Introduction

This Points of Interest application that I had to develop for this course was a very interesting project, fun at time and frustrating at other times, but always pushing me to learn and understand more. I have chosen to use the SQL database provided. My development environment consisted of the WAMP package, that allowed me easy access to my MySql database and a server, and Visual Studio Code, one of the most popular code editors. My main folder contained two separate folders for the backend and the frontend respectively, the node modules, my package.json file and my .env file. The point of access for the API was the server.js file and for my frontend, the index.html file.

Tasks

Part A – The simple API

Task 1

Displaying all the points of interest in a region

For this task I installed express and created a basic server. The entry to my application is the server.js file found in the backend folder. I also installed mysql2 and I created a db.js file where I connected to my SQL database into which I had previously imported the provided SQL database.

A screenshot of a computer

Description automatically generated with medium confidence

Fig. 1. Connecting to the database

Then, I created a simple GET endpoint that was looking for all the points of interest in the database, making sure to return the results as JSON. I tested it in the browser and I got the desired information.

Text

Description automatically generated

Fig. 2. Controller for finding all points

A screenshot of a computer

Description automatically generated with medium confidence

Fig. 3. Model for finding all points in the database

Task 2

Adding a new point of interest

I created another endpoint with a POST verb that created a new point of interest in the database. I used Postman to be able to send the data to the endpoint and test if it worked correctly.

A screenshot of a computer

Description automatically generated

Fig. 4. Creating a new point of interest controller

Text

Description automatically generated

Fig. 5. Model for creating a new point of interest

A screenshot of a computer

Description automatically generated with medium confidence

Fig. 6. Testing new point creation with Postman

Task 3

Recommending a point of interest

In order to recommend a point of interest, I accessed the recommendations field in the database and increased it by one.

Text

Description automatically generated

Fig. 7. Controller for recommending a point of interest

Text

Description automatically generated

Fig. 8. Updating a recommendation in the points of interest model

**Part B – The AJAX-based Javascript front-end**

Task 4

A HTML page allowing to search points of interest by region

In my frontend folder I created a simple HTML page called index.html and an index.js file that I connected to the HTML page. I also connected it to Bootstrap through a link to a CDN. I also created a simple navbar that contained links to the Home page and the other eventual pages. In the body of the HTML I created a simple input with the id “region”, and a button next to it with the id of “regBtn”. I also created a table that contained only the headers, mapped to the information I was about to get from the points of interest database. I left the tbody of the table empty and gave it an id of “results”.

Text

Description automatically generated

Fig. 9. Empty HTMLtable to be populated with fetched data

In the index.js file I attached a click event to the “regBtn” button, that would call a function named getByRegion with the value of the input “region” as an argument.

Text

Description automatically generated

Fig. 10. Attaching an event listener to the search button

Then, I created the getByRegion() function that would make a request to the backend using the fetch API to the endpoint [http://localhost:3000/points/${region}](http://localhost:3000/points/$%7bregion%7d). I parsed the data obtained from there with a “for” function so that I could dynamically create td’s containing the information and needed and attach them to a table row that I finally attached to my table body.

Text

Description automatically generated

Fig. 11. Fetching points of interest data and attaching it to the table

Task 5

The HTML page for creating points of interest

I created another HTML page, named it createPOI, and connected it to the home page through the navigation top bar. I created a form containing fields for all the data that was needed in the points of interest database and connected it to “createForm” function in the index.js file through the onSubmit attribute.

Text

Description automatically generated

Fig. 12. The form for creating a new point of interest

Then, in the index.js file I created the “createForm” function. First, it prevents the form from submitting, so that I was able to gather the data entered and use in our own fetch call to our backend. When making the call, I made sure that the method was set to “POST” and that the collected data was converted to a JSON string through the JSON.stringify() method. The data was sent to the server and a new entry was created in the database.

Text

Description automatically generated

Fig. 13. Sending a call to the backend to create a point of interest

The function waited for a positive or negative answer from the server and displayed it dynamically in the browser, for the user to see.

Task 6

Adding recommend button to the fetched results

In order to add a “recommend” button on each of the results returned by searching by region, I nested a fetch call to the [http://localhost:3000/points/recommend/${item["ID"]}](http://localhost:3000/points/recommend/$%7bitem%5b%22ID%22%5d%7d) endpoint. Inside the getByRegion() function, as every row was being created by the for loop using the results from the first fetch call, I dynamically added to it a button and to every button a click event. Thus, every button could make a fetch call based on the id of the row in the database. The method of this call was PUT, as it would update the recommend field by increasing it by one. Unfortunately, this method did not allow me to show the results in real time, the used would have to refresh the page to see that the recommendations field was updated. I did however let the user know that the action was successful by a message in the browser.

A screenshot of a computer

Description automatically generated with medium confidence

Fig. 14. Adding a fetch call to recommendations on every button

Part C - Errors

Task 7

Adding error-checking

Error checking is done in the Points of Interest API, in the controllers. If there are no points of interest that match the user query, the server sends a message to the client together with a status code.

Text

Description automatically generated

Fig. 15. Checking for errors

This message is picked up by the displayMessage() function. In order to make displaying the response on the browser easier, I created a helper function called “displayMessage()” that would take the JSON message from the server, format it and add it to the HTML dynamically, also making it disappear after a number of seconds.

Text

Description automatically generated

Fig. 16. Function for displaying messages from the server

Part D – Adding a map

Task 8

Adding a map with Leaflet

Using the Leaflet library to add a map turned out to be fairly easy. I initialized the map and centred it on the UK. I also created a layer group so that I could add or remove markers and popups whenever I needed.

Text

Description automatically generated

Fig. 17. Initializing the map

Then, in the “getByRegion()” function, in the “for” loop, I added markers using the latitude and longitude fetched from the database. To the markers I bound a popup which had its content set with the name and description of the point of interest. I added the markers to the markerslayer and the layer to the map. I used a markerslayer so I could remove the markers at every fresh reload of the map.

A screenshot of a computer

Description automatically generated with medium confidence

Fig. 18. Adding markers and popups

Task 9

Adding a Point of Interest by clicking on the map

I created a new function in the index.js file called “createPOI()”. In this function, I initialized a new popup class and I captured the coordinates on the map through the click event. I also copied the form from the “createPOI.html” file in its entirety, setting on it the coordinates returned from the click event. I then set the popup to open on a click on the map, at the set coordinates and displaying the form.

A screenshot of a computer

Description automatically generated with medium confidence

Fig. 19. Adding a form to create a point of interest directly on the map

Pard E – Logins and sessions

Task 10

Implementing a session-based login system

I created a simple login form on a new login.html page that can be accessed from the navbar.

Text

Description automatically generated

Fig. 20. HTML poage for the user login form

The form was submitted through a “loginForm()” function found in the index.js file that use the same pattern of sending the date to the backend through the fetch API and returning a response to the client.

Text

Description automatically generated

Fig. 21. Sending user data to the backend

In the backend, the information was received in a controller that checked if the details entered were found in the database. If they were found, it created a session in the database with a unique id, and the response would set a cookie in the browser so that the user remained authenticated for 60 minutes or until it logged out. The logout function signs out the user by setting the session to null and clearing the cookie.

Text

Description automatically generated

Fig. 22. Controller and router for authentication

In order to be able to record sessions, I had to install the express-session and express-mysql-session npm packages and then initialize and configure the session in the server.js file, using a secret to encode the id and setting the lifespan of the session to one hour.

Text

Description automatically generated

Fig. 23. Session configuration

This is the part of my project that I struggled with the most. In my development environment, I was serving the backend from a php server and the frontend from the VSC server, both from the localhost. This created CORS errors that would not permit my cookies to be set. I managed to resolve the problem by explicitly setting the CORS options in the server.js file:

Text

Description automatically generated

Fig. 24. Setting up CORS

I also had to set the Access-Control-Allow-Origin on every request, to which purpose I created a piece of middleware:

Text

Description automatically generated

Fig. 25. Setting the headers for CORS

With all these in place, there still is a CORS error if the frontend is server from 127.0.0.1 instead of from “localhost”.

Task 11

Adding authorisation

After implementing authentication with sessions and cookies, I moved on to authorisation. This was done easily by creating a piece of middleware that would only allow users to create a point of interest if the username was found on the session object when a request was sent.

A screenshot of a computer

Description automatically generated with medium confidence

Fig. 26. Middleware for authorisation

This middleware was then used in the server.js file in the app.use() calls to the other endpoints.

Graphical user interface, text, application

Description automatically generated

Fig. 27. Using authorisation middelware

**Part F – Implementing a review system**

Task 12

Adding review in the backend

The review system was easy to implement as I just followed the same patterns used to create the points of interest system. I created the models, controllers and routes for the reviews and then then hooked the controllers to the app, remembering to use the authorisation middleware.

A screenshot of a computer

Description automatically generated with medium confidence

Fig. 28. Review controllers

A screenshot of a computer

Description automatically generated with medium confidence

Fig. 29. Reviews models

A screenshot of a computer

Description automatically generated with medium confidence

Fig. 30. Reviews routes



Fig. 31. Connecting the reviews controller to the route

Task 13

Allow reviews to be entered through the map

I had to make some changes in the index.js file to accommodate the display of the reviews in the popups. The data for the review was collected through a form and was sent to the backend when the “submitReview()” function was triggered on submitting the form. This followed the same pattern of working with the fetch API.

Text

Description automatically generated

Fig. 32. Submitting data for the review

Part G – Improving the application

Creating a well-structured node application

I have used the MVC model in my application, as can be seen by the folder structure. The flow of information can be easily followed; the user hits an enpoint and the express app connects it to the controller. The controller makes requests to the database and sends responses to the client, making sure to handle any errors and send server statutes and messages to the client. The models are organized as classes that take the connection as a parameter and creates the appropriate queries.

Using middleware

I have used two middleware functions in the app and they can be found in the middleware folder. One of them handles the headers sent to the client in order to manage the CORS and the other is used for authorizing a logged in user to create points of interest and reviews.

Using Passport

I have used installed the npm passport package and successfully used it for a more robust and supple user authentication.

Text

Description automatically generated

Fig. 33. Authentication using passport

Conclusion

The Points of Interests project has taught me a lot about building an API and fetching data from it with a client site. I found the frontend part very easy to build and the Leaflet library very accessible, with a good documentation. I learned a lot about the request and response cycle and especially about how sessions and cookies are managed. Particularly useful was learning how to build a DAO class as a model and how to query the database more easily. I also understood how crucial middleware is for creating a robust app and how much it can simplify our workload.