Trail Microservice

Introduction

This document outlines the development and implementation of the Trail Microservice. The document is sorted into theses sections: starting off with the background of the microservice and its place within the larger Trail application is a part of.

This is followed by the any Legal, Social, Ethical, and Professional (LSEP) considerations the project has faced, and the measures taken to address them, such as data privacy, integrity, and security. This is preceded by a detailed design section featuring UML diagrams and the entity relationship diagram (ERD) which will demonstrate the database these features are built off from. This will be built upon in the Implementation section, going into the technical detail alongside the technologies used to create this microservice. It will conclude with an evaluation, giving a thorough though of on strengths, weaknesses, and potential improvements of my implementation.

Below is the GitHub which contains the source code and the docker image that needs to run to access the microservice.

GitHub Repo: https://github.com/MxFrgsn/COMP2001-CW2
Docker Image: mxfrgsn/comp2001_cw2

Background

The trail microservice intends to be responsible for all CRUD operations of a trail, allowing people to view, edit, create and delete trails where necessary. This extends to trail attractions, the points of interests and activities the trail is best suited for like dog walking or a museum. In addition, a table and suitable CRUD operations for the location points across the trail – necessary as a trail is made up of a series of location points.

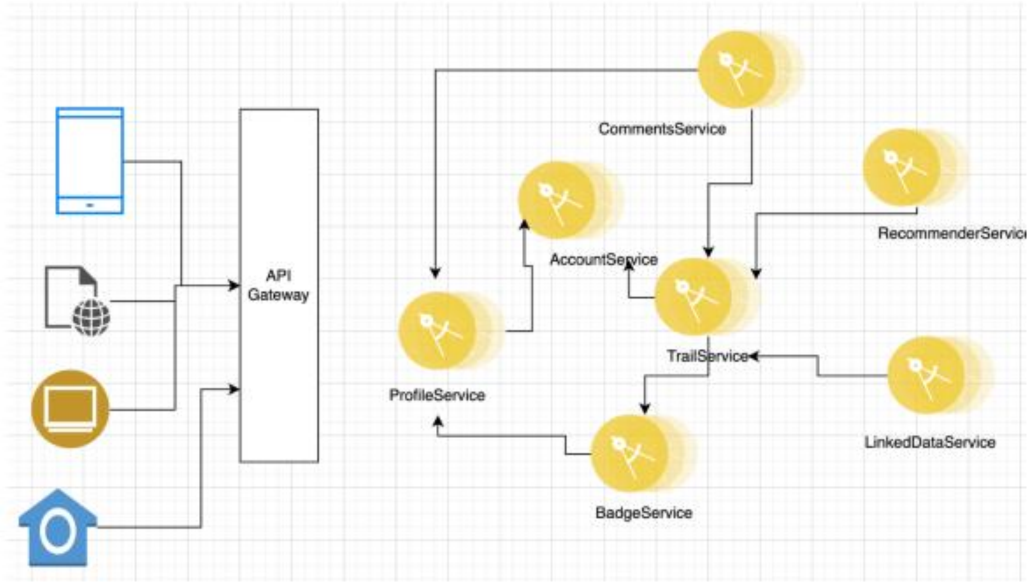
This is intended to be part of a larger Trail application. The Trail application intends to be a service to enhance user wellbeing, providing a reason to explore a particular area through the usage of trails to encourage physical activity and mindfulness. This intends to target people of all types, however the audience most likely to gain usefulness of this app is hikers, fitness and/or outdoor enthusiasts within the teenage to adult demographic.

The microservice primary functionality intends to play a crucial role of the app, by storing and handling the trail information through CRUD (Create, read, update and delete) operations. This will be done through a robust set RESTful API endpoints operation to allow the user to seamlessly interact with the database intending to at least meet these requirements:

- CRUD operation on trails.
- Anyone can view a trail but has limited view.
- Trails are a series of location points.
- Trails are owned by a user.
- As an admin I wish to create a new trail with information.
- As an admin I wish to edit an existing trail.
- As an admin, I wish to delete an existing trail.
- As a user, I wish to create a new trail.
- As a user, I wish to edit the trail(s) I own
- As a user, I wish to delete the trail(s) I own.

- To be protected from OWASP top 10 vulnerabilities where possible.

Here is how the entire Trail app would look, this document covers the trail service at the centre.



Legal, Social, Ethical and Professional (LSEP).

Legal

Legally, compliance with GDPR is essential, requiring transparency about what data is stored and where, ensuring only relevant data is collected. This microservice focuses on trial data, but the inclusion of authentication methods requires passwords and emails to be stored. Currently, these are not hashed, which would be critical for public release. The only personal data stored is names, a low-risk detail, with no age, address, or similar information, adhering to data minimization principles.

How location data is stored needs to be considered, especially as the application evolves to include features like suggesting nearby trails.

Social & Ethical

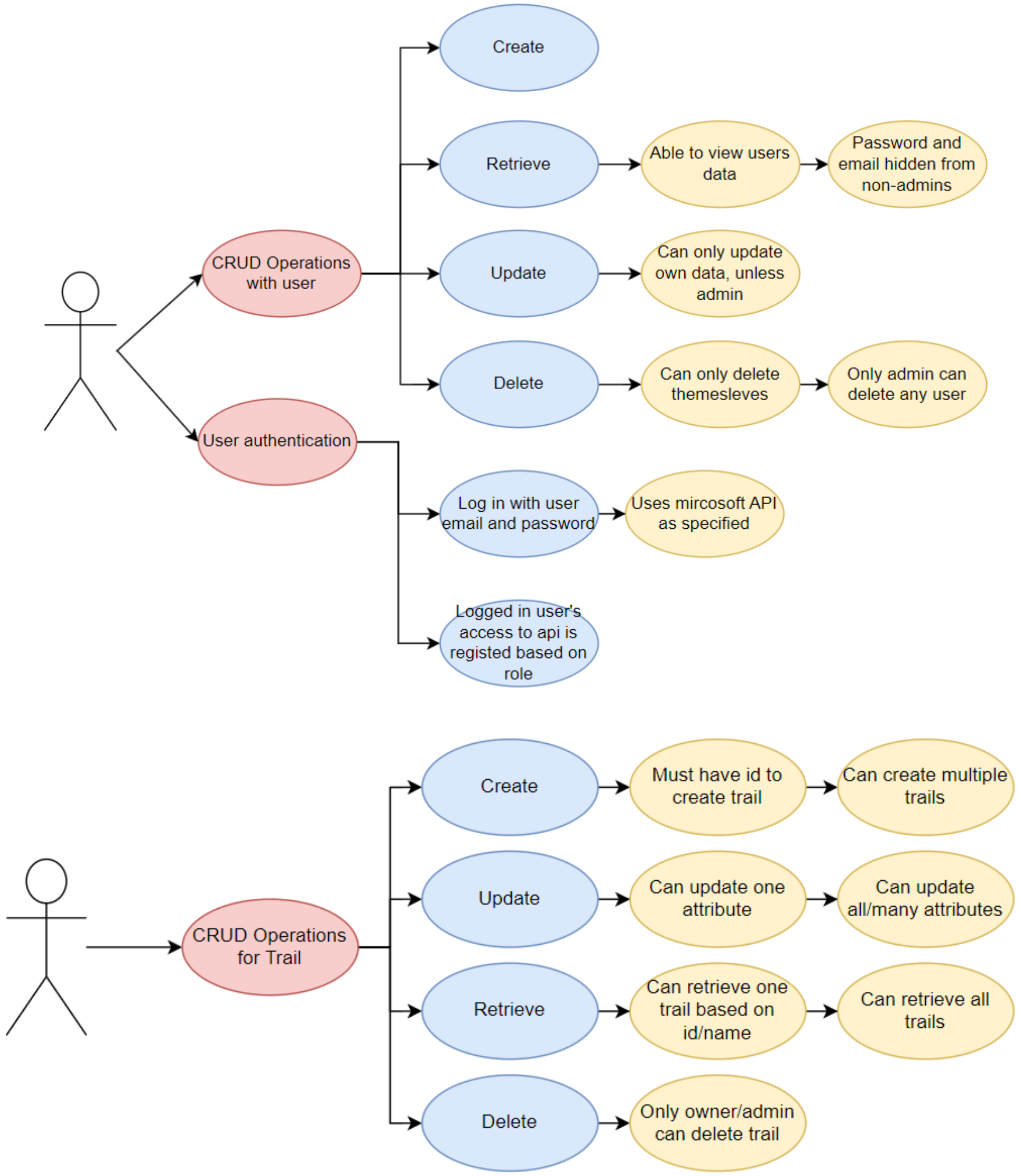
There are minimal ethical concerns aside from ensuring transparency about data store and manipulation. To address this, accounting should be introduced to improve oversight and trust.

Professional

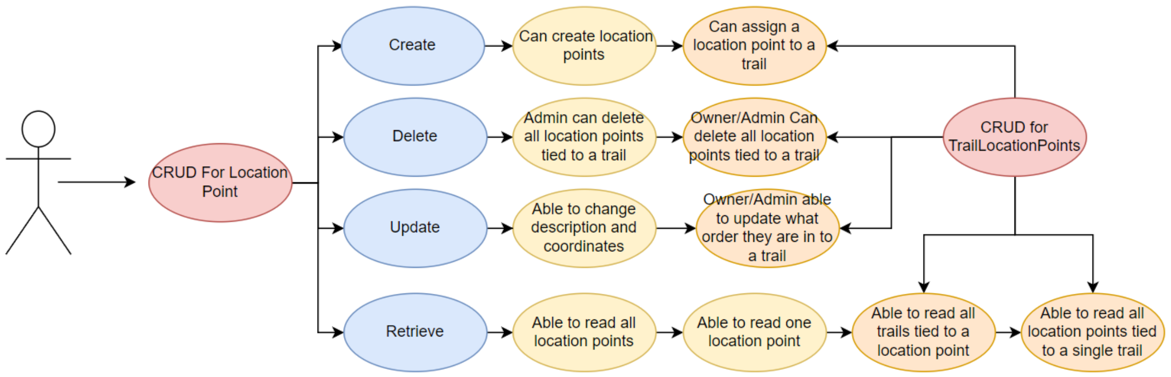
There are numerous professional issues to consider. Firstly, scalability and performance are always a key consideration. To ensure this, a normalized database and fast network ensure high performance but will need fine-tuning as this microservice integrates with others, particularly on software level. The use of ODBC and Python may limit portability due ODBC inflexible nature, causing potential issues if differing databases are used in other microservices. While the functions are internally tested, the absence of robust unit testing may overlook potential issues or performance bottlenecks in real-world use.

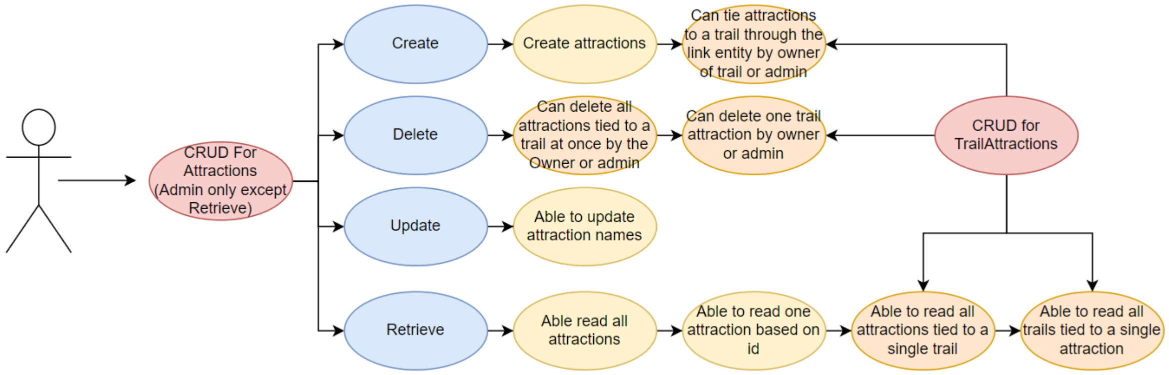
Design

Main features of Implemented Service.



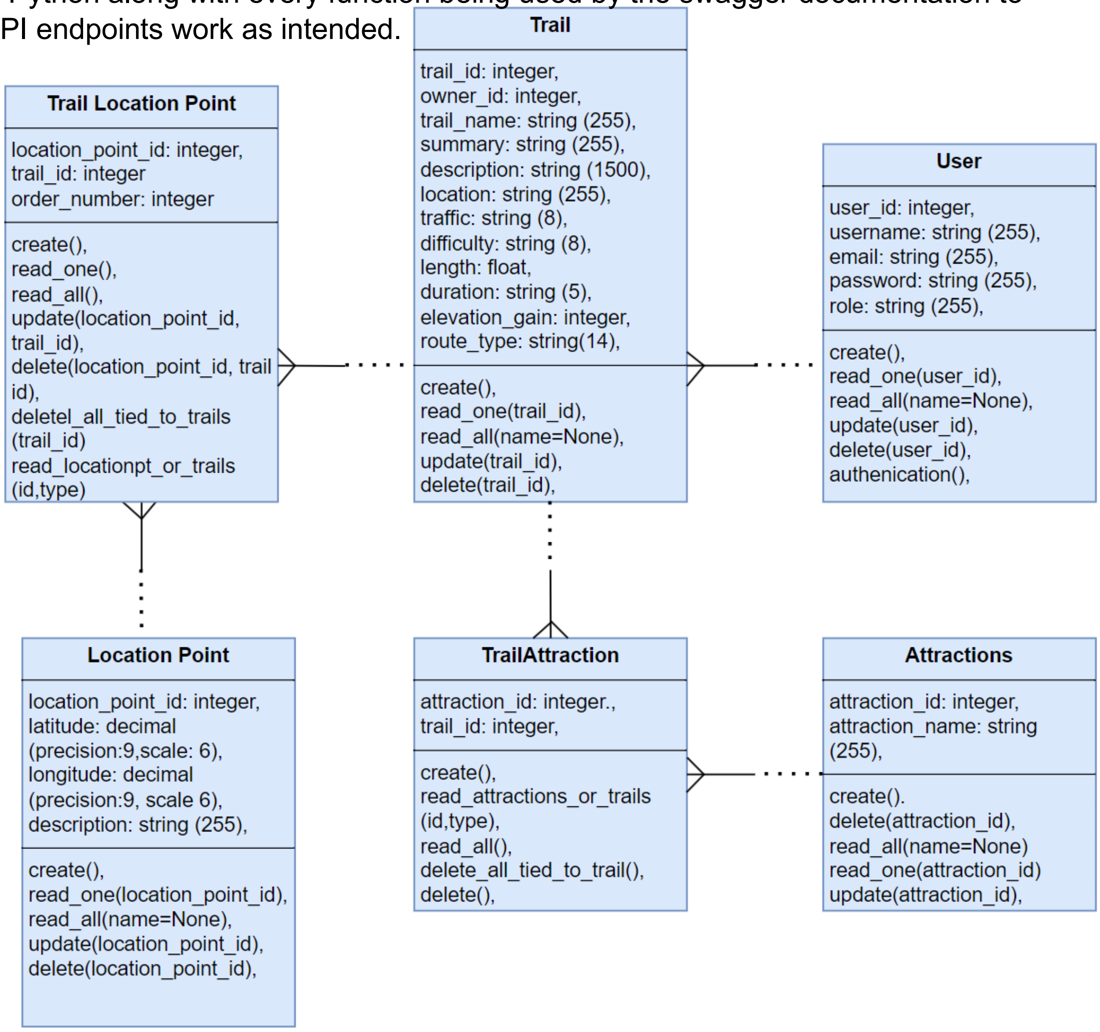
Create
Retrieve
Able to view users data
Password and email hidden from non-admins
CRUD Operations with user
Update
Can only update own data, unless admin
Delete
Can only delete themesleves
Only admin can delete any user
User authentication
Log in with user email and password
Uses mircosoft API as specified
Logged in user's access to api is registed based on role
Create
Must have id to create trail
Can create multiple trails
Update
Can update one attribute
Can update all/many attributes
CRUD Operations for Trail
Retrieve
Can retrieve one trail based on id/name
Can retrieve all trails
Delete
Only owner/admin can delete trail


CRUD For Location Point
Create
Can create location points
Can assign a location point to a trail
Delete
Admin can delete all location points tied to a trail
Owner/Admin Can delete all location points tied to a trail
CRUD for TrailLocationPoints
Update
Able to change description and coordinates
Owner/Admin able to update what order they are in to a trail
Retrieve
Able to read all location points
Able to read one location point
Able to read all trails tied to a location point
Able to read all location points tied to a single trail


CRUD For Attractions (Admin only except Retrieve)
Create
Create attractions
Can tie attractions to a trail through the link entity by owner of trail or admin
Delete
Can delete all attractions tied to a trail at once by the Owner or admin
Can delete one trail attraction by owner or admin
CRUD for TrailAttractions
Update
Able to update attraction names
Retrieve
Able read all attractions
Able to read one attraction based on id
Able to read all attractions tied to a single trail
Able to read all trails tied to a single attraction

Class diagram

This shows all the attributes of each table used within the database, the exact data type used within Python along with every function being used by the swagger documentation to make the API endpoints work as intended.



Trail

trail_id: integer,
owner_id: integer,
trail_name: string (255),
summary: string (255),
description: string (1500),
location: string (255),
traffic: string (8),
difficulty: string (8),
length: float,
duration: string (5),
elevation_gain: integer,
route_type: string(14),

create(),
read_one(trail_id),
read_all(name=None),
update(trail_id),
delete(trail_id),

Trail Location Point

location_point_id: integer,
trail_id: integer
order_number: integer

create(),
read_one(),
read_all(),
update(location_point_id, trail_id),
delete(location_point_id, trail id),
deletel_all_tied_to_trails (trail_id)
read_locationpt_or_trails (id,type)

User

user_id: integer,
username: string (255),
email: string (255),
password: string (255),
role: string (255),

create(),
read_one(user_id),
read_all(name=None),
update(user_id),
delete(user_id),
authenication(),

Location Point

location_point_id: integer,
latitude: decimal (precision:9,scale: 6),
longitude: decimal (precision:9, scale 6),
description: string (255),

create(),
read_one(location_point_id),
read_all(name=None),
update(location_point_id),
delete(location_point_id),

TrailAttraction

attraction_id: integer.,
trail_id: integer,

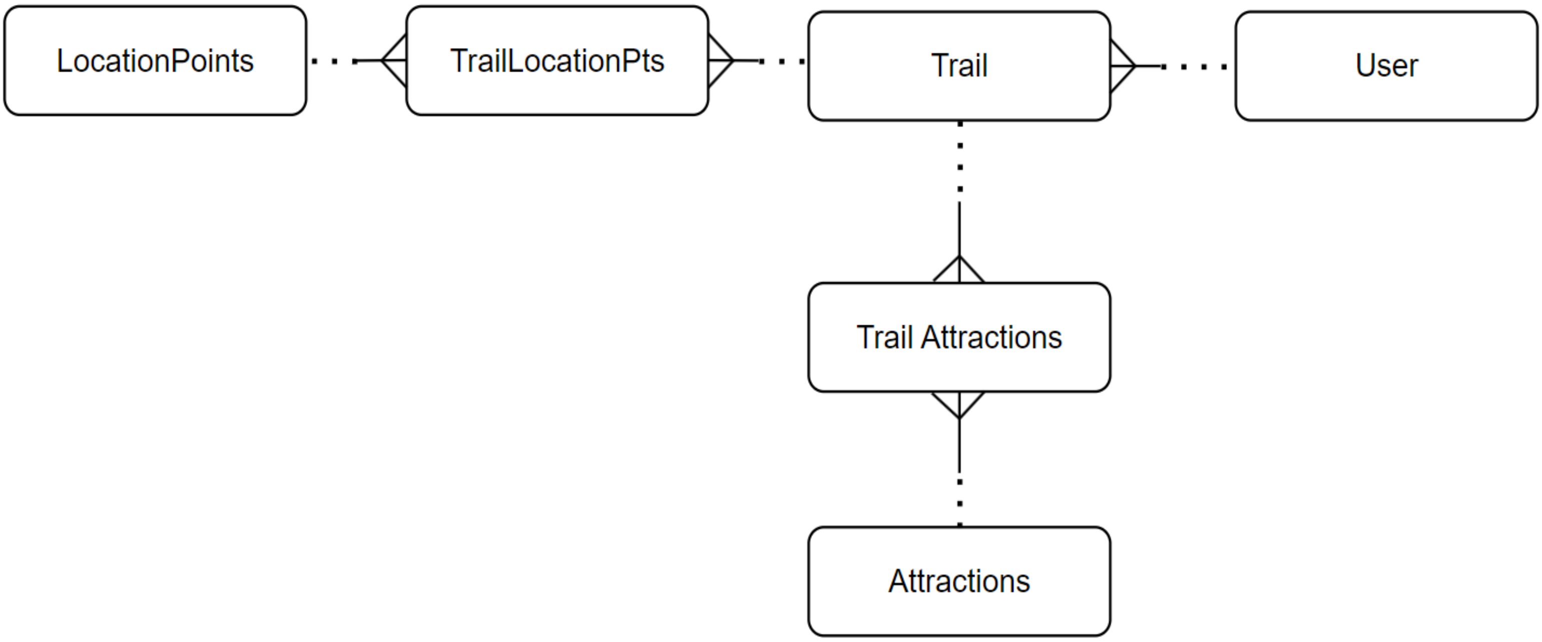
create(),
read_attractions_or_trails (id,type),
read_all(),
delete_all_tied_to_trail(),
delete(),

Attractions

attraction_id: integer,
attraction_name: string (255),

create().
delete(attraction_id),
read_all(name=None)
read_one(attraction_id)
update(attraction_id),

ERD

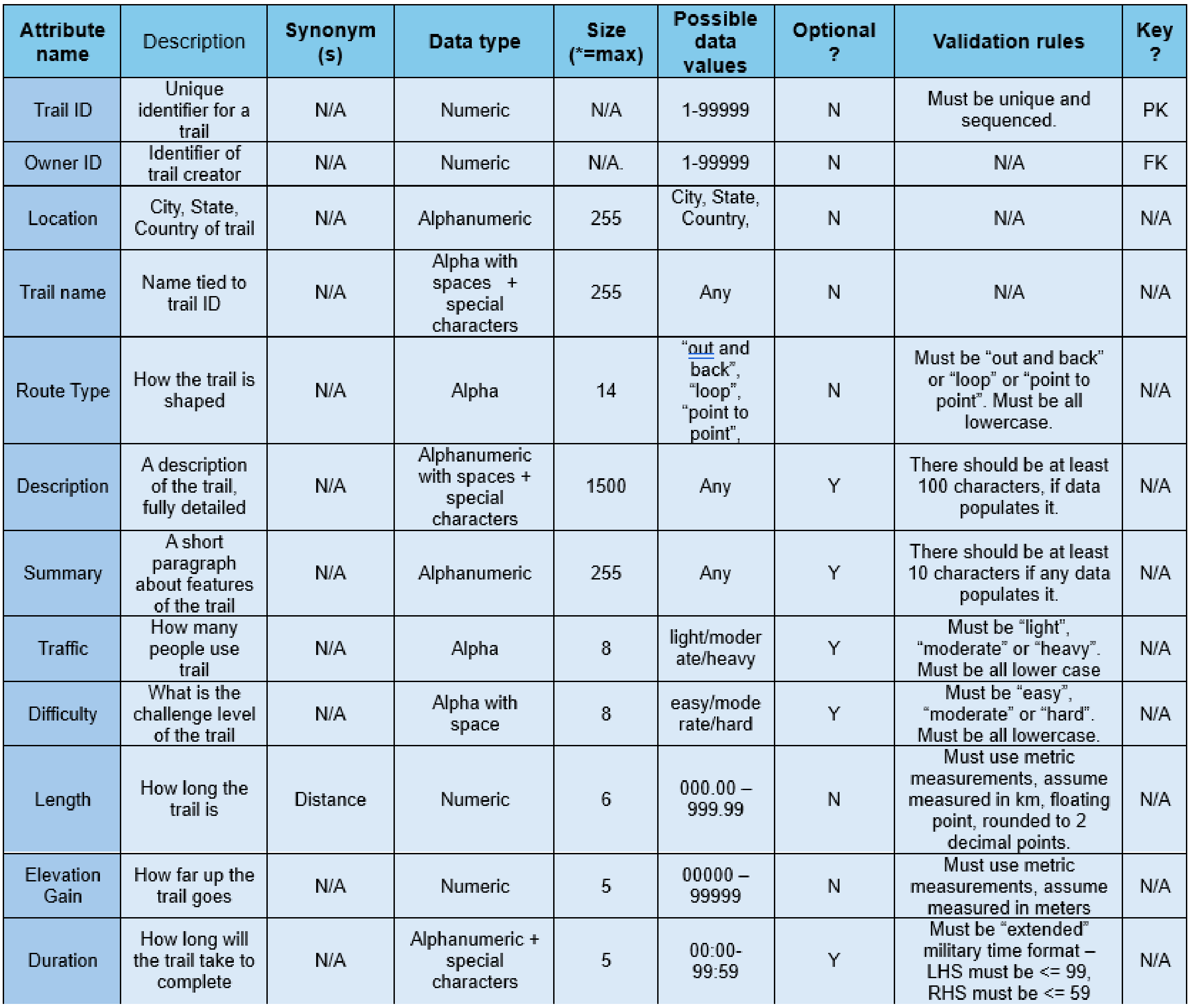


Entities in database:

Entities	Description	Relationship
User	Holds user data, used for to know who's signed in and their permissions.	Many to one to trail
Trail	Holds trail data and who the trail	Many to one to both trail attraction and trail location point
Attraction	Holds attraction data that could be tied to a trail	Many to one to trail attraction
Trail Attraction	Link entity that ties attractions to trails	Many to one to attraction and trail
Location Point	Holds location points, used to create location points across a trail	Many to one to trail location point
Trail Location Point	Link entity tying what location points are along the trail and in what order	Many to one to trail and location point

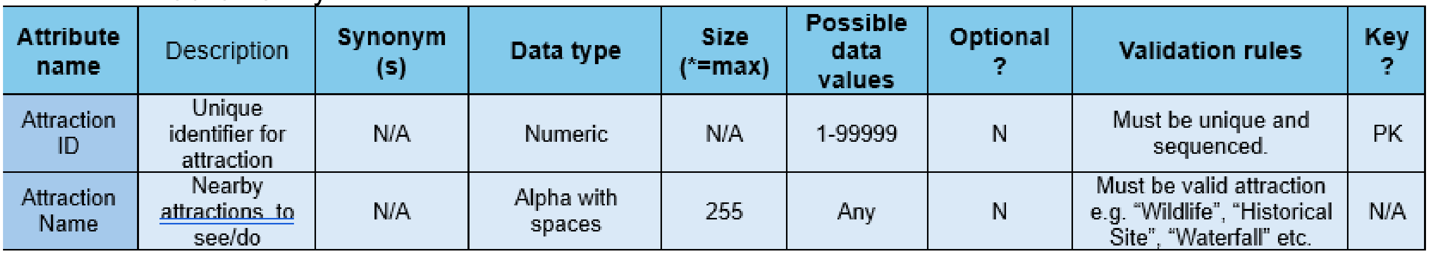
Field definition grids of the entities.

Trail Entity



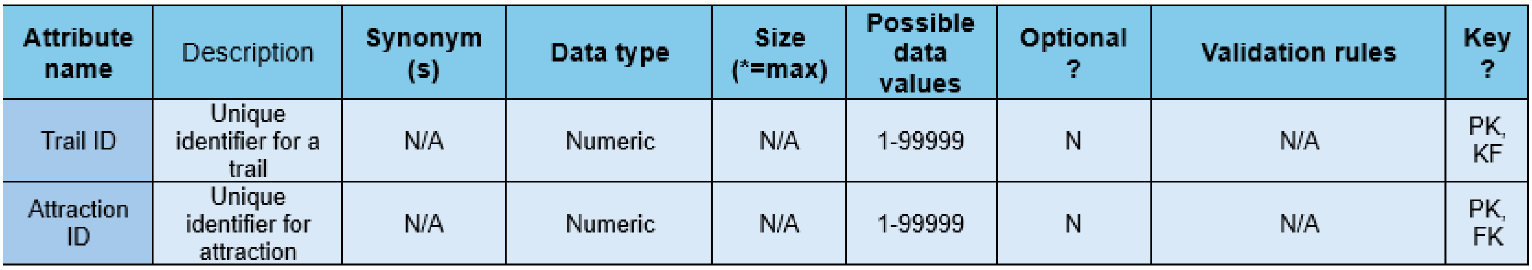
Attribute name	Description	Synonym (s)	Data type	Size (*=max)	Possible data values	Optional ?	Validation rules	Key ?
Trail ID	Unique identifier for a trail	N/A	Numeric	N/A	1-99999	N	Must be unique and sequenced.	PK
Owner ID	Identifier of trail creator	N/A	Numeric	N/A.	1-99999	N	N/A	FK
Location	City, State, Country of trail	N/A	Alphanumeric	255	City, State, Country,	N	N/A	N/A
Trail name	Name tied to trail ID	N/A	Alpha with spaces + special characters	255	Any	N	N/A	N/A
Route Type	How the trail is shaped	N/A	Alpha	14	"out and back", "loop", "point to point",	N	Must be "out and back" or "loop" or "point to point". Must be all lowercase.	N/A
Description	A description of the trail, fully detailed	N/A	Alphanumeric with spaces + special characters	1500	Any	Y	There should be at least 100 characters, if data populates it.	N/A
Summary	A short paragraph about features of the trail	N/A	Alphanumeric	255	Any	Y	There should be at least 10 characters if any data populates it.	N/A
Traffic	How many people use trail	N/A	Alpha	8	light/moderate/heavy	Y	Must be "light", "moderate" or "heavy". Must be all lower case	N/A
Difficulty	What is the challenge level of the trail	N/A	Alpha with space	8	easy/moderate/hard	Y	Must be "easy", "moderate" or "hard". Must be all lowercase.	N/A
Length	How long the trail is	Distance	Numeric	6	000.00 – 999.99	N	Must use metric measurements, assume measured in km, floating point, rounded to 2 decimal points.	N/A
Elevation Gain	How far up the trail goes	N/A	Numeric	5	00000 – 99999	N	Must use metric measurements, assume measured in meters	N/A
Duration	How long will the trail take to complete	N/A	Alphanumeric + special characters	5	00:00-99:59	Y	Must be "extended" military time format – LHS must be <= 99, RHS must be <= 59	N/A

Attraction entity



Attribute name	Description	Synonym (s)	Data type	Size (*=max)	Possible data values	Optional ?	Validation rules	Key ?
Attraction ID	Unique identifier for attraction	N/A	Numeric	N/A	1-99999	N	Must be unique and sequenced.	PK
Attraction Name	Nearby attractions to see/do	N/A	Alpha with spaces	255	Any	N	Must be valid attraction e.g. "Wildlife", "Historical Site", "Waterfall" etc.	N/A

Trail Attraction entity



Attribute name	Description	Synonym (s)	Data type	Size (*=max)	Possible data values	Optional ?	Validation rules	Key ?
Trail ID	Unique identifier for a trail	N/A	Numeric	N/A	1-99999	N	N/A	PK, KF
Attraction ID	Unique identifier for attraction	N/A	Numeric	N/A	1-99999	N	N/A	PK, FK

User entity

Attribute name	Description	Synonym (s)	Data type	Size (*=max)	Possible data values	Optional ?	Validation rules	Key ?
User ID	Unique Identifier for User	N/A	Numeric	N/A	1-99999	N	Must be unique and sequenced.	PK
Name	Named tied to user id	N/A	Alpha with Spaces + special characters	255	Any	N	Any alpha, only accents and hyphens acceptable.	N/A
Email	Email tied to user id	N/A	Alphanumeric + special characters	255	AAAA1111@gmail.com	N	Must fit standard email format, must contain "@" and ".", no spaces, must be unique	N/A
Password	Password tied to user id	N/A	Alphanumeric + special characters	128	Any	N	Must be at least 8 characters long, to enforce a degree of security	N/A

Location point entity

Attribute name	Description	Synonym (s)	Data type	Size (*=max)	Possible data values	Optional ?	Validation rules	Key ?
Location Point ID	Unique Identifier for location	N/A	Numeric	N/A	1-99999	N	Must be unique and sequenced.	PK,
Latitude	Latitude coordinate of for location point	N/A	Float	8	-90.000000° - +90.000000°,	N	Must be between 90.0000 and +90.0000, must always be a floating-point number, rounded to 6 decimal places	N/A
Longitude	Longitude coordinate of for location point	N/A	Float	9	-180.000000° - +180.000000°	N	Must be between 180.0000 - +180.0000, must always be a floating-point number, rounded to 6 decimal places	N/A
Description	Important information tied to location point	N/A	Any	255	Any	Y	N/A	N/A

Trail Location Point Entity

Attribute name	Description	Synonym (s)	Data type	Size (*=max)	Possible data values	Optional ?	Validation rules	Key ?
Location Point ID	Unique Identifier for location	N/A	Numeric	N/A	1-99999	N	Must be unique and sequenced.	PK, FK
Trail ID	Unique identifier for a trail	N/A	Numeric	N/A	1-99999	N	N/A	PK, KF
Order number	The order of which the location point goes across the trail	N/A	Numeric	N/A	1-99999	N	N/A	PK, KF

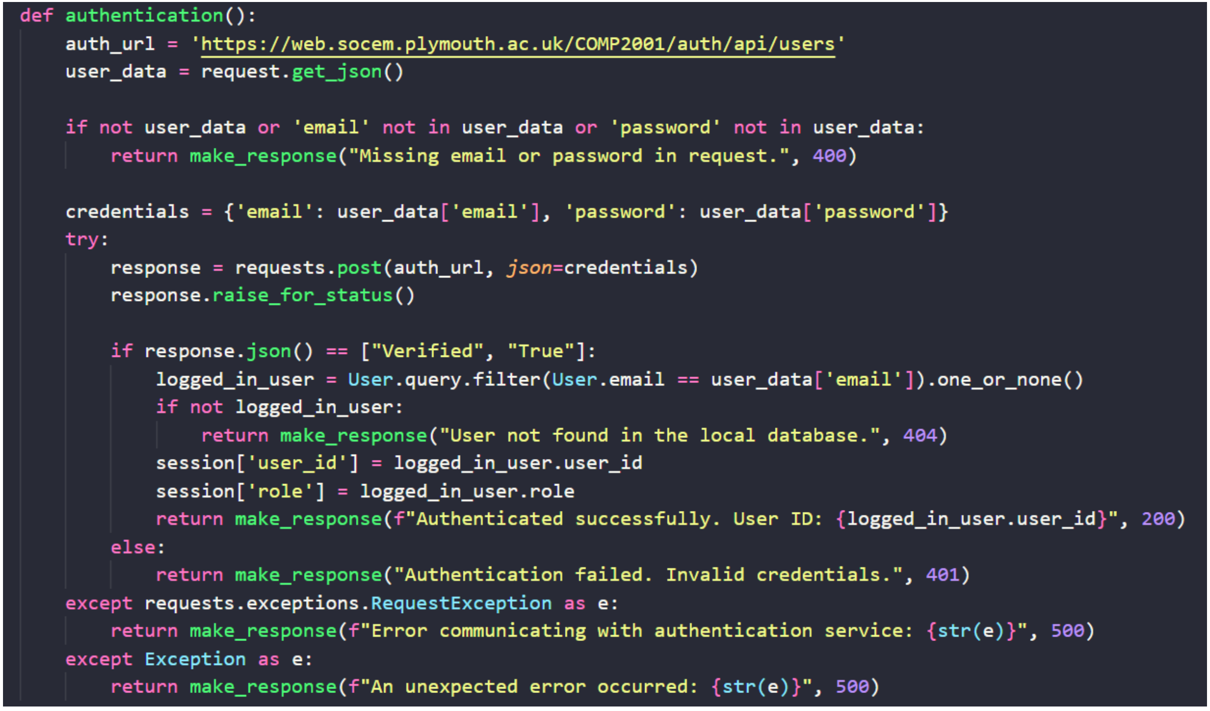
Implementation

Technologies used:

This microservice server-side code is written in Python, utilizing a Microsoft SQL Server for the database. Initial SQL setup operations are executed through a Jupyter Notebook using the Pyodbc library, leveraging ODBC driver. The Flask framework is employed to handle API routing, with Swagger providing documentation.

In conjunction, SQL Alchemy is utilized to manage data as an Object-Relational Mapper (ORM), mapping each table to a model in models.py, enforcing the necessary relationships and validation that match the SQL, which reduces the risk of SQL injections and maintains data integrity. Marshmallow is employed for serialization and deserialization between request bodies and Python objects, defining the schemas that control data visibility to the signed-in user.

The currently signed-in user is determined using Flask's built-in session management. By default, user 1 is signed in, set up with the admin role for testing purposes. Authentication occurs via the auth endpoint, changing the role based on the given role in the database if inputted data is verified by authenticator. This allows users to sign in using one of three accounts in the database, each having the user role with reduced permissions.

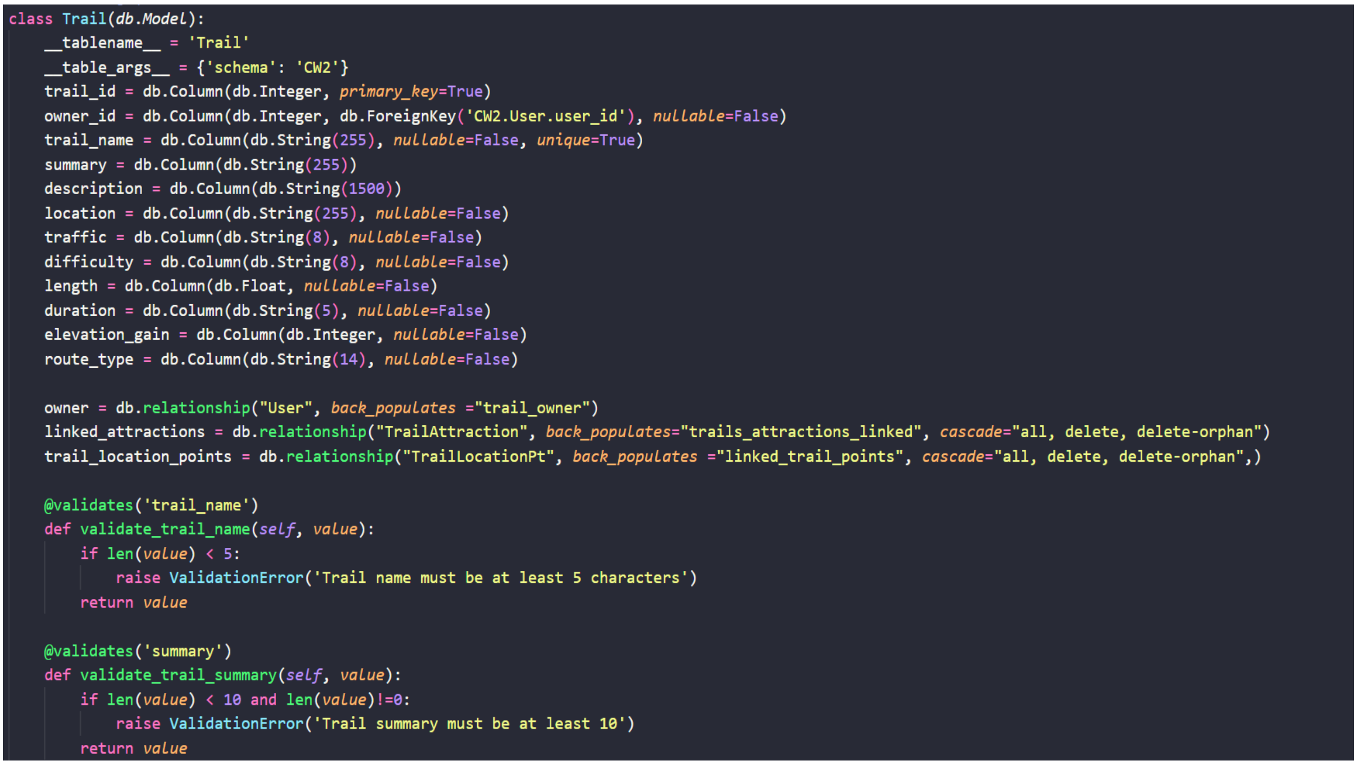


```python
def authentication():
    auth_url = 'https://web.socem.plymouth.ac.uk/COMP2001/auth/api/users'
    user_data = request.get_json()

    if not user_data or 'email' not in user_data or 'password' not in user_data:
        return make_response("Missing email or password in request.", 400)

    credentials = {'email': user_data['email'], 'password': user_data['password']}
    try:
        response = requests.post(auth_url, json=credentials)
        response.raise_for_status()

        if response.json() == ["Verified", "True"]:
            logged_in_user = User.query.filter(User.email == user_data['email']).one_or_none()
            if not logged_in_user:
                return make_response("User not found in the local database.", 404)
            session['user_id'] = logged_in_user.user_id
            session['role'] = logged_in_user.role
            return make_response(f"Authenticated successfully. User ID: {logged_in_user.user_id}", 200)
        else:
            return make_response("Authentication failed. Invalid credentials.", 401)
    except requests.exceptions.RequestException as e:
        return make_response(f"Error communicating with authentication service: {str(e)}", 500)
    except Exception as e:
        return make_response(f"An unexpected error occurred: {str(e)}", 500)
```

To host the microservice, a Docker image is provided, as specified at the end of the introduction. Alternatively, you can run it locally by following the GitHub README instructions.

SQL and Models:

The database that has been constructed within the SQL server follows the entity relationship diagram and the field definition grids within the design section of the document. By extension, all the models in models.py follow it. Below is a minor snippet of the trail model.



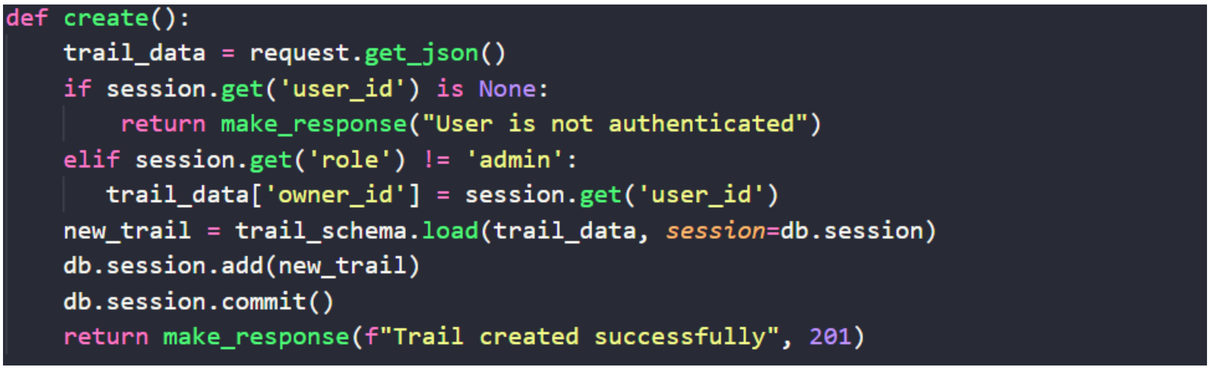
```python
class Trail(db.Model):
    __tablename__ = 'Trail'
    __table_args__ = {'schema': 'CW2'}
    trail_id = db.Column(db.Integer, primary_key=True)
    owner_id = db.Column(db.Integer, db.ForeignKey('CW2.User.user_id'), nullable=False)
    trail_name = db.Column(db.String(255), nullable=False, unique=True)
    summary = db.Column(db.String(255))
    description = db.Column(db.String(1500))
    location = db.Column(db.String(255), nullable=False)
    traffic = db.Column(db.String(8), nullable=False)
    difficulty = db.Column(db.String(8), nullable=False)
    length = db.Column(db.Float, nullable=False)
    duration = db.Column(db.String(5), nullable=False)
    elevation_gain = db.Column(db.Integer, nullable=False)
    route_type = db.Column(db.String(14), nullable=False)

    owner = db.relationship("User", back_populates ="trail_owner")
    linked_attractions = db.relationship("TrailAttraction", back_populates="trails_attractions_linked", cascade="all, delete, delete-orphan")
    trail_location_points = db.relationship("TrailLocationPt", back_populates ="linked_trail_points", cascade="all, delete, delete-orphan",)

    @validates('trail_name')
    def validate_trail_name(self, value):
        if len(value) < 5:
            raise ValidationError('Trail name must be at least 5 characters')
        return value

    @validates('summary')
    def validate_trail_summary(self, value):
        if len(value) < 10 and len(value)!=0:
            raise ValidationError('Trail summary must be at least 10')
        return value
```

CRUD:

Each table has a CRUD operation implemented, uses the HTTP request methods. The code for each function is similar, so I'll focus on the implementation on Trail functions, as its the primary functionality of this service. There is, however, some differences for the link entities of trail attraction and trail location point which will be discussed later.

Create

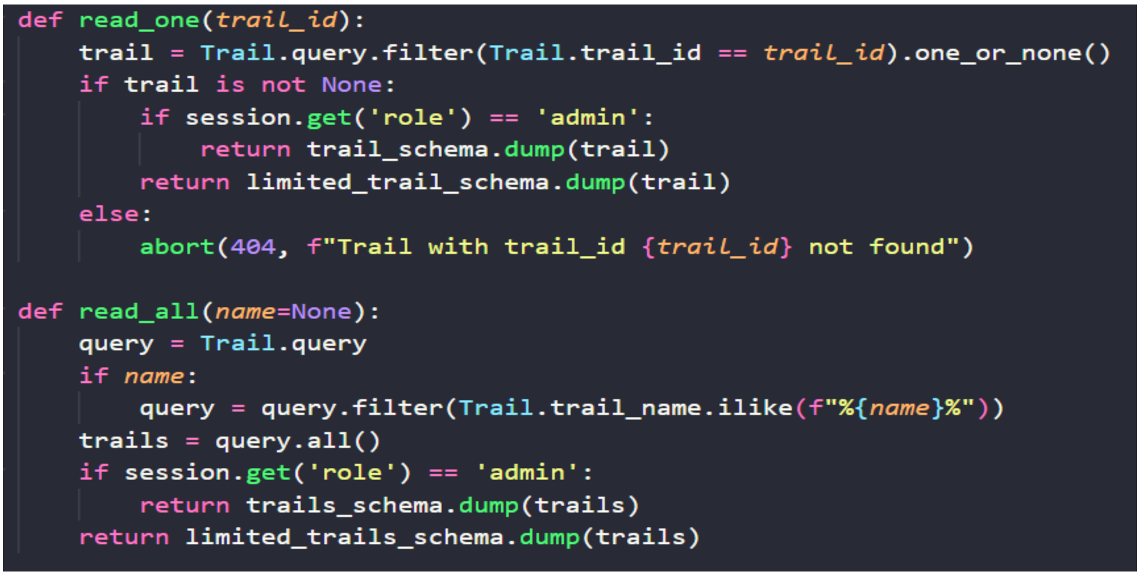
This function is taking in the request body using the get_json() function within the request within flask. Then, using the built in flask session management, it'll take the role of who's signed in, if it's an admin, it'll take the owner id within the request body, giving them the ultimate control, but if it it's a user, not matter the owner id they inputted, it'll change it to the currently signed in user and then commit the trail the database. The SQL database and the marshmallow ensures that trail information submitted is valid before submission.



```python
def create():
    trail_data = request.get_json()
    if session.get('user_id') is None:
        return make_response("User is not authenticated")
    elif session.get('role') != 'admin':
        trail_data['owner_id'] = session.get('user_id')
    new_trail = trail_schema.load(trail_data, session=db.session)
    db.session.add(new_trail)
    db.session.commit()
    return make_response(f"Trail created successfully", 201)
```

Retrieve:

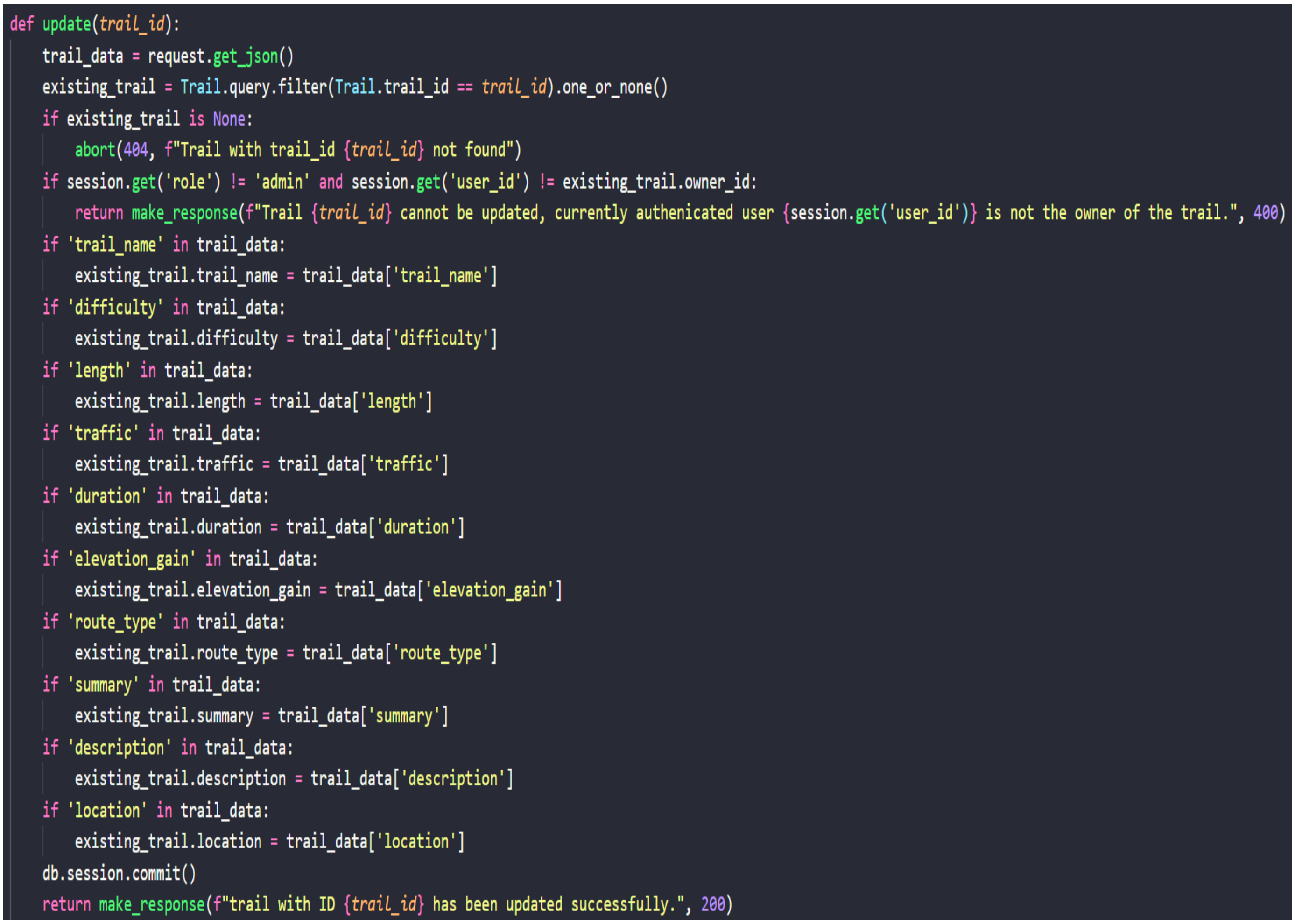
The read one is taking in a singular id, and returning all the information tied to that id using the schema serialized using Marshmallow. The read all just returns all the trail data stored in the database, however with a "name" parameter that allows the user to query a trail on the trail name, making it more user accessible. In both examples, the amount of information is limited if the signed in user is not an admin, hiding data backend data like the trail id.



```
def read_one(trail_id):
    trail = Trail.query.filter(Trail.trail_id == trail_id).one_or_none()
    if trail is not None:
        if session.get('role') == 'admin':
            return trail_schema.dump(trail)
        return limited_trail_schema.dump(trail)
    else:
        abort(404, f"Trail with trail_id {trail_id} not found")

def read_all(name=None):
    query = Trail.query
    if name:
        query = query.filter(Trail.trail_name.ilike(f"%{name}%"))
    trails = query.all()
    if session.get('role') == 'admin':
        return trails_schema.dump(trails)
    return limited_trails_schema.dump(trails)
```

Update

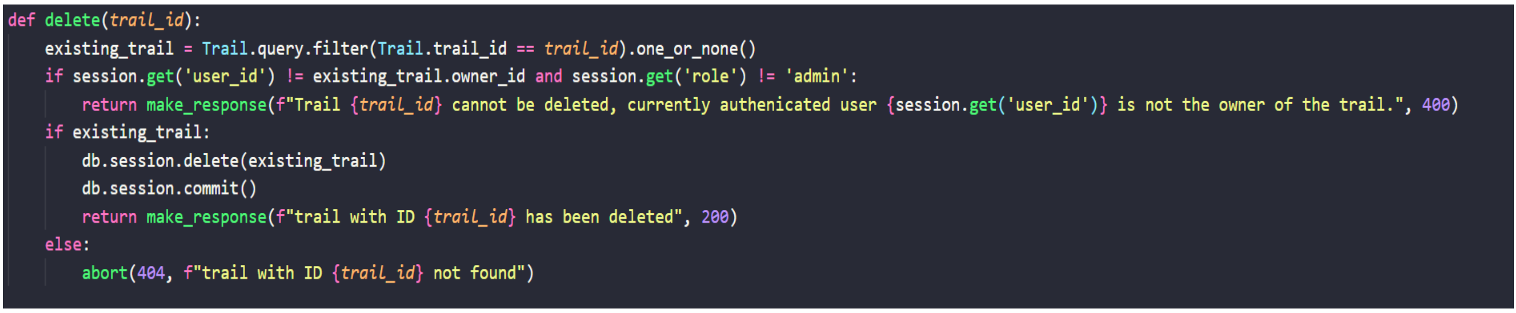
The update function takes in the trail id and similarly takes in the request body using the get_json function. First, it checks whether the trail exists in the first place, then checks against the signed in user to the owner of the id if they aren't an admin, ensuring only the owner can update their own trail. Then, as the PATCH method is used over PUT, to ensure that only the specified fields are updated rather than everything, the request body must be scanned for each attribute before commit the changed.



```
def update(trail_id):
    trail_data = request.get_json()
    existing_trail = Trail.query.filter(Trail.trail_id == trail_id).one_or_none()
    if existing_trail is None:
        abort(404, f"Trail with trail_id {trail_id} not found")
    if session.get('role') != 'admin' and session.get('user_id') != existing_trail.owner_id:
        return make_response(f"Trail {trail_id} cannot be updated, currently authenicated user {session.get('user_id')} is not the owner of the trail.", 400)
    if 'trail_name' in trail_data:
        existing_trail.trail_name = trail_data['trail_name']
    if 'difficulty' in trail_data:
        existing_trail.difficulty = trail_data['difficulty']
    if 'length' in trail_data:
        existing_trail.length = trail_data['length']
    if 'traffic' in trail_data:
        existing_trail.traffic = trail_data['traffic']
    if 'duration' in trail_data:
        existing_trail.duration = trail_data['duration']
    if 'elevation_gain' in trail_data:
        existing_trail.elevation_gain = trail_data['elevation_gain']
    if 'route_type' in trail_data:
        existing_trail.route_type = trail_data['route_type']
    if 'summary' in trail_data:
        existing_trail.summary = trail_data['summary']
    if 'description' in trail_data:
        existing_trail.description = trail_data['description']
    if 'location' in trail_data:
        existing_trail.location = trail_data['location']
    db.session.commit()
    return make_response(f"trail with ID {trail_id} has been updated successfully.", 200)
```

Delete

The delete function also checks the who the owner of the trail is if the signed in user isn’t an admin, before allowing them to delete the trail. As relationships are properly defined with models.py, this also means that any attractions/location points linked to the trail using the link entities are also deleted. This works similarly in other functions, with the main exception within user.py delete function, reassigning all the owner id of trails the user might have owned to the id 1, which is the id of the admin to maintain data integrity.

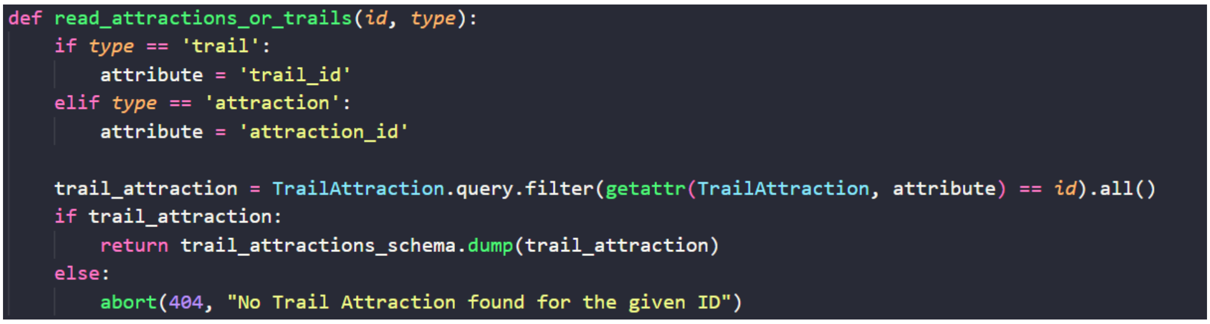


```
def delete(trail_id):
    existing_trail = Trail.query.filter(Trail.trail_id == trail_id).one_or_none()
    if session.get('user_id') != existing_trail.owner_id and session.get('role') != 'admin':
        return make_response(f"Trail {trail_id} cannot be deleted, currently authenicated user {session.get('user_id')} is not the owner of the trail.", 400)
    if existing_trail:
        db.session.delete(existing_trail)
        db.session.commit()
        return make_response(f"trail with ID {trail_id} has been deleted", 200)
    else:
        abort(404, f"trail with ID {trail_id} not found")
```

How link entities are different

The primary difference is the lack a "read one" function due to their compound key structure and absence of standalone data worth retrieving. Instead, a "get" function retrieves all IDs tied to a given input ID. For attractions, inputting an attraction ID returns all trails linked to it, and vice versa. This relies on specifying the ID type, crucial as IDs are auto-assigned integers via SQL's IDENTITY function. Additionally, there’s no update function for trail attractions, as updating an entry is nearly identical to creating one with added steps.

Below is code used in trail_attraction.py, a similar implementation used within trail_location_point.py.



```
def read_attractions_or_trails(id, type):
    if type == 'trail':
        attribute = 'trail_id'
    elif type == 'attraction':
        attribute = 'attraction_id'

    trail_attraction = TrailAttraction.query.filter(getattr(TrailAttraction, attribute) == id).all()
    if trail_attraction:
        return trail_attractions_schema.dump(trail_attraction)
    else:
        abort(404, "No Trail Attraction found for the given ID")
```

Permissions Overall:

As an admin, you have access to all data, with permissions to create, read, update, and delete. The main difference for users is amongst two things: hiding some data such as trail id and the username and passwords of other users when viewing all users, by extension there are restrictions in place to ensure that users can only perform CRUD operations on their own trails. Additionally, the limitations extend to the Attraction endpoints, which are restricted to Admins for creating and deleting.

Evaluation

Overall, the project successfully achieved its objective of implementing the backend code for the trail microservice, fulfilling the requirements outlined in the background. However, several improvements could improve its functionality, security, and scalability:

Database Improvements

1. **Location Table**:
    - Adding a table to store location information (e.g., city, state, and country) would prevent duplication and ensures consistency plus scalability.
2. **Enhanced Data Retrieval**:
    - When requesting trail data, including location point data and attraction details would provide a more comprehensive response, making the service more user-friendly and useful.

Design Decisions

3. **User Role Permissions for Location Points**:
    - Allowing users to create their own location points could enable more trails to be created entirely by users. However, this introduces risks, such as misuse and unnecessary data storage, potentially leading to increased long-term costs.
    - In the current implementation, users can create location points, but a decision should be made to balance user empowerment and system efficiency.

Testing and Debugging

4. **Default Session Role**:
    - For debugging and testing, the session defaults to the admin role, which simplifies testing but is unrealistic for a real-world scenario.
    - Ideally, the session should default to the signed-in user or a guest role, but this feature is out of scope for this project and would typically be managed by other components of the microservice.

Permission and Privilege Management

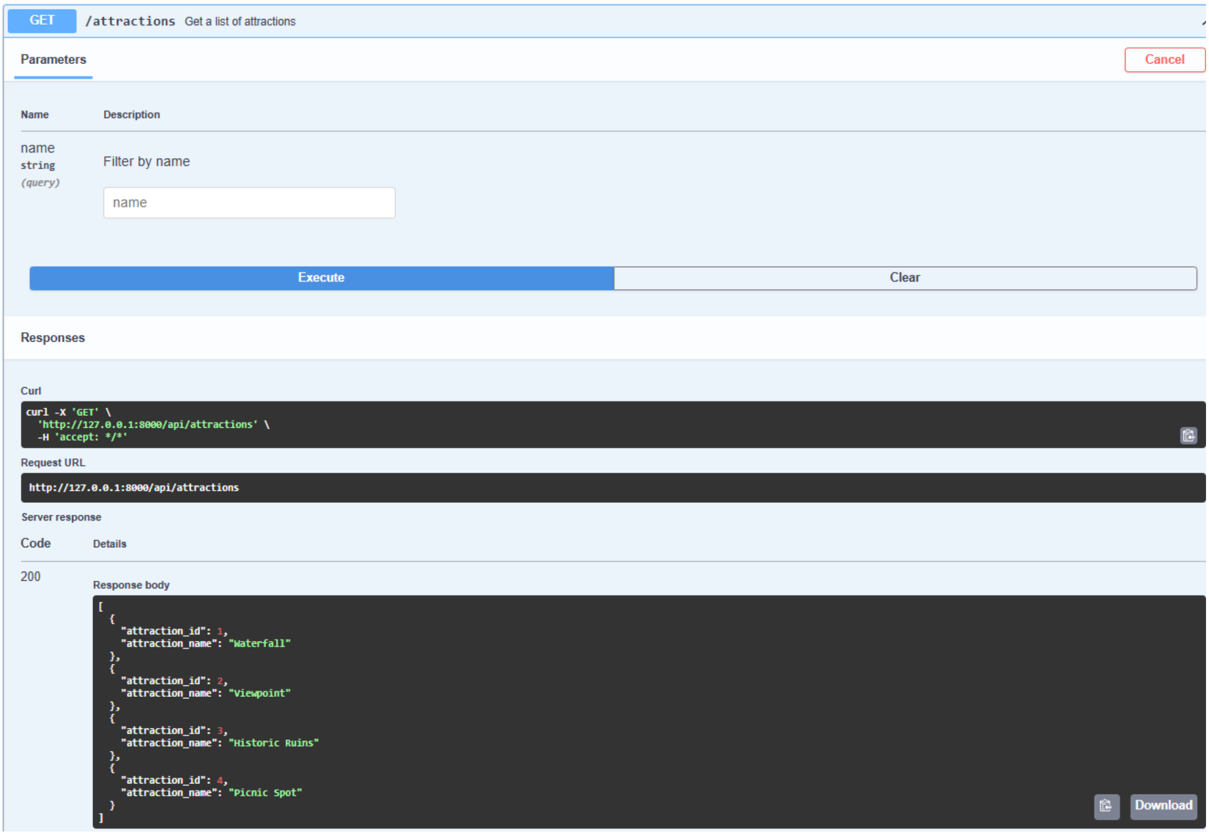
5. **Updating Attributes**:
    - Currently, certain attributes, like the owner_id, cannot be updated, even by an admin. While this limitation rarely impacts functionality, enabling admins to modify all data would align with their intended role.
6. **Stricter Privilege Systems**:
    - Implementing a least-privilege system would improve security by ensuring users have access only to the data and functions necessary for their role.

Security Enhancements

7. **Data Security**:
    - Hashing sensitive data, such as passwords, and implementing encryption during data transmission would significantly enhance security.

Screenshots of testing

Attraction:



Name	Description
name string *(query)*	Filter by name name

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/api/attractions' \
  -H 'accept: */*'
```

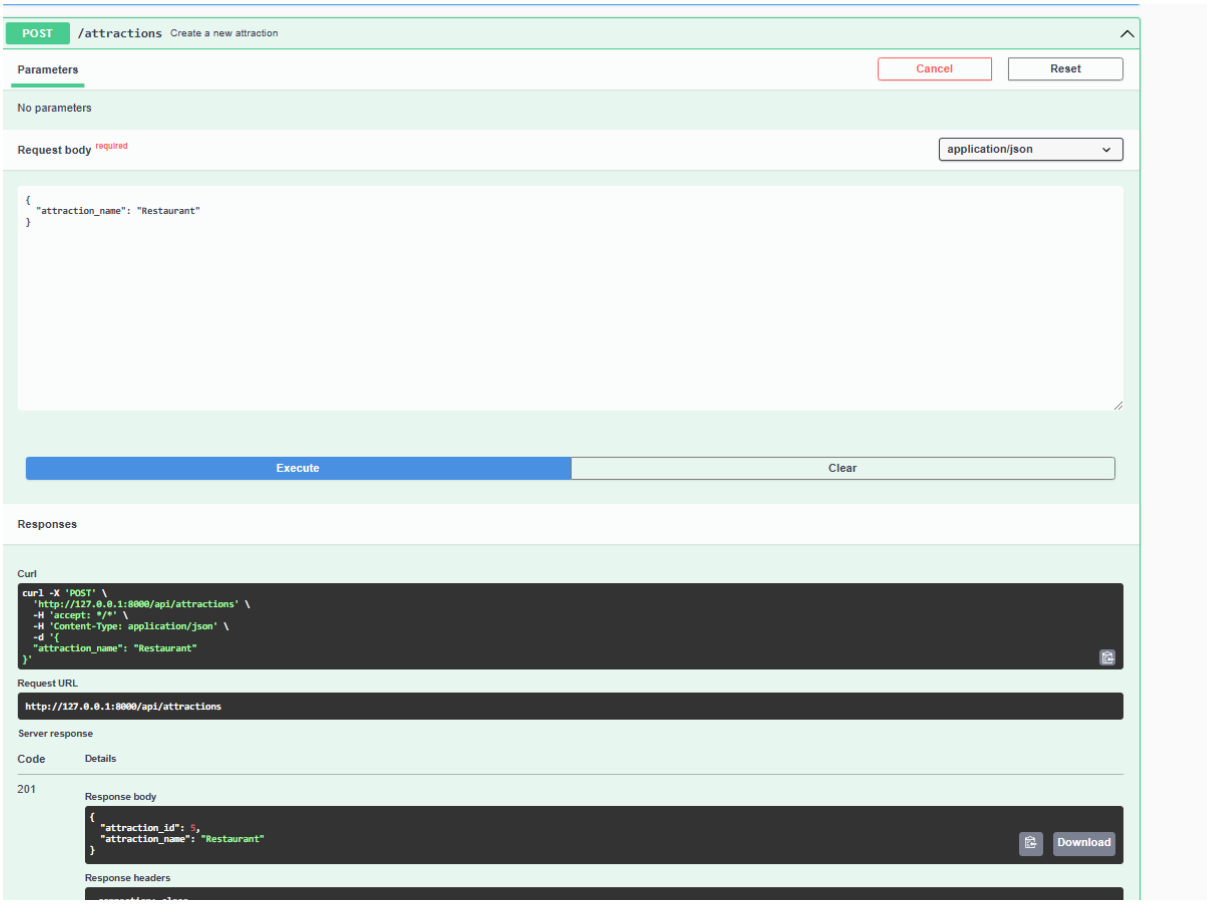
Request URL

```
http://127.0.0.1:8000/api/attractions
```

Server response

Code	Details
200	Response body

```
[
  {
    "attraction_id": 1,
    "attraction_name": "Waterfall"
  },
  {
    "attraction_id": 2,
    "attraction_name": "Viewpoint"
  },
  {
    "attraction_id": 3,
    "attraction_name": "Historic Ruins"
  },
  {
    "attraction_id": 4,
    "attraction_name": "Picnic Spot"
  }
]
```



No parameters

Request body required

application/json

```
{
  "attraction_name": "Restaurant"
}
```

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/api/attractions' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "attraction_name": "Restaurant"
}'
```

Request URL

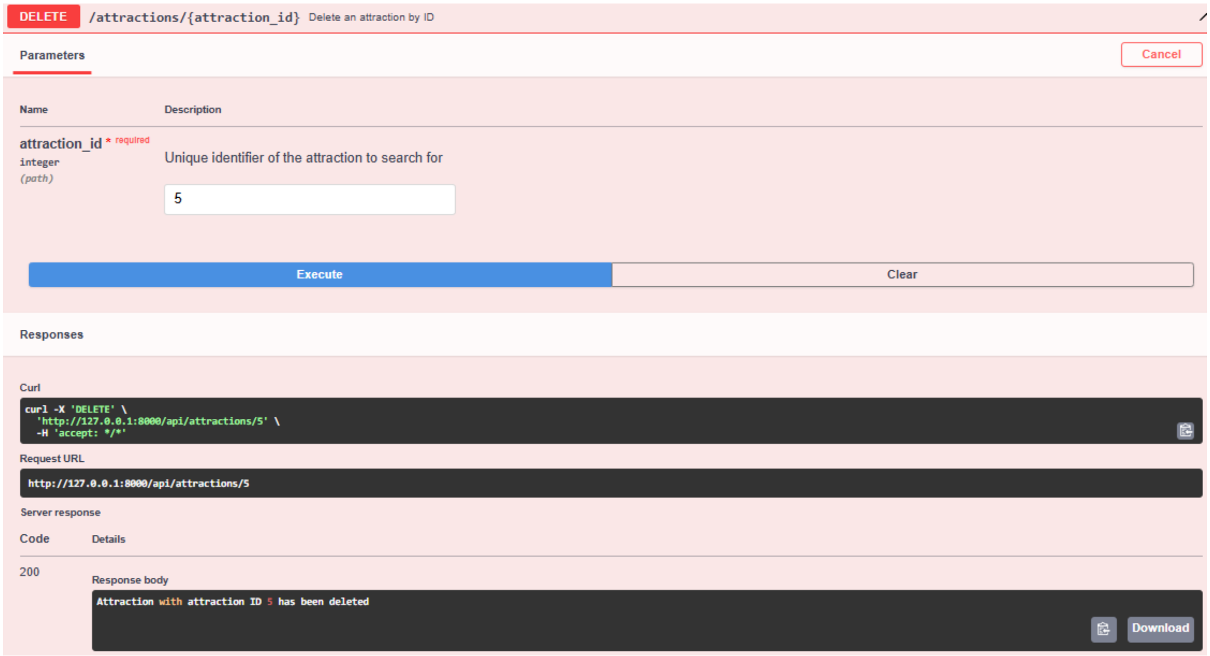
```
http://127.0.0.1:8000/api/attractions
```

Server response

Code	Details
201	Response body

```
{
  "attraction_id": 5,
  "attraction_name": "Restaurant"
}
```

Response headers



/attractions/{attraction_id} Delete an attraction by ID

Name: attraction_id * required
integer
(path)

Description: Unique identifier of the attraction to search for

5

Responses

Curl

```
curl -X 'DELETE' \
  'http://127.0.0.1:8000/api/attractions/5' \
  -H 'accept: */*'
```

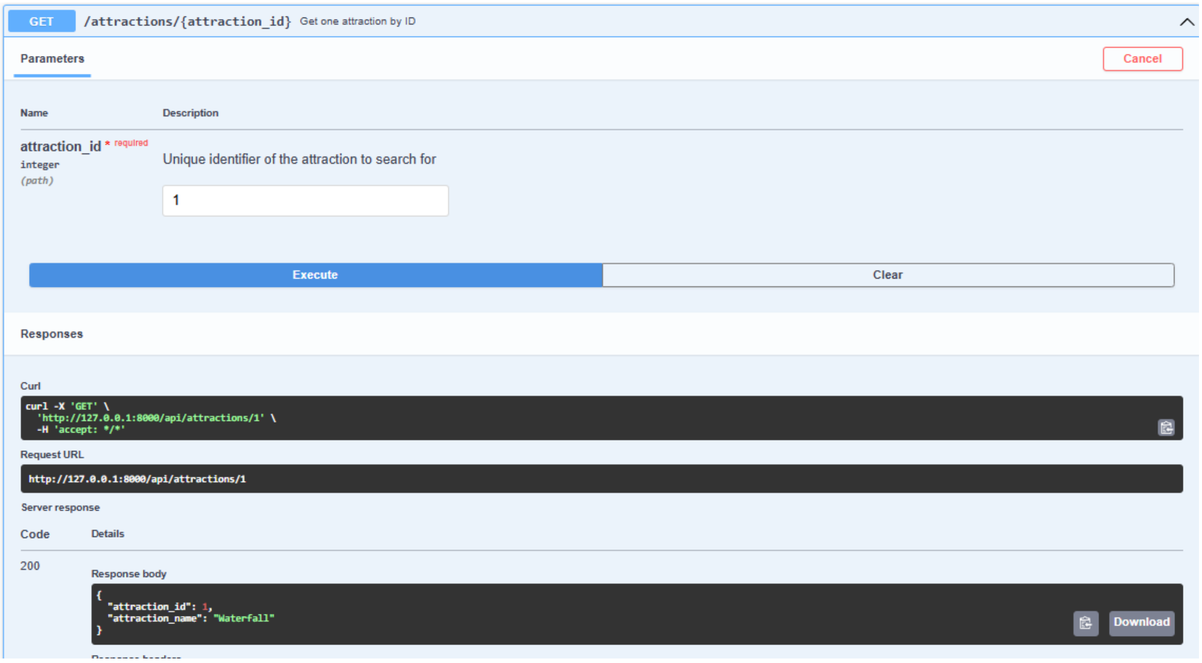
Request URL

```
http://127.0.0.1:8000/api/attractions/5
```

Server response

Code	Details
200	Response body

```
Attraction with attraction ID 5 has been deleted
```



/attractions/{attraction_id} Get one attraction by ID

Name: attraction_id * required
integer
(path)

Description: Unique identifier of the attraction to search for

1

Responses

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/api/attractions/1' \
  -H 'accept: */*'
```

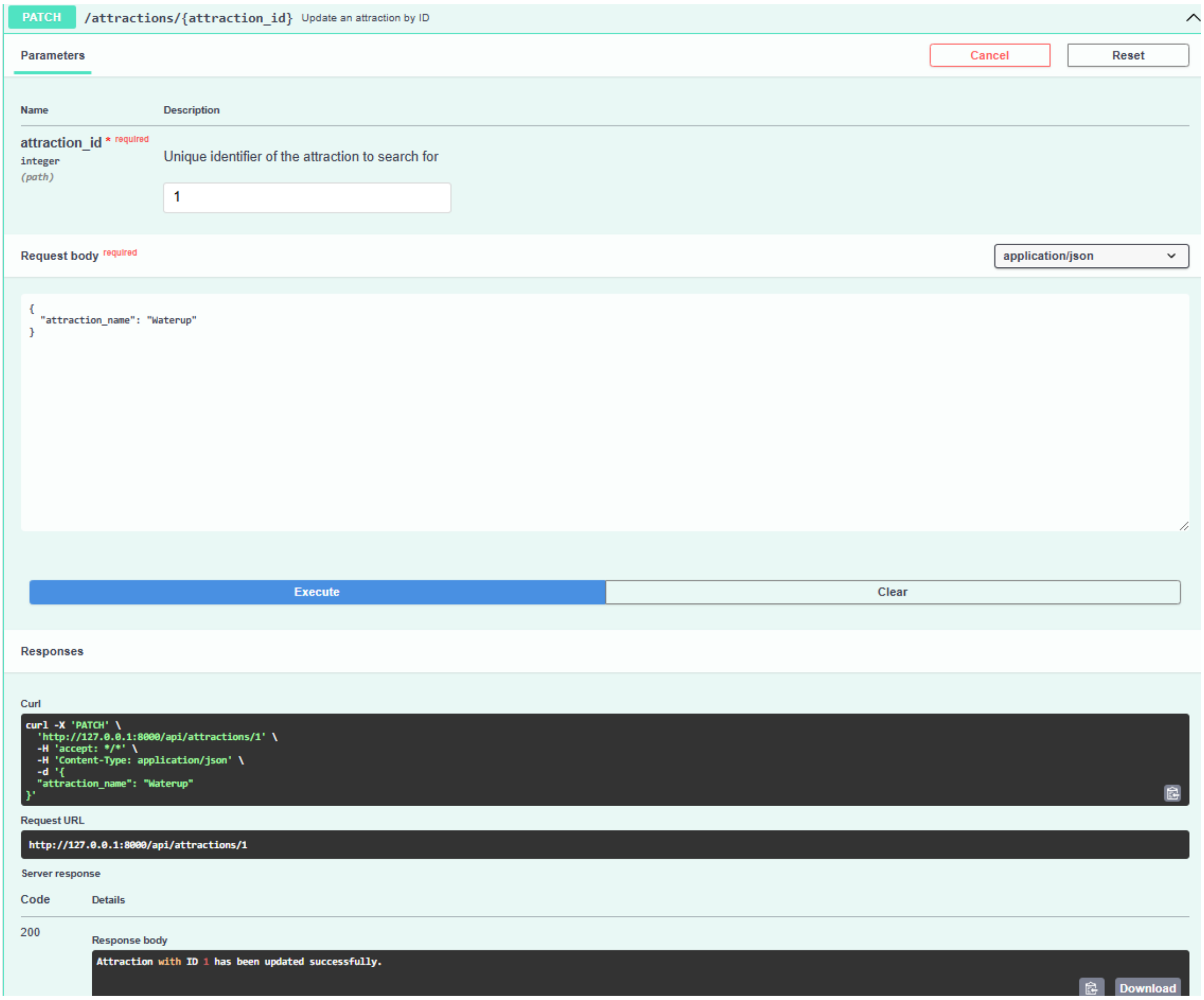
Request URL

```
http://127.0.0.1:8000/api/attractions/1
```

Server response

Code	Details
200	Response body

```
{
  "attraction_id": 1,
  "attraction_name": "Waterfall"
}
```



attraction_id * required
integer
(path)

Unique identifier of the attraction to search for

1

```
{
  "attraction_name": "Waterup"
}
```

Curl

```
curl -X 'PATCH' \
  'http://127.0.0.1:8000/api/attractions/1' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "attraction_name": "Waterup"
}'
```

Request URL

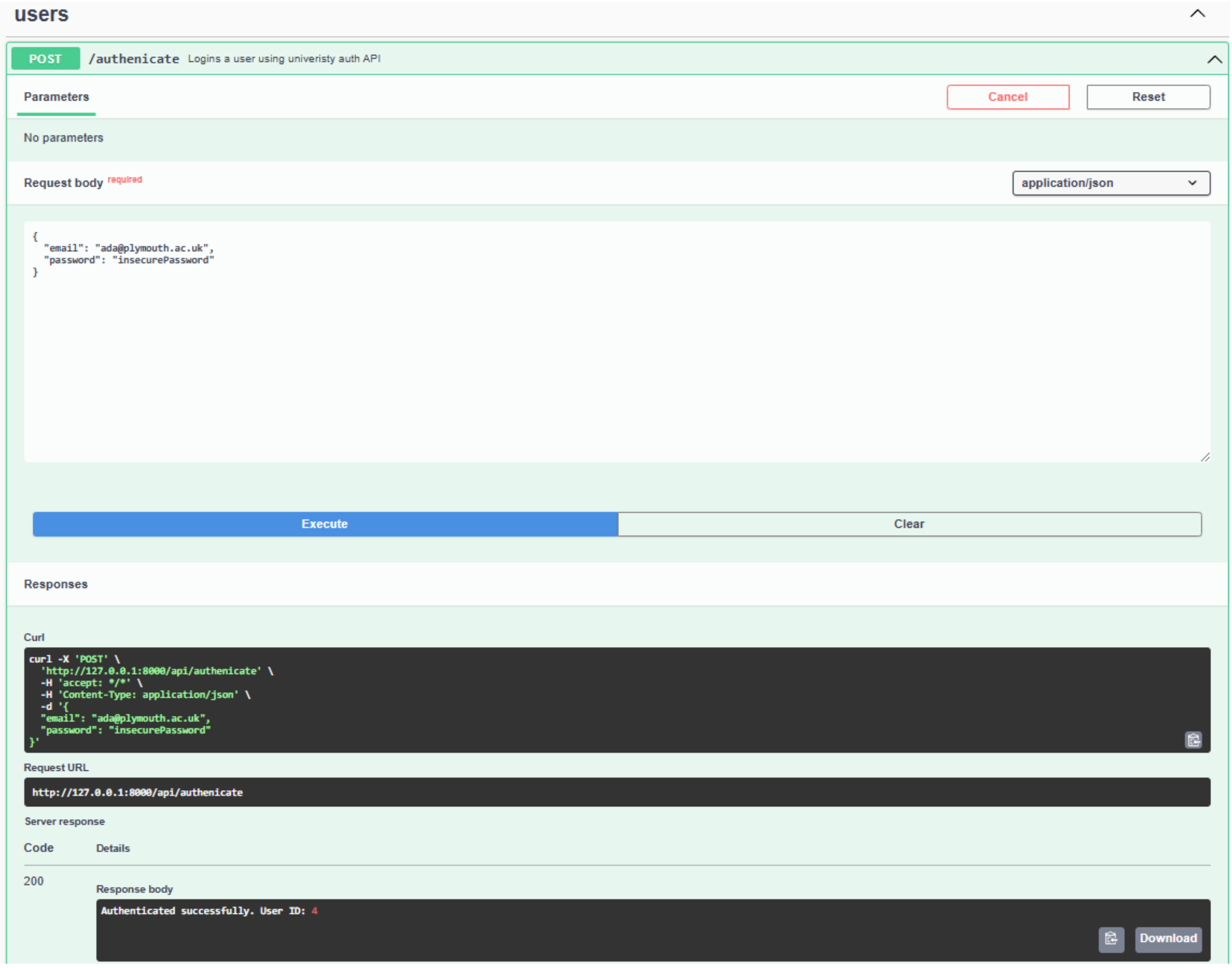
```
http://127.0.0.1:8000/api/attractions/1
```

Server response

Code	Details
200	Response body

```
Attraction with ID 1 has been updated successfully.
```

Users



/authenicate Logins a user using univeristy auth API

No parameters

Request body required

```
{
  "email": "ada@plymouth.ac.uk",
  "password": "insecurePassword"
}
```

Responses

Curl

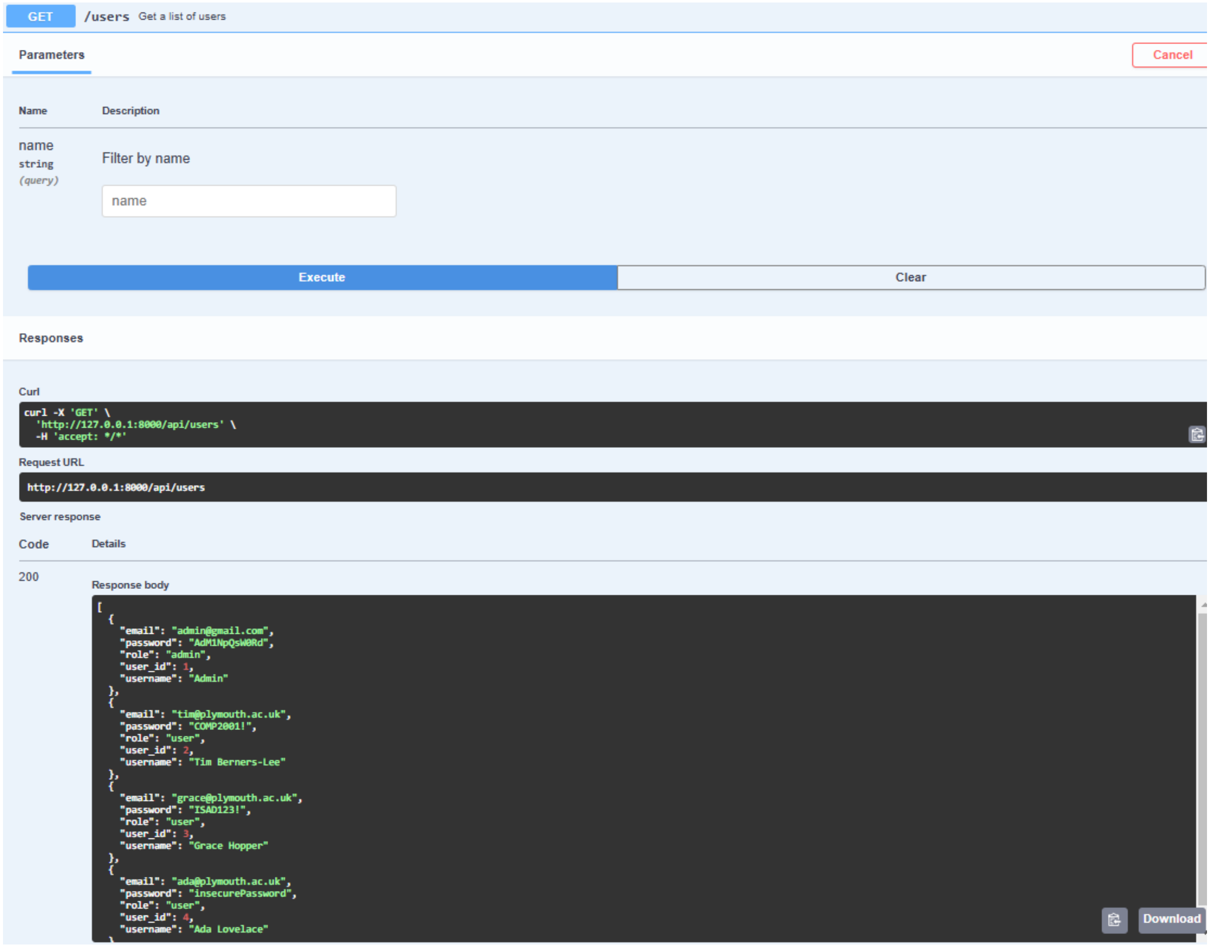
```
curl -X 'POST' \
  'http://127.0.0.1:8000/api/authenicate' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "email": "ada@plymouth.ac.uk",
  "password": "insecurePassword"
}'
```

Request URL

```
http://127.0.0.1:8000/api/authenicate
```

Server response

Code	Details
200	Response body `Authenticated successfully. User ID: 4`



```
curl -X 'GET' \
  'http://127.0.0.1:8000/api/users' \
  -H 'accept: */*'
```

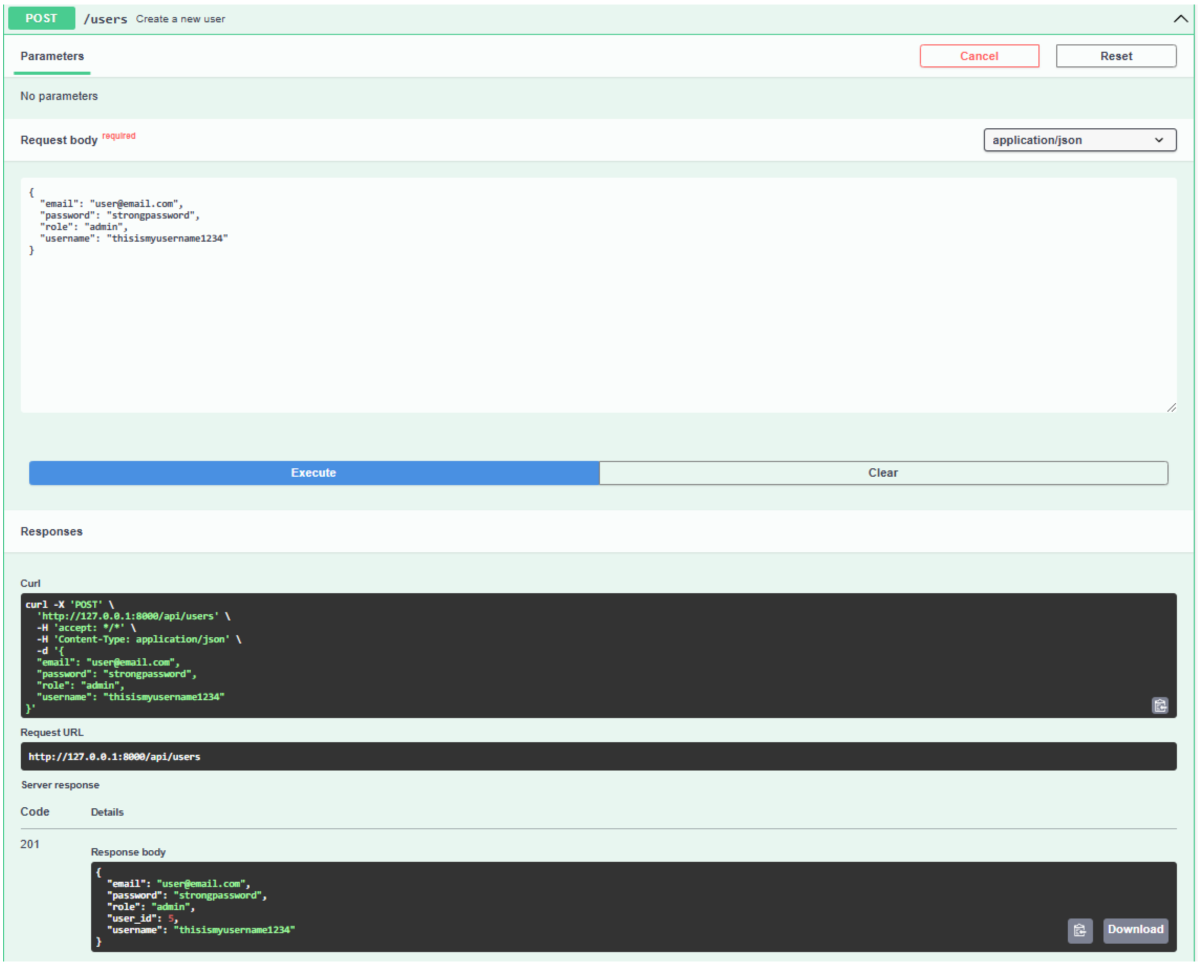
Request URL

```
http://127.0.0.1:8000/api/users
```

200

Response body

```
[
  {
    "email": "admin@gmail.com",
    "password": "AdM1NpQsW0Rd",
    "role": "admin",
    "user_id": 1,
    "username": "Admin"
  },
  {
    "email": "tim@plymouth.ac.uk",
    "password": "COMP2001!",
    "role": "user",
    "user_id": 2,
    "username": "Tim Berners-Lee"
  },
  {
    "email": "grace@plymouth.ac.uk",
    "password": "ISAD123!",
    "role": "user",
    "user_id": 3,
    "username": "Grace Hopper"
  },
  {
    "email": "ada@plymouth.ac.uk",
    "password": "insecurePassword",
    "role": "user",
    "user_id": 4,
    "username": "Ada Lovelace"
```



```
{
  "email": "user@email.com",
  "password": "strongpassword",
  "role": "admin",
  "username": "thisismyusername1234"
}
```

```
curl -X 'POST' \
  'http://127.0.0.1:8000/api/users' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "email": "user@email.com",
  "password": "strongpassword",
  "role": "admin",
  "username": "thisismyusername1234"
}'
```

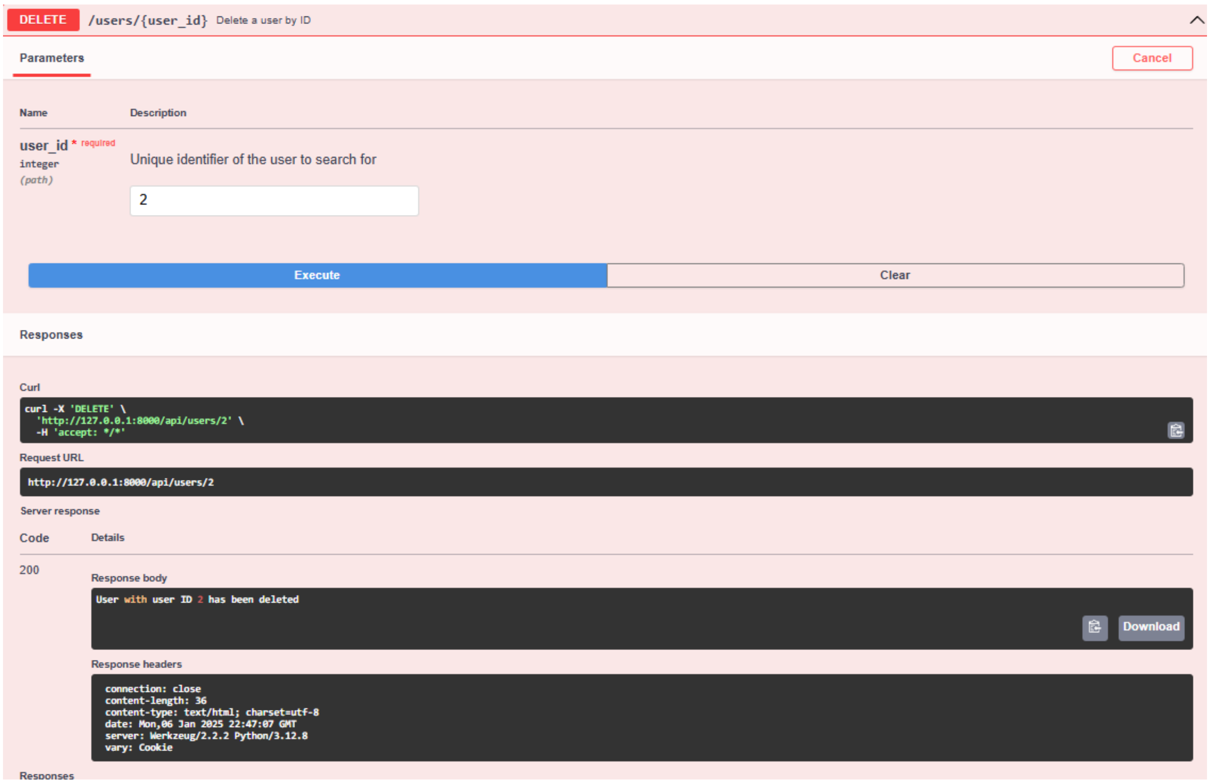
Request URL

```
http://127.0.0.1:8000/api/users
```

201

Response body

```
{
  "email": "user@email.com",
  "password": "strongpassword",
  "role": "admin",
  "user_id": 5,
  "username": "thisismyusername1234"
}
```



/users/{user_id} Delete a user by ID

Name	Description
user_id * required integer *(path)*	Unique identifier of the user to search for 2

Responses

Curl

```
curl -X 'DELETE' \
  'http://127.0.0.1:8000/api/users/2' \
  -H 'accept: */*'
```

Request URL

```
http://127.0.0.1:8000/api/users/2
```

Server response

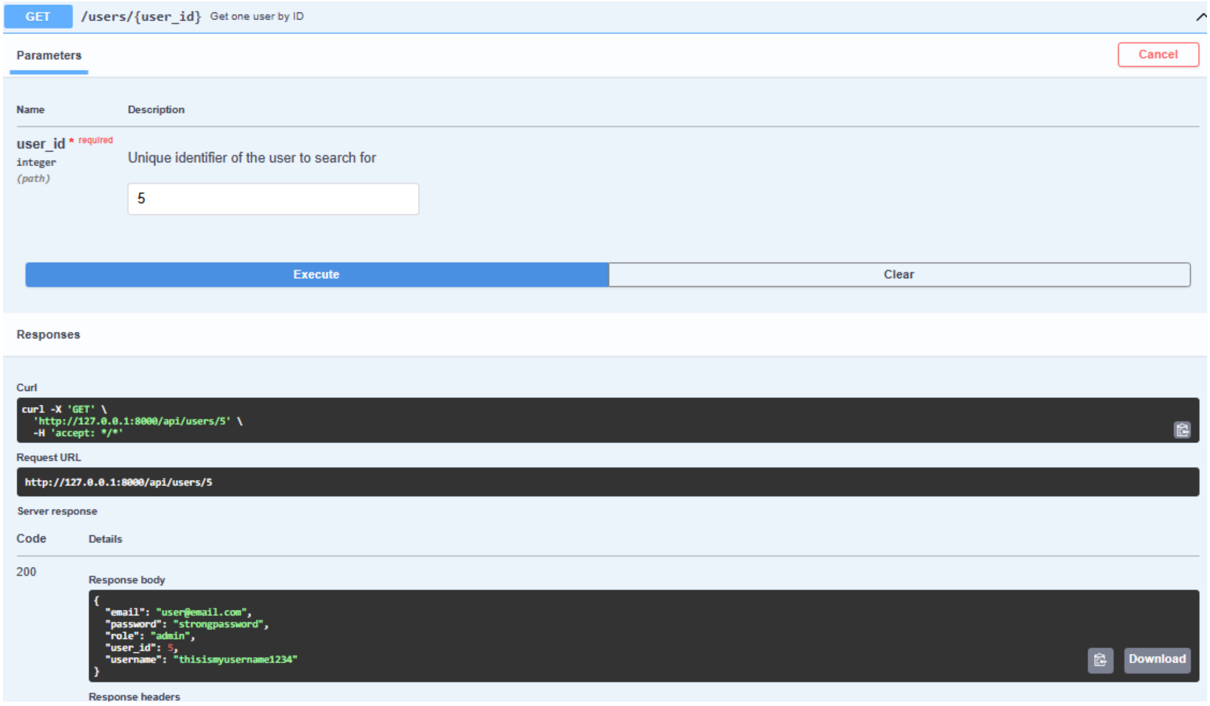
Code	Details
200	Response body

```
User with user ID 2 has been deleted
```

Response headers

```
 connection: close
 content-length: 36
 content-type: text/html; charset=utf-8
 date: Mon,06 Jan 2025 22:47:07 GMT
 server: Werkzeug/2.2.2 Python/3.12.8
 vary: Cookie
```

Responses



/users/{user_id} Get one user by ID

Name	Description
user_id * required integer *(path)*	Unique identifier of the user to search for 5

Responses

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/api/users/5' \
  -H 'accept: */*'
```

Request URL

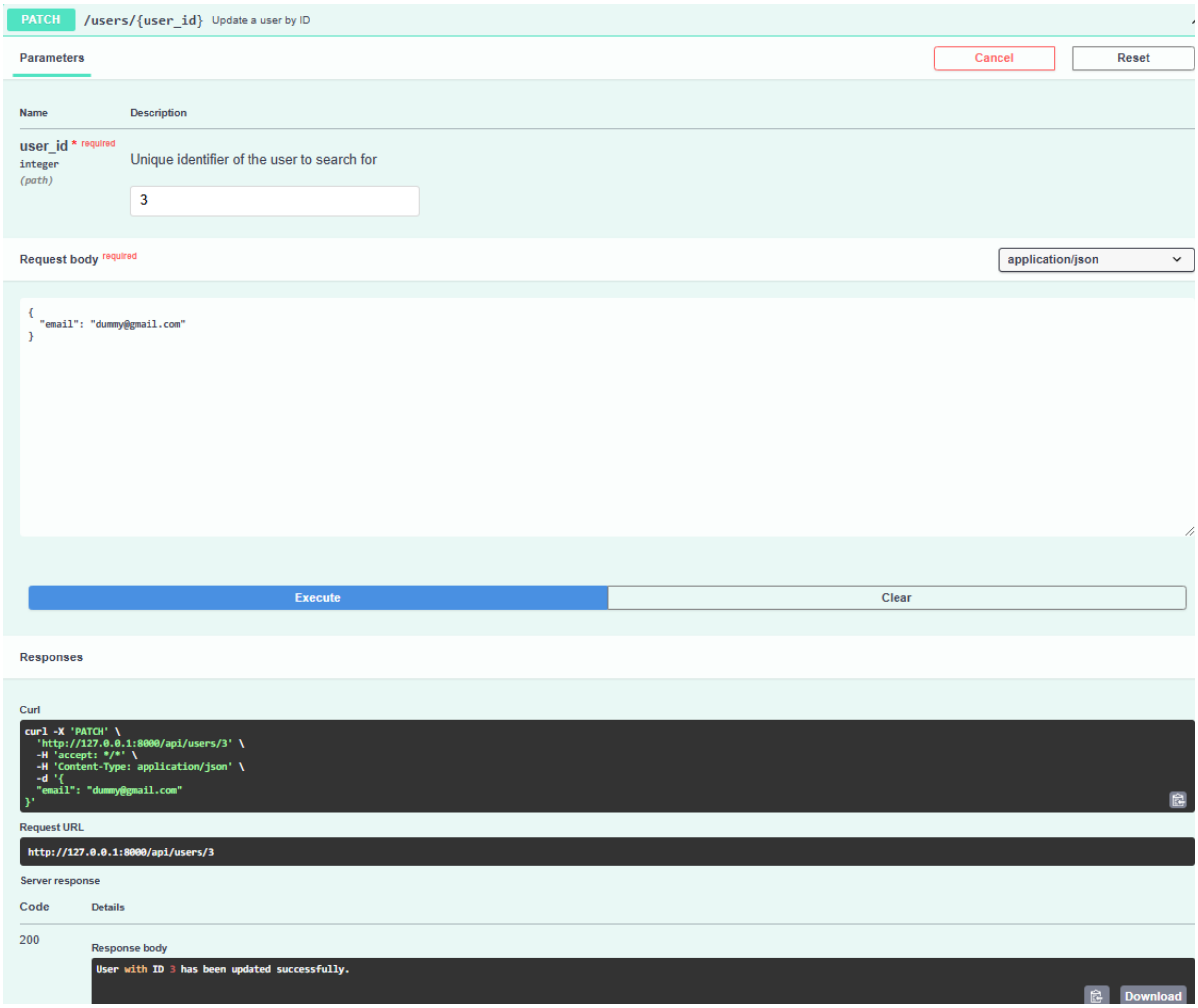
```
http://127.0.0.1:8000/api/users/5
```

Server response

Code	Details
200	Response body

```
{
  "email": "user@email.com",
  "password": "strongpassword",
  "role": "admin",
  "user_id": 5,
  "username": "thisismyusername1234"
}
```

Response headers


PATCH
/users/{user_id} Update a user by ID
Parameters
Cancel
Reset
Name
Description
user_id * required
integer
(path)
Unique identifier of the user to search for

3

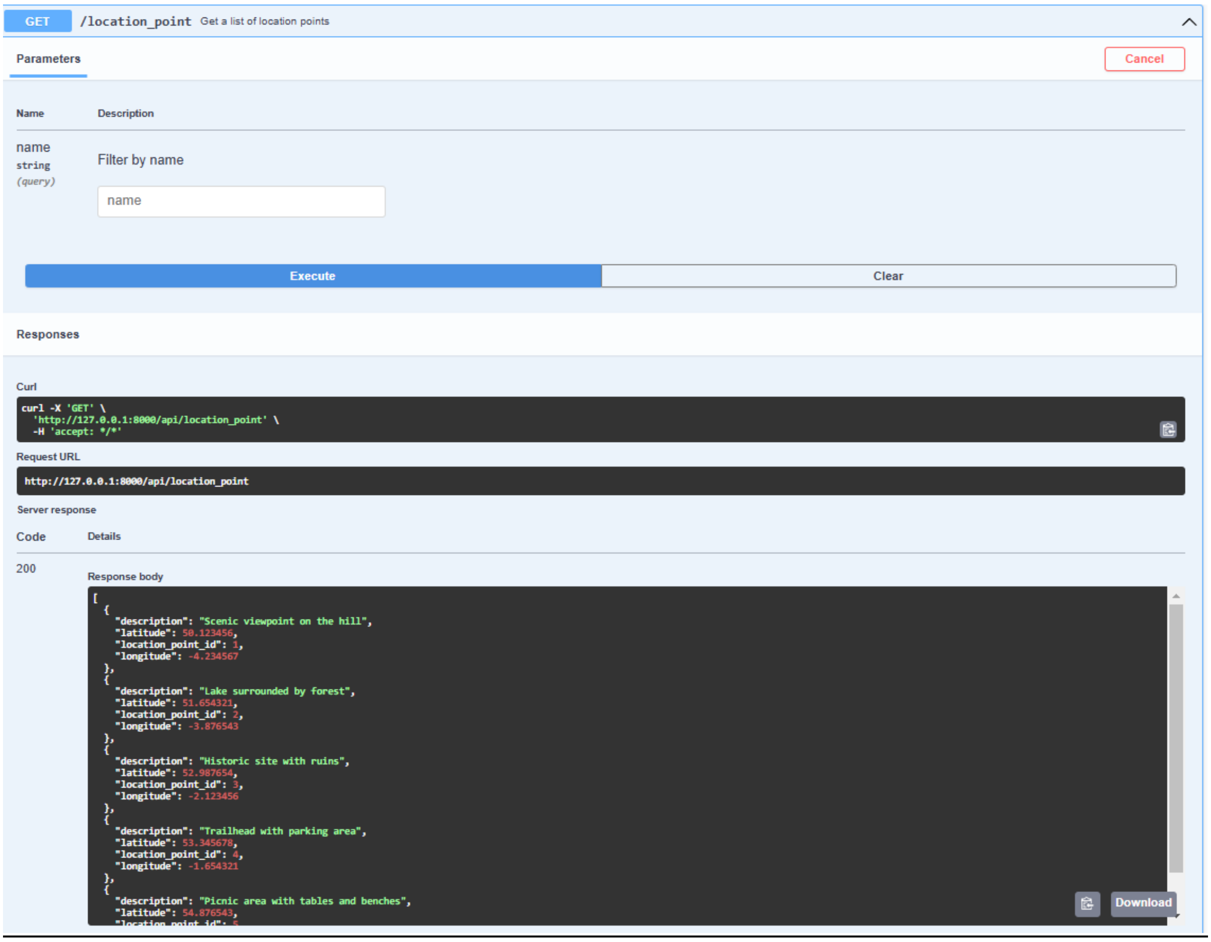
```
{
  "email": "dummy@gmail.com"
}
```

```
curl -X 'PATCH' \
  'http://127.0.0.1:8000/api/users/3' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "email": "dummy@gmail.com"
}'
```

```
http://127.0.0.1:8000/api/users/3
```

```
User with ID 3 has been updated successfully.
```

LocationPoints



Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/api/location_point' \
  -H 'accept: */*'
```

Request URL

```
http://127.0.0.1:8000/api/location_point
```

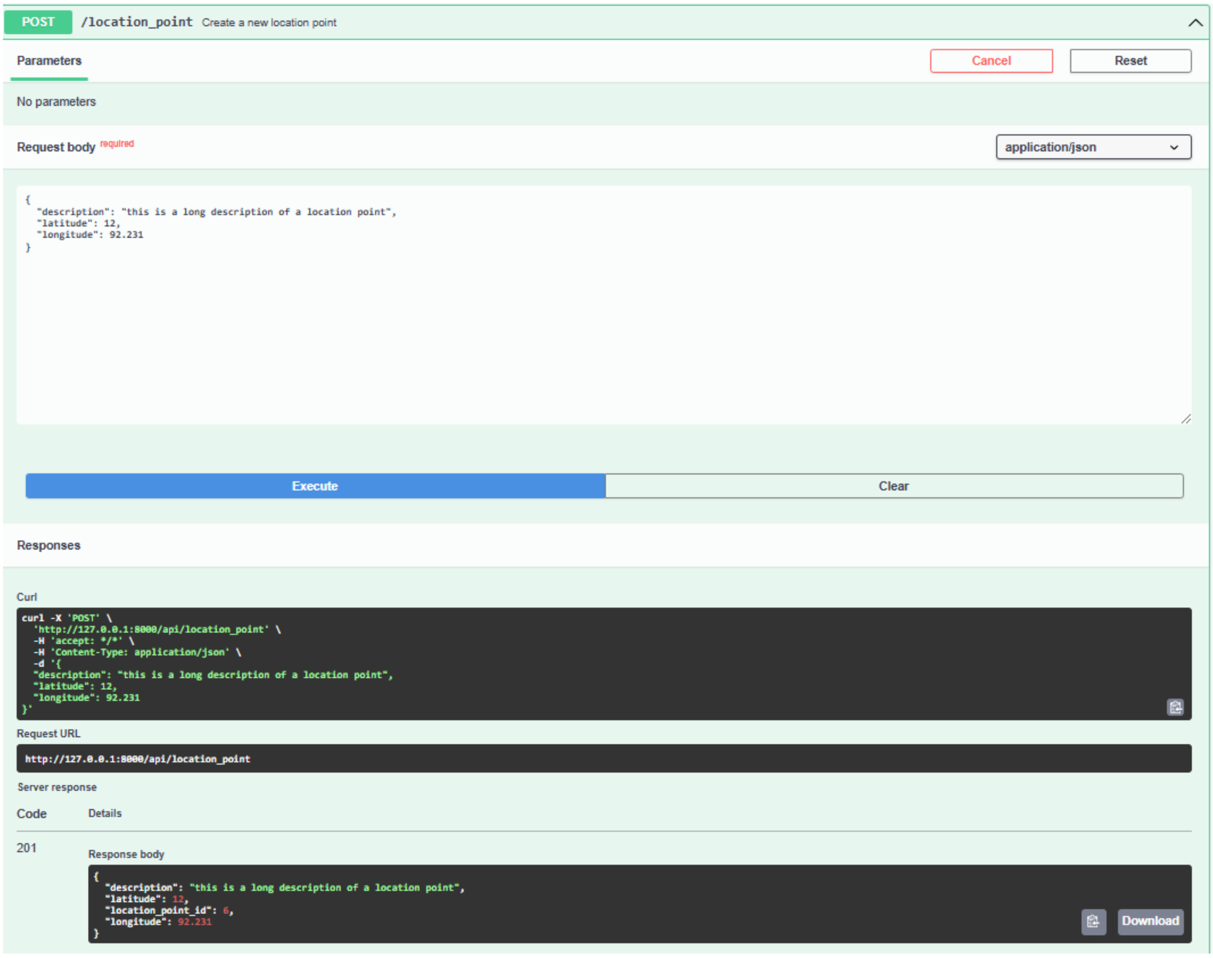
Server response

Code | Details

200

Response body

```
[
  {
    "description": "Scenic viewpoint on the hill",
    "latitude": 50.123456,
    "location_point_id": 1,
    "longitude": -4.234567
  },
  {
    "description": "Lake surrounded by forest",
    "latitude": 51.654321,
    "location_point_id": 2,
    "longitude": -3.876543
  },
  {
    "description": "Historic site with ruins",
    "latitude": 52.987654,
    "location_point_id": 3,
    "longitude": -2.123456
  },
  {
    "description": "Trailhead with parking area",
    "latitude": 53.345678,
    "location_point_id": 4,
    "longitude": -1.654321
  },
  {
    "description": "Picnic area with tables and benches",
    "latitude": 54.876543,
    "location_point_id": 5
```



```
{
  "description": "this is a long description of a location point",
  "latitude": 12,
  "longitude": 92.231
}
```

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/api/location_point' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "description": "this is a long description of a location point",
  "latitude": 12,
  "longitude": 92.231
}'
```

Request URL

```
http://127.0.0.1:8000/api/location_point
```

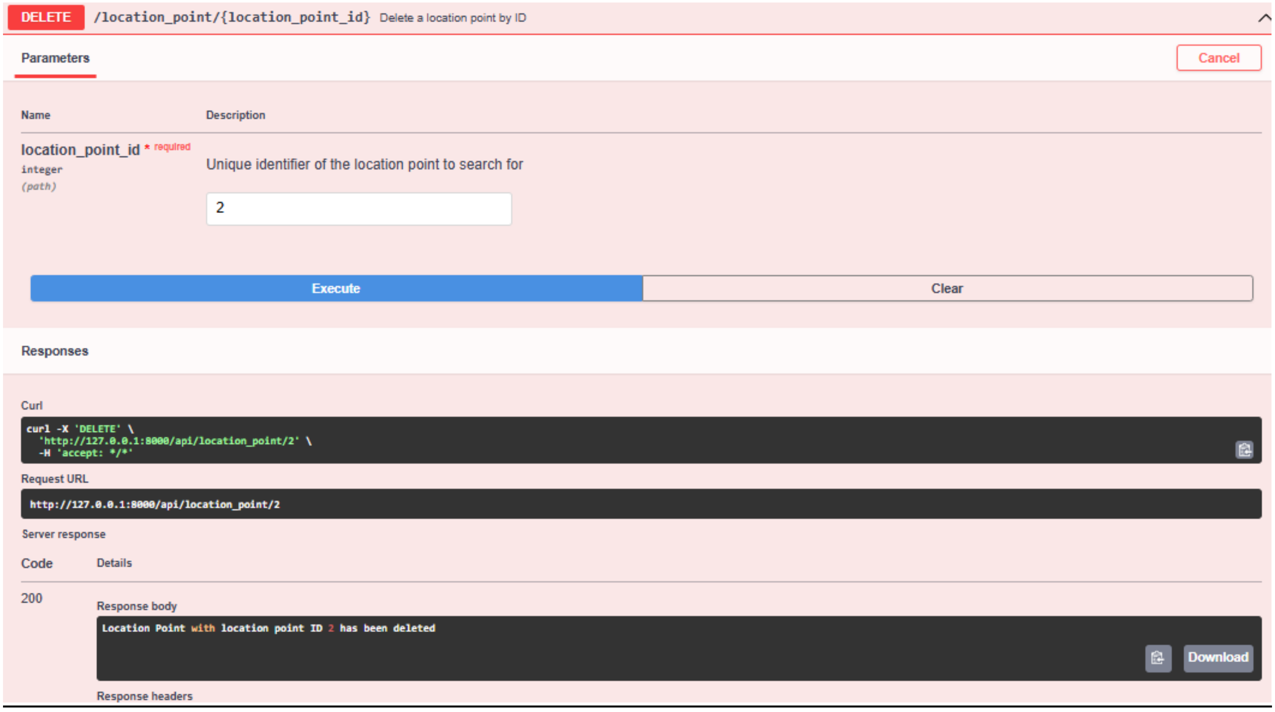
Server response

Code | Details

201

Response body

```
{
  "description": "this is a long description of a location point",
  "latitude": 12,
  "location_point_id": 6,
  "longitude": 92.231
}
```



`/location_point/{location_point_id}` Delete a location point by ID

Name	Description
location_point_id * required integer *(path)*	Unique identifier of the location point to search for 2

Responses

Curl

```
curl -X 'DELETE' \
  'http://127.0.0.1:8000/api/location_point/2' \
  -H 'accept: */*'
```

Request URL

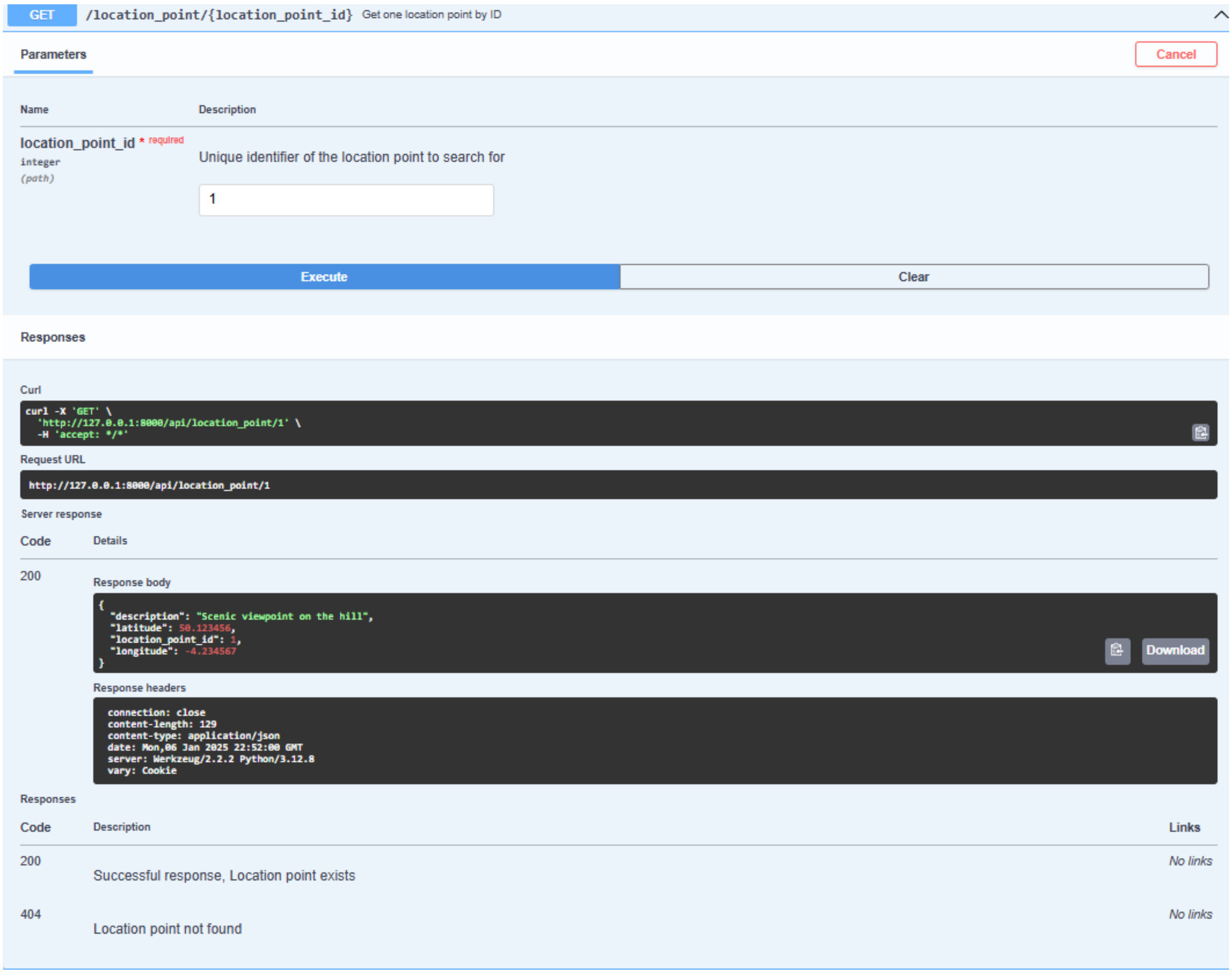
```
http://127.0.0.1:8000/api/location_point/2
```

Server response

Code	Details
200	Response body

```
Location Point with location point ID 2 has been deleted
```

Response headers



`/location_point/{location_point_id}` Get one location point by ID

Name	Description
location_point_id * required integer *(path)*	Unique identifier of the location point to search for 1

Responses

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/api/location_point/1' \
  -H 'accept: */*'
```

Request URL

```
http://127.0.0.1:8000/api/location_point/1
```

Server response

Code	Details
200	Response body

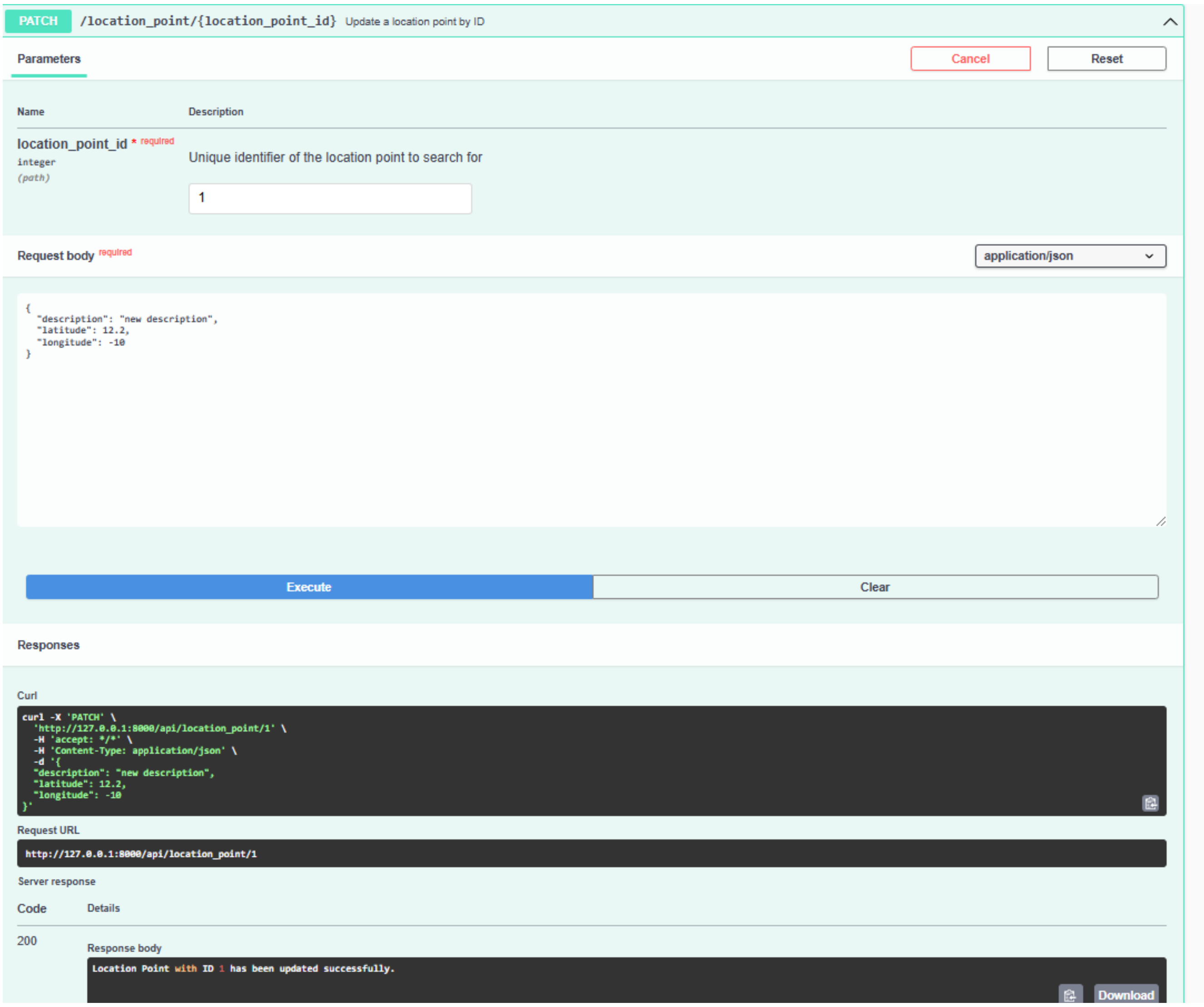
```
{
  "description": "Scenic viewpoint on the hill",
  "latitude": 50.123456,
  "location_point_id": 1,
  "longitude": -4.234567
}
```

Response headers

```
 connection: close
 content-length: 129
 content-type: application/json
 date: Mon,06 Jan 2025 22:52:00 GMT
 server: Werkzeug/2.2.2 Python/3.12.8
 vary: Cookie
```

Responses

Code	Description	Links
200	Successful response, Location point exists	*No links*
404	Location point not found	*No links*



/location_point/{location_point_id} Update a location point by ID

Name	Description
location_point_id * required integer *(path)*	Unique identifier of the location point to search for 1

Request body required

application/json

```
{
  "description": "new description",
  "latitude": 12.2,
  "longitude": -10
}
```

Responses

Curl

```
curl -X 'PATCH' \
  'http://127.0.0.1:8000/api/location_point/1' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "description": "new description",
  "latitude": 12.2,
  "longitude": -10
}'
```

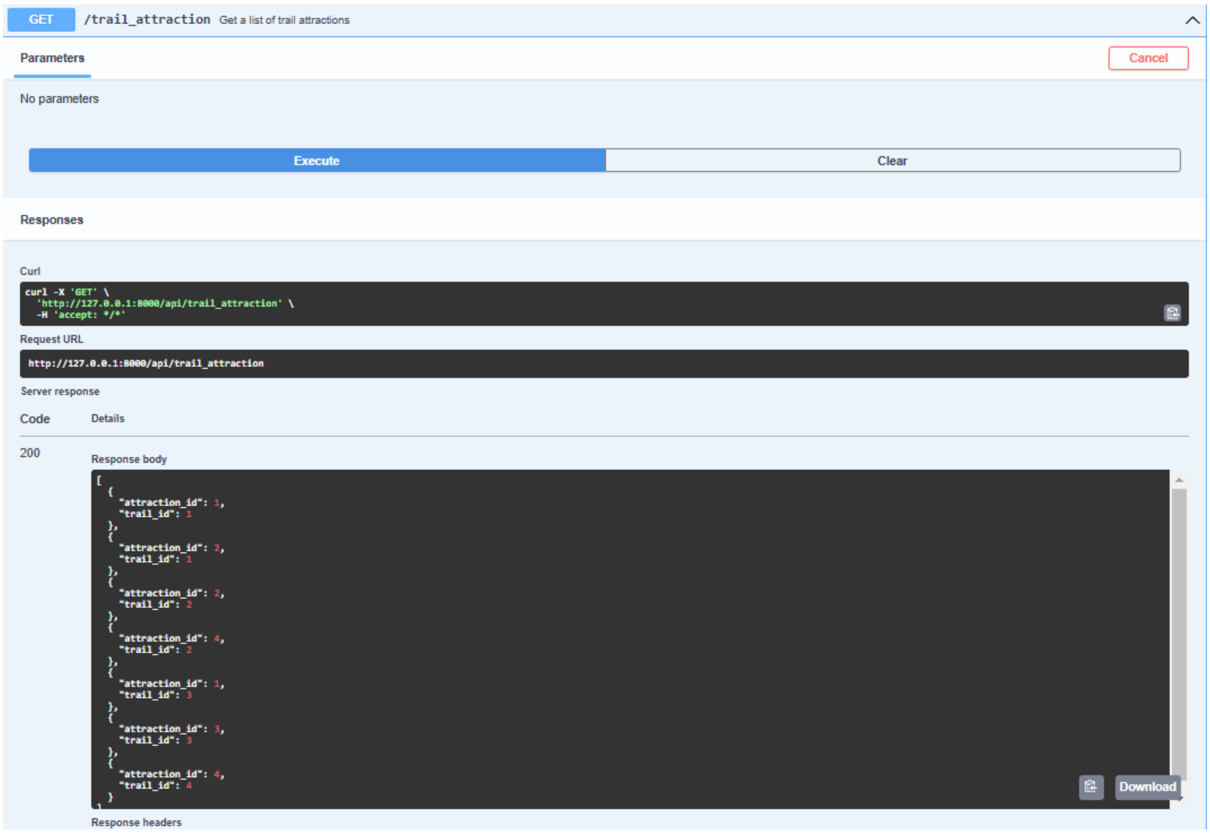
Request URL

```
http://127.0.0.1:8000/api/location_point/1
```

Server response

Code	Details
200	Response body Location Point with ID 1 has been updated successfully.

TrailAttractions



No parameters

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/api/trail_attraction' \
  -H 'accept: */*'
```

Request URL

```
http://127.0.0.1:8000/api/trail_attraction
```

Server response

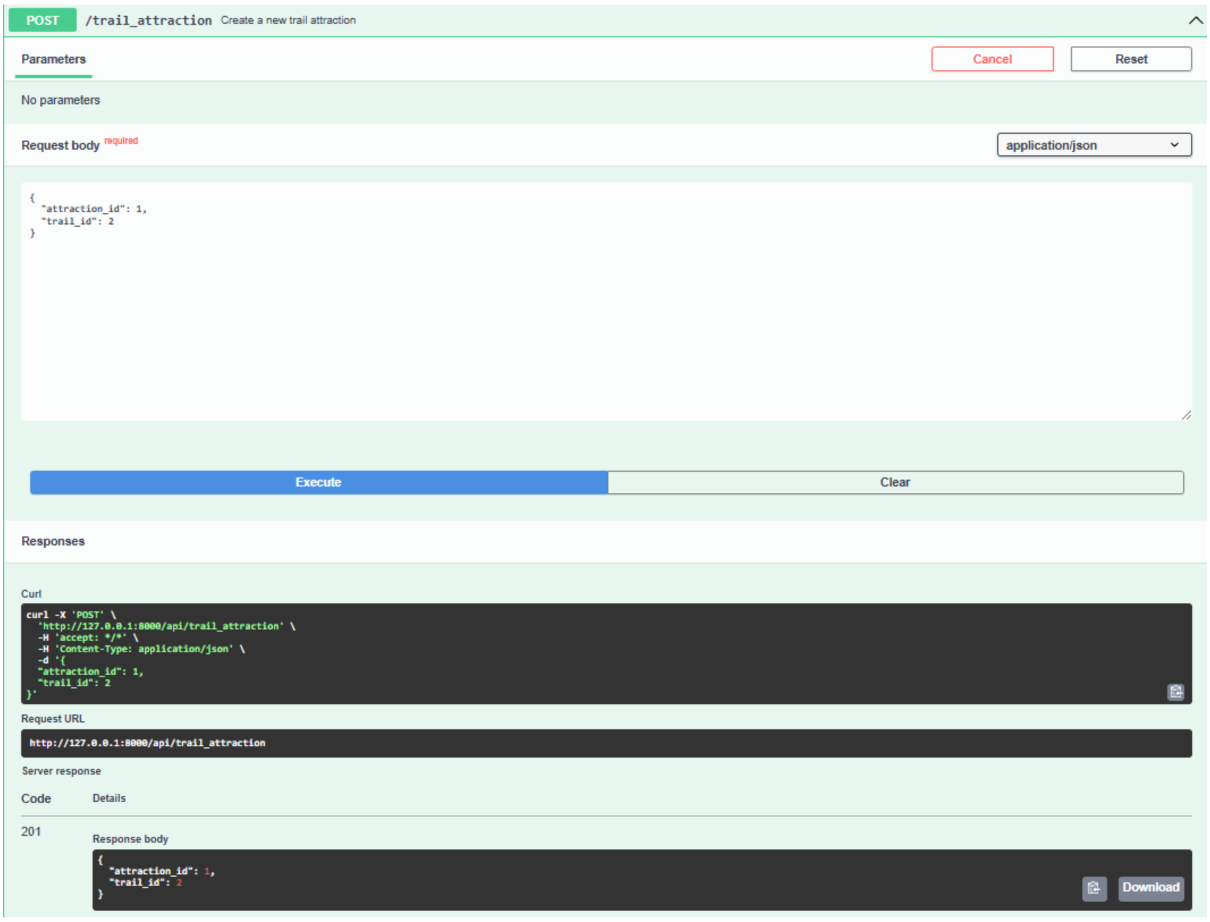
Code Details

200

Response body

```
[
  {
    "attraction_id": 1,
    "trail_id": 1
  },
  {
    "attraction_id": 2,
    "trail_id": 1
  },
  {
    "attraction_id": 2,
    "trail_id": 2
  },
  {
    "attraction_id": 4,
    "trail_id": 2
  },
  {
    "attraction_id": 1,
    "trail_id": 3
  },
  {
    "attraction_id": 3,
    "trail_id": 3
  },
  {
    "attraction_id": 4,
    "trail_id": 4
  }
]
```

Response headers



No parameters

Request body required

application/json

```
{
  "attraction_id": 1,
  "trail_id": 2
}
```

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/api/trail_attraction' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "attraction_id": 1,
  "trail_id": 2
}'
```

Request URL

```
http://127.0.0.1:8000/api/trail_attraction
```

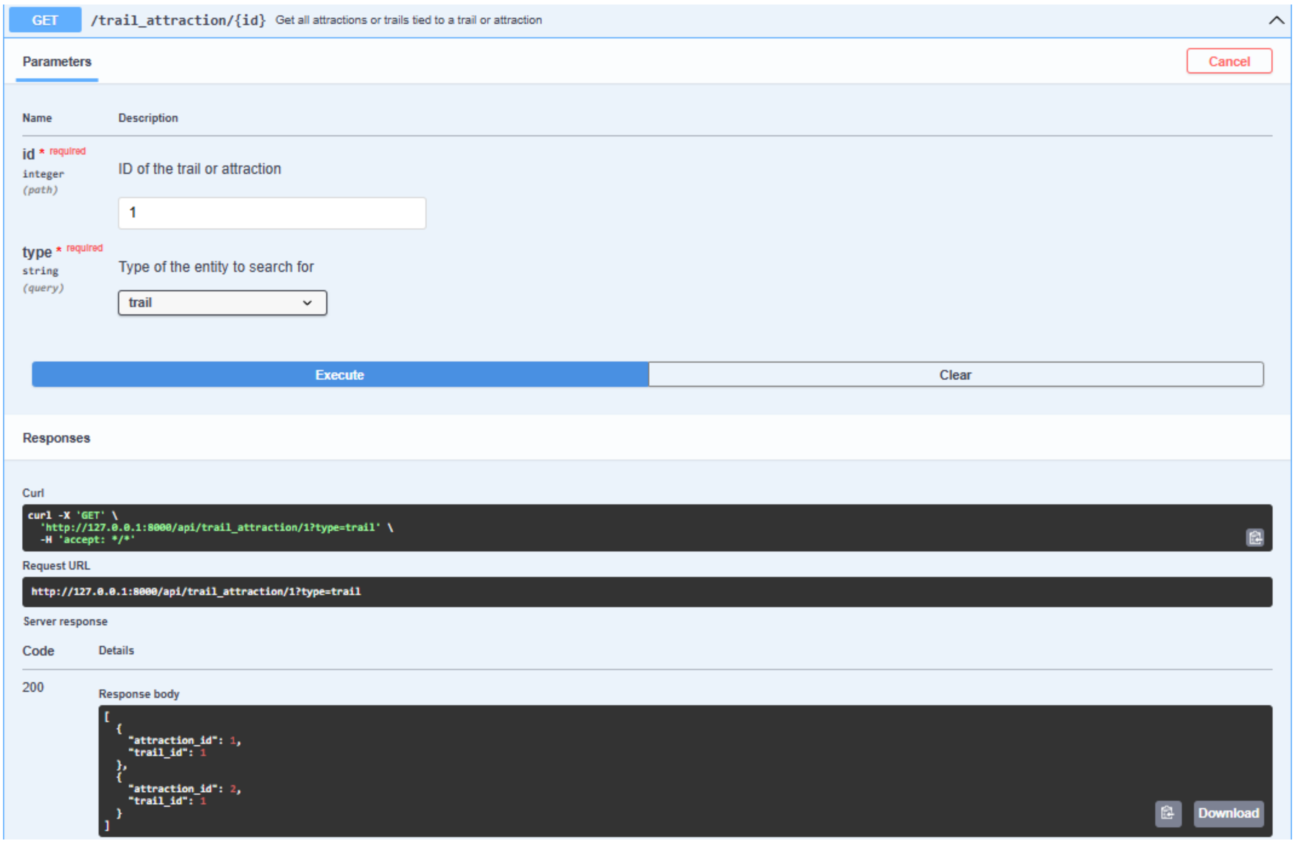
Server response

Code Details

201

Response body

```
{
  "attraction_id": 1,
  "trail_id": 2
}
```



/trail_attraction/{id} Get all attractions or trails tied to a trail or attraction

Parameters

Name	Description
id * required integer *(path)*	ID of the trail or attraction 1
type * required string *(query)*	Type of the entity to search for trail

Responses

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/api/trail_attraction/1?type=trail' \
  -H 'accept: */*'
```

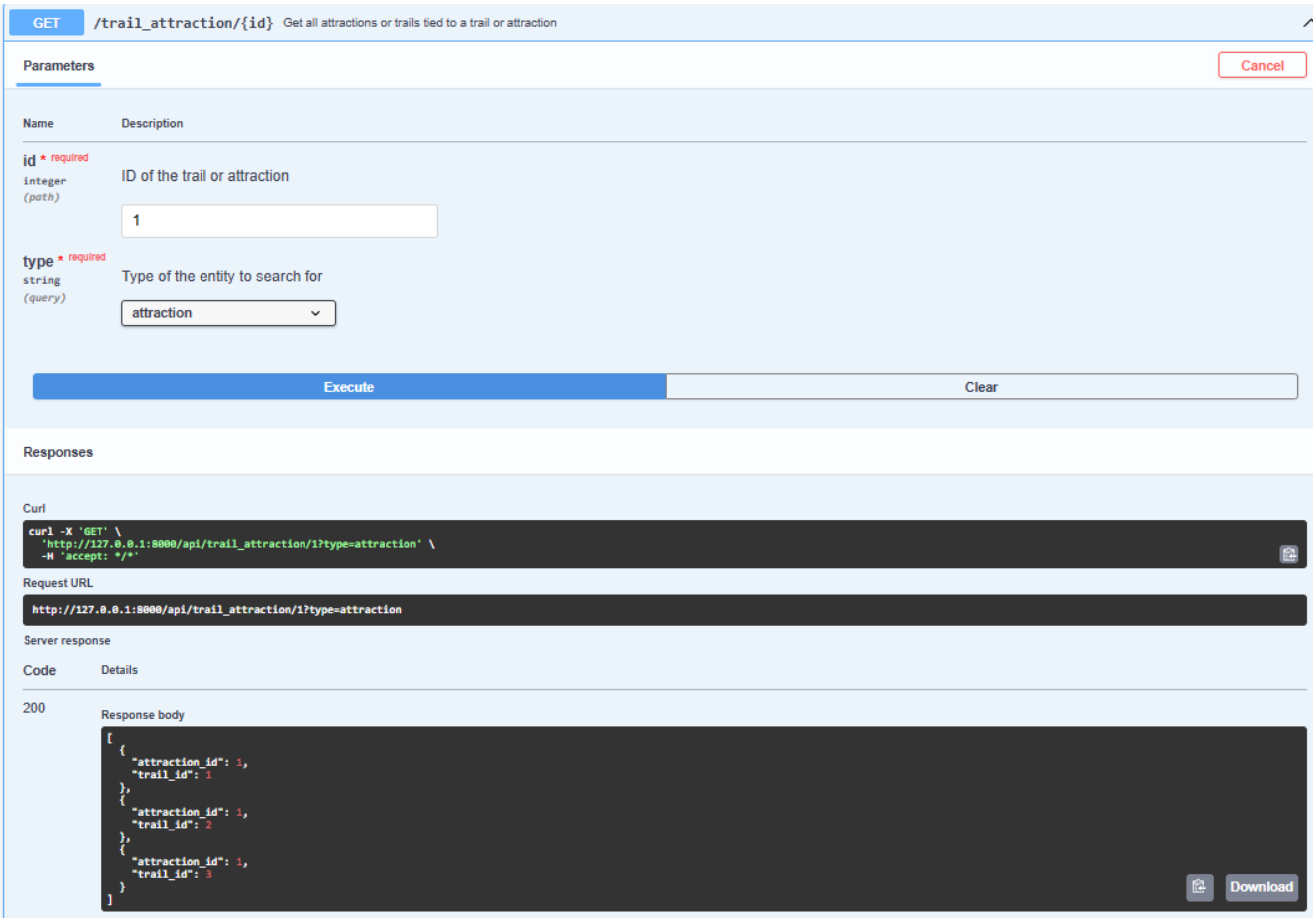
Request URL

```
http://127.0.0.1:8000/api/trail_attraction/1?type=trail
```

Server response

Code	Details
200	Response body

```
[
  {
    "attraction_id": 1,
    "trail_id": 1
  },
  {
    "attraction_id": 2,
    "trail_id": 1
  }
]
```



/trail_attraction/{id} Get all attractions or trails tied to a trail or attraction

Parameters

Name	Description
id * required integer *(path)*	ID of the trail or attraction 1
type * required string *(query)*	Type of the entity to search for attraction

Responses

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/api/trail_attraction/1?type=attraction' \
  -H 'accept: */*'
```

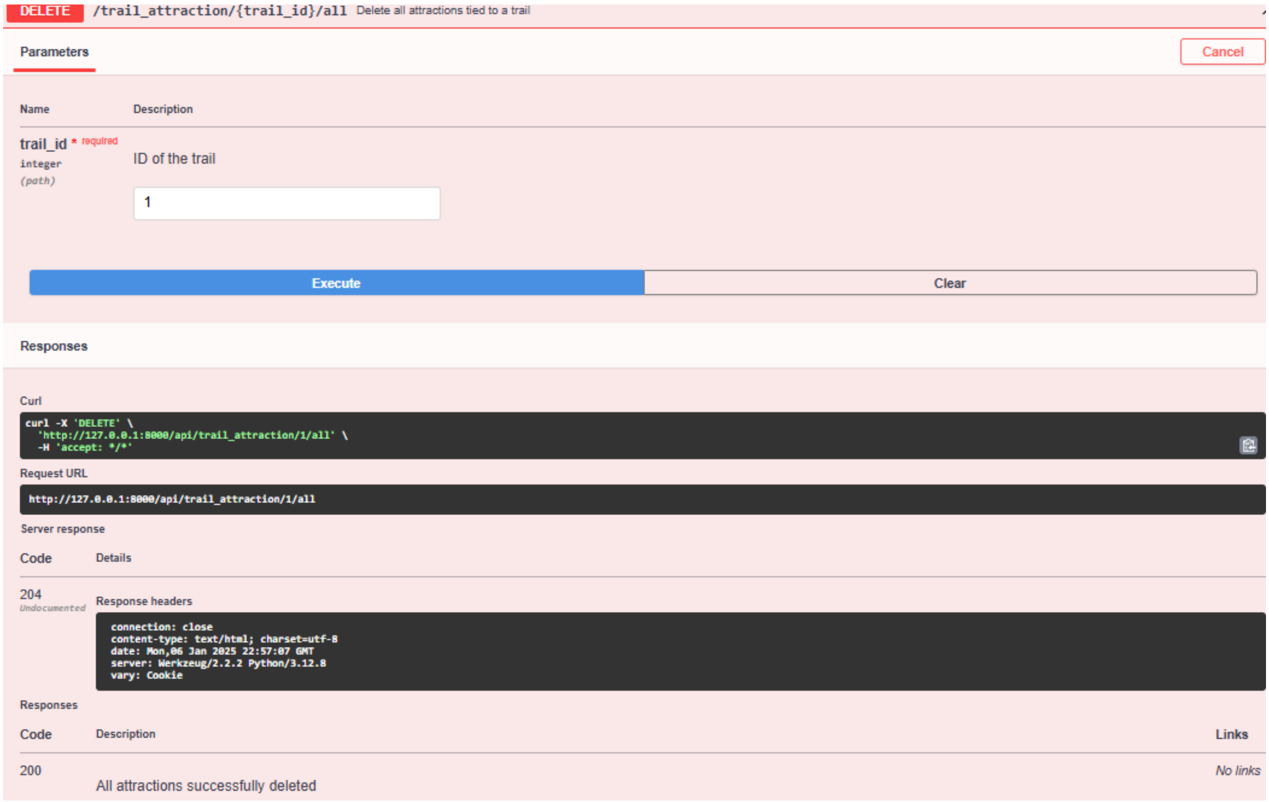
Request URL

```
http://127.0.0.1:8000/api/trail_attraction/1?type=attraction
```

Server response

Code	Details
200	Response body

```
[
  {
    "attraction_id": 1,
    "trail_id": 1
  },
  {
    "attraction_id": 1,
    "trail_id": 2
  },
  {
    "attraction_id": 1,
    "trail_id": 3
  }
]
```


DELETE
/trail_attraction/{trail_id}/all
Delete all attractions tied to a trail
Parameters
Cancel
Execute
Clear

Name	Description
trail_id * required integer *(path)*	ID of the trail 1

Responses

Curl

```
curl -X 'DELETE' \
  'http://127.0.0.1:8000/api/trail_attraction/1/all' \
  -H 'accept: */*'
```

Request URL

```
http://127.0.0.1:8000/api/trail_attraction/1/all
```

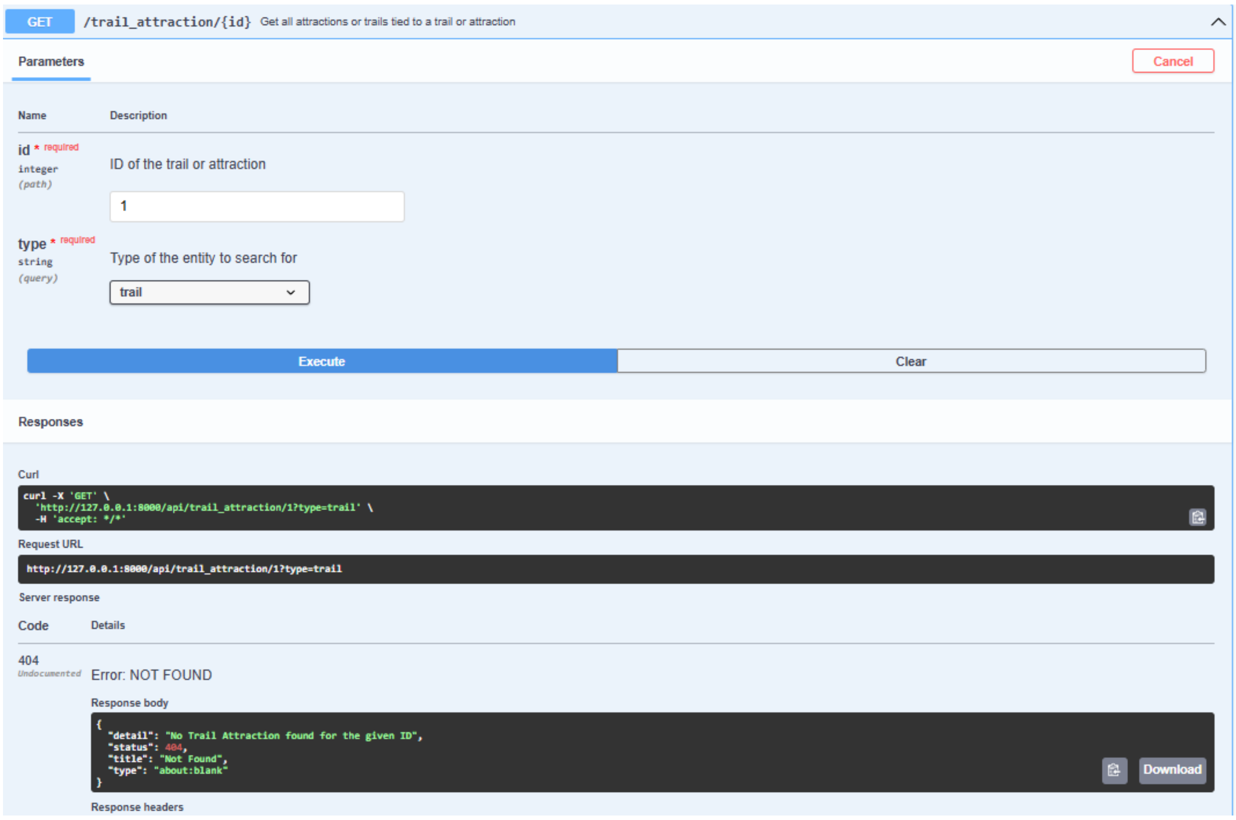
Server response

Code	Details
204 *Undocumented*	Response headers

```
connection: close
content-type: text/html; charset=utf-8
date: Mon,06 Jan 2025 22:57:07 GMT
server: Werkzeug/2.2.2 Python/3.12.8
vary: Cookie
```

Responses

Code	Description	Links
200	All attractions successfully deleted	*No links*



Name	Description
id * required integer *(path)*	ID of the trail or attraction 1
type * required string *(query)*	Type of the entity to search for trail

Responses

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/api/trail_attraction/1?type=trail' \
  -H 'accept: */*'
```

Request URL

```
http://127.0.0.1:8000/api/trail_attraction/1?type=trail
```

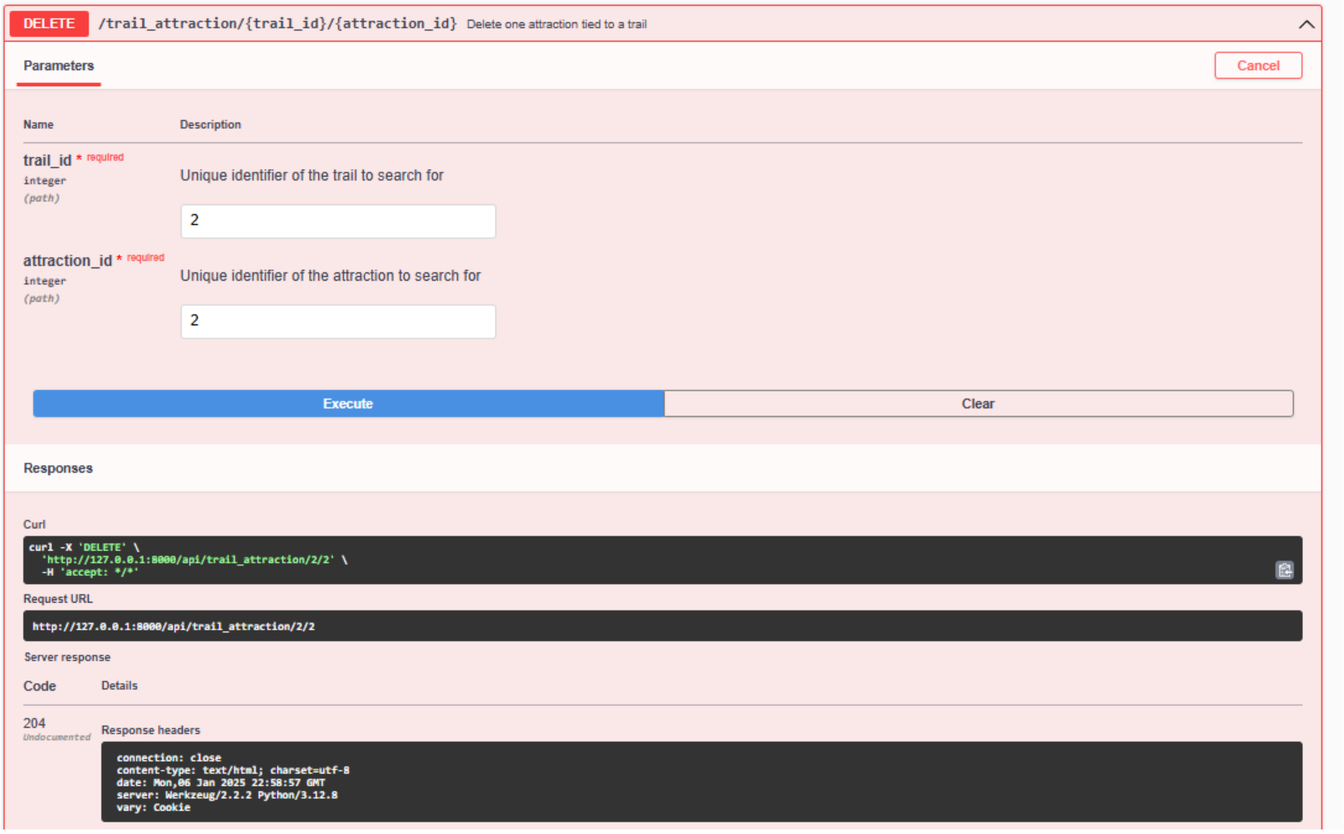
Server response

Code	Details
404 *Undocumented*	Error: NOT FOUND

Response body

```
{
  "detail": "No Trail Attraction found for the given ID",
  "status": 404,
  "title": "Not Found",
  "type": "about:blank"
}
```

Response headers



`/trail_attraction/{trail_id}/{attraction_id}` Delete one attraction tied to a trail

Name	Description
trail_id * required integer *(path)*	Unique identifier of the trail to search for 2
attraction_id * required integer *(path)*	Unique identifier of the attraction to search for 2

Curl

```
curl -X 'DELETE' \
  'http://127.0.0.1:8000/api/trail_attraction/2/2' \
  -H 'accept: */*'
```

Request URL

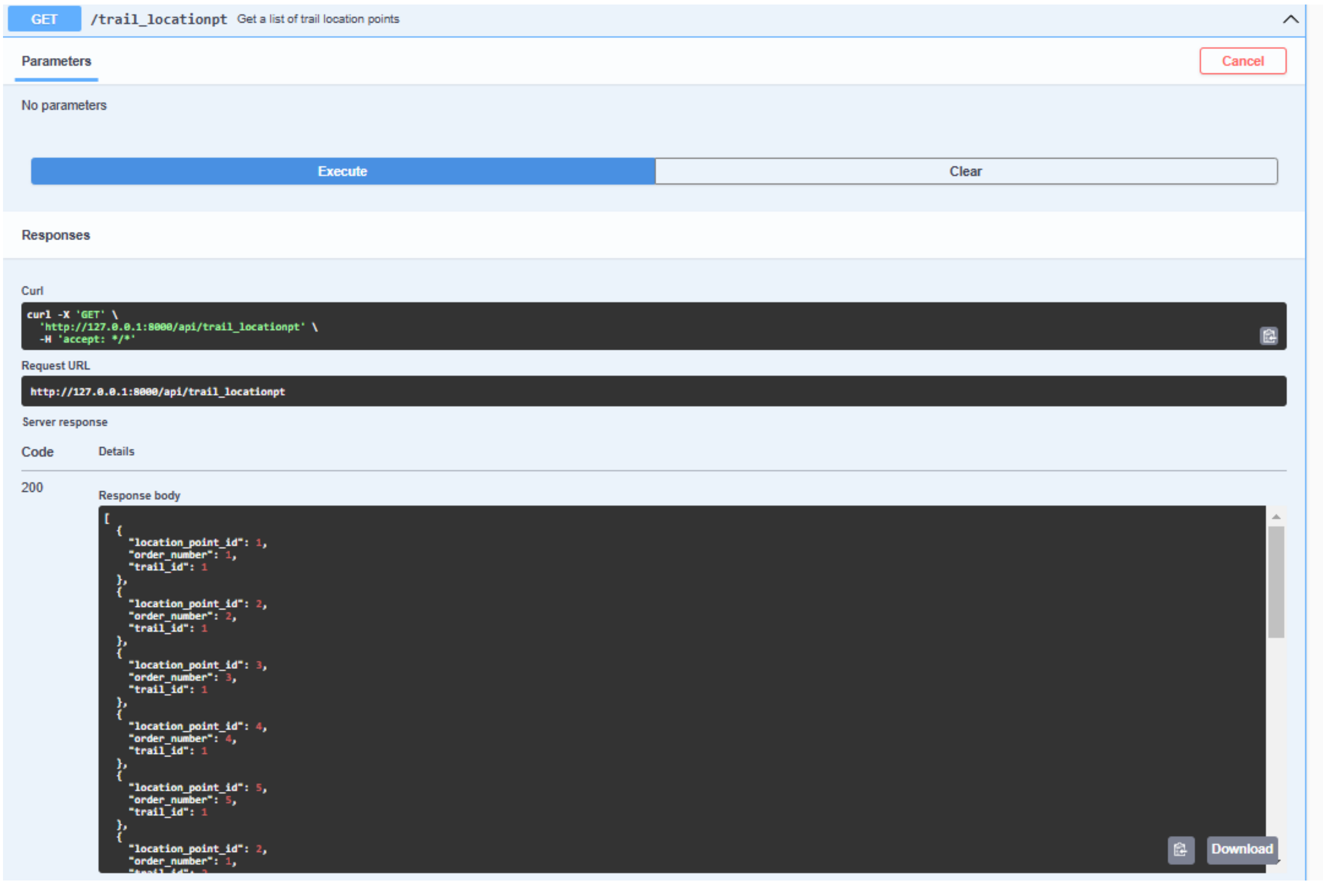
```
http://127.0.0.1:8000/api/trail_attraction/2/2
```

Server response

Code	Details
204 *Undocumented*	Response headers

```
connection: close
content-type: text/html; charset=utf-8
date: Mon,06 Jan 2025 22:58:57 GMT
server: Werkzeug/2.2.2 Python/3.12.8
vary: Cookie
```

TrailLocationPoints



`/trail_locationpt` Get a list of trail location points

No parameters

Responses

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/api/trail_locationpt' \
  -H 'accept: */*'
```

Request URL

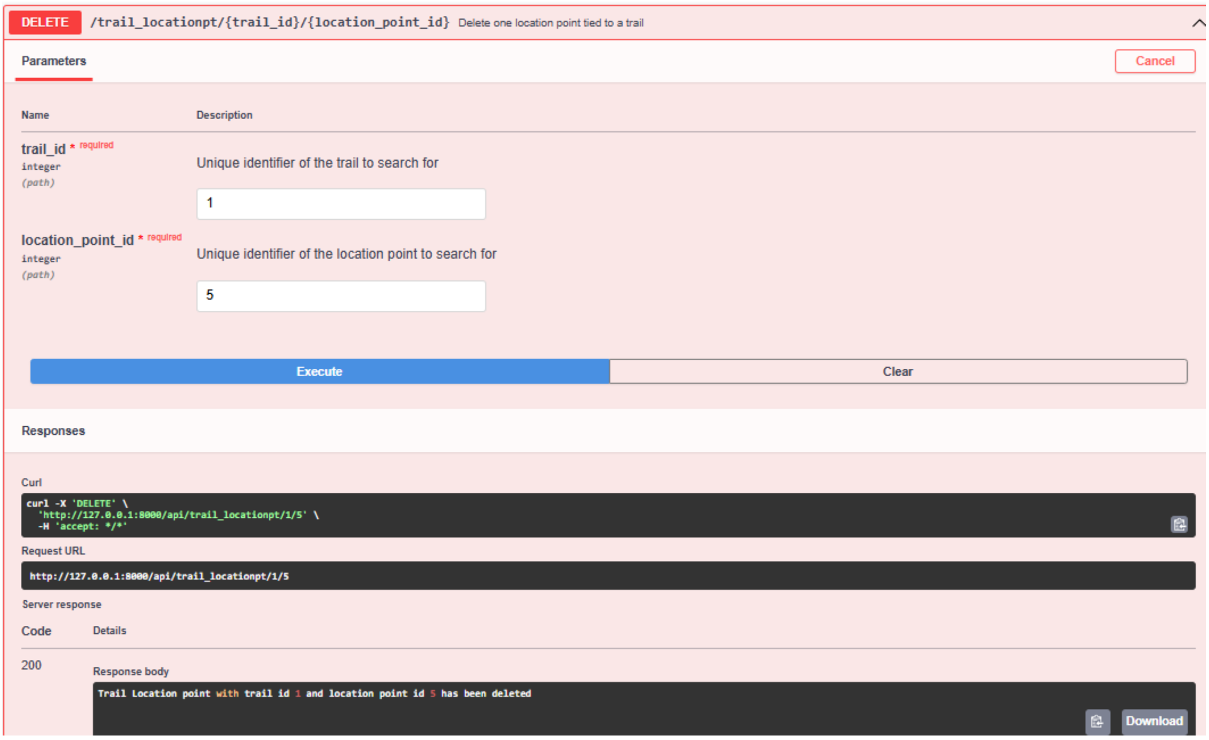
```
http://127.0.0.1:8000/api/trail_locationpt
```

Server response

Code Details

200 Response body

```
[
  {
    "location_point_id": 1,
    "order_number": 1,
    "trail_id": 1
  },
  {
    "location_point_id": 2,
    "order_number": 2,
    "trail_id": 1
  },
  {
    "location_point_id": 3,
    "order_number": 3,
    "trail_id": 1
  },
  {
    "location_point_id": 4,
    "order_number": 4,
    "trail_id": 1
  },
  {
    "location_point_id": 5,
    "order_number": 5,
    "trail_id": 1
  },
  {
    "location_point_id": 2,
    "order_number": 1,
```



`/trail_locationpt/{trail_id}/{location_point_id}` Delete one location point tied to a trail

Name	Description
trail_id * required integer *(path)*	Unique identifier of the trail to search for 1
location_point_id * required integer *(path)*	Unique identifier of the location point to search for 5

Responses

Curl

```
curl -X 'DELETE' \
  'http://127.0.0.1:8000/api/trail_locationpt/1/5' \
  -H 'accept: */*'
```

Request URL

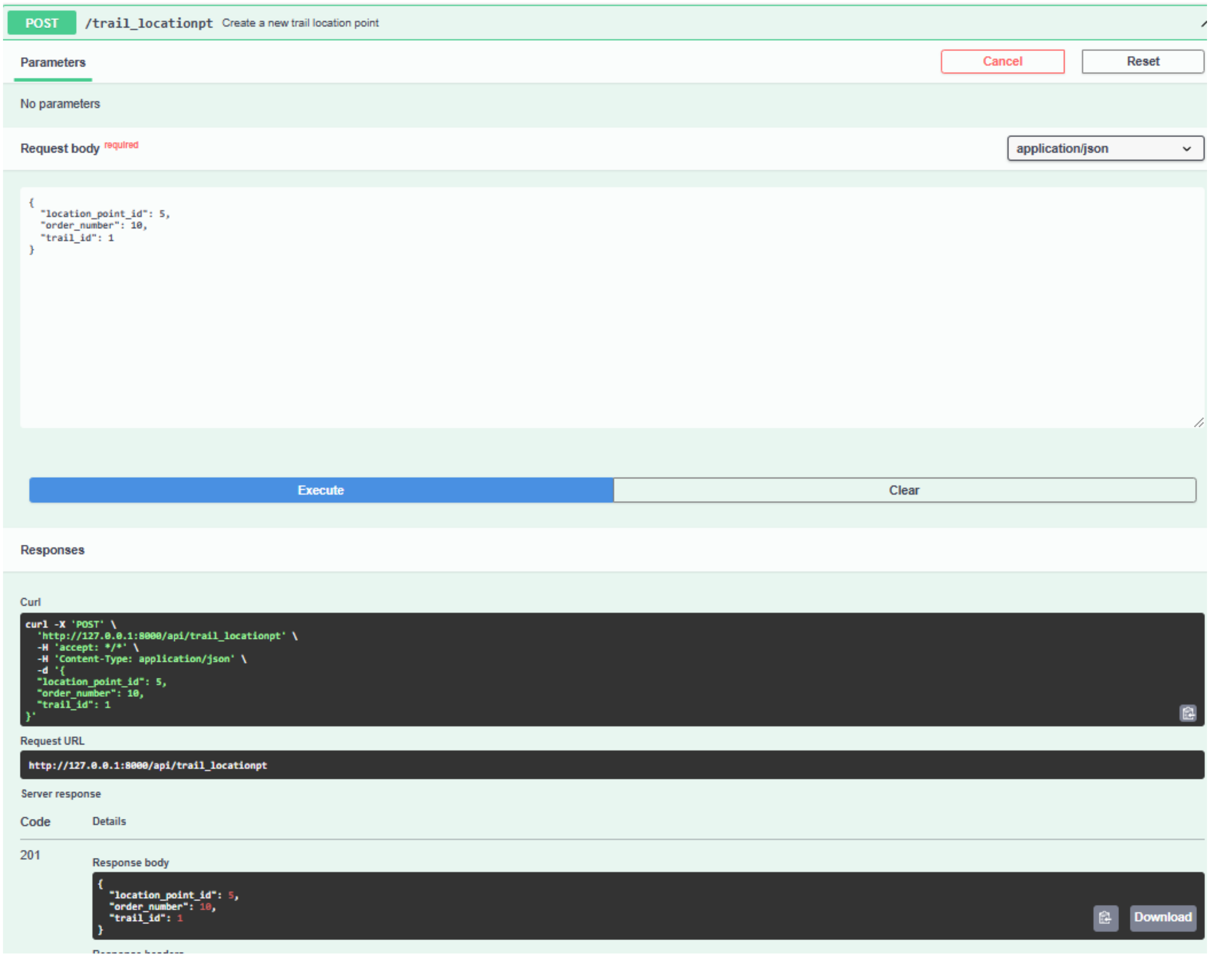
```
http://127.0.0.1:8000/api/trail_locationpt/1/5
```

Server response

Code Details

200 Response body

```
Trail Location point with trail id 1 and location point id 5 has been deleted
```



/trail_locationpt Create a new trail location point

No parameters

Request body required application/json

```
{
  "location_point_id": 5,
  "order_number": 10,
  "trail_id": 1
}
```

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/api/trail_locationpt' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "location_point_id": 5,
  "order_number": 10,
  "trail_id": 1
}'
```

Request URL

```
http://127.0.0.1:8000/api/trail_locationpt
```

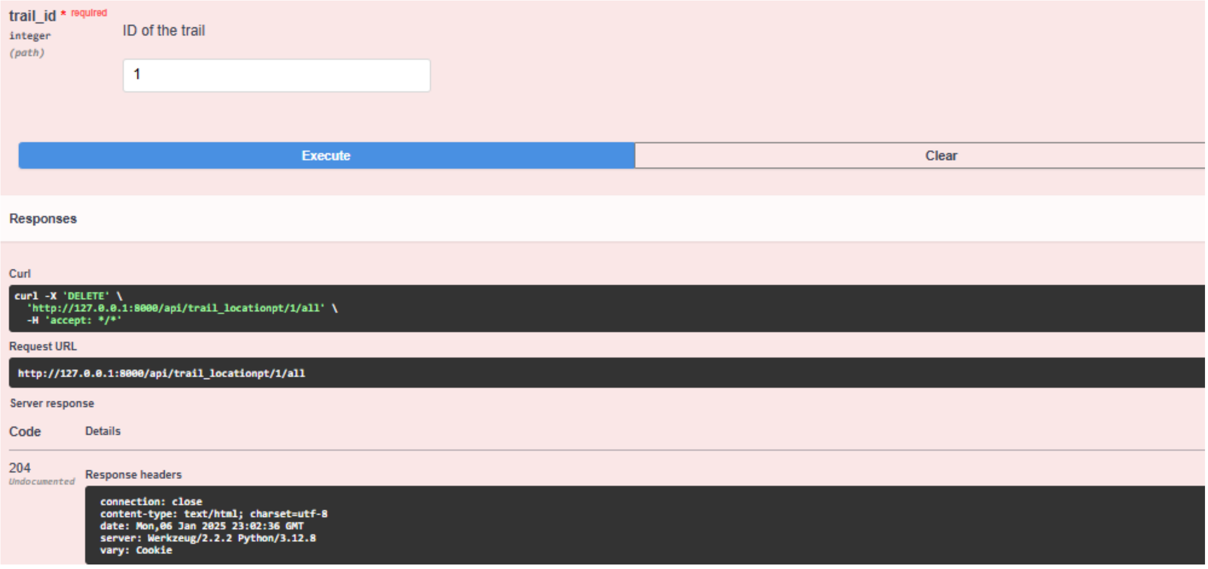
Server response

Code Details

201

Response body

```
{
  "location_point_id": 5,
  "order_number": 10,
  "trail_id": 1
}
```



trail_id * required
integer
(path)

ID of the trail

1

Curl

```
curl -X 'DELETE' \
  'http://127.0.0.1:8000/api/trail_locationpt/1/all' \
  -H 'accept: */*'
```

Request URL

```
http://127.0.0.1:8000/api/trail_locationpt/1/all
```

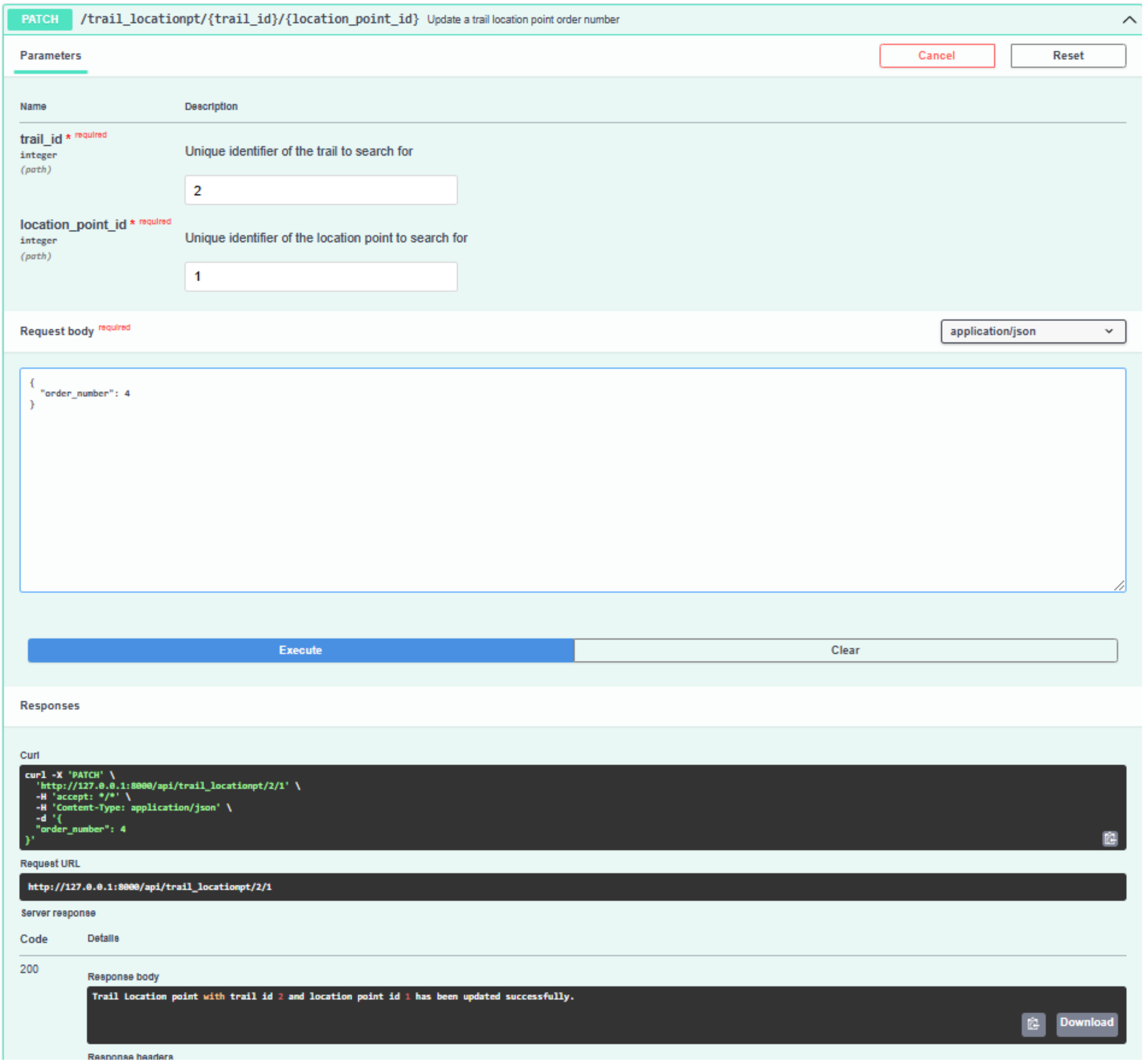
Server response

Code Details

204
Undocumented

Response headers

```
connection: close
content-type: text/html; charset=utf-8
date: Mon,06 Jan 2025 23:02:36 GMT
server: Werkzeug/2.2.2 Python/3.12.8
vary: Cookie
```



/trail_locationpt/{trail_id}/{location_point_id} Update a trail location point order number

Name	Description
trail_id * required integer (path)	Unique identifier of the trail to search for 2
location_point_id * required integer (path)	Unique identifier of the location point to search for 1

Request body required

application/json

```
{
  "order_number": 4
}
```

Responses

Curl

```
curl -X 'PATCH' \
  'http://127.0.0.1:8000/api/trail_locationpt/2/1' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "order_number": 4
}'
```

Request URL

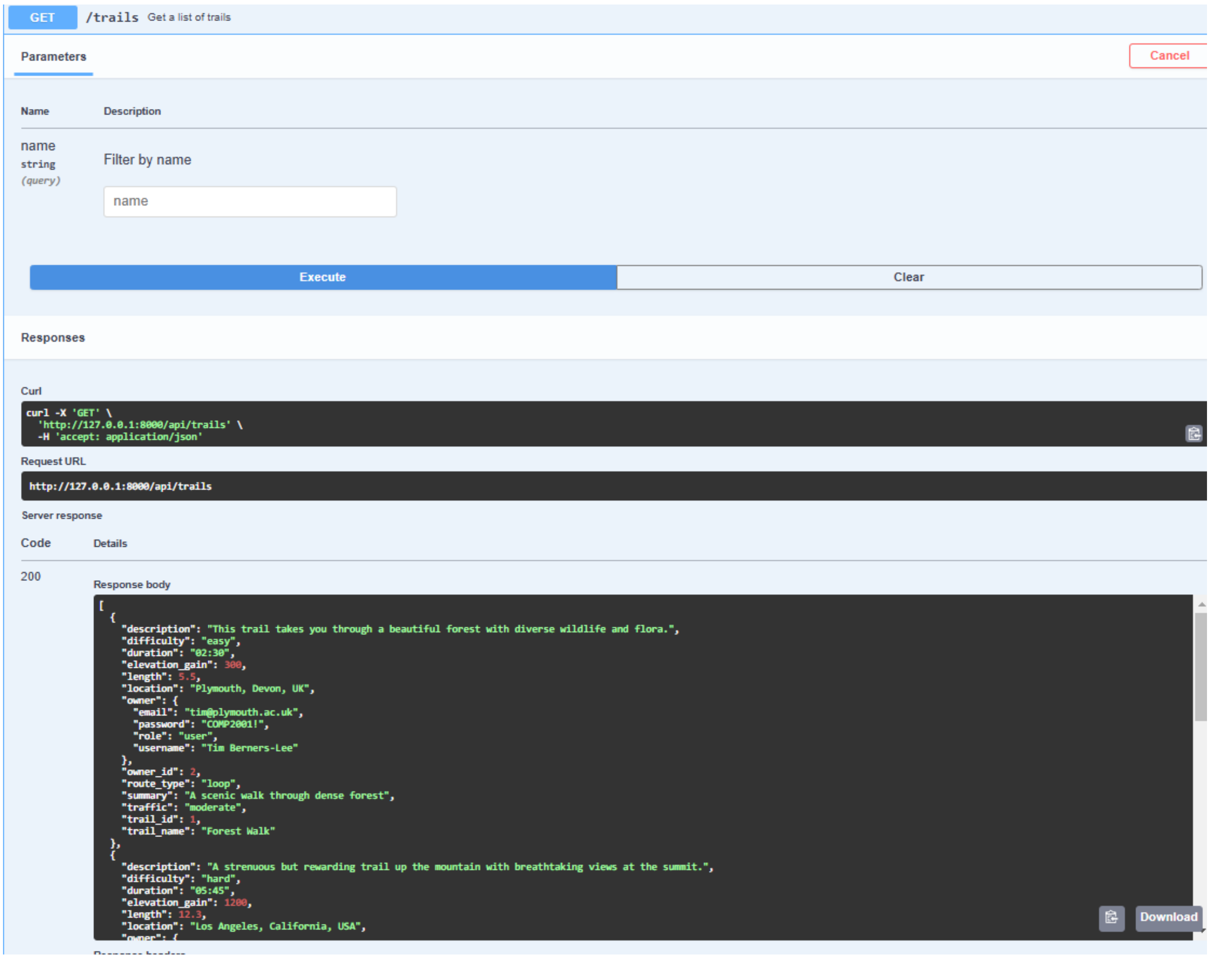
```
http://127.0.0.1:8000/api/trail_locationpt/2/1
```

Server response

Code	Details
200	Response body Trail Location point with trail id 2 and location point id 1 has been updated successfully.

Response headers

Trails



name
string
(query)

Filter by name

name

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/api/trails' \
  -H 'accept: application/json'
```

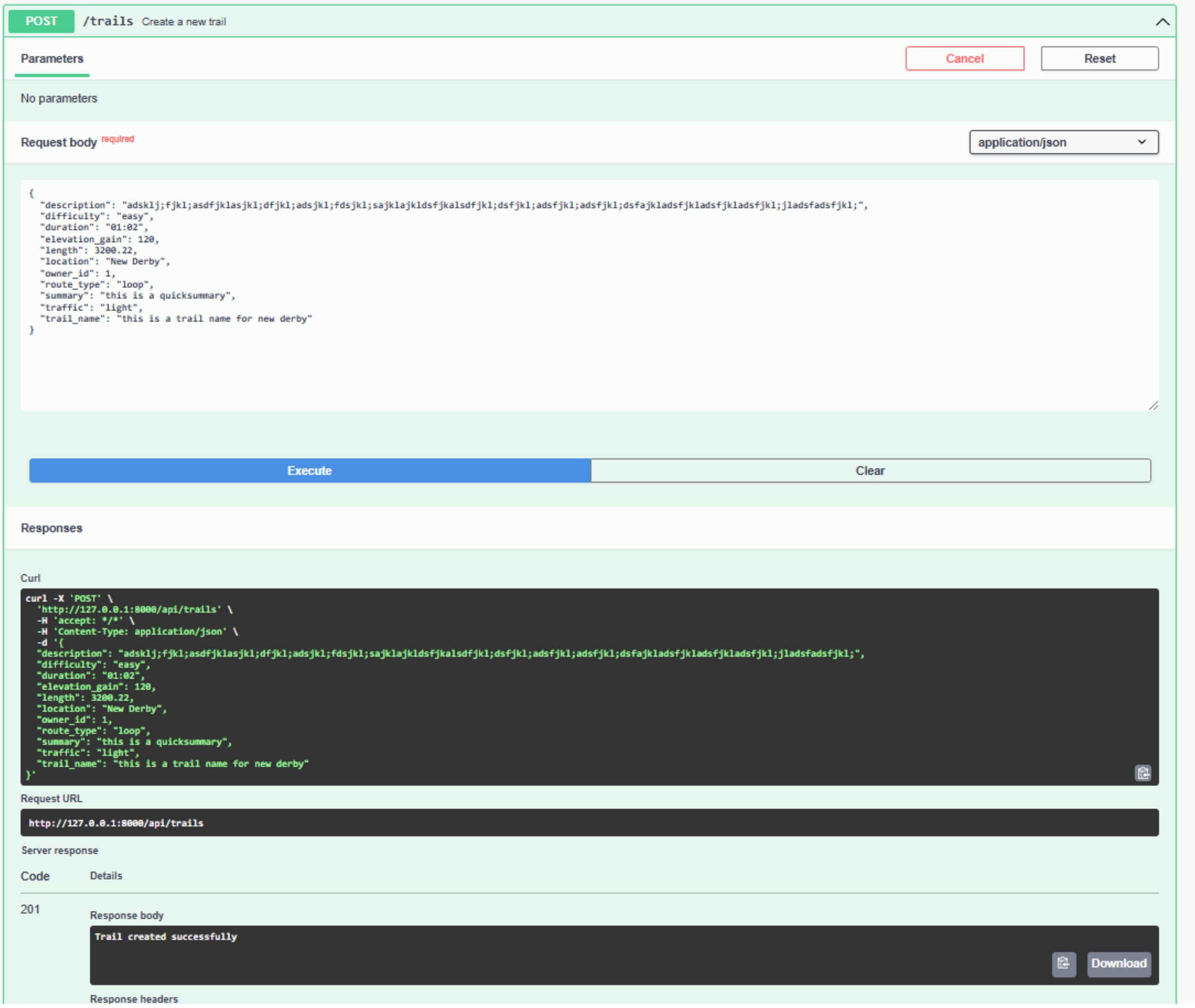
Request URL

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

Server response

Code	Details
200	Response body

```
[
  {
    "description": "This trail takes you through a beautiful forest with diverse wildlife and flora.",
    "difficulty": "easy",
    "duration": "02:30",
    "elevation_gain": 300,
    "length": 5.5,
    "location": "Plymouth, Devon, UK",
    "owner": {
      "email": "tim@plymouth.ac.uk",
      "password": "COMP2001!",
      "role": "user",
      "username": "Tim Berners-Lee"
    },
    "owner_id": 2,
    "route_type": "loop",
    "summary": "A scenic walk through dense forest",
    "traffic": "moderate",
    "trail_id": 1,
    "trail_name": "Forest Walk"
  },
  {
    "description": "A strenuous but rewarding trail up the mountain with breathtaking views at the summit.",
    "difficulty": "hard",
    "duration": "05:45",
    "elevation_gain": 1200,
    "length": 12.3,
    "location": "Los Angeles, California, USA",
    "owner": {
```



No parameters

Request body required

```
{
  "description": "adsklj;fjkl;asdfjklasjkl;dfjkl;adsjkl;fdsjkl;sajklajkldsfjkalsdfjkl;dsfjkl;adsfjkl;adsfjkl;dsfajkladsfjkladsfjkladsfjkl;jladsfadsfjkl;",
  "difficulty": "easy",
  "duration": "01:02",
  "elevation_gain": 120,
  "length": 3200.22,
  "location": "New Derby",
  "owner_id": 1,
  "route_type": "loop",
  "summary": "this is a quicksummary",
  "traffic": "light",
  "trail_name": "this is a trail name for new derby"
}
```

Responses

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/api/trails' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "description": "adsklj;fjkl;asdfjklasjkl;dfjkl;adsjkl;fdsjkl;sajklajkldsfjkalsdfjkl;dsfjkl;adsfjkl;adsfjkl;dsfajkladsfjkladsfjkladsfjkl;jladsfadsfjkl;",
  "difficulty": "easy",
  "duration": "01:02",
  "elevation_gain": 120,
  "length": 3200.22,
  "location": "New Derby",
  "owner_id": 1,
  "route_type": "loop",
  "summary": "this is a quicksummary",
  "traffic": "light",
  "trail_name": "this is a trail name for new derby"
}'
```

Request URL

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

Server response

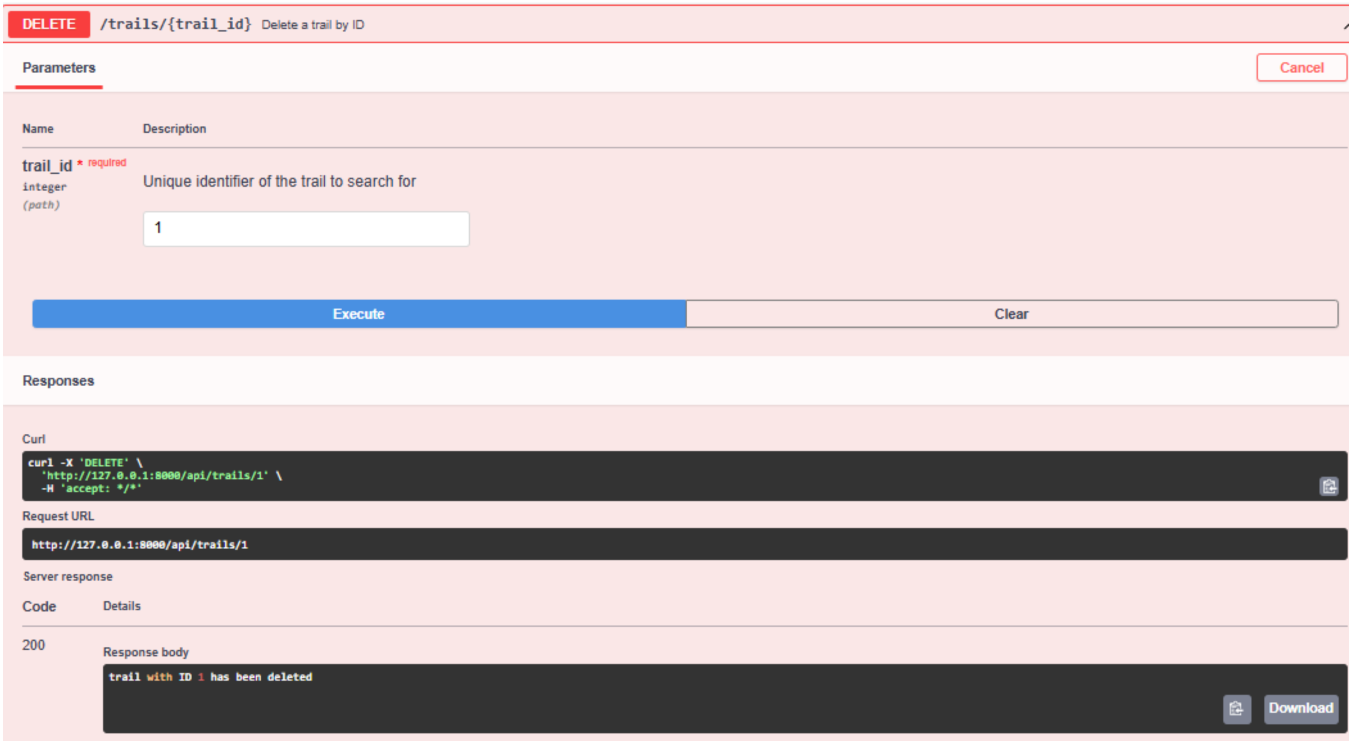
Code Details

201

Response body

```
Trail created successfully
```

Response headers



Name Description

trail_id * required
integer
(path)

Unique identifier of the trail to search for

1

Responses

Curl

```
curl -X 'DELETE' \
  'http://127.0.0.1:8000/api/trails/1' \
  -H 'accept: */*'
```

Request URL

```
http://127.0.0.1:8000/api/trails/1
```

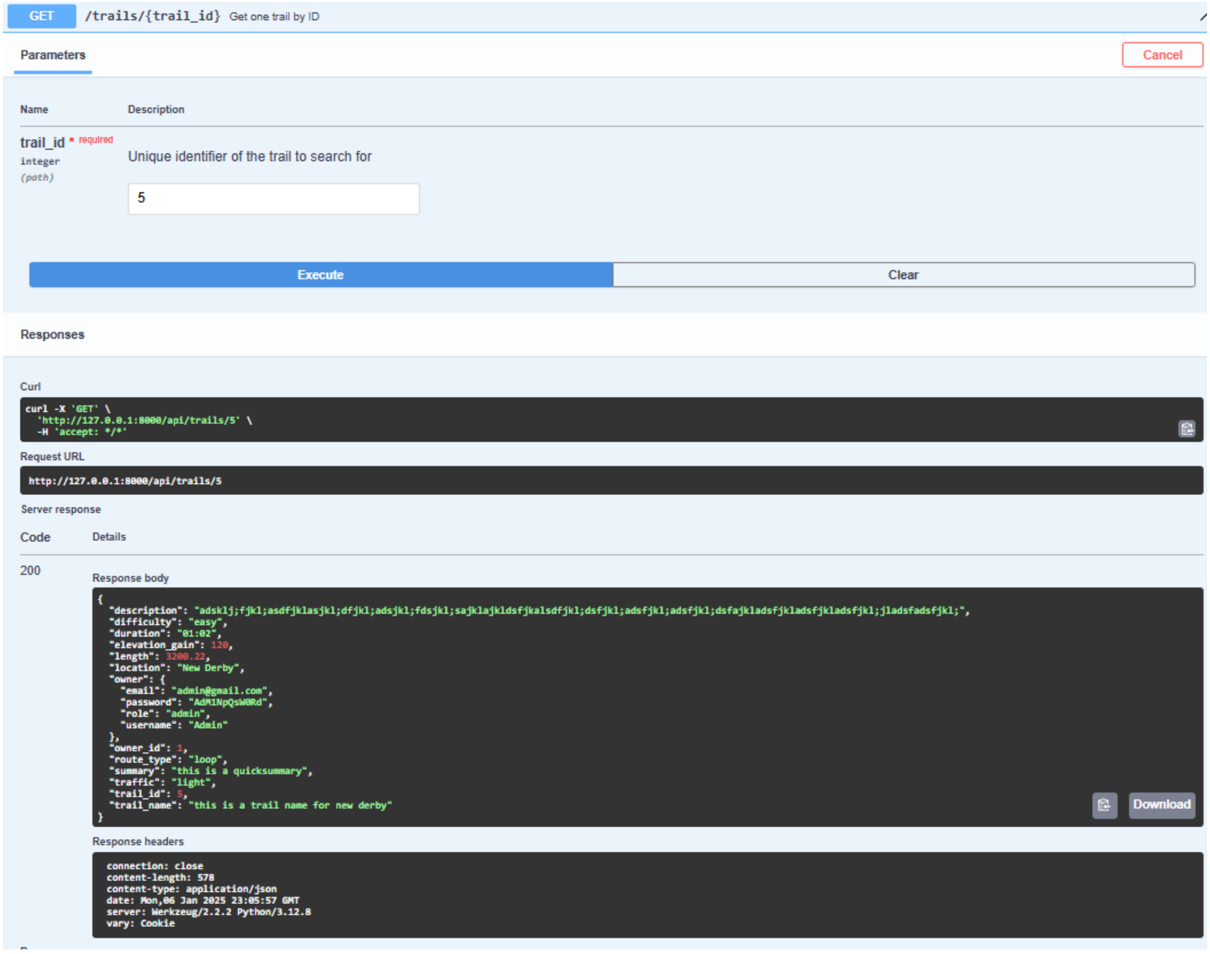
Server response

Code Details

200

Response body

```
trail with ID 1 has been deleted
```



/trails/{trail_id} Get one trail by ID

Parameters

Name | Description

trail_id * required
integer
(path)

Unique identifier of the trail to search for

5

Responses

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/api/trails/5' \
  -H 'accept: */*'
```

Request URL

```
http://127.0.0.1:8000/api/trails/5
```

Server response

Code | Details

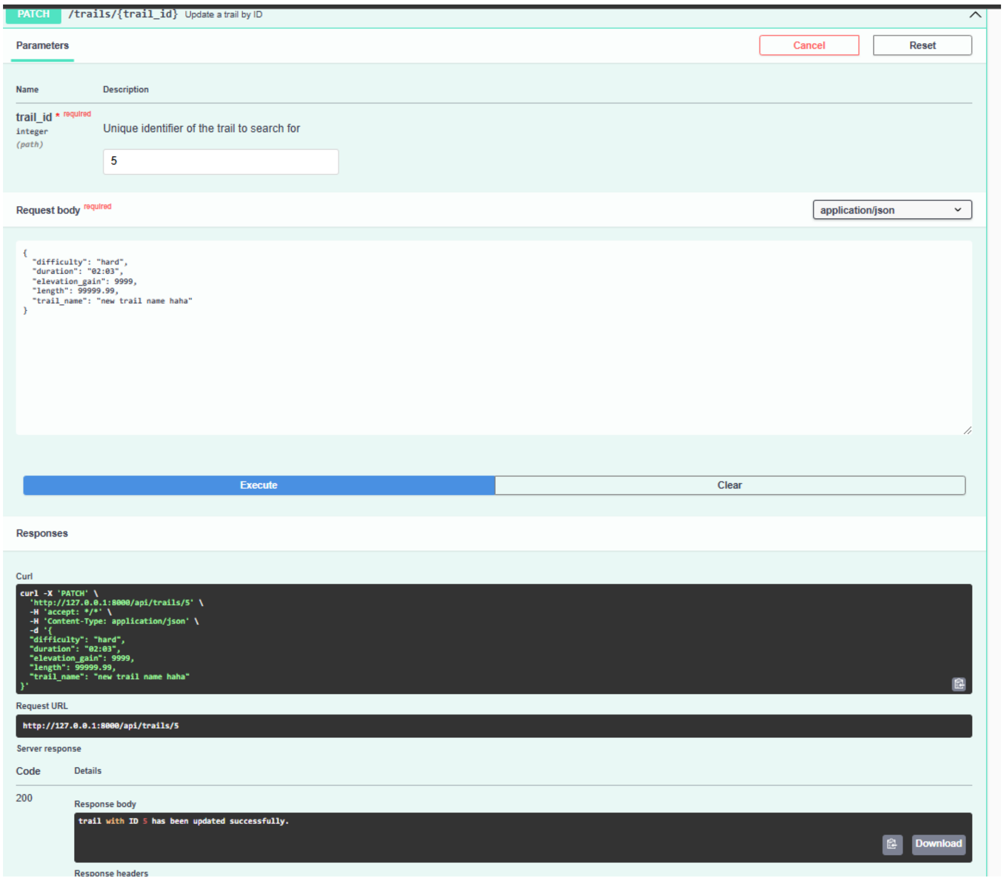
200

Response body

```
{
  "description": "adsklj;fjkl;asdfjklasjkl;dfjkl;adsjkl;fdsjkl;sajklajkldsfjkalsdfjkl;dsfjkl;adsfjkl;adsfjkl;dsfajkladsfjkladsfjkladsfjkl;jladsfadsfjkl;",
  "difficulty": "easy",
  "duration": "01:02",
  "elevation_gain": 120,
  "length": 3200.22,
  "location": "New Derby",
  "owner": {
    "email": "admin@gmail.com",
    "password": "AdM1NpQsW0Rd",
    "role": "admin",
    "username": "Admin"
  },
  "owner_id": 1,
  "route_type": "loop",
  "summary": "this is a quicksummary",
  "traffic": "light",
  "trail_id": 5,
  "trail_name": "this is a trail name for new derby"
}
```

Response headers

```
connection: close
content-length: 578
content-type: application/json
date: Mon,06 Jan 2025 23:05:57 GMT
server: Werkzeug/2.2.2 Python/3.12.8
vary: Cookie
```



/trails/{trail_id} Update a trail by ID

Parameters

Name | Description

trail_id * required
integer
(path)

Unique identifier of the trail to search for

5

Request body required

application/json

```
{
  "difficulty": "hard",
  "duration": "02:03",
  "elevation_gain": 9999,
  "length": 99999.99,
  "trail_name": "new trail name haha"
}
```

Responses

Curl

```
curl -X 'PATCH' \
  'http://127.0.0.1:8000/api/trails/5' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "difficulty": "hard",
  "duration": "02:03",
  "elevation_gain": 9999,
  "length": 99999.99,
  "trail_name": "new trail name haha"
}'
```

Request URL

```
http://127.0.0.1:8000/api/trails/5
```

Server response

Code | Details

200

Response body

```
trail with ID 5 has been updated successfully.
```

Response headers