Report of Boldizsar Banfia

WEB APPLICATION DEVELOPMENT (COM518)

Part A: REST API

1. The code follows the Model-View-Controller (MVC) design pattern, with a DAO serving as the Model, a controller defining application logic as the Controller, and a router defining HTTP routes as the View. This pattern separates concerns into distinct components, improving modularity and maintainability.

The getPoiByRegion() method in the DAO uses a simple SQL query to fetch all POIs in the application that match a specific region, returning an array of objects. The Controller implements the business logic of the application and uses the req and res parameters to handle the HTTP request and response.

The function extracts the region parameter from the request object and uses the apiGetPoiByRegion() method in the controller to retrieve data from the database and send it back as a JSON response with an appropriate status. The Controller handles errors and acts as an intermediary between the client and the DAO, ensuring valid and consistent responses. The poiRouter.get() method in the router maps the /region endpoint to the apiGetAllPoi() method in the controller, using dynamic parameters. The handler function is bound to the poiControl object using the bind method.

1. Same three main components.

The DAO takes all the parameters of the POI, that will be inserted into the table. Notable difference here, instead of sql.all(), sql.run() is used that returns an object that represents the result of the query execution, which includes information about the number of rows affected or any errors that occurred. Its is also important to note that this parameterized query approach is the best practice to follow (Fadlallah 2022) to prevent SQL injection attacks. In this type, the SQL statement is precompiled with placeholders for the input values, and the input values are then passed separately to the database engine, rather than being directly concatenated into the SQL statement.

The controller extracts the POI information from the request parameters, validates that the user is logged in, which will later be important and all required information are present. Then calls the postPoi method of a DAO object to add the new POI to the database. If successful, the function returns a 201 response with a success message, otherwise, it returns an error response.

The endpoint URL is specified with placeholders for the POI information, which are then extracted from the URL parameters by the controller function. Notable difference aside from the amount of variables, is the .post method, which defines the HTTP request, and indicates that information will be uploaded to the server.

Note that this will go obsolete since it does not return the created id of the row, which will be key at the last task.

1. The purpose of the DAO is to update a row in a database table with a given ID, by incrementing the recommendations field by 1.

Controller has the identical purpose with only one parameter this time.

Router is likewise having the same syntax.

Throughout the application the same principles were followed including ECMAScript 6 that enabled a more modular application along with all DAOs, controllers and routes. By using .mjs file extension the import and export of functions facilitate the overall transparency and re-usability of the application. The db.mjs very well exemplify that, instead of having to initiate the database in every file, a simple import and  the database is ready to be used. The whole MVC stack architectural pattern was by module imports on server-side part. Route modules define a set of routes that redirects the incoming requests to the corresponding route, this allows better organization  . Unfortunately, the same modularity at the front-end could not be reached due to software limitations and lack of React-DOOM library due using solely NCD and babel transpiler result in incompatibility with ECMAScript 6 modules (Ihechikara Vincent Abba 2021). React uses something called ES6 that falls out of the scope of this module.

Part B: Ajax Front-end

1. The client side was implemented with React. Task four takes place on the index page. The whole front-end code is wrapped around a React function, then createRoot method creates a root-level component for a React application. The document.getElementById('root') expression retrieves the HTML element with the ID "root", which is an empty container. Babel used to convert JSX  to JavaScript and render it onto the page.The HTML components in this code provide a user interface for searching Points of Interest (POIs) by region. The interface includes an input field for entering the region name, and a search button for initiating the search. When the user clicks the search button, a React event listener is triggered  which calls the corresponding search function. The function retrieves POI data from the server based on the input region name and renders the results to the DOM.The function first retrieves the region name, this is used as a parameter in an HTTP GET request to the server, using the fetch API.The server response is then processed using a promise chain. It reads the response from the server and notifies the user if an issue occurred. Otherwise, the response data is parsed as JSON using the json method, and then passed to the setPoi function , which updates the state of the poi array with the retrieved POI data. If the array is not empty, the data will be presented to the user in a professional way with additional features. The searchByRegion() function checks for any errors, and displays the appropriate massage to the user, the function will be further enhanced at task 13.
2. Note, the hyper link is only becoming visible if the user logged in. Due to later task, even tough it would not post data without the user logged in, to make it professional the link is hidden. Similar structure to the index file, it returns a JSX element that renders the relevant input fields to the page, that are made to look professional with added CSS. When submit button clicked, react on click triggered and calls the function. The postPoi function handles the submission of the form data to the server. The function first retrieves the values entered in the input fields and performs error checking to ensure that all required fields are filled in. If any required fields are missing, the function highlights the missing fields in red and displays an error message listing the missing fields. The function then sends a POST request using the fetch API. If the request is successful, the function displays a success message and redirects the user back to the homepage. If the request fails, the function displays an error message.
3. Upon search for region, when the data is returned, an additional line of code is added  that onclick, calls the recommend function that grabs the id of the chosen POI, and with a fetch API sends a post requests. The back end will increment the referred recommendation of the POI. Server-side error handling deemed sufficient due to the lack of delete function, the POI id will always exist. The user will be notified of the success of the post request.

Part C: Error checking

A screenshot of a computer

Description automatically generated

These error handling methods are already mentioned, highlighted on the picture, all the existence of variables are checked in the server side and the client side too, Text

Description automatically generated plus sending the outcome of the request may that be error or success, which is being interpreted in the front-end side. Text

Description automatically generated

If the response is not successful, an appropriate error message is displayed in an alert, and the input box associated with the error is highlighted by changing its border color to red. Finally, the function returns the response data in JSON format. This code block demonstrates error handling and user feedback.

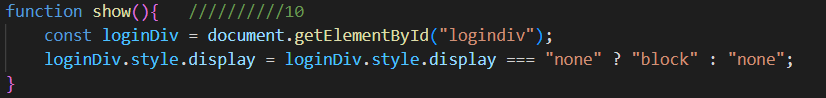
Part D: Adding a map.

1. For the map to only initialize once, we use effect. Inside the useEffect hook, the
2. L.map function is used to create a new map instance, and a tile layer is added to the map using OpenStreetMap data. The map is then centered on a specific location using the setView method. Using react prevent the usage of global variables, unless the State used, which proved “slow” in many instances. One of the main challenges was to circumvent this as the application advanced. Key variables for latitude and longitude were initialized to be available for sub functions. Within the map click event listener, on click onto the map (not on a marker) will populate those two variables and place a marker to that place with the pre – created form containing the rest of the POI details.  . As instructed the marker only stays on place if post fetch API was successful, otherwise it will be removed from the map . It is achieved by a simple if statement Text

   Description automatically generated, assigning the true value to the dependent variable in a case of successful response from the server Text

   Description automatically generated . Once all the input value was collected, it checks for blank fields, front and back-end error checking added to avoid errors occurring. Value validation could have been added with a more through error checking. This function was further advanced with image upload capability, more on that later. As instructed, if the post request was successful the marker was bind with the name and description of the POI, if no picture was added.
3. The session middleware is set up by using two modules in this framework. Express session creates and retrieves session cookies automatically, and stores session data in a session store . The session data is stored in a table called express\_session as a JSON object with the session ID and session variables. The middleware is then configured with various options, such as a secret for signing session cookies , a maximum age for the session cookie , and options for resaving the session and not saving uninitialized sessions.

The login function is defined as a callback function that is called when the user submits their login credentials. It extracts the username and password from the form input and sends an HTTP POST request to the /user/login endpoint with the login details in the request body instead of HTTP parameter. The DAO and controller serve a very similar purpose, it checks the value of input against the database, little difference here if the data does exist, then the username will be saved as session variable  . This is, as mentioned before used to prevent post request without user logged in. If the response status code is 200, indicating a successful login, an alert message is displayed, and the page is reloaded. Note that this is the only artificial reload, needed for the HTML tags to appear in react. The response data is also passed to a callback function that was passed in as an argument. If the response status code is 401, indicating incorrect credentials the user also notified. If status code has any other value, the user gets notified with the error code. As mentioned, the HTML tags are only visible when the user logged in, achieved this with Graphical user interface, text

Description automatically generated react, moreover the username and password input at login also disappears by click  . When the user logged in, the log in <li> is hidden, instead the Logout <li> is visible and when clicked, it calls the logout function.

The function sends an HTTP POST request to the /user/logout endpoint when called which calls destroys the session variable Text

Description automatically generated . If the response status code is 200, indicating a successful logout, an alert message is displayed, and the page is reloaded. If the response status code is any other value, indicating an error, an alert message is displayed. Within the effect hook, another fetch API Text

Description automatically generated with medium confidence is checking if Text

Description automatically generated the user session is empty or the user is logged in, and populates the data .If there is no  session variable, it will go back to displaying the login.

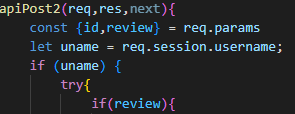
1. Each post requested constructed in a way that if there is no session variable, ergo the user is not logged in, Text

   Description automatically generated there will be an alert notifying the user to do so. Another middleware function that intercepts incoming POST requests to the "poiRouter" and checks if there is a user logged in by verifying the existence of a "username" value in the "session" object of the request.

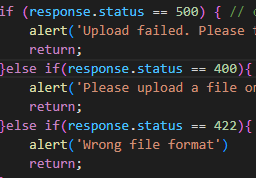
If there is no logged-in user (i.e., the "username" value is null or undefined), the function sends an HTTP response with a 401 Unauthorized status code and a JSON object containing an error message A screenshot of a computer

Description automatically generated with medium confidence.

1. The API, server code including the separate DAO, Controller, and route the same with a small difference in the SQL query. Since the display of any review is had not been defined hence the latest review being displayed upon clicking on the marker. For that when querying the database, yet all the result will be returned,  the last one first, and when displaying  the very first one, ergo the most recent will be displayed. In terms of error handling and exact same precautions were taken along with preventing post request when the user is not logged in. Last slight difference in the two similar DAO insert and controller post request, is the retrieval of the id variable, once it happens via parameter once requesting from the body. The reason for that is, since the whole functionality could not have been written as a separate function due to lack of global variables, it had to be included upon the creation of the markers when searching for region, plus when the take me there function called searching for POI and clicking on marker. When placing all markers onto the map, it was easier to push data via HTTP parameters, whereas when clicking on the name of one particular poi, it was feasible to push data via a form data requesting it from the body. Error checking is in place in both case  of client side and server side too Text

   Description automatically generated .
2. Upon clicking on any marker on the map, allows the user to leave a review about that poi, simply by clicking on the add review button. It is achieved to by adding a text area to popping up marker and hiding it  unless the add review button clicked  . The revController is responsible for sending response and status back to the front-end, including all conventional already mentioned approaches, including preventing post request in case of lacking logged in user or empty value  .
3. React used throughout the assessment for the client-side part.

Part H: Improvements:

* Some CSS applied to make it look more professional.
* ECMAScript 6 classes used in server side, well-structured server side.
* Separate Controllers, DAOs and Router files used.
* Easily modifiable host, port, and database path in the .env file.
* Custom middleware files are demonstrated in the file upload facility.
* File upload facility implemented: (example in Portsmouth!!)
  + For the upload the module express – file upload was used.
  + Due to time limit the DAO, controller and route was not separated.
  + The function takes place in the effect hook, when clicking on the map, when creating a POI. After all the details of POI was input, and the response is 201, the marker is bind to another div, created to upload the picture. There is an option to opt out, in that case the marker is bind with conventional details such us name and description. Otherwise, the file before gets uploaded, goes through some checking. First at the front-end the file gets named after the id of the POI  with its original extracted  extension. Then it is uploaded with a fetch API to the server. Before it is saved it goes through the middleware  that checks whether the file exist at all, and sends the appropriate response  . Then checks the extension of the file. Note that the middleware was written in a way that it can be customized when it is called . Once the allowed extensions provided it loops through them and checks the file against it  . If it is not the right format  it sends the respective response back and  the client side acts in accordance.
  + Last limitation imposed on the file is the actual size  that is initialized in the app.mjs and the value is declared in the .env file.
  + If goes through all checks the marker is bind with the picture.
  + Next step is crucial for image retrieval. Took a while to contrive a way, to be able to retrieve an image per POI. Even though the actual file is named after the POI id, the extension could have been three different, hence it could not have been retrieved. Tried to rename the file with the extension upon upload but that hindered the extension middleware. So after the middleware checked, it retrieves the file again with a fetch API, then rename the extension to a conventional .jpeg format, still leaving the initial file intact, but now retrievable when clicking on any marker.
  + Last step was to add the img tag when the region search was called 

This instant finds the picture if there is any. Note that there is no image displayed if there is an error (404). Hence on clicking the marker will result in a 404 get request in the console, so it is designed that way!!!