Ismail Kefokeris

Assignment Description

I am tasked with developing a website named PlacesToStay, the site should allow users to lookup information on places they may want to stay whilst away, such as hotels, bed and breakfasts, and hostels. The website should make use of Node and Express as the back end, and MySQL for the database.

COM518 - AE1

**Contents**

[**Task 1** 2](#_Toc101954012)

[**Task 2** 4](#_Toc101954013)

[**Task 3** 5](#_Toc101954014)

[**Task 4** 7](#_Toc101954015)

[**Task 5** 10](#_Toc101954016)

[**Task 6** 11](#_Toc101954017)

[**Task 7** 11](#_Toc101954018)

[**Task 8** 16](#_Toc101954019)

[**Task 9** 17](#_Toc101954020)

[**Task 10** 18](#_Toc101954021)

[**Task 11** 18](#_Toc101954022)

[**Task 12 – Part G** 18](#_Toc101954023)

[**Database Changes** 18](#_Toc101954024)

# **Task 1**

**Complete: Yes**

For task 1 I needed to implement an Express route to find all accommodations in the given location. All routes in the API are in separate folders and for this task my route will root point will be “/acc” which I specify to be used in “app.js”.

A screenshot of a computer

Description automatically generated with medium confidence

The first tasks route will be “/acc/search/:location” (where location is the param that will be used to query the database). This will be an Express GET route as its retrieving data from the server.



When the route is accessed by a device connected to the API it will call the Accommodation Controller (accController) which handles the logic and communicating with the DAO (Database) – keeping all the messy MySQL code away in one file. It is also vital when we call the “findAccByLocation” function it is connecting back to itself using “bind” so to preserve the context of the “this” object.

Text

Description automatically generated

Located inside the Controller we can find the async function “findAccByLocation” which take two parameters “req” and “res”, this allows the method to act like a route handler and complete tasks such as render or send back a status. The controller will (inside a try…catch statement to avoid errors) call the DAO’s function “findAccByLocation”.

A screenshot of a computer

Description automatically generated with medium confidence

The DAO will receive the request and return a new promise which allows us to not worry about call-backs and allowing us to write sequential code using async/await. The promise will start of by querying the database, if an error were to occur the function would reject the promise, otherwise the function will check whether the query has returned any results and depending on the answer to that it would either send back nothing to the controller or the information it has.

Text

Description automatically generated

Coming back to the controller after the DAO is complete, the controller will check whether anything was sent back from the DAO and if nothing was sent back the controller would pick this up and return both a user-friendly message back to the user and a HTTP status code of 404. Otherwise, the API will return the results.

# **Task 2**

**Complete: Yes**

For task 2, I must find all accommodation of a specific type in a specific location, the root point of this tasks route will also be “/acc” and the full route will be “/acc/search/:type/:location”, where type and location would be changed out for actual values.



Consistent with task 1 this route will be an Express GET route which will call the Accommodation Controller (accController), the controller will call a function which allows for not only location but type to be looked at “findAccByLocationType”.

Text

Description automatically generated

This function works identically to the function called in task 1, inside the try…catch statement the controller will call the DAO’s function “findAccByLocationType” while passing it the two parameters retrieved from the URL.

Text

Description automatically generated

The DAO will query the database and handle the results respectively as explained in task 1.

# **Task 3**

**Complete: Yes**

For Task 3 I am tasked with inputting data into the database rather than retrieving it, I need to create a system to book a place at any given accommodation. The API must receive the accommodation ID, Number of People, The Date, and username. The username was not part of the original question but when looking through the database there was a column for username, and it seemed best to add it and keep track of who booked what place.



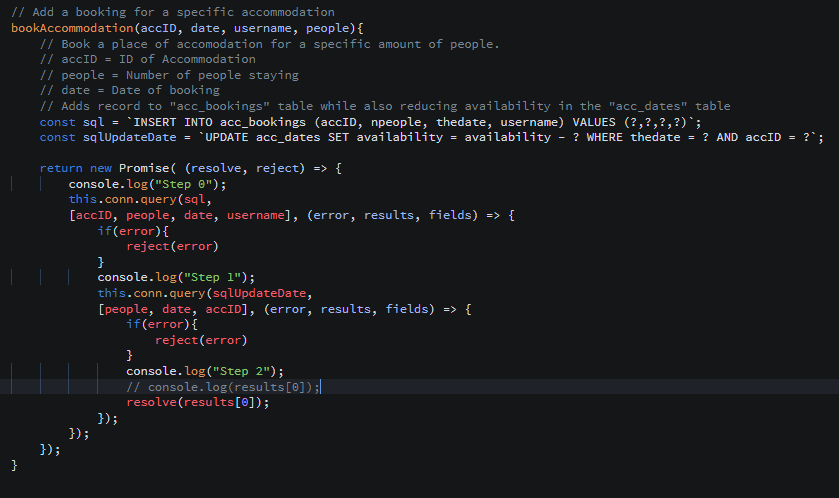
Similar to task 1 and task 2 the root of my route will be “/acc” making the full route “/acc/book/:accID/:numP/:date/:user”. This route has a total of four parameters accID being the accommodation ID, numP being the Number of People, date being the Date booked, user being the username. To differ from tasks 1 and 2 we are creating an Express POST route as we need to send data and create and update listings in the database.

The post route like task 1 and task 2 routes uses a controller, the controller its using is specific to booking called the bookController which calls the function “bookAccommodation” and again binding it to itself.

Text

Description automatically generated

The controller first does a simple check ensuring none of the parameters that were provided via the URL are null, once this passes we can open up a try…catch statement and call the DAO’s “bookAccommodation” function passing through our variables.



The DAO “bookAccommodation” function starts first just like every function would in a DAO by returning a new Promise, within the promise the database is queried to first insert into the table “acc\_bookings” the accommodation ID, number of people, the date, and username. The “acc\_bookings” table will keep track of all bookings made through the website, after this query is run and complete the next query runs (still within the promise) which is an update query to update the “acc\_dates” table. The second query will reduce the availability from the “acc\_dates” table by only the amount of people that are staying.

# **Task 4**

**Complete: Yes**

Task 4 stated that I must now create a way for the user to interact with the API created in tasks 1-3, to complete this task I have decided to use a templating system known as EJS.

Text

Description automatically generated Chart, scatter chart

Description automatically generated

The user has the option to either enter only a location or both the Type of accommodation and Location, once entered and they hit Search (The Submit button of the form) a background script will run.

Text

Description automatically generatedThe script starts by retrieving the values from the text boxes of both “location” and “type”. A check will be performed ensuring a location has been provided and if not nothing will happen, afterwards there is another check to see if both location and type were provided, if both were provided we would run the function with both parameters filled otherwise only location would be filled.

A screenshot of a computer screen

Description automatically generated

The “accSearch” function takes two parameters, a location and a type but the type is by default set to null as the default way for searching for a place to stay is through location. The function defines a few variables at the start “accommodationsHTML” being the body for the table that all results are going to be placed into once collected, “messageLocation” being a place to relay any success or error messages, and finally “accommodationRow” which passes through the variable accommodation to fill in the template JavaScript, this will be different for each result found and will be added as a row in the table.

Text

Description automatically generated

Graphical user interface, application

Description automatically generatedThe next step is to check whether type was provided or not, if it was provided the function will fetch the appropriate link to search for accommodation with the correct type in the correct location, it will then run through the very same error check before turning the response once again into json (accInfo). The following check checks whether the fetch returns anything for the parameters provided, if so the function will iterate through each accommodation returned and fill it into the template before pushing it to a list which will then be joined later on and sent to the innerHTML of the element the function found at the start.

A screenshot of a computer

Description automatically generated

Text

Description automatically generated

In the scenario in which the user may have only provided a location this snippet of code would run, everything that runs is identical to the previous block of code we went through except the response is fetching a different URL which allows for location only search.

# **Task 5**

**Complete: Yes**

For task 5 I had to display a book button for each accommodation result provided from task 4.

Text

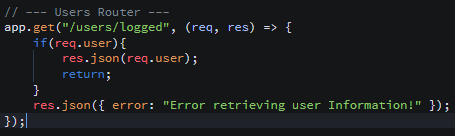
Description automatically generated

In the last task we used this template to render out the accommodation found, in the template there is an “a” tag which is used as the book button, it has a name which is “acc-(ID of associated accommodation)” and an id, once this button is clicked on the home page an AJAX script will send a POST request to the REST API from task 3.

Text

Description automatically generated

The script is always listening for clicks but will only register clicks on Elements with the tagName “a” once it finds one it will separate the name at the separator “-“and leave us with the id which will can call the booking function (with hard codded values).

A screenshot of a computer

Description automatically generated with medium confidence

The function starts of by retrieving the user information which tests whether the user is logged in or not, this is an additional test to what happens server side when the user tries to access the bookings route.



Then the function does a quick check ensuring it has received information back from the fetch.

If this goes through okay without any issues the response will be turned into json before setting the username to a variable in the JavaScript to be later used.

A screenshot of a computer

Description automatically generated with medium confidence

Once the program has the username it can now commence the post request to the url from task 3 which if successful will return a user-friendly message and have booked the specified date with the specified user and number of people.

# **Task 6**

**Complete: Yes**

Shown in other tasks.

# **Task 7**

**Complete: Yes**

For Task 7 I was required to add an OpenStreetMap map to the application using Leaflet and it must communicate with Task 4 showing up on the map markers of places to stay when searched.



A screenshot of a computer

Description automatically generated with medium confidence

The map is added to the home page through leaflet, and as I have had problems getting it to work in the past I kept both the text based results and map results using a button to toggle the view between the two.

Text

Description automatically generated

The script waits for the check box to change state, and once the state has changed the function will run checking which state its in, depending on its state it will either show the map or hide the map.

Map

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

Above is the initial setup of the map.

Text

Description automatically generated

The same as task 4 the script waits until we get a click on the search button and does all the same checks as task 4 but after it runs “accSearchMap” function instead.

Text

Description automatically generatedThere are two other functions inside the script worth mentioning, “setNewMarker” which simply has all the marker information stored inside it and is able to be called upon to add a new marker in an instant, “clearAllMarkers” which cleans up the map from old markers in preparation for a new search.

These two functions were created to avoid repeating code many times.

Text

Description automatically generated

The “accSearchMap” function starts the exact same way as task 4 “accSearch”. This function must first before plotting markers on the map remove all markers (this avoids duplicates), it can then find out which location is associated with this specific search, to do this it only picks the first of the found accommodation and does a location search.

Text

Description automatically generated

Text

Description automatically generatedThe location route works like all the other routes and will return the cords for the location.

Text

Description automatically generated

With the information sent back from the fetch method we can now determine the appropriate zoom level for the specific location.

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

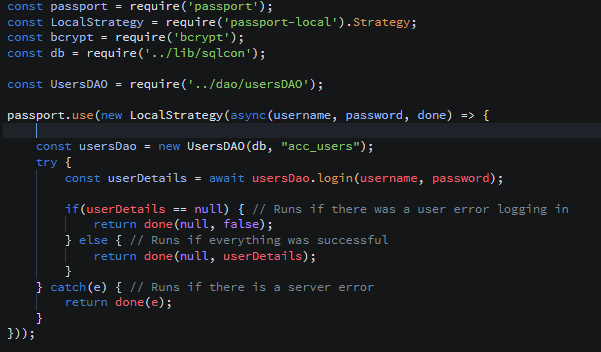
Description automatically generated with medium confidenceBefore finally plotting all the markers for each accommodation onto the map.

Like task 4 searching without a type is identical it simply fetches a different URL.

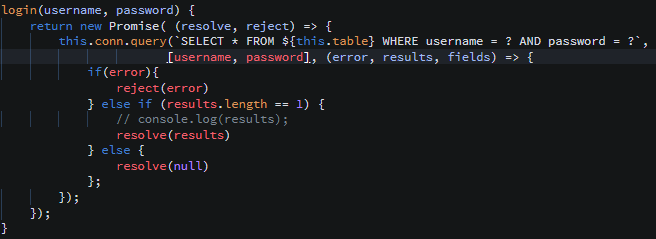
# **Task 8**

**Complete: Yes**

Task 8 was modified and made use of a technology called passport to take care of the logging in system for the website.



Passport is defined and setup in its own file. In the application we use passports “LocalStrategy” which can be seen being used above. In LocalStartegy’s verify function we connect to the Users DAO which allows for the function to call the login method from the DAO testing whether the username or password provided is valid.



The Login method queries the database to find the user row with the specified username and matching password so it can return the information back to the verify function.

Text

Description automatically generated

The verify function will retrieve and check whether it has received information from the DAO or not, if it hasn’t the function will call the “done” callback with the first parameter as null and the second as false which indicates authentication was not successful causing Passport to automatically send a status code of 401. If everything was successful the “done” callback would be called again but instead of false it would place the user details which will later be attached to the “req” object as “req.user” which can be used later to verify whether the user is logged in and what routes they can access.

# **Task 9**

**Complete: Yes**

Task 9 states users must be logged in to book accommodations, which means task 3 must be changed to accommodate this.

Text

Description automatically generated

To do this I started by using a middleware I wrote in a previous assignment to help keep track of whether the user was logged in or not by storing their id in a global variable, in this case although I already had full user details and usage of them using “req.user” I found that I couldn’t use it in my ejs pages but I can use global variables, so I stored the “req.user” in a global variable and was able to accomplish the username on the header.

Text

Description automatically generated

It also allowed me to add verification to the middleware applied to the route “/acc/book”.

# **Task 10**

**Complete: No**

# **Task 11**

**Complete: No**

# **Task 12 – Part G**

**Complete: No**

The Only thing completed was the usage of **Passport** as shown in task 8**.**

# **Database Changes**

The Database files new name is WAD22MODDED.sql (located in cli)

The Database file was changed to include a new table called locations:

Text

Description automatically generated

This was done to allow me to search for the locations location and pan the map to the specific location area.

Additionally, I added a LocationID to the accommodation table which will be used to find the location.

Text

Description automatically generated