Hannah-Ann Nana-Hackman  
  
GitHub Link : <https://github.com/hannahannx/PlacesToStay>

14970244

# Introduction

This report will discuss the methodology for developing the Web Application PlacesToStay, a full-stack web application.

This application was developed using key technologies and frameworks such as Node.js, React, and Express for front-end development, utilising HTML (Hyper Text Markup Language), JavaScript, and AJAX (Asynchronous JavaScript and XML). The user interface, created by these components, allowed the user to interact with the API, which processed the user input. SQLite Studio was used to manage the SQLite database with SQL queries, and the better-sqlite3 module was downloaded onto the server side to allow the interaction between the server for the server-side script and the database.

When the user interacts with the user interface on the web browser client, the request created from the user interaction is sent via JavaScript to the web server. The web server processes the information with the corresponding RESTful API request (POST, GET, DELETE) and interacts with the database. The database takes that information and sends it to the web server. The web server sends the data retrieved from the database in JSON to JavaScript, allowing HTML, CSS, and JavaScript to manipulate the data returned. This manipulation allows the JSON of the accommodation information to be displayed user-friendly (Mishra et al., 2021).

Furthermore, the mapping features Leaflet and OpenStreetMap were used. These features enhance the user experience by visually representing accommodation options on an interactive map using markers. This implementation of markers, with a popup for the corresponding details for each marker, made the web application more fluid in its design approach.

Security sessions were used to check the integrity of the web application. This allowed each user who visited the web application to log into their account, which was also communicated to the database.

Critically analyse the methodology used, giving advantages and disadvantages.

## Part A

### Task 1- Look up all accommodations in a given location

The first task uses the Express framework route to search for the location, which the user inputs in the route to search for accommodation in the database. We create a GET route from the express application on the app object. GET requests are used when collecting data. In this case, we are searching for the location name; therefore, this is the appropriate HTTP request. We make a call-back function that runs when the specific route is displayed and matches the requested route. This route takes on one parameter, location and the inputted location provided by the user on the client side. We then create a try-catch, which will handle the error if unexpected data is retrieved from the attempt to connect to the database and display the results in JSON. Error requests will allow us to see where errors in the code are for better resolution.

We create a variable called smth, which takes the connection object of the database initialised at the beginning of the server file (app.mjs) and queries the SQL statement onto the places to stay database. It selects all fields from the accommodation table where the location of the accommodation field is equivalent to the location requested by the user on the client side. We create a variable called results, which stores the required information that relies on the value passed into from the parameters in the route.

Then, res.json() takes the results as a parameter and returns the results that meet the criteria based on the SQL query request from the database. The results are formatted in key-value pairs, where the first item in the pair is the key, and the second is the value stored inside that key.

A screen shot of a computer code

Description automatically generated

Create your reference to this picture and add it to the reference list at the end of the report.

### Task 2 – Look up all accommodations of a given type in a given location

After developing the second API route, I retrieved an error message. To decode this error, I looked at the type of error the JSON displayed. SQLITE\_ERROR would refer to an error created by connecting to the database. Therefore, to debug this, the message looked at the SQL statement. I found that this was a syntax error, as the name of the database, “accommodation”, in the SQL statement was misspelt. I changed “accomodation” to “accommodation” to fix this error.

A close-up of a computer screen

Description automatically generated

The second task uses the Express framework route to search based on the type and location of the accommodation. The process used for this task is very similar to the first task. The main difference is the parameters and the SQL statement used. In the GET route, more than one parameter is required. This allows the user to enter more than one parameter: the type of accommodation and the location of the accommodation. These are inputted by the user on the client side. In our variable smth, the SQL statement selects all the fields of the accommodation table in the places to stay database where the location and type are equivalent to the two values passed in by the parameters. As two parameters are required, we must create another parameter for the method smth. all to be run on the database.  
  
A screen shot of a computer program

Description automatically generated

### Task 3 - Book a place of accommodation for a given number of people on a given date

The third task uses the express framework route to book accommodation. We create a POST request from the express application on the app object. POST requests are used when updating or inserting data; we would do both in this case. Updating the database for availability and inserting a new booking in the acc\_bookings table would be the appropriate HTTP request. This task would be slightly different from the previous task, as this is a POST request. We would not be passing values as parameters in the POST route. This is because, instead, we would be embedding the data retrieved from the database inside the request body as JSON. Additionally, this method would be more secure, as if sensitive information were retrieved from the database, it would not be exposed to the parameters. We then create a try-catch, handling the error if unexpected data is retrieved.   
  
The following line checks if there are any blank or erroneous values in the embedded data. Suppose there are no blank or 0 values; we create a variable called createBooking, which takes the connection object of the database and initialises the query to the SQL statement. It Inserts into the acc\_booking table new values (accID, the date, username, people), which are passed into by the embedded data that meet the key-value pair criteria by JSON. We then store into a new variable called booking results the required information needed for the create

Booking variable and run this method. We also do a similar technique to reducavaiblityResults and availability results, respectively. Still, the SQL statements update the acc\_dates table in the placestostay database and reduce the availability of the accommodation based on the required embedded data for the date and accID.

To check if the database update was successful, we ran the changes()on the results returned from the availability results. If a change happens, we can see if one row is affected. If this is the case, we insert the new data retrieved by the embedded JSON from the SQL query and add it to the last column in the database—.lastInsertRowId. This is done by assigning the new data to the previous row ID plus one (which would mean a new result). If there had not been one change, no results would have been returned. We then send an error message to the user to show that the fields were incorrect, meaning there would be no such results with that information.

## A computer screen shot of a program code Description automatically generated

## Part B and G Using React.js)

### Task 4 - Write an HTML page (which MUST be named index.html) that allows the user to search for all accommodations in a given location.

For the CSS for the HTML page, I initially used inline CSS, but I used a CSS loader with an external stylesheet instead. However, the CSS page was not linked, so I used npm install—save styleloader to resolve the issue. Then, I found that applying inline CSS to each page would be tedious, so I used external stylesheets for each page so they would have specific components to work on. Inline CSS would be beneficial if I didn’t have many components to apply CSS to. Still, external stylesheets would be better in this case due to the size of the project, which allows some reusability for the different components and a bit more flexibility.

Firstly, I created state variables to store the current value of the accommodation being searched and another state variable to save the search results retrieved from the search route in the servers. The initialised state (default value) of the currentaccom and setcurrentaccom set would be a blank string. However, the initialised date of search and set search results would be an empty list. We stole these values in the state because it automatically renders when the data changes. When you first load the page, the values will be empty. So, later on, we can append the data to the list. When you first load the page, the values will be empty. Later on, we can append the data to the list and string, respectively.

All accommodations would be the list of JSON, which is returned from the API POST route on the client side by using an asynchronous function called display allocation. We can create an asynchronous function that returns JSX containing the code to be displayed to the user. JSX allows us to write HTML in React. It also converts HTML tags into React components/elements so we can use them. The function “updateLocationName” updates the function's name to be searched for, which triggers the rerendering in React. This is so that event handlers need access to the component's state, which must be written inside the component. We create all accommodations and a mapping function equivalent to the FOR LOOP, where the information is rendered based on the database for every accommodation comparable to every set of JSON key-value pairs returned from the database.

deatilsToBook was created to store the embedded data, which must be sent to the POST route in the RESTful API. The try-catch handles the error if unexpected data is retrieved from the

### Task 5 – Modify your code to process the search results so that you can create a Book button for each result.

I used a button and assigned it to the ID of bookAccommButton to be referred to later. This button also takes a function called bookAccomm(), which takes the key that the mapping function would be currently on as a parameter. Unfortunately, this returns an “undefined” acid on the database when it populates the acc\_bookings database. I tried different methods, including ${accommodation. Id}, which returns NULL instead of undefined. This would be due to how I am retrieving the accommodation ID. Which I am not sure how to resolve. Suppose I hard-code the number inside the bookAccomm function. In that case, it successfully populates the database, so I need to think about my approach regarding passing the unique ID for the accommodation being clicked into the function.

## Part C

### Task 6 – Add error checking to task 3

This task does further error checking. The if statement checks if accID is a blank string or 0 (the zero was not needed but allows it), and the other values in the embedded JSON passed in are blank strings by the database. This allows for validation to ensure that the user has entered information when testing the POST route and that no information that would be updated or inputted inside the database would be undefined or NULL.

## Part D

### Task 7 – Adding a map

The map successfully adds a marker to the information, which is put inside the two input columns. To get the information which is the accommodation results. I would need to store those results inside a list, link this to the map with a marker component, and traverse each key-value pair based on latitude and longitude, but I needed to figure out how to do this. Therefore, it would be possible to manually put the information on the results directly inside the input column.

Map. current takes the current state of the map and setsPos() and continues to add new markers based on the information entered by the user via the input field. It also has a popup that is bound to the marker. When the marker is clicked, the information passed in through the user's input of the longitude and latitude is displayed, with more than one marker corresponding to the corresponding position on the map.

## Part G

I used React to complete this web application. The main areas in which I used React were through the lifting upstate. Lifting the state in React allowed me to display more than one component on the index page (which I called the AppComponent). As I separated each functionality into more than one component, this made them reusable, and I would be able to contact the component more than once per page and get a unique result for each one.

# Conclusion

In this report, we discussed the methodology I used to develop the web application Places to Stay, a full-stack web application. We talked about the reasoning behind the techniques and errors that I encountered and how I would like to resolve the issue or, however, resolve this issue. Upon reflection, I concluded that I needed more research to understand how to handle data through React. I wanted to challenge myself with the React framework compared to non-react, so I implemented React over a non-React version. In the future, I would need to revise the basics of non-web applications before applying the knowledge to based applications.

# References

Mishra, D. P., Rout, K. K., & Salkuti, S. R. (2021). Modern tools and current trends in web-development. Indonesian Journal of Electrical Engineering and Computer Science, 24(2), 978. https://doi.org/10.11591/ijeecs.v24.i2.pp978-985