CW2 Report

- Larissa Ayres

Introduction

This document presents a comprehensive overview of the design, development, and implementation of a microservice for the COMP2001 coursework. The micro-service, named TrailService, plays a pivotal role in the Trail Application, a platform aimed at promoting outdoor activities and enhancing users' well-being through the management of digital trails. These trails are represented as sequences of location points curated by users to encourage exploration and engagement with nature.

The report provides a look at the TrailService's objectives, integration with the Trail Application, and the developmental processes undertaken to deliver a robust and functional solution. Key sections include the rationale behind its design, the strategies employed during implementation, an evaluation of its effectiveness, and the legal, social, ethical, and professional (LSEP) considerations integral to the project's success.

For further details, the source code is accessible via https://github.com/lissie9a/2001-CW2-Larissa-Ayres and docker image lissie9ayres/imagecw1:latest at https://github.com/lissie9a/2001-CW2-Larissa-Ayres/blob/main/DockerImage.zip

Please see https://www.alltrails.com/trail/england/devon/plymbridge-circular as an example of sample data.

Background

The TrailService microservice is an essential component of the Trail Application, a forward-thinking platform designed to promote physical and mental well-being by encouraging outdoor exploration. By offering users the ability to create, manage, and share digital trails, the application fosters a sense of connection with the natural environment. A trail, in this context, is defined as a series of geolocated points that form a route, curated by users for activities such as walking, running, or cycling.

The TrailService is tasked with managing these trails via CRUD (Create, Read, Update, Delete) operations. It ensures seamless interaction with the application's front-end, enabling users to efficiently manage their trail data. To maintain security and authenticity, the micro-service integrates with an existing Authenticator API for user authentication, ensuring each trail is securely linked to its creator. The use of RESTful endpoints ensures compatibility with modern software standards, facilitating smooth data handling and scalability.

By addressing core functionalities like data integrity, user authentication, and operational reliability, the TrailService streamlines data management while enhancing user experience. This service exemplifies how innovative technology can simplify complex tasks, delivering value through accuracy, usability, and adherence to contemporary design principles.

Legal, Social, Ethical and Professional (LSEP)

In developing the TrailService microservice, emphasis was placed on addressing legal, social, ethical, and professional (LSEP) considerations to ensure reliability and trustworthiness. This reflects broader efforts to integrate sustainability and ethical awareness into Computer Science curricula (Andrew Gordon, n.d.; Gordon, 2014).

Data privacy was a key focus. The micro-service integrates with the Authenticator API to securely handle user authentication, ensuring sensitive information such as passwords remains encrypted. Role-based access controls further safeguard personal data by limiting access to authorized users. These measures align with sustainable development principles, emphasizing fairness and accountability (Gordon, 2014).

To maintain information integrity and security, the design incorporates parameterized queries to prevent SQL injection. RESTful endpoints are protected with authentication tokens to ensure secure and tamper-proof data exchanges. Regular log audits enhance transparency and operational accountability, aligning with sustainable practices that promote resilience (Andrew Gordon, n.d.).

The General Data Protection Regulation (GDPR) is a robust framework designed to protect the privacy and personal data of individuals within the European Union (Gov.UK, 2018). It imposes strict requirements on how organizations collect, process, store, and share personal data, emphasizing transparency and accountability. In the context of the TrailService microservice, GDPR compliance was a critical consideration, ensuring that data privacy is maintained. Measures such as limiting data collection to only what is necessary, encrypting sensitive information, and obtaining user consent where applicable help safeguard personal data and uphold users' rights. These practices align with GDPR's core principles, fostering trust and legal compliance.

Additionally, data preservation strategies were implemented through a dedicated log table. This captures key actions, such as trail creation and updates, with timestamps and user identifiers. These measures not only ensure a clear activity history but also support system recovery if needed. Collectively, the design addresses OWASP guidelines while fostering an ethical approach to data management.

Design

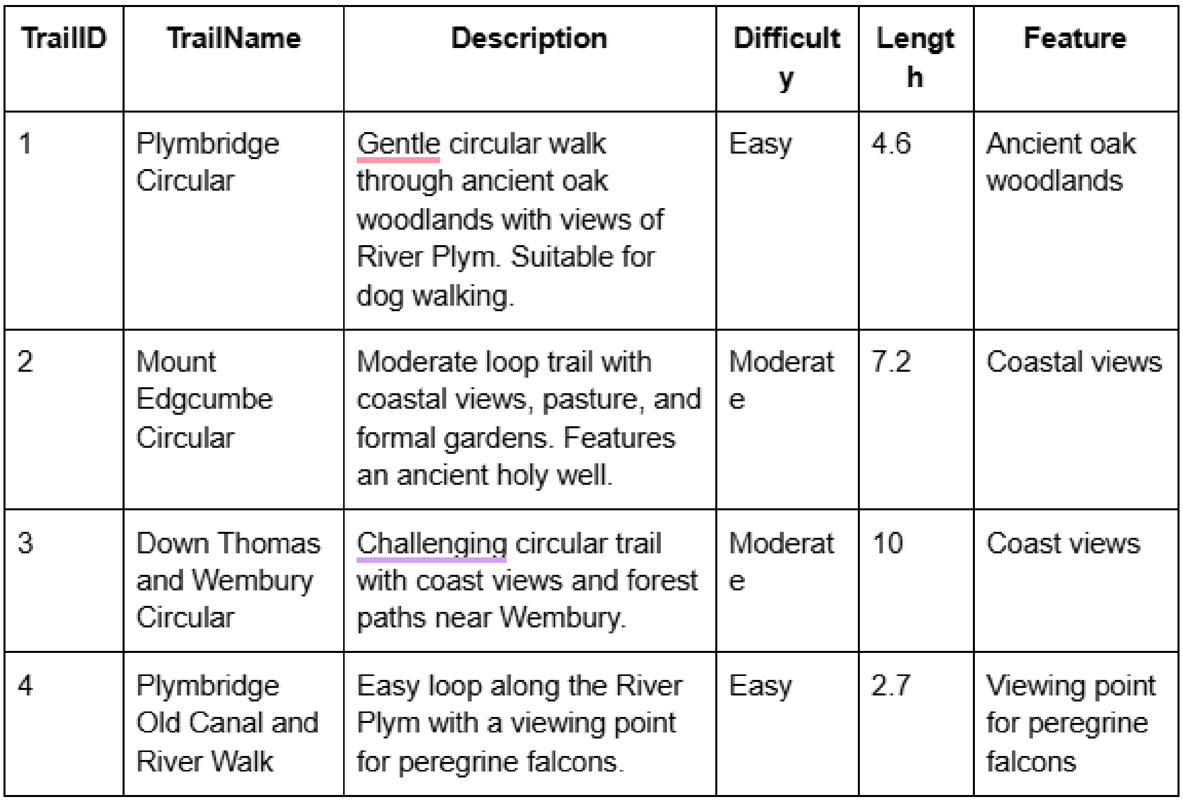
The database design for the TrailService microservice follows a structured normalization process to ensure data integrity, scalability, and flexibility. Through normalization, the data is organized into logical entities, eliminating redundancy and maintaining consistency. The design includes separate tables for trails, features, users, and their relationships, using bridge tables like Trail-Feature and Trail-User to manage many-to-many connections. This structure allows for easy addition of new data, such as trails or features, without altering the schema. By ensuring that each table stores only relevant data and relationships, the design enhances readability, supports efficient querying, and provides a robust foundation for managing well-being trail data.

Normalisation

1NF (First Normal Form)

Objective: Ensure all values are atomic (no repeating groups or multivalued attributes).

Trail Table



TrailID	TrailName	Description	Difficulty	Length	Feature
1	Plymbridge Circular	Gentle circular walk through ancient oak woodlands with views of River Plym. Suitable for dog walking.	Easy	4.6	Ancient oak woodlands
2	Mount Edgcumbe Circular	Moderate loop trail with coastal views, pasture, and formal gardens. Features an ancient holy well.	Moderate	7.2	Coastal views
3	Down Thomas and Wembury Circular	Challenging circular trail with coast views and forest paths near Wembury.	Moderate	10	Coast views
4	Plymbridge Old Canal and River Walk	Easy loop along the River Plym with a viewing point for peregrine falcons.	Easy	2.7	Viewing point for peregrine falcons

2NF (Second Normal Form)

Objective: Eliminate partial dependencies by separating data into distinct entities.

Trail Table

TrailID	TrailName	Difficulty	Length	Description
1	Plymbridge Circular	Easy	4.6	Gentle circular walk through ancient oak woodlands with views of River Plym. Suitable for dog walking.
2	Mount Edgcumbe Circular	Moderate	7.2	Moderate loop trail with coastal views, pasture, and formal gardens. Features an ancient holy well.
3	Down Thomas and Wembury Circular	Moderate	10	Challenging circular trail with coast views and forest paths near Wembury.
4	Plymbridge Old Canal and River Walk	Easy	2.7	Easy loop along the River Plym with a viewing point for peregrine falcons.

Feature Table

FeatureID	Feature
1	Ancient oak woodlands
2	Coastal views
3	Viewing point for peregrine falcons

Trail-Feature Table

TrailID	FeatureID
1	1
2	2
3	2
4	3

3NF (Third Normal Form)

Objective: Remove transitive dependencies by ensuring all non-primary key attributes are only dependent on the primary key.

Trail Table

TrailID	TrailName	Difficulty	Length	Description
1	Plymbridge Circular	Easy	4.6	Gentle circular walk through ancient oak woodlands with views of River Plym. Suitable for dog walking.
2	Mount Edgcumbe Circular	Moderate	7.2	Moderate loop trail with coastal views, pasture, and formal gardens. Features an ancient holy well.
3	Down Thomas and Wembury Circular	Moderate	10	Challenging circular trail with coast views and forest paths near Wembury.
4	Plymbridge Old Canal and River Walk	Easy	2.7	Easy loop along the River Plym with a viewing point for peregrine falcons.

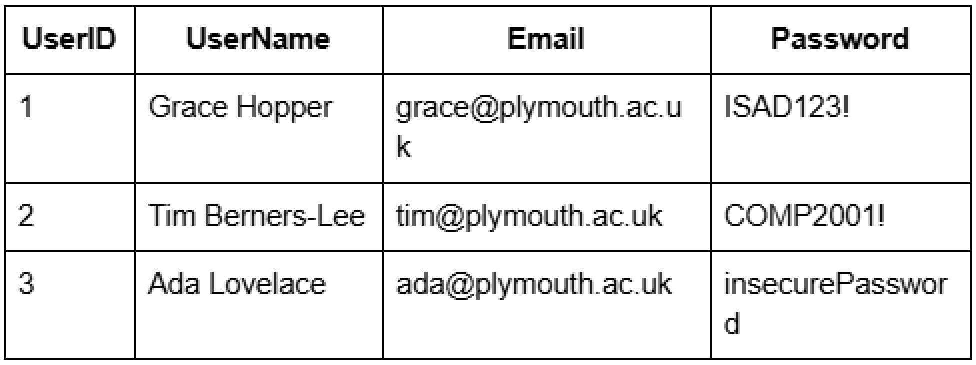
Feature Table

FeatureID	Feature
1	Ancient oak woodlands
2	Coastal views
3	Viewing point for peregrine falcons

Trail-Feature Table

TrailID	FeatureID
1	1
2	2
3	2
4	3

User Table

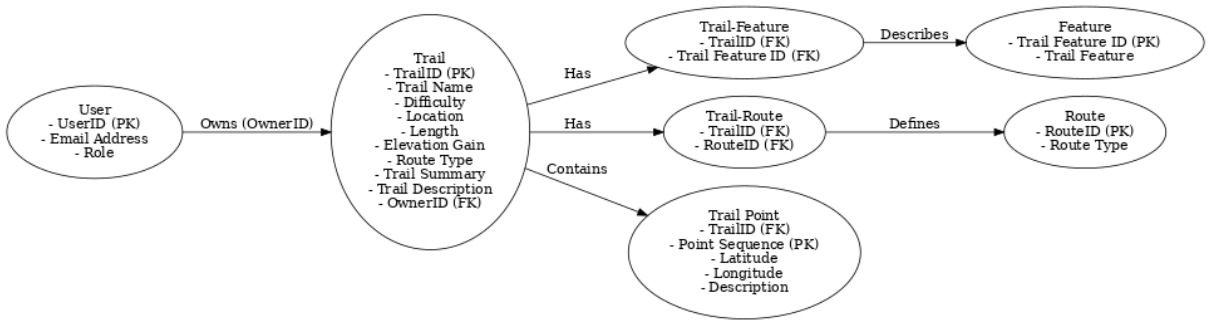


UserID	UserName	Email	Password
1	Grace Hopper	grace@plymouth.ac.uk	ISAD123!
2	Tim Berners-Lee	tim@plymouth.ac.uk	COMP2001!
3	Ada Lovelace	ada@plymouth.ac.uk	insecurePassword

Trail-User Table

TrailID	UserID
1	1
2	2
3	3
4	1

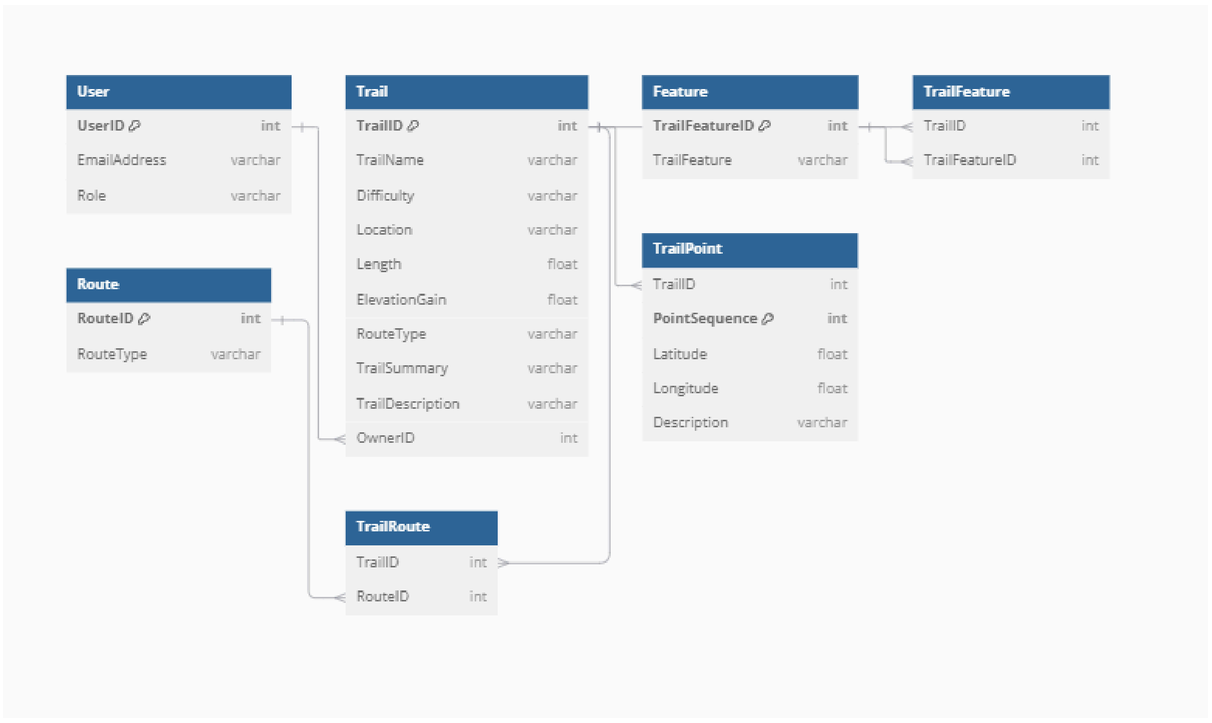
ERD



1.

Key Benefits of This Design:

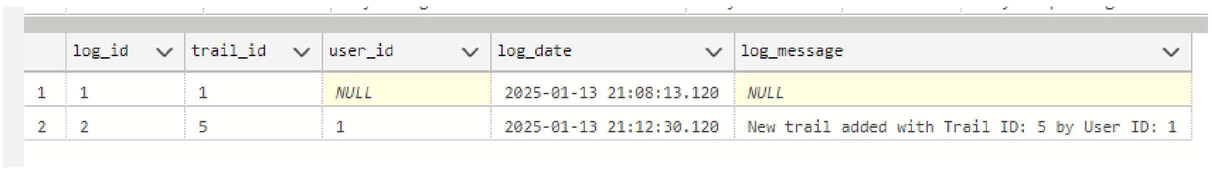
- **Data Integrity**: Normalized to reduce redundancy and maintain consistency across related data.
- **Scalability**: Easily accommodates new trails, features, users, or route types without altering the structure.
- **Flexibility**: Supports complex relationships, such as many-to-many connections between trails and features or trails and routes.
- **Readability**: Clearly separates data into logical entities, making it easier to understand and query.



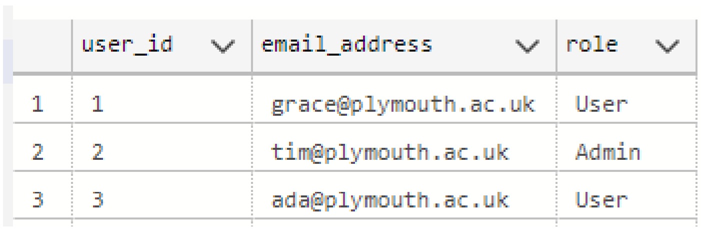
Implementation

SQL

Due to the CW1 SQL files not fully running triggers i re-did them as there was no way to just fix the triggers. They are now fully working and under the CW2 schema.

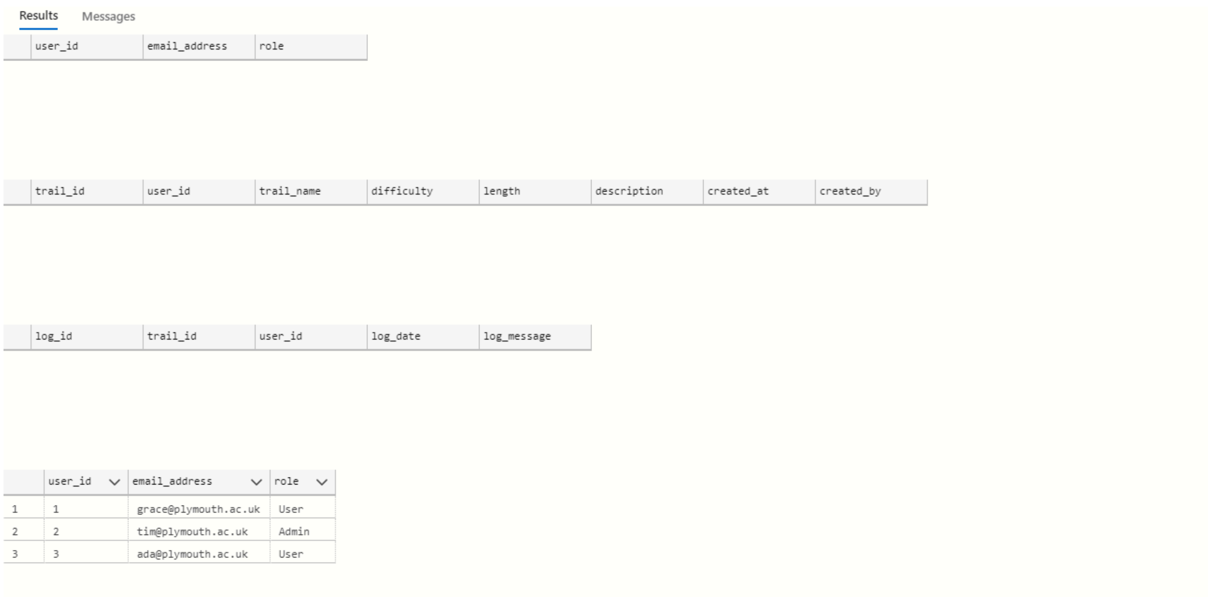


	log_id	trail_id	user_id	log_date	log_message
1	1	1	NULL	2025-01-13 21:08:13.120	NULL
2	2	5	1	2025-01-13 21:12:30.120	New trail added with Trail ID: 5 by User ID: 1



	user_id	email_address	role
1	1	grace@plymouth.ac.uk	User
2	2	tim@plymouth.ac.uk	Admin
3	3	ada@plymouth.ac.uk	User

Below are the screenshots of the data (user and trail))being added to the tables.

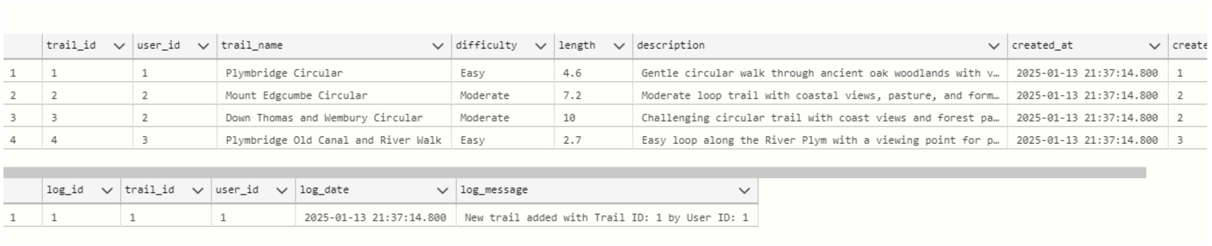


user_id	email_address	role

trail_id	user_id	trail_name	difficulty	length	description	created_at	created_by

log_id	trail_id	user_id	log_date	log_message

	user_id	email_address	role
1	1	grace@plymouth.ac.uk	User
2	2	tim@plymouth.ac.uk	Admin
3	3	ada@plymouth.ac.uk	User



	trail_id	user_id	trail_name	difficulty	length	description	created_at	create
1	1	1	Plymbridge Circular	Easy	4.6	Gentle circular walk through ancient oak woodlands with v…	2025-01-13 21:37:14.800	1
2	2	2	Mount Edgcumbe Circular	Moderate	7.2	Moderate loop trail with coastal views, pasture, and form…	2025-01-13 21:37:14.800	2
3	3	2	Down Thomas and Wembury Circular	Moderate	10	Challenging circular trail with coast views and forest pa…	2025-01-13 21:37:14.800	2
4	4	3	Plymbridge Old Canal and River Walk	Easy	2.7	Easy loop along the River Plym with a viewing point for p…	2025-01-13 21:37:14.800	3

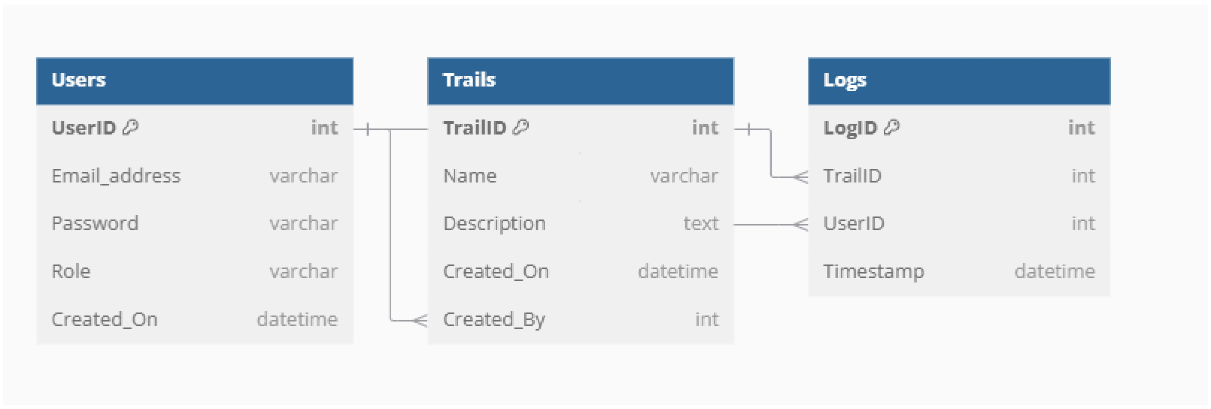
	log_id	trail_id	user_id	log_date	log_message
1	1	1	1	2025-01-13 21:37:14.800	New trail added with Trail ID: 1 by User ID: 1

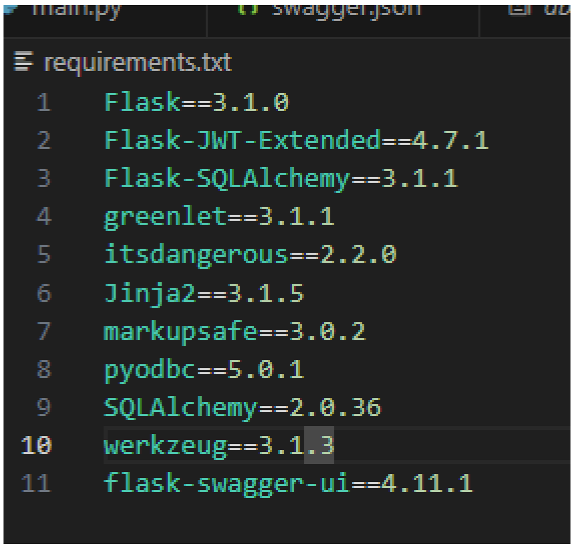
Python and Swagger

API Development:
A RESTful API was implemented using Python to provide endpoints for interacting with the database. The API was designed to support CRUD operations for trails and their associated data, ensuring compliance with REST principles for clarity and usability.

The TrailService was implemented using Python, with a Microsoft SQL Server backend. Key implementation includes:

- Stored Procedures: CRUD operations are encapsulated within stored procedures under the CW2 schema, ensuring modularity and maintainability.
- RESTful Endpoints: The API adheres to REST principles, providing clear and consistent methods for managing trail data.
- Integration with Authenticator API: The micro-service relies on the Authenticator API for user validation, ensuring that only authorized users can create or modify trails.



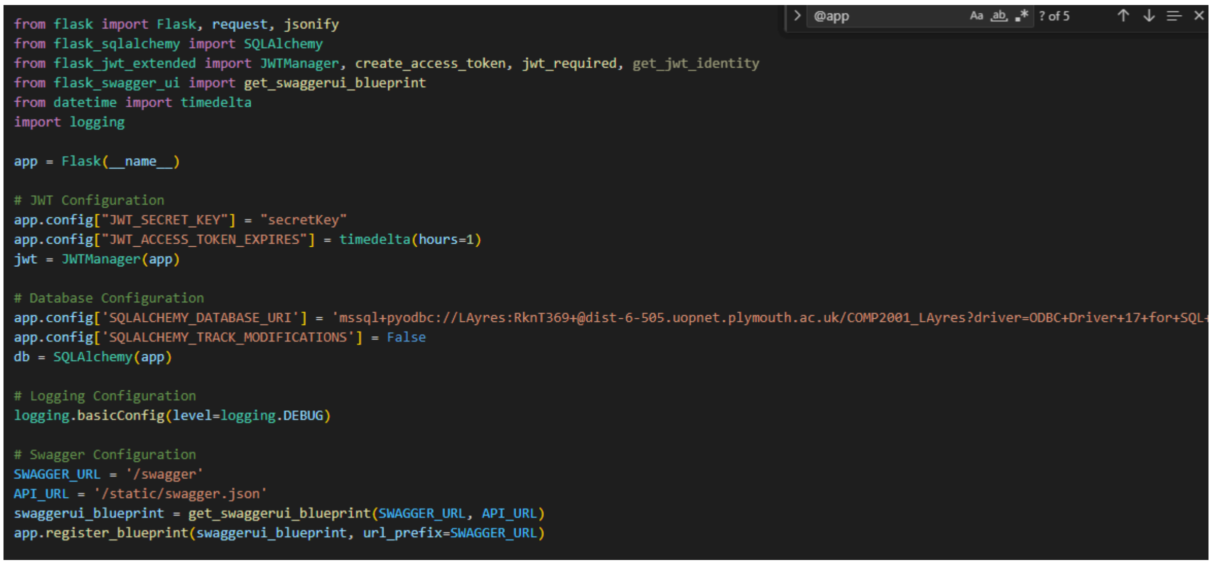


```
Flask==3.1.0
Flask-JWT-Extended==4.7.1
Flask-SQLAlchemy==3.1.1
greenlet==3.1.1
itsdangerous==2.2.0
Jinja2==3.1.5
markupsafe==3.0.2
pyodbc==5.0.1
SQLAlchemy==2.0.36
werkzeug==3.1.3
flask-swagger-ui==4.11.1
```

Here are my requirements kept in a separate .txt file so they can be called on and installed in one go in the powershell.

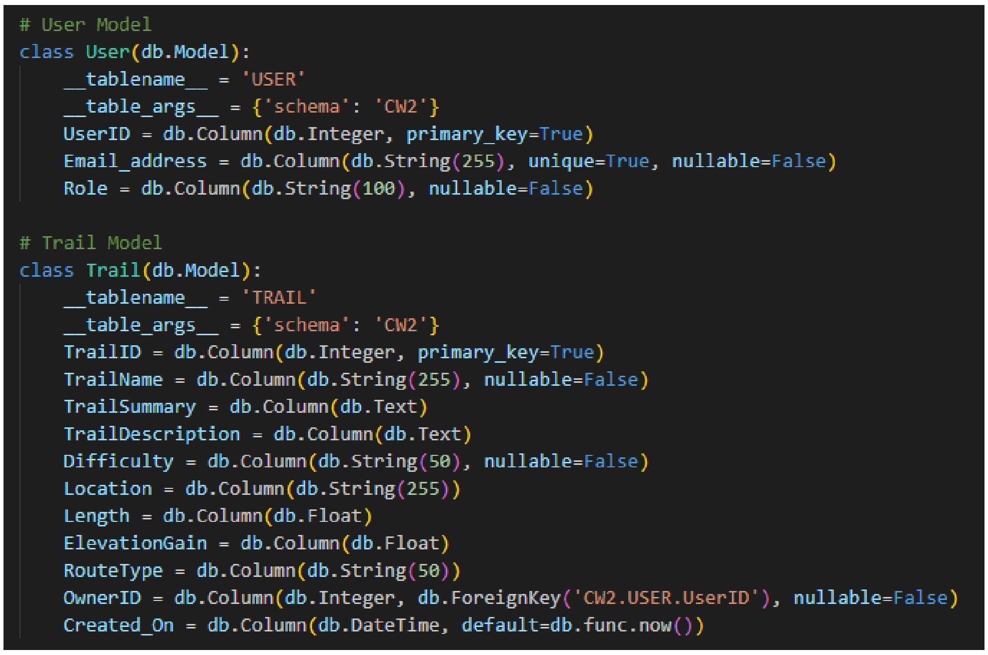
Below is the main.py file that used to be four .py files including config.py, __init__.py, routes.py and

Below it is shown to import all the libraries, and configure the files.



```python
from flask import Flask, request, jsonify
from flask_sqlalchemy import SQLAlchemy
from flask_jwt_extended import JWTManager, create_access_token, jwt_required, get_jwt_identity
from flask_swagger_ui import get_swaggerui_blueprint
from datetime import timedelta
import logging

app = Flask(__name__)

# JWT Configuration
app.config["JWT_SECRET_KEY"] = "secretKey"
app.config["JWT_ACCESS_TOKEN_EXPIRES"] = timedelta(hours=1)
jwt = JWTManager(app)

# Database Configuration
app.config['SQLALCHEMY_DATABASE_URI'] = 'mssql+pyodbc://LAyres:RknT369+@dist-6-505.uopnet.plymouth.ac.uk/COMP2001_LAyres?driver=ODBC+Driver+17+for+SQL+
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)

# Logging Configuration
logging.basicConfig(level=logging.DEBUG)

# Swagger Configuration
SWAGGER_URL = '/swagger'
API_URL = '/static/swagger.json'
swaggerui_blueprint = get_swaggerui_blueprint(SWAGGER_URL, API_URL)
app.register_blueprint(swaggerui_blueprint, url_prefix=SWAGGER_URL)
```

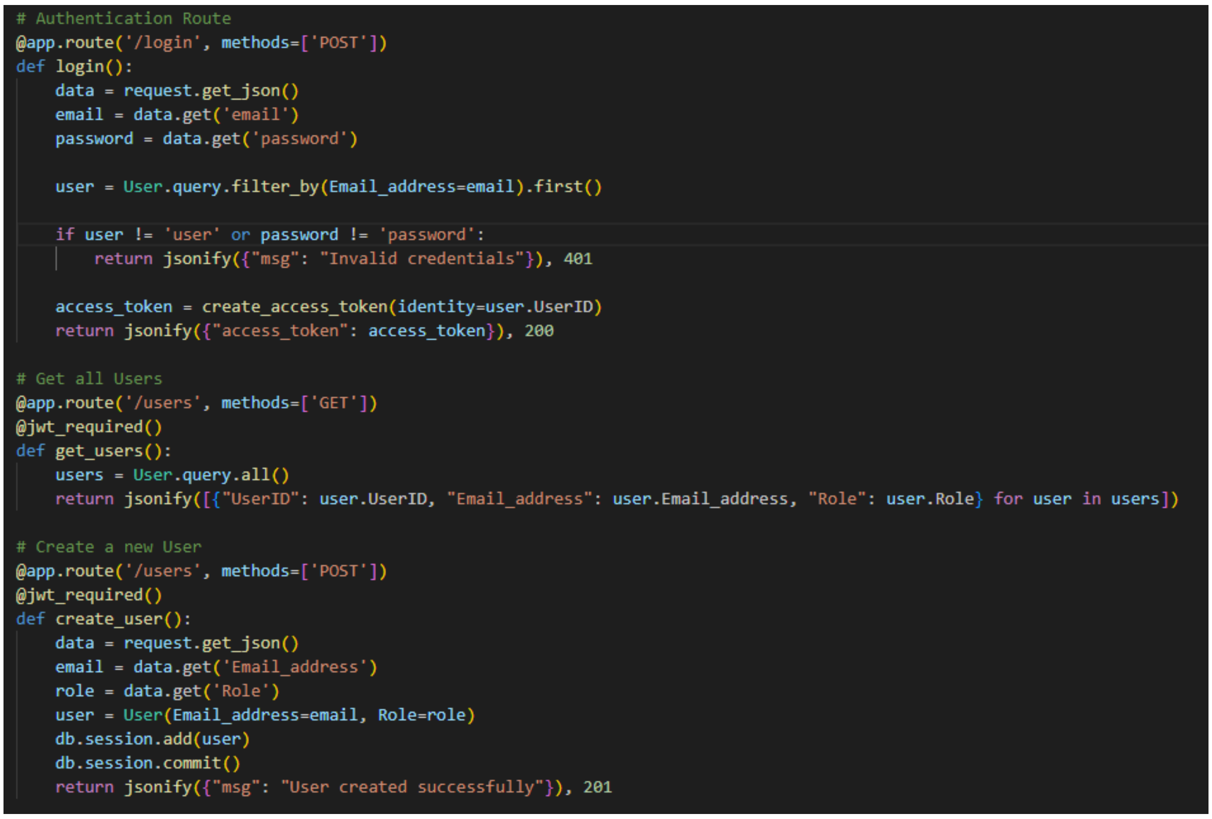
Below is setting the trail and user models with the same data as the tables.



```python
# User Model
class User(db.Model):
    __tablename__ = 'USER'
    __table_args__ = {'schema': 'CW2'}
    UserID = db.Column(db.Integer, primary_key=True)
    Email_address = db.Column(db.String(255), unique=True, nullable=False)
    Role = db.Column(db.String(100), nullable=False)

# Trail Model
class Trail(db.Model):
    __tablename__ = 'TRAIL'
    __table_args__ = {'schema': 'CW2'}
    TrailID = db.Column(db.Integer, primary_key=True)
    TrailName = db.Column(db.String(255), nullable=False)
    TrailSummary = db.Column(db.Text)
    TrailDescription = db.Column(db.Text)
    Difficulty = db.Column(db.String(50), nullable=False)
    Location = db.Column(db.String(255))
    Length = db.Column(db.Float)
    ElevationGain = db.Column(db.Float)
    RouteType = db.Column(db.String(50))
    OwnerID = db.Column(db.Integer, db.ForeignKey('CW2.USER.UserID'), nullable=False)
    Created_On = db.Column(db.DateTime, default=db.func.now())
```

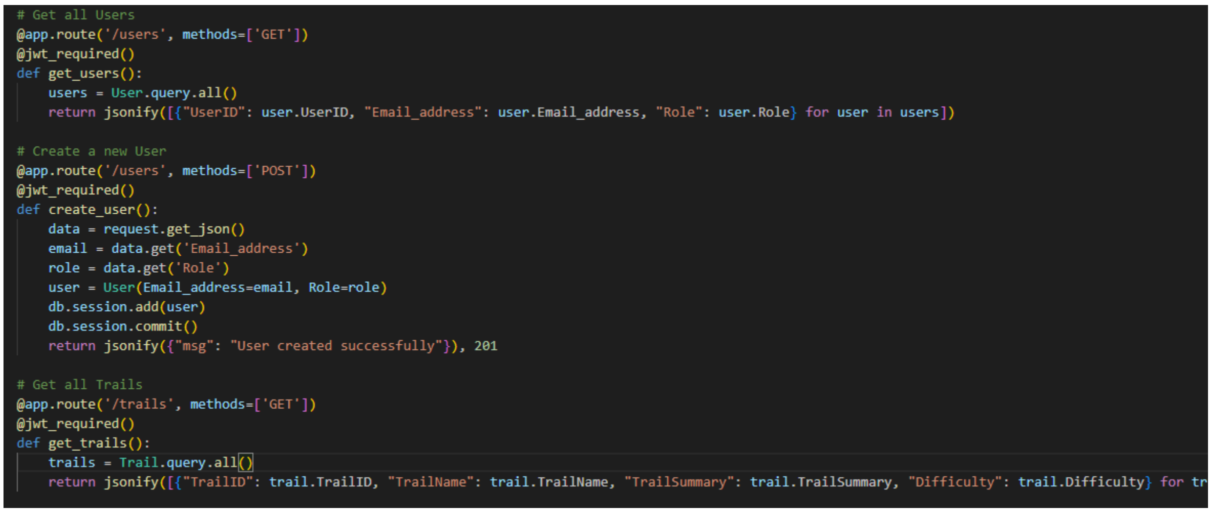
Integration with Authenticator API:
The micro-service was integrated with an existing Authenticator API to manage user authentication. This would ensure that only authorized users could create, update, or delete trails. Tokens were used to secure endpoints and validate user access.

Below is the authentication route, meaning the login screen.
I couldn't get the login screen to work because it kept giving the 'Couldn't get method' error.

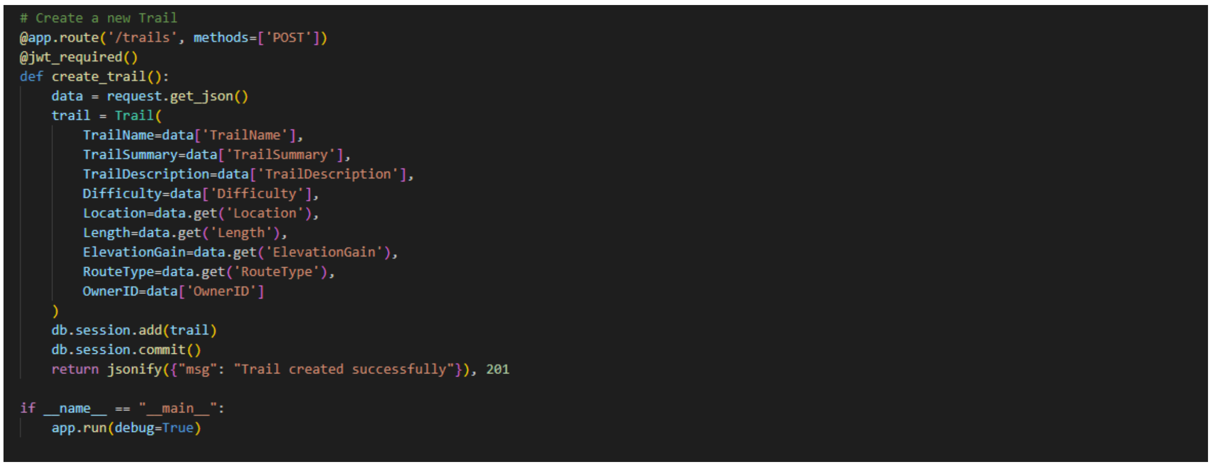


```python
# Authentication Route
@app.route('/login', methods=['POST'])
def login():
    data = request.get_json()
    email = data.get('email')
    password = data.get('password')

    user = User.query.filter_by(Email_address=email).first()

    if user != 'user' or password != 'password':
        return jsonify({"msg": "Invalid credentials"}), 401

    access_token = create_access_token(identity=user.UserID)
    return jsonify({"access_token": access_token}), 200

# Get all Users
@app.route('/users', methods=['GET'])
@jwt_required()
def get_users():
    users = User.query.all()
    return jsonify([{"UserID": user.UserID, "Email_address": user.Email_address, "Role": user.Role} for user in users])

# Create a new User
@app.route('/users', methods=['POST'])
@jwt_required()
def create_user():
    data = request.get_json()
    email = data.get('Email_address')
    role = data.get('Role')
    user = User(Email_address=email, Role=role)
    db.session.add(user)
    db.session.commit()
    return jsonify({"msg": "User created successfully"}), 201
```

These create the new user or trail when the user tries to add one. This can only happen after authentication however to comply with GDPR.

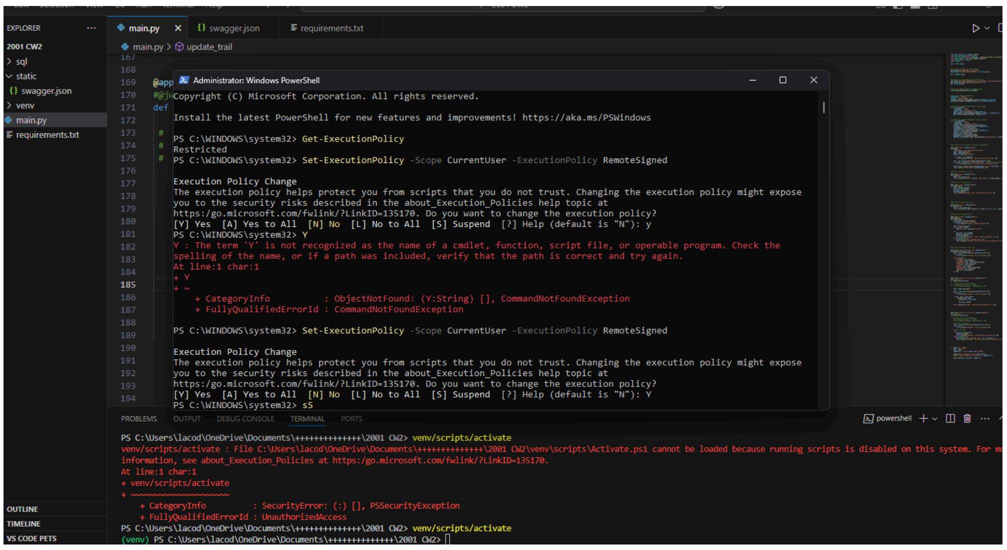


```python
# Get all Users
@app.route('/users', methods=['GET'])
@jwt_required()
def get_users():
    users = User.query.all()
    return jsonify([{"UserID": user.UserID, "Email_address": user.Email_address, "Role": user.Role} for user in users])

# Create a new User
@app.route('/users', methods=['POST'])
@jwt_required()
def create_user():
    data = request.get_json()
    email = data.get('Email_address')
    role = data.get('Role')
    user = User(Email_address=email, Role=role)
    db.session.add(user)
    db.session.commit()
    return jsonify({"msg": "User created successfully"}), 201

# Get all Trails
@app.route('/trails', methods=['GET'])
@jwt_required()
def get_trails():
    trails = Trail.query.all()
    return jsonify([{"TrailID": trail.TrailID, "TrailName": trail.TrailName, "TrailSummary": trail.TrailSummary, "Difficulty": trail.Difficulty} for tr
```



```python
# Create a new Trail
@app.route('/trails', methods=['POST'])
@jwt_required()
def create_trail():
    data = request.get_json()
    trail = Trail(
        TrailName=data['TrailName'],
        TrailSummary=data['TrailSummary'],
        TrailDescription=data['TrailDescription'],
        Difficulty=data['Difficulty'],
        Location=data.get('Location'),
        Length=data.get('Length'),
        ElevationGain=data.get('ElevationGain'),
        RouteType=data.get('RouteType'),
        OwnerID=data['OwnerID']
    )
    db.session.add(trail)
    db.session.commit()
    return jsonify({"msg": "Trail created successfully"}), 201

if __name__ == "__main__":
    app.run(debug=True)
```

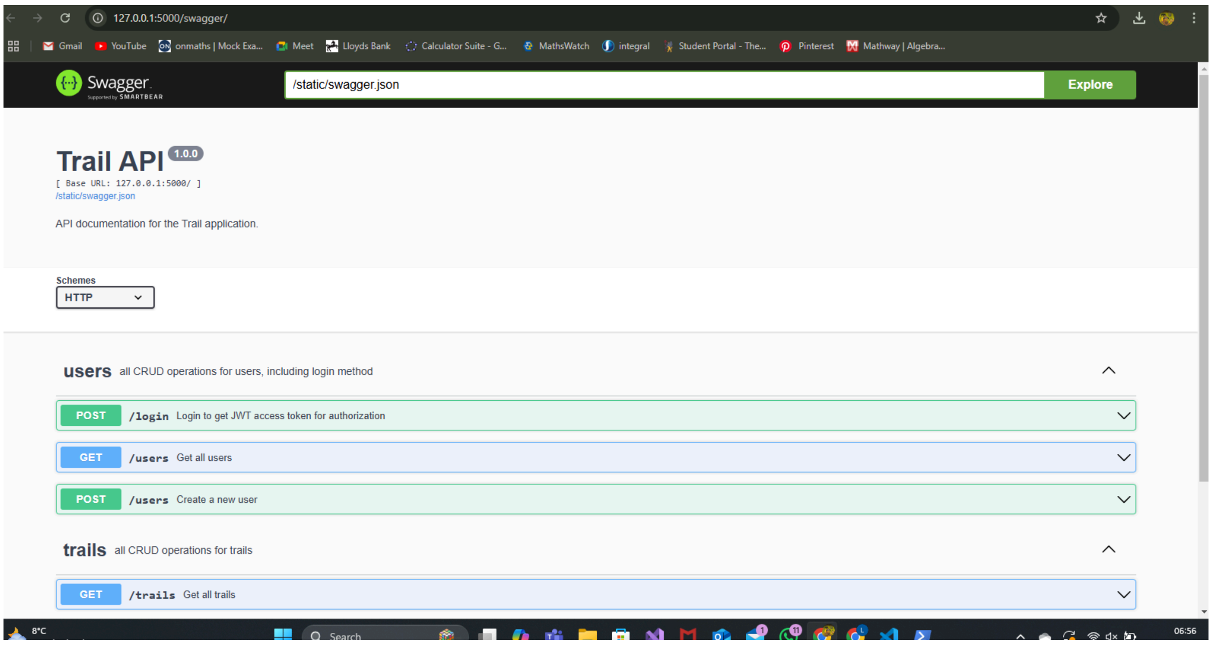
The script wouldn't run because I hadn't been given permission to run it in visual studio so I had to go into powershell and turn on permissions manually.



```
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\WINDOWS\system32> Get-ExecutionPolicy
Restricted
PS C:\WINDOWS\system32> Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned

Execution Policy Change
The execution policy helps protect you from scripts that you do not trust. Changing the execution policy might expose
you to the security risks described in the about_Execution_Policies help topic at
https:/go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the execution policy?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "N"): y
PS C:\WINDOWS\system32> Y
Y : The term 'Y' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the
spelling of the name, or if a path was included, verify that the path is correct and try again.
At line:1 char:1
+ Y
+ ~
    + CategoryInfo          : ObjectNotFound: (Y:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException

PS C:\WINDOWS\system32> Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned

Execution Policy Change
The execution policy helps protect you from scripts that you do not trust. Changing the execution policy might expose
you to the security risks described in the about_Execution_Policies help topic at
https:/go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the execution policy?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "N"): Y
PS C:\WINDOWS\system32> sS
```

```
PS C:\Users\lacod\OneDrive\Documents\++++++++++++++\2001 CW2> venv/scripts/activate
venv/scripts/activate : File C:\Users\lacod\OneDrive\Documents\++++++++++++++\2001 CW2\venv\scripts\Activate.ps1 cannot be loaded because running scripts is disabled on this system. For m
information, see about_Execution_Policies at https:/go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:1
+ venv/scripts/activate
+ ~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : SecurityError: (:) [], PSSecurityException
    + FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Users\lacod\OneDrive\Documents\++++++++++++++\2001 CW2> venv/scripts/activate
(venv) PS C:\Users\lacod\OneDrive\Documents\++++++++++++++\2001 CW2>
```

This was the view when the link was opened although testing didn't go ahead as for reasons stated in my EC i didn't have time.



Trail API 1.0.0

[Base URL: 127.0.0.1:5000/]

/static/swagger.json

API documentation for the Trail application.

Schemes HTTP

users all CRUD operations for users, including login method

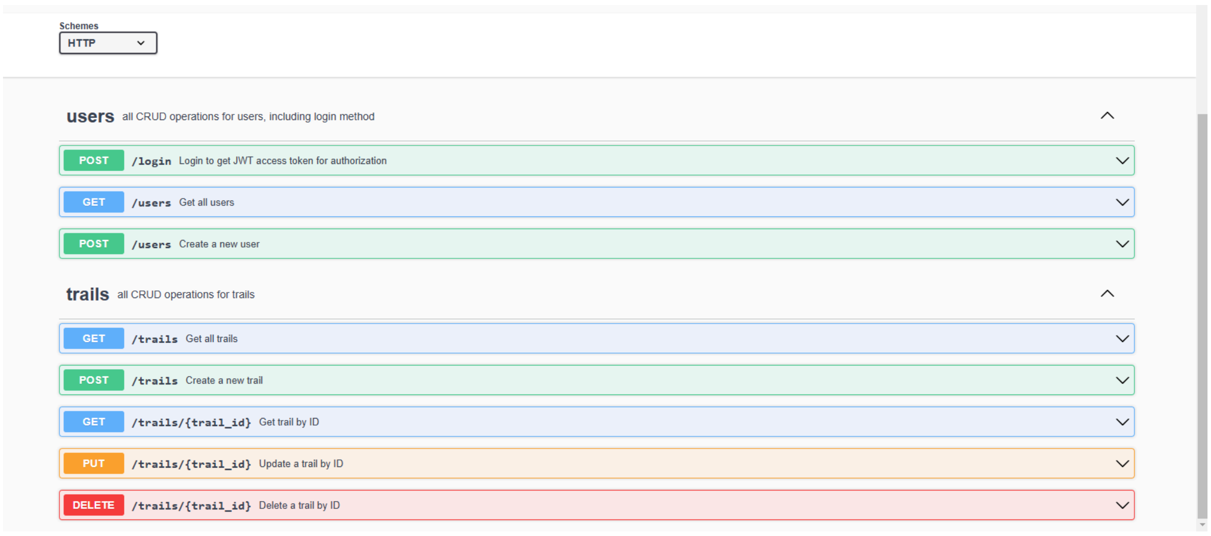
POST `/login` Login to get JWT access token for authorization

GET `/users` Get all users

POST `/users` Create a new user

trails all CRUD operations for trails

GET `/trails` Get all trails



Schemes HTTP

users all CRUD operations for users, including login method

POST `/login` Login to get JWT access token for authorization

GET `/users` Get all users

POST `/users` Create a new user

trails all CRUD operations for trails

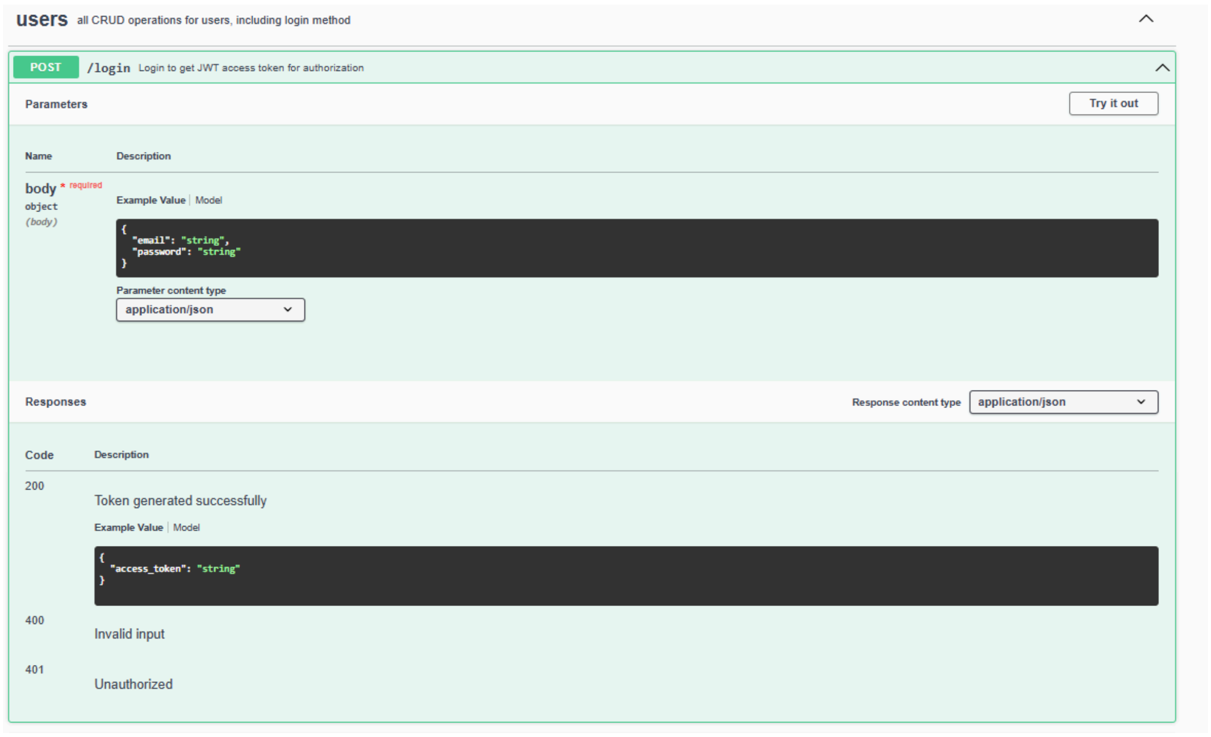
GET `/trails` Get all trails

POST `/trails` Create a new trail

GET `/trails/{trail_id}` Get trail by ID

PUT `/trails/{trail_id}` Update a trail by ID

DELETE `/trails/{trail_id}` Delete a trail by ID



users all CRUD operations for users, including login method

POST `/login` Login to get JWT access token for authorization

Parameters Try it out

Name	Description
body * required object *(body)*	Example Value | Model

```
{
  "email": "string",
  "password": "string"
}
```

Parameter content type: application/json

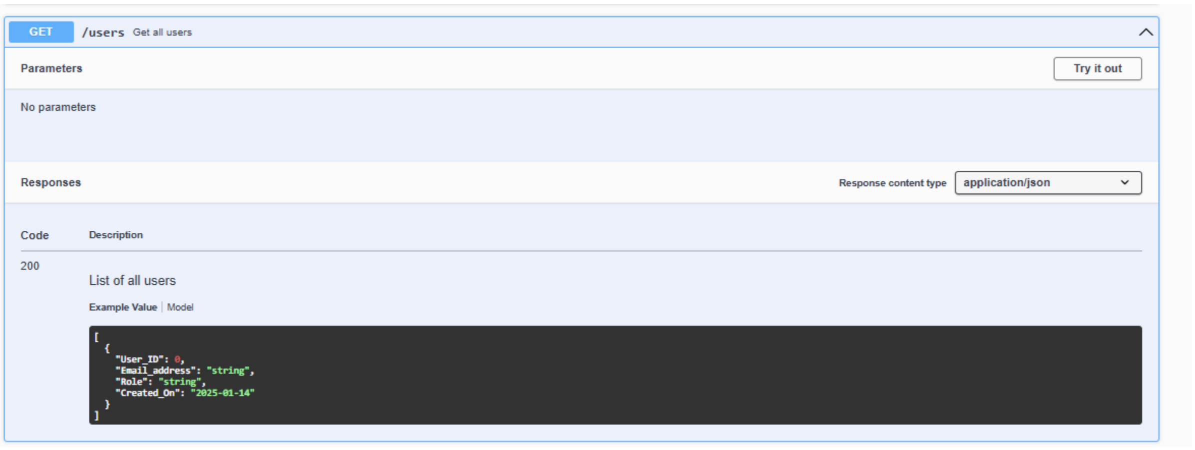
Responses Response content type: application/json

Code	Description
200	Token generated successfully

Example Value | Model

```
{
  "access_token": "string"
}
```

Code	Description
400	Invalid input
401	Unauthorized



GET `/users` Get all users

Parameters Try it out

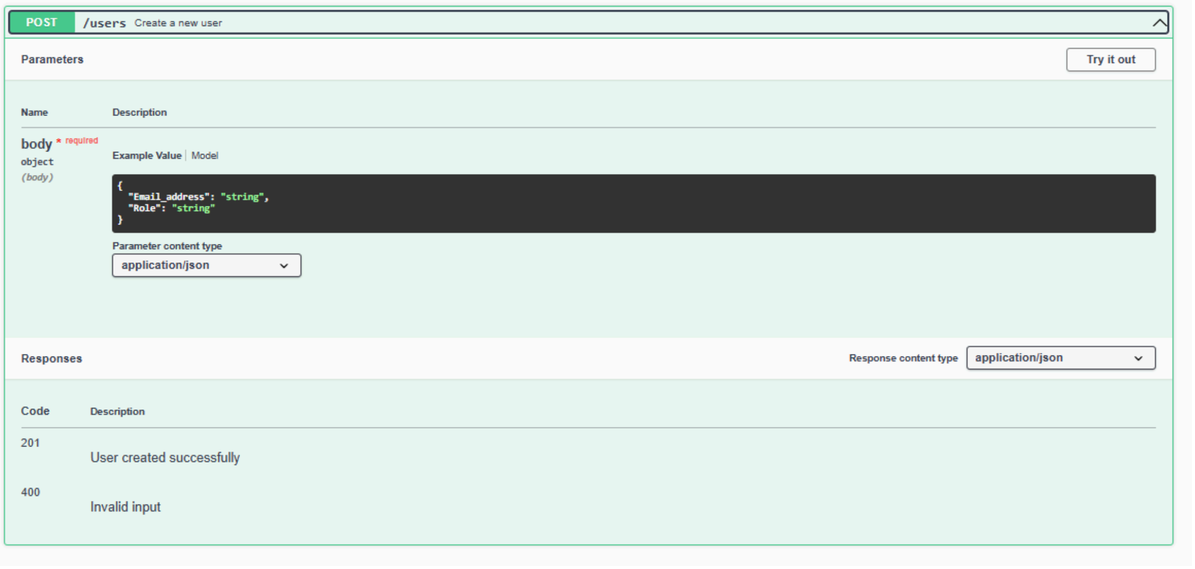
No parameters

Responses Response content type: application/json

Code	Description
200	List of all users

Example Value | Model

```
[
  {
    "User_ID": 0,
    "Email_address": "string",
    "Role": "string",
    "Created_On": "2025-01-14"
  }
]
```



POST `/users` Create a new user

Parameters Try it out

Name	Description
body * required object *(body)*	Example Value | Model

```
{
  "Email_address": "string",
  "Role": "string"
}
```

Parameter content type: application/json

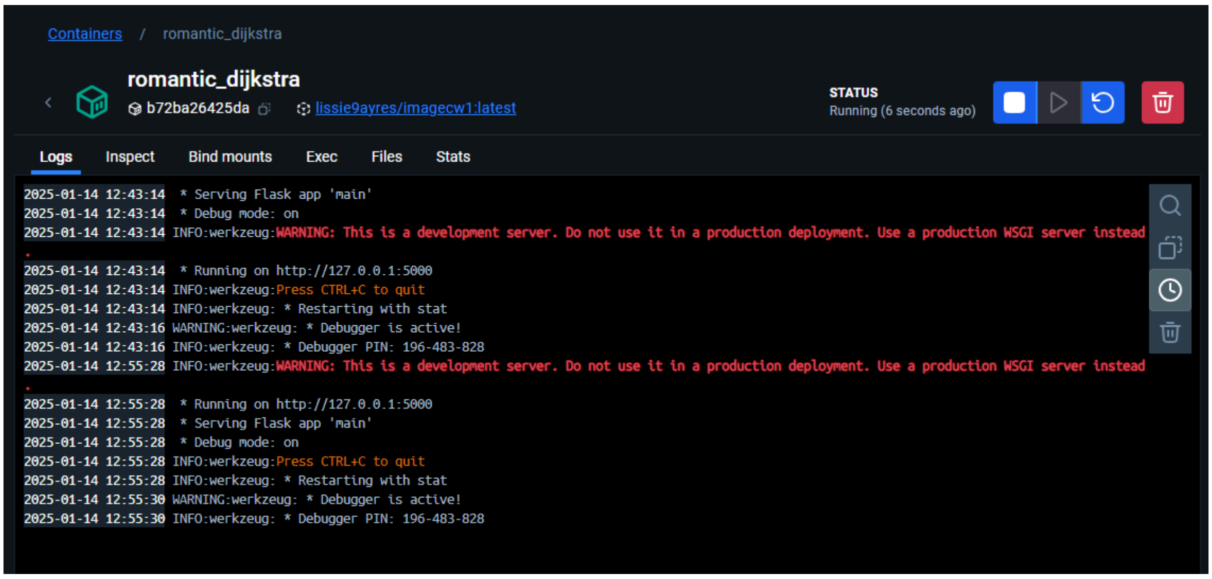
Responses Response content type: application/json

Code	Description
201	User created successfully
400	Invalid input

Docker

The micro-service was containerized using Docker, creating an image (`lissie9ayres/imagecw1:latest`) for easy deployment and scalability. This ensured consistency across development and production environments.

The micro-service was deployed using Docker, making it accessible via a public URL. Sample data was loaded into the database to simulate real-world use, allowing further evaluation of the service.



```
2025-01-14 12:43:14  * Serving Flask app 'main'
2025-01-14 12:43:14  * Debug mode: on
2025-01-14 12:43:14 INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead
.
2025-01-14 12:43:14  * Running on http://127.0.0.1:5000
2025-01-14 12:43:14 INFO:werkzeug:Press CTRL+C to quit
2025-01-14 12:43:14 INFO:werkzeug: * Restarting with stat
2025-01-14 12:43:16 WARNING:werkzeug: * Debugger is active!
2025-01-14 12:43:16 INFO:werkzeug: * Debugger PIN: 196-483-828
2025-01-14 12:55:28 INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead
.
2025-01-14 12:55:28  * Running on http://127.0.0.1:5000
2025-01-14 12:55:28  * Serving Flask app 'main'
2025-01-14 12:55:28  * Debug mode: on
2025-01-14 12:55:28 INFO:werkzeug:Press CTRL+C to quit
2025-01-14 12:55:28 INFO:werkzeug: * Restarting with stat
2025-01-14 12:55:30 WARNING:werkzeug: * Debugger is active!
2025-01-14 12:55:30 INFO:werkzeug: * Debugger PIN: 196-483-828
```

Evaluation

The development of the TrailService microservice demonstrated a strong understanding of database design principles, RESTful API development, and integration with external authentication systems. The service effectively manages CRUD operations for trails and adheres to modern software engineering practices, such as using Docker for containerization and deploying the service in a consistent environment. The use of normalized tables, stored procedures, and modular design ensures data integrity, scalability, and clarity, providing a solid foundation for future enhancements.

However, there are areas for improvement. Testing, a critical phase in any development project, was not fully completed due to time constraints. This limited the ability to validate the service's robustness, handle edge cases, and ensure seamless interaction between components. Comprehensive testing, including unit tests, integration tests, and user acceptance tests, should be prioritized in future iterations to ensure the service meets functional and performance requirements.

Accessibility and functionality are additional areas where the service could improve. While the API supports CRUD operations and integrates with the Authenticator API, future developments could focus on enhancing usability for diverse user groups. Features such as better error handling, detailed response messages, and accessibility-compliant interfaces would significantly improve the user experience. Adding functionality such as advanced search, trail recommendations, or interactive visualizations could also make the application more engaging and useful.

In conclusion, the project successfully delivered a working microservice with a robust foundational design, demonstrating key technical competencies. By addressing the gaps in testing, accessibility, and functionality in future iterations, the TrailService has the potential to become a highly reliable and user-friendly solution for managing well-being trails.

GitHub Link

https://github.com/lissie9a/2001-CW2-Larissa-Ayres

References

Andrew Gordon, D.N. (n.d.). *The Impact of embedding Sustainable Development within the Teaching of Computing*. [online] Hull : Department of Computer Science University of Hull , p.5. Available at: https://d1wqtxts1xzle7.cloudfront.net/46131662/The_Impact_of_embedding_Sustainable_Deve20160601-12306-1nxtlg3-libre.pdf?1464788457=&response-content-disposition=inline%3B+filename%3DThe_Impact_of_embedding_Sustainable_Deve.pdf&Expires=1736867121&Signature=WxLTqyWukGyojW9Ker3qYDf6PGY5M9t80y5LSBmlw6CcGus8zHPj8DUwWkUPSD7nHxc20YcR4~oslHVg7jBiXMerfL~LSOOoc7K1t~h3E71-eBAOtlCaszFFuSzux1Gq-CtB8NEwyE90K9KcVTaOZPIQfB3K0-YUcAjdynVkkJoA3e--9fBjt-4XaCmh2TfwW2BjndL5Y5FweP5fM~nAkkAHc3eA89S1fPoPS5Fb6UzS-PpWqny5gEfHml~Gb1iQ9IJfkYc9WvXm1Lc-q1PhovIuv8WVMOz3MRMS6j~w8q4GtTZf1oO3Vhs888l1AyTv6UzhGd7788StltJgxSjtsA__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA.

Gordon, N. (2014). Sustainable Development as a Framework for Ethics and Skills in Higher Education Computing Courses. *World sustainability series*, pp.345–357. doi:https://doi.org/10.1007/978-3-319-10690-8_24.

Gov.UK (2018). *Data Protection Act*. [online] GOV.UK. Available at: https://www.gov.uk/data-protection.

Sharma, K., Verma, A. and Verma, P. (2023). Analysis and Implementation of Microservices Using Docker. *Communications in Computer and Information Science*, pp.413–421. doi:https://doi.org/10.1007/978-3-031-28180-8_28.