOSwianRpIjoiZTE3NzVhMGMtMWE2Zi00ZDY2LWFkYmYtNjBlZmJiYmMyNTc3IiwidHlwZSI6ImFjY2VzcyIsInN1YiI6IjQiLCJu
-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": 19,
"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)

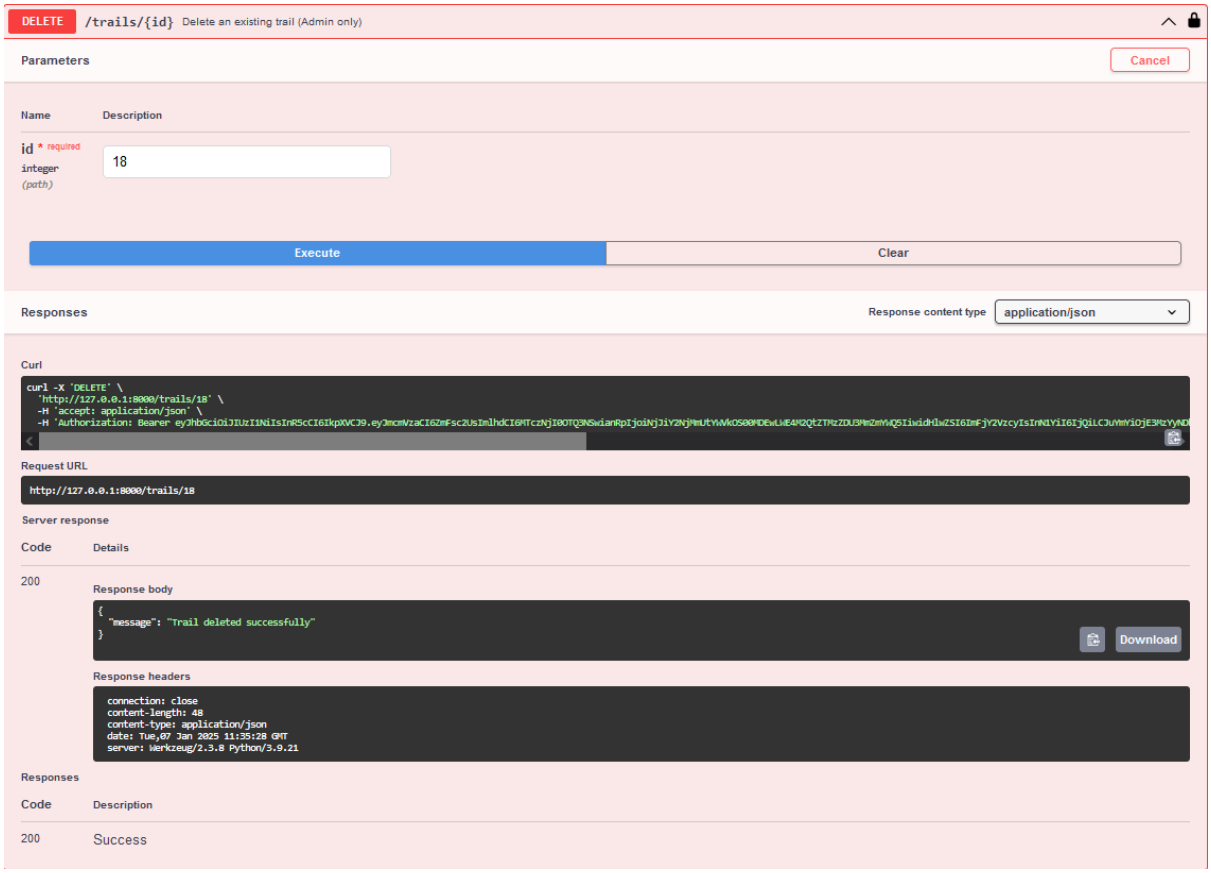


PUT /trails/(int id)


PUT
/trails/{id}
Update an existing trail (Admin only)
Parameters
Cancel
Name
Description
payload * required
object
(body)
Edit Value | Model
{
"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
integer
(path)
18
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.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczNjI0OTQ3NSwianRpIjoiNjJiY2NjMmUtYWVkOS00MDEwLWE4M2QtZTMzZDU3MmZmYWQ5IiwidHlwZSI6ImFjY2VzcyIsInN1YiI6IjQiLCJuYmYiOjE3MzYyND
-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
curl -X 'PUT' \
'http://127.0.0.1:8000/trails/18' \
-H 'accept: application/json' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczNjI0OTQ3NSwianRpIjoiNjJiY2NjMmUtYWVkOS00MDEwLWE4M2QtZTMzZDU3MmZmYWQ5IiwidHlwZSI6ImFjY2VzcyIsInN1YiI6IjQiLCJuYmYiOjE3MzYyND
-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
}'
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].