Software Test Document

Testing has been carried out using Postman to send HTTP requests to the API's endpoints.

Before testing, ensure that the latest version of the Docker container is being used (` *docker pull coreyrichardson1/trails-api*` , ` *docker run -p 8000:8000 coreyrichardson1/trails-api*`) and that you are connected to the University of Plymouth network, either directly or through the FortiClient VPN.

Testing was carried out on a fresh, empty instance of the schema using files *Schema\DROP.sql*, *Schema\init.sql* and *Schema\VIEW.sql*.

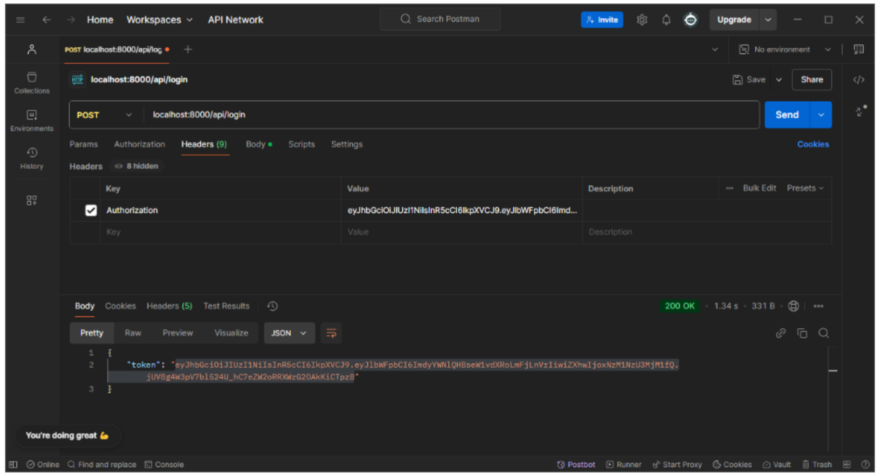
Test	Request Body	Expected Response	Actual Response
Check that frontend displays content as expected.			
http://localhost:8000/	N/A	Ensure *http://localhost:8000/* displays a box with a summary of the application and a navigation link to the Swagger UI page.	As expected.
http://localhost:8000/api/ui/	N/A	Ensure *http://localhost:8000/api/ui/* displays the API documentation, with section tags for Authentication, Features, Points, Trails, Trail-Feature Links and Users.	As expected.
Test that GET/READ endpoints don't require authentication.			
A GET request to */feature* returns OK.	N/A	200 OK []	200 OK []

A GET request to /feature/1 returns NOT FOUND.	N/A	404 NOT FOUND
		{
		"detail": ...,
		"status": 404,
		"title": "Not Found",
		"type": ...
		}
			404 NOT FOUND
			{
			"detail": ...,
			"status": 404,
			"title": "Not Found",
			"type": ...
			}

A GET request to */point* returns OK.	N/A	200 OK	200 OK
A GET request to /point/1 returns NOT FOUND.	N/A	404 NOT FOUND	404 NOT FOUND
A GET request to */trail* returns OK.	N/A	200 OK	200 OK
A GET request to /trail/1 returns NOT FOUND.	N/A	404 NOT FOUND	404 NOT FOUND
A GET request to */trail-feature* returns OK.	N/A	200 OK	200 OK
A GET request to */trail-feature/1/features* returns NOT FOUND.	N/A	404 NOT FOUND	404 NOT FOUND
A GET request to */trail-feature/1/1* returns NOT FOUND.	N/A	404 NOT FOUND	404 NOT FOUND
Test that GET endpoints associated with User objects ARE protected and require authentication.			
A GET request to */user* is blocked.	N/A	401 UNAUTHORIZED	401 UNAUTHORIZED
A GET request to /user/1 returns UNAUTHORIZED.	N/A	401 UNAUTHORIZED	401 UNAUTHORIZED
Test that the */login* endpoint returns a JWT token on POST of valid credentials.			
Authentication highlights missing credentials.	{ "email" : "john.doe@example.com }	400 BAD REQUEST	400 BAD REQUEST
Authentication rejects invalid credentials.	{ "email" : "john.doe@example.com", "password" : "Password123!" }	401 UNAUTHORIZED	401 UNAUTHORIZED
Authentication returns JWT token on post of valid credentials.	{ "email" : "grace@plymouth.ac.uk", "password" : "ISAD123!" }	200 OK	500 INTERNAL SERVER ERROR “AttributeError: module 'jwt' has no attribute 'encode'” Caused by a naming conflict between PIP modules ‘jwt’ and ‘PyJWT’.

Authentication returns JWT token on post of valid credentials.	{ "email" : "grace@plymouth.ac.uk", "password" : "ISAD123!" }	200 OK { "token": "x.y.z" }	200 OK { "token": "x.y.z" }

The authentication endpoint returns a JSON Web Token (JWT) which can be passed into the *Authorization* header of a request to allow access to Create, Update and Delete operations via POST, PUT and DELETE HTTP requests. Using Postman, this header can either be manually set as a Custom Header or it can be provided in the *Authorization* tab as a *Bearer Token*. With the Authorization header set, protected endpoints will now be accessible.



POST localhost:8000/api/login

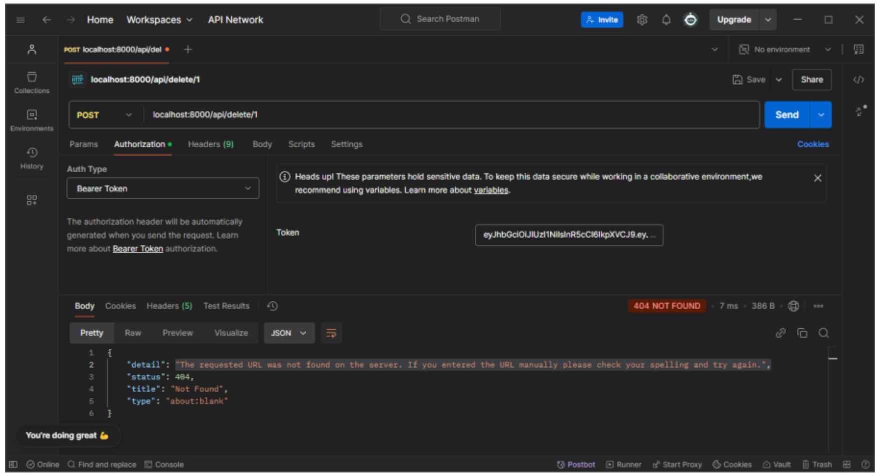
Headers 8 hidden

	Key	Value	Description
✓	Authorization	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6Imd...	

200 OK · 1.34 s · 331 B

```
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6ImdyYWNlQHBseW1vdXRoLmFjLnVrIiwiZXhwIjoxNzM1NzU3MjM1fQ.
        jUV8g4W3pV7bl524U_hC7eZW2oRRXWzG20AkKiCTpz8"
}
```

Figure 1: Custom Authorization Header



POST localhost:8000/api/delete/1

Auth Type: Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about Bearer Token authorization.

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment,we recommend using variables. Learn more about variables.

Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey. ...

404 NOT FOUND · 7 ms · 386 B

```
{
    "detail": "The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.",
    "status": 404,
    "title": "Not Found",
    "type": "about:blank"
}
```

Figure 2: Authorization > Auth Type > Bearer Token

Test	Request Body	Expected Response	Actual Response
Check user was added to the User table during authentication.			
A GET request to */user* shows that “grace@plymouth.ac.uk” is present in the table with the role “ADMIN”.	N/A	200 OK JSON List containing User	200 OK [{ "email": "grace@plymouth.ac.uk", "id": 1, "role": "ADMIN" }]
A GET request to */user/1* returns “grace@plymouth.ac.uk”	N/A	200 OK JSON Object containing User	200 OK { "email": "grace@plymouth.ac.uk", "id": 1, "role": "ADMIN" }
Test Feature Endpoints			
A POST request to */feature* creates a new Feature and adds it to the database.	{ "feature" : "Aeroport" }	201 CREATED	201 CREATED
A PUT request to */feature/1* will update a Feature’s attributes.	{ "feature" : "Airport" }	200 OK	200 OK
A GET request to */feature* returns a single Feature from the database.	N/A	200 OK	200 OK
A DELETE request to */feature/1* deletes a Feature from the database.	N/A	204 NO CONTENT	204 NO CONTENT
A GET request to */feature* returns an empty JSON list.	N/A	200 OK	200 OK

Test Point Endpoints			

A POST request to */point* creates a new Point and adds it to the database.	{
	"latitude": 50.423698,
	"longitude": -4.110593,
	"description": "Point 1"
	}
		201 CREATED	201 CREATED

A POST request to */point* rejects a new Point if the next or previous linked Points don't exist.	{
	"latitude": 50.420222,
	"longitude": -4.099503,
	"description": "Point 2",
	"next_point_id": 100,
	"previous_point_id": 98
	}
		404 NOT FOUND	404 NOT FOUND

A PUT request to */point/1* will update a Point's attributes.	{
	"latitude": 50.424958,
	"longitude": -4.108179
	}
		200 OK	200 OK

A GET request to */point* returns a single Point from the database.	N/A	200 OK	200 OK
A DELETE request to */point/1* deletes a Point from the database.	N/A	204 NO CONTENT	204 NO CONTENT
A GET request to */point* returns an empty JSON List.	N/A	200 OK	200 OK

Test Trail Endpoints			

A POST request to */trail* creates a new Trail and adds it to the database.	{
	  "author_id": 1,
	  "description": "Trail description",
	  "difficulty": "Easy",
	  "elevation_gain": 10,
	  "length": 100.5,
	  "location": "Plymouth, UK",
	  "name": "Trail name",
	  "route_type": "Loop",
	  "summary": "string"
	}
		201 CREATED	201 CREATED

A POST request to */trail* rejects a new Trail if the Starting Point doesn’t exist.	{
	  "author_id": 1,
	  "description": "Trail description",
	  "difficulty": "Easy",
	  "elevation_gain": 10,
	  "length": 100.5,
	  "location": "Plymouth, UK",
	  "name": "Trail name",
	  "route_type": "Loop",
	  "summary": "string",
	  "starting_point_id": 100
	}
		404 NOT FOUND	404 NOT FOUND

A POST request to */trail* rejects a new Trail if the Difficulty isn't in the enumerated type.	{
	  "author_id": 1,
	  "description": "Trail description",
	  "difficulty": "Super Duper Tricky",
	  "elevation_gain": 10,
	  "length": 100.5,
	  "location": "Plymouth, UK",
	  "name": "Trail name",
	  "route_type": "Loop",
	  "summary": "string"
	}
		400 BAD REQUEST	400 BAD REQUEST
			
			'Super Duper Tricky' is not one of ['Easy', 'Moderate', 'Hard'] - 'difficulty'

A POST request to */trail* rejects a new Trail if the Route Type isn't in the enumerated type.	{
	  "author_id": 1,
	  "description": "Trail description",
	  "difficulty": "Easy",
	  "elevation_gain": 10,
	  "length": 100.5,
	  "location": "Plymouth, UK",
	  "name": "Trail name",
	  "route_type": "Circular",
	  "summary": "string"
	}
		400 BAD REQUEST	400 BAD REQUEST
			
			'Circular' is not one of ['Loop', 'Out & back', 'Point to point'] - 'route_type'

A PUT request to */trail/1* will update a Trail's attributes.	{
	  "summary": "Updated Trail Summary"
	}
		200 OK	200 OK

A GET request to */trail* returns a single Trail from the database.	N/A	200 OK	200 OK
A DELETE request to */trail/1* deletes a Trail from the database.	N/A	204 NO CONTENT	204 NO CONTENT
A GET request to */trail* returns an empty JSON List.	N/A	200 OK	200 OK

Test Trail-Feature Link Endpoints			

Create a temporary Trail using a POST request to */trail* to use to test the Trail-Feature link.	{
	"author_id": 1,
	"description": "There and Back Again",
	"difficulty": "Hard",
	"elevation_gain": 3500,
	"length": 950,
	"location": "Middle Earth",
	"name": "A Quest to Slay a Dragon",
	"route_type": "Out & back",
	"summary": "Trail Summary"
	}
		201 CREATED	401 UNAUTHORIZED (Token expired!)
			
			201 CREATED
			
			Returned ID: 2

Create a temporary Feature using a POST request to */feature* to use to test the Trail-Feature link. #1	{
	"feature": "Trolls"
	}
		201 CREATED	201 CREATED
			
			Returned ID: 3

Create a temporary Feature using a POST request to */feature* to use to test the Trail-Feature link. #2	{
	"feature": "Dragons"
	}
		201 CREATED	201 CREATED
			
			Returned ID: 4

A POST request to */trail-feature* creates a link between a Trail and a Feature. #1	{
	"trail_id": 2,
	"feature_id": 3
	}
		201 CREATED	201 CREATED

A PUT request to */trail-feature/2/3* updates a link between a Trail and a Feature	{
	"feature_id": 4
	}
		200 OK	200 OK

A POST request to */trail-feature* creates a link between a Trail and a Feature. #2	{
	"trail_id": 2,
	"feature_id": 3
	}
		201 CREATED	201 CREATED

A GET request to */trail-feature/2/features* returns a JSON list with two *Features*.	N/A	200 OK	200 OK

A DELETE request to */trail-feature/2/3* removes a link between a Trail and a Feature.	N/A	204 NO CONTENT	204 NO CONTENT
A GET request to */trail-feature* returns a JSON list with a single Trail-Feature Link.	N/A	200 OK	200 OK
A DELETE request to */trail-feature/2/4* removes a link between a Trail and a Feature.	N/A	204 NO CONTENT	204 NO CONTENT
Delete the temporary Trail using a DELETE request to */trail/2*.	N/A	204 NO CONTENT	204 NO CONTENT
Delete a temporary Feature using a DELETE request to */feature/3*.	N/A	204 NO CONTENT	204 NO CONTENT
Delete a temporary Feature using a DELETE request to */feature/4*.	N/A	204 NO CONTENT	204 NO CONTENT

Test User Endpoints			

A POST request to */user* creates a new User and adds them to the database.	{
	"email": "john.doe@example.com",
	"role": "USER"
	}
		201 CREATED	201 CREATED

A PUT request to */user/3* will reject an update on a User's attributes if the role isn't in the enumerated type.	{
	"role": "ADMINISTRATOR"
	}
		400 BAD REQUEST	400 BAD REQUEST

A PUT request to */user/3* will update a User's attributes.	{
	"role": "ADMIN"
	}
		200 OK	200 OK

A GET request to */user/3* returns a single User from the database.	N/A	200 OK	200 OK
A DELETE request to */user/3* deletes a User from the database.	N/A	204 NO CONTENT	204 NO CONTENT
A GET request to */user* returns a single User: "grace@plymouth.ac.uk".	N/A	200 OK	200 OK

Test the Trail Creation Workflow			
Create a series of Points using POST requests to */point*.	{ "latitude": 50.423698, "longitude": -4.110593, "description": "Point 1" }	200 OK	200 OK Returned ID: 3
	{ "latitude": 50.424958, "longitude": -4.108179, "description": "Point 2" }	200 OK	200 OK Returned ID: 4
	{ "latitude": 50.420222, "longitude": -4.099503, "description": "Point 3" }	200 OK	200 OK Returned ID: 5
	{ "latitude": 50.422134, "longitude": -4.113191, "description": "Point 4" }	200 OK	200 OK Returned ID: 6
	{ "latitude": 50.424262, "longitude": -4.109453, "description": "Point 5" }	200 OK	200 OK Returned ID: 7

Link a series of Points using PUT requests to */point/{point_id}*.	*/point/3*
	
	{
	  "next_point_id": 4,
	  "previous_point_id": 7
	}
		200 OK	200 OK

	/point/4
	
	{
	  "next_point_id": 5,
	  "previous_point_id": 3
	}
		200 OK	200 OK

	/point/5
	
	{
	  "next_point_id": 6,
	  "previous_point_id": 4
	}
		200 OK	200 OK

	/point/6
	
	{
	  "next_point_id": 7,
	  "previous_point_id": 5
	}
		200 OK	200 OK

	/point/7
	
	{
	  "next_point_id": 3,
	  "previous_point_id": 6
	}
		200 OK	200 OK

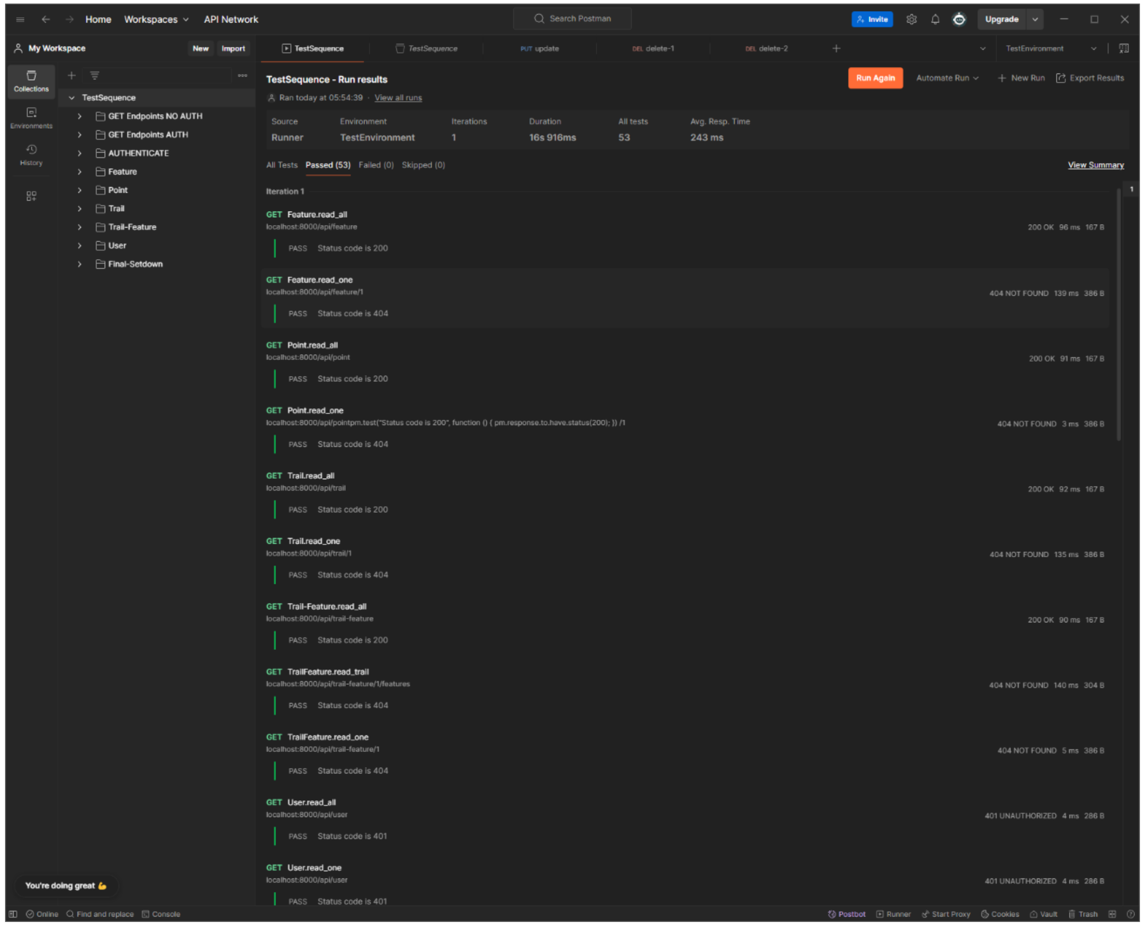
Create a Trail using a POST request to */trail* using Point 3 as the *starting_point_id*. This trail is a Loop.	{
	"author_id": 1,
	"starting_point_id": 3,
	"name": "Plymouth Airport Runway",
	"summary": "A walk that follows Plymouth Airports runway",
	"description": "This trail follows the Plymouth airport runway. Not technically legal to walk this one.",
	"difficulty": "Easy",
	"location": "Plymouth, UK",
	"length": 5.0,
	"elevation_gain": 1,
	"route_type": "Loop"
	}
		201 CREATED	201 CREATED
			
			Returned ID: 3

Create a Feature using a POST request to */feature*.	{
	"feature": "Runway Walk"
	}
		201 CREATED	201 CREATED
			
			Returned ID: 5

Link the created Trail and Feature using a POST request to */trail-feature*.	{
	"trail_id": 3,
	"feature_id": 5
	}
		201 CREATED	201 CREATED

Send a GET request to */trail-feature/3/features* to confirm the link has been created.	N/A	200 OK	200 OK
			
			[
			{
			"feature": "Runway Walk",
			"id": 5
			}
			]

I also utilised Postman's *Collection* feature to create a Test Sequence which will run through a predefined series of HTTP requests and compare the results against a set response status code. This allows for faster testing improving the efficiency at which new features could be added. This test Collection can be seen in my GitHub repository at **/Testing/TestSequence.postman_collection.json** and the exported results can be viewed at **/Testing/TestSequence.postman_test_run.json**.



TestSequence - Run results

Ran today at 05:54:39 · View all runs

Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	TestEnvironment	1	16s 916ms	53	243 ms

All Tests   Passed (53)   Failed (0)   Skipped (0)

Iteration 1

```
GET Feature.read_all
localhost:8000/api/feature                                              200 OK  96 ms  167 B
    PASS  Status code is 200

GET Feature.read_one
localhost:8000/api/feature/1                                            404 NOT FOUND  139 ms  386 B
    PASS  Status code is 404

GET Point.read_all
localhost:8000/api/point                                                200 OK  91 ms  167 B
    PASS  Status code is 200

GET Point.read_one
localhost:8000/api/pointpm.test("Status code is 200", function () { pm.response.to.have.status(200); }) /1    404 NOT FOUND  3 ms  386 B
    PASS  Status code is 404

GET Trail.read_all
localhost:8000/api/trail                                                200 OK  92 ms  167 B
    PASS  Status code is 200

GET Trail.read_one
localhost:8000/api/trail/1                                              404 NOT FOUND  135 ms  386 B
    PASS  Status code is 404

GET Trail-Feature.read_all
localhost:8000/api/trail-feature                                        200 OK  90 ms  167 B
    PASS  Status code is 200

GET TrailFeature.read_trail
localhost:8000/api/trail-feature/1/features                             404 NOT FOUND  140 ms  304 B
    PASS  Status code is 404

GET TrailFeature.read_one
localhost:8000/api/trail-feature/1                                      404 NOT FOUND  5 ms  386 B
    PASS  Status code is 404

GET User.read_all
localhost:8000/api/user                                                 401 UNAUTHORIZED  4 ms  286 B
    PASS  Status code is 401

GET User.read_one
localhost:8000/api/user                                                 401 UNAUTHORIZED  4 ms  286 B
    PASS  Status code is 401
```

Figure 3: Results of running a Postman Test Collection