CS2003 Practical 4 report

Tutor: David Morrison

Matriculation ID: 230016906

2023-11-18

Overview

In this practical we were asked to create an API in express which parses in movie data from a static JSON file, stores it within a store that is provided to us using asynchronous operations, create endpoints that validate and process data from both the user and the given movie data and write Jest unit tests which comprehensively test the whole API using the 'supertest' package to send http requests and evaluate the response received back.

Design

R1.1

To read in the data from supplied JSON file I have used the createReadStream() method from the pred-defined fs module to start a stream and read in each part of JSON. While doing this however, I ran into the problem of attempting to use asynchronous code within a "forEach" loop. This created many problems as the order of the data stored in the media store was not in the same order as the data in the JSON file, this meant that every time the server restarted, a lot of my tests were failing as they are checking for hardcoded data that should be present but was obviously in the wrong order. To overcome this problem, I used a basic for loop which can handle asynchronous operations perfectly normally.

Apart from that, to maintain my code I made it modular by extracting the validation checks method as that is used in multiple places.

R1.2

To return all the data from the media store, there is conveniently a "retrieveAll()" method which I call from within the "/media" endpoint. The alternative would be to manually loop through each id in the media store and use the "retrieve" method to get each object from the media store, this is however incredibly inefficient as with each call the media store it takes a

variable amount of time to complete, it is by far more efficient to do this once rather than for each element.

R1.3

To get a specific bit of data from the media store it would make sense to use the "retrieve" method as we will always know the id of the movie that we are looking for. After getting the movie from the store I manually appended the start part: "/media/" to the id and returned it.

R1.4

To create a new movie, I have used the "create" method within the media store, after saving the movie the correct error handling is implemented.

R1.5

To update a new movie, I have used the "update" method and reused the same validation method to ensure that the updated data is clean.

R1.6

To delete a movie. I used the "delete" method, but first checked to ensure that the object first exists within the store, this is useful to later determine if the response's status code should be 404 or 500.

R1.7

To implement pagination within this API, I have chosen to loop through from the offset to the offset + limit and manually get each movie. I went this option as the alternative was to retrieve every single movie in the store and then parse through that to extract the required data. Although the latter approach would be more efficient if the overall number of movies remained small, under the impression that there will be many more movies added to the store it becomes more efficient to only request the required ones.

R1.8

To implement querying, I have used a flag for each query and used that to check the results I get without querying to decide if the object needs to be filtered out or not.

There are 2 steps in this entire process:

- Get all the objects from the store normally, keeping in account the pagination.

- Filter out the objects that match the query and push them onto the finalResults array.

It would be more efficient to filter out the objects while getting them, but this negates the effect of the pagination, to have the functionality of both pagination and querying this is the only way.

R1.9

I have used Jest and supertest to test my whole application.

R1.10

For validation I have initialized a validTransfer method which validates that both the source and target are present. The endpoint parses off the id from the source and sends a request to the target endpoint using the "axios" package which enables me to make http requests from a backend express app and finally deletes the movie from its own store.

Testing

For local testing (testing while developing) I downloaded the software: Postman which sends API requests to any endpoint and helps to structure the response JSON. As mentioned before, I have used Jest and supertest for automation testing in this project. I have two testing files; one tests each endpoint to ensure that they function accordingly and return the correct errors when required. "errors.test.js" file enables errorMode on the media store which ensures that the actual store always fails allowing me to test that each endpoint can return a 500 error. Here is the testing table:

index.test.js

Get /media

Name of test	Input	Output	Pass/Fail
Should be able to call the endpoint.		Status code 200	PASS

Should be paginated when offset is 0.	Pass in an offset of 0 as a query parameter.	Should return the whole media store with status code of 200.	PASS
Should work normally.	Pass in a normal limit and offset as query parameters.	Should return the 5 movies with a status code of 200.	PASS
Should return null for previous when offset is 0.	Pass in an offset of 0 as a query parameter.	Should return a previous of null and status code 200.	PASS
Should return null for next if the limit is too high.	Pass in a ridiculously large limit as a query parameter.	Should return a next of null and status code 200.	PASS
Should return a 204 when result is empty.	Pass in a non-existent query parameter.	Should return a status code of 204.	PASS
Should return all the movies of type 'DVD'.	Pass in a type of DVD as a query parameter.	Should return an array of all the DVD's in the store.	PASS
Should return a specific movie by name.	Pass in a specific name of a movie that is inside of the store as a query parameter.	Should return all the information about the movie with that name.	PASS
Should be able to return the movie from a partial description.	Pass in a partial description of a movie as a query parameter.	Should return all the information about that movie.	PASS

GET /movie/id

Name of test	Input	Output	Pass/Fail
Should return a specific movie by using the id as a parameter.	The id of a movie.	Should return the information about the movie with that id.	PASS
Should return a 404 if movie doesn't exist	An id that doesn't exist.	Should throw a 404 not found.	PASS

POST /media

Name of test	Input	Output	Pass/Fail
Should accept new data and save it.	A valid movie object to add.		PASS
Should return a 201 when created.	A valid movie object to add.		PASS

PUT /media

Name of test	Input	Output	Pass/Fail
Should update the media object in the store.	A valid media object.		PASS
Should return a 200 if successfull	A valid media object.	Status code of 200	PASS
Should return a 400 with incorrect input data.	A non-valid media object.	Status code of 400	PASS

DELETE /media/id

Name of test	Input	Output	Pass/Fail
Should delete the media object with the specified id.	The id of the movie you want to remove from store.		PASS
Should return a 404 if movie doesn't exist.	The id of a movie that does not exist.	Status code of 404	PASS

errors.test.js

GET /media

Name of test	Input	Output	Pass/Fail
Should return a 500 without any query parameters.		A status code of 500.	PASS
Should return a 500 with query parameters.	Send in random query parameters.	A status code of 500.	PASS

POST /media

Name of test	Input	Output	Pass/Fail
Should return a 500		A status code of 500	PASS

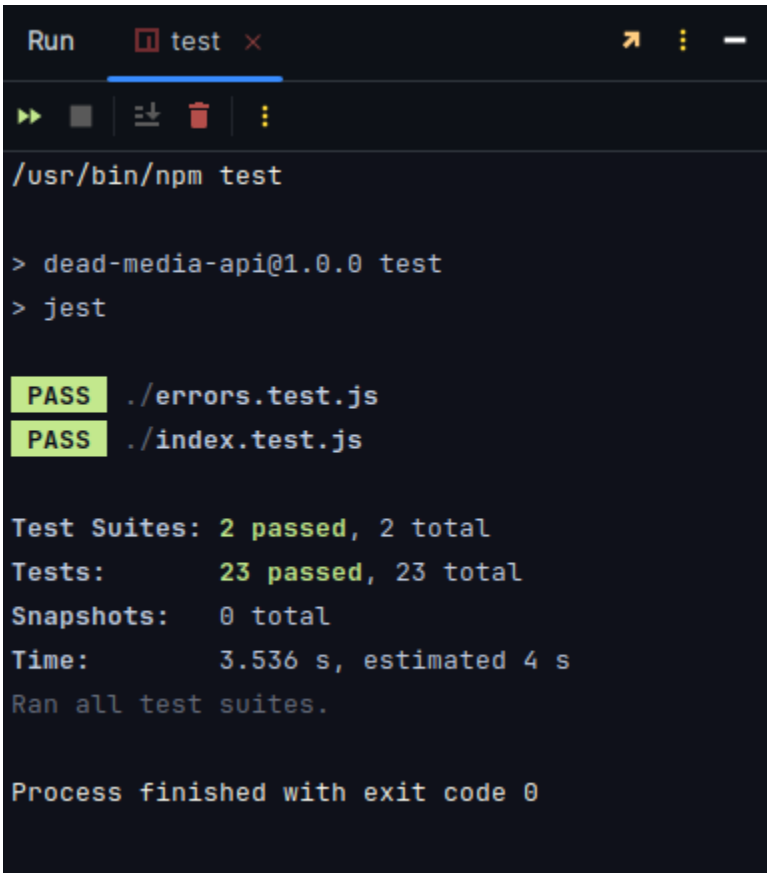
PUT /media

Name of test	Input	Output	Pass/Fail
Should return a 500	A valid movie object to be updated.	A status code of 500.	PASS

DELETE /media/id

Name of test	Input	Output	Pass/Fail
Should return a 500	A valid id of a movie in the store.	A status code if 500.	PASS

Terminal output:



Evaluation

My application meets all the requirements and has sufficient testing to prove it, I have successfully created an application that is able to listen to multiple endpoints, parse data from local JSON files, validate and send data.

Conclusion

Overall, during this practical there were many challenges the most prominent one being asynchronous programming, which took a while to get the hang of, but over the course of this practical I have grown to appreciate JavaScript's way of handling asynchronous behavior. If I had more time, I would go back and validate more using regex.

Bibliography

Axios: https://github.com/axios/axios