REPORT

Web Development Capstone

Team 4: Christopher Horsfall

18/12/2024

Table of Contents

Introduction

This project was about building a React and Node.js based web application to provide users with custom data visualisations based on provide tourism data. The users are intended to be organisations involved in the tourism and tourism marketing industries as well has local government. The project was to have both a ‘client’ side and a ‘server’ side. The client half was to be focused on the web app and the user experience whilst the server was to focus of responding to client requests and connecting to a MySQL database.

The project also required extensive project planning and team management. ‘Agile’ software development practices were required and the project was split into 2 ‘sprints’. Sprint 2 is covered in this report.

Difficulties face in this project was the small team size, only one person, and their inexperience in developing a project this size. Many features were planned but not implemented. But the features that were implement work as intended.

The complexity of the folder structure and uncertainty over the usefulness of the application’s features were also detrimental to the projects progress.

The greatest outcome of the project was the experience gained by the developer in working on a web application of significant ambition.

Project Plan Sprint 2

Introduction

As part of the project plan for sprint 1, a major review of features was scheduled for the start sprint 2. This was to help guide the project and offer an opportunity for reflection and a change in course.

Project Goal

The original goal of this project is to create a dynamic web application to provide users bespoke data centric insights about their local tourism industry. However, as the project progressed the goal of 'data centric insights' was refined to 'data visualisations'. These visualisations would help the users develop their own insights into trend and patterns occurring in the tourism industry.

Sprint 1 Review

A key part of sprint 2 is to review features and aspects of sprint 1. This will help inform and refine the direction of development for sprint 2.

Review of User Requirements

The users for this web app are expected to include local Tourism Boards(Destination Marketing Organisations), Local Government Area Bodies, and Tourism Operators. It was noted in some Tourism Boards end of year reports that they are dependent on funding from Local Government and voluntary contributions from tourism operators. It has been concluded that a useful purpose for the web application is to provide compelling illustrations of the effectiveness of a Tourism Board's advertising campaigns. Measuring tourism data over time and comparing it to the launching of related campaigns would help improve confidence in the Tourism Boards activities. Marketing campaigns by operators could also be measured using the visualised data.

Thus, the visual comparison of data to con-current events is the primary method of delivering useful insights for the project users.

Review of Folder Structure and Project Architecture

The folder structure/ project architecture from sprint 1 is sound when considering the knowledge and experience available to the developer at the time of the projects creation. Now that the project has developed serval criticisms of the projects architecture are becoming apparent.

A single page on the web app has multiple nested components that make navigating and editing the project confusing and difficult. The users Home page contains an 'user welcome' component (that handles the users location value) and the component 'Industry watch' – an area for charts about the users local tourism industry'. This component contains a Hook component, a few data processing components, a chart component. The chart component also contains data processing components and a users location prop to be passed down from the 'user welcome' component.

In the developers' opinion, its is difficult and confusing to navigate through and edit hierarchical 'tree' structures.

This problem could maybe be solved by a better IDE, one adapt at dealing with tree folder structures, or a better understanding of the capabilities of VScode and it's available extensions.

The difficulty of trees could be avoided by limiting the depth of the structure and avoiding 'diagonal' dependencies. I.e. a 'child' element being dependent on a value from an 'uncle' or 'cousin' relative instead of a vertical 'parent' element.

It may be too time consuming to change the projects architecture in large ways to avoid this difficulty. Small alterations will have to suffice.

In future projects, page files should contain all components calls. The layout, structure, features and dependencies on a page should be visible from the page file.

Review of Server Code

The server code for this project is adequate for the current projects' size. One of the main learnings from the second half of subject IFQ716 'Advanced Web Development' was the technique of splitting server endpoints into their own files. Currently all endpoints are in a single file and single 'if' statement. This strategy is nearing its limits and if the project is to expand its number of endpoints they will have to be arranged into the new file structure lest they become unwieldy to work with.

Currently the data available is limited to the last two years, is limited to 5 locations, and is limited to 4 attributes. If it was expanded to include all Australian locations, the last 10 years, and 10 attributes, it is likely many different server endpoints will need to be built. Serving the client the whole database every time they use the web app will be undesirable.

The data files are currently being stored in the server folder. This approach was acceptable at the start of the project when data was limited and knowledge of how to handle data was still developing. However, for Sprint 2, the data files provided should be moved to a MySQL database. This transition aligns with practices used in professional-grade projects, where databases ensure better scalability, security, and data integrity. With MySQL, data can be efficiently queried, filtered, and manipulated before being sent to the client, which might prove invaluable for future iterations of the project, such as implementing advanced analytics, supporting dynamic queries, or enabling integration with other systems.

Similarly, user credentials need to be moved from the server folder to the MySQL database. The user login and register features should be implemented following what was taught in 'Advanced Web Development'. Further security improvement could be made, and would need to be made for this web app to be made public, but that is beyond the scope of this project.

Review of Aesthetics and Web App Layout

The appearance of some parts of the web is sound. Serval aspects of the project are holding back the progress of the appearance of the web app. The folder structure makes it difficult to judge and edit the contents of a page in terms of layout and margins. The general strategy for making a web app visually appealing seems to involve a circular process of trial and error, as this developer still struggle to visualize how changes to the HTML or JSX code will translate to the final design.

A CSS framework may help solve this problematic aspect of the project's design. A CSS framework provides pre-defined components and utilities to facilitate the easier design and creation of web apps.

Currently the project uses Bootstrap. Bootstrap offers predesigned components and responsive features. However it is likely not being used to its full potential as it is so feature rich that finding an appropriate feature becomes too time consuming.

Tailwind CSS is a framework that focuses on offering utility classes that can be used to build custom designs without the writing CSS. Other teams were successful in using Tailwind so it might be worth considering.

Semantic UI is an framework that features intuitive class names, pre-built UI components and a customisable theming system. It is useful when readability is a priority.

Primer CSS is a framework that focuses on creating simple and maintainable designs. It has been developed by GitHub and is known for being easy to customise.

Utilising a more appropriate CSS framework will likely help the design aspect of this project or projects in the future.

Review of Charts

The main feature of this web app is to view quality data visualisations to help generate insights into the tourism industry. This web apps charts are its most important feature.

The chart created for sprint 1, a line chart comparing occupancy rates year on year for a single location, was adequate but time consuming to build. It was decided that comparing data from different years was most important as the tourism industry has a cyclical yearly nature. Tourism industry demands correlate with yearly events like public holidays, school holidays, sporting calanders and the seasons. Comparing occupancy in June to July is not as useful as comparing occupancy June 2023 to June 2024.

The data had to be considerably pre-processed before being charted, and the AG Charting library was difficult to customise. More importantly, many AG charting features where paid features not available to the project. The improvement of the projects charts are limited by the AG Charting library.

It has been concluded that a different charting library should be considered, one that is free and highly customisable. Important features should be the ability to draw multi-line charts that can be 'zoomable' and have several annotations like school holiday periods highlighted. Other charts types like pie charts or calendar heatmaps would also be valuable.

Chart.js

A simple library that allows developers to quickly create charts. Animations and styling options available but there appears to be a limit on the number of charts types and customisations options.

D3

A complex charting library that offers the full degree of customisations. Other well known charting libraries are based on D3, they include Nivo and Plotly. D3 has all the features required for the project. It appears to have the potential to produce really impressive and interesting charts. Other charting libraries seem to have greater ease of use at the expense of the charts' potential.

Plotly

A declarative charting library based on D3. Has advanced scientific and 3d charting options. Plotly can be used in a variety of languages. It is not clear what benefits Plotly could bring to the project that D3 could not also bring.

It is concluded that because the project goal is to provide bespoke data visualisations and D3 is the charting library most capable of producing custom data visualisations, the project should move from using the AG Charting library to the D3 charting library. With D3, the projects' potential will be greatly increased.

Out of the various chart types that exist, only the line chart and possibly the calendar heat map seem to be useful in presenting the data on hand. All data available is chronological in nature and lines charts are typically use to present time series data. A calendar heat map could be useful in visualising the effect of important dates, like ANZAC day, have on the data attributes.

Pie charts and bar charts are used when data is in discrete allotments, which is not the case for this data set. The data set could be modified into discrete allotments, like an average length of stay per month set, but the usefulness of this to the user is not fully understood.

An area heat map with the different locations around the country could be interesting, useful and impressive. However, it is a very advanced chart type that might not be that impressive with only 5 locations. A prototype could be built to showcase the charts potential with some mock data.

As the developer is not a data scientist it is hard to know what chart types would be useful in extracting insights from the data provided. Especially since all the data attributes have already been averaged.

It would be nice to have a list of all the tourist bookings in a location and time. This would allow the wed app to provide a histogram for each 'length of stay', 'daily rate' and 'booking window' attribute. This would allow the user to see if attributes were clustered around the average, or spread out widely, or if the average was being pulled upwards by a few outliers.

It is also not known how large the sample size is for the data collected. Some locations likely have many fewer hotels than others, this could lead to the random decisions of a few tourists having oversized influence on the data.

Access to less processed data would make the project more complicated but could also increase the usefulness of the visualisations.

Sprint 2 Plan and Outcomes

Priority Grade	Task/s	Comments/Outcome
A (Most Important)	• Set up new MySQL connection. • Import CSV datafiles to a ‘Schema’. • Ensure schema matches old CSV structure. • Install Knex into project. • Write knexfile.js • Modify server.js routing function to use knex for database queries. • Test web app to ensure knex and MySQL are working.	Completed
A	• Install D3 into the project. • Research multiline graphs • Build new multiline chart to display ‘occupancy’ data – start with basic features first. • Add lines and shaded rectangles to hight light school holiday periods. • Build button or check box to toggle on/off holiday period highlights. • Add toggleable highlights for other important dates/ periods. These toggles maybe need to be moved to a drop-down selector. • Research and implement ‘zoomable’ feature. • Test chart in chrome.	Completed, except ‘zoomable’ feature and other important dates highlighter.
A	• Move user credentials to MySQL server • Modify/add to server endpoints to access user credentials in MySQL via knex. Register feature will need to modify user credential database. • Build Register page and components on Client side of web app folder structure. • Test login and register components in chrome, ensure MySQL data base is being updated when new user is signed on.	Not Completed
A	• Use D3 and existing multiline occupancy chart as a template to build multiline charts displaying	Not Completed

	‘Average Length of Stay’, ‘Average Booking Window’ and ‘Average Daily Rate’ attribute for a single location. • Alternatively existing multiline chart can be modified to accept ‘attribute’ prop, so it can b e used to display different data attributes. A drop down will need to be generated so that the user can change which attribute to display.	
B (less important but still valuable)	• Build a multiline chart that compares a data attribute across different locations. The x-axis will be time, y-axis the attribute value, and the different lines the different locations. • A dropdown selector needs to be added to allow the user to choose which locations should be displayed.	Not Completed
B	• Build and add ‘Meta Data’/’Summary’ statistics for each dataset attribute. • Build data processing functions to supply these meta statistics. • Ensure feature sit well with other elements on the web app page.	Not Completed
B	• Research and Build D3 based ‘Calendar Heat map’ to display data attributes. This could be a valuable alternative to line charts. • Outline area on calendar that correspond to important dates (Holidays, festivals, tourism campaigns)	Not Completed
B	• Research and store in database dates corresponding to tourism campaigns run by users. • Add these dates to toggle options for timeseries charts. This will allow Tourism boards to view the start/end dates for their campaigns and their effect on tourist data attributes.	Not Completed
C (Should be done after all other priority grades are complete)	• Improve Web app layout and aesthetics. This will be difficult to do before all app features are implemented.	Not Completed
C	• Reorganise folder structure to be more intuitive, this can be done in small bit as the project progresses.	Not Completed

Application Design

The major focus of the application design was to provide a useful series of features for the registered user. Because of the value associated with the data used in this web app it was decided that the features should be accessible only to users how have already registered. The web app outside of user login would be focused on advertising the usefulness of the site.

A professional aesthetic, inspired by some of our users own websites, would help locate the app within the industry as a professional tool. Images of tourist destinations were used to help make the site appealing.

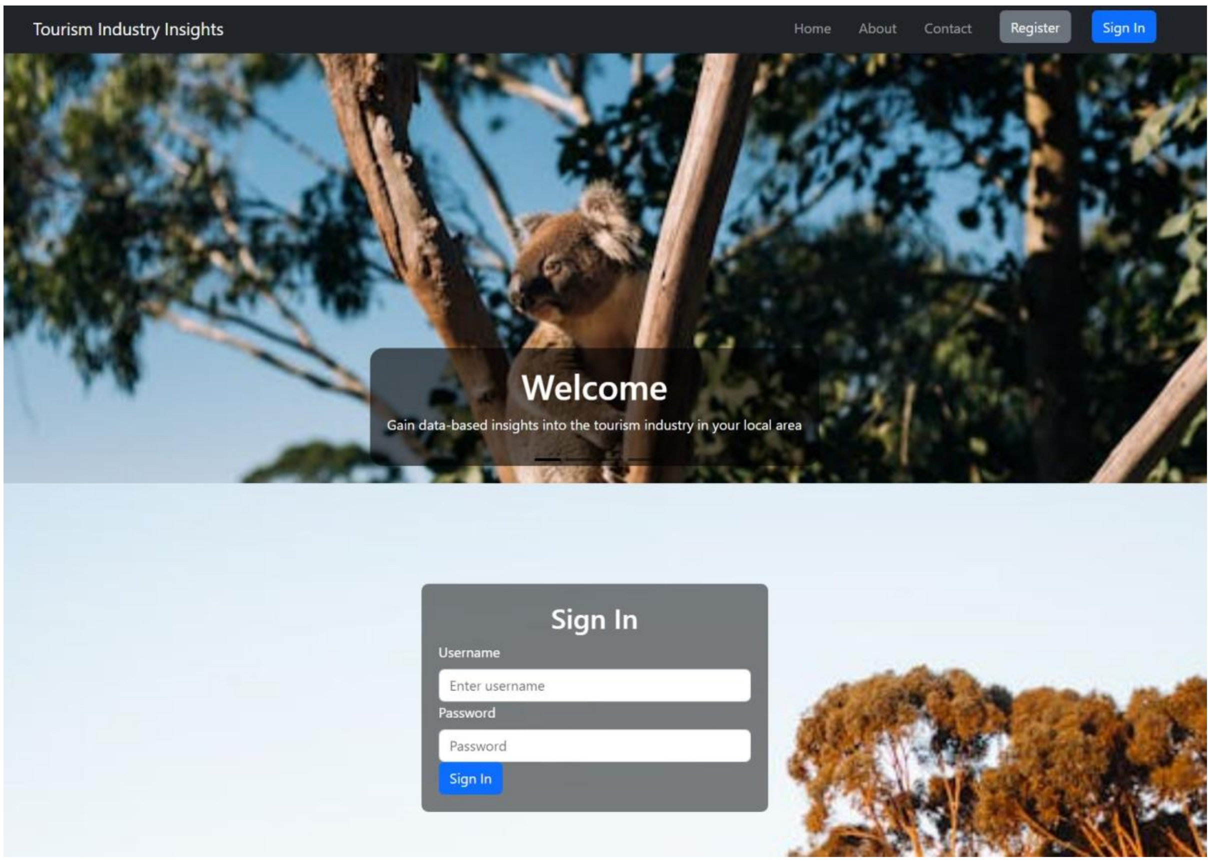
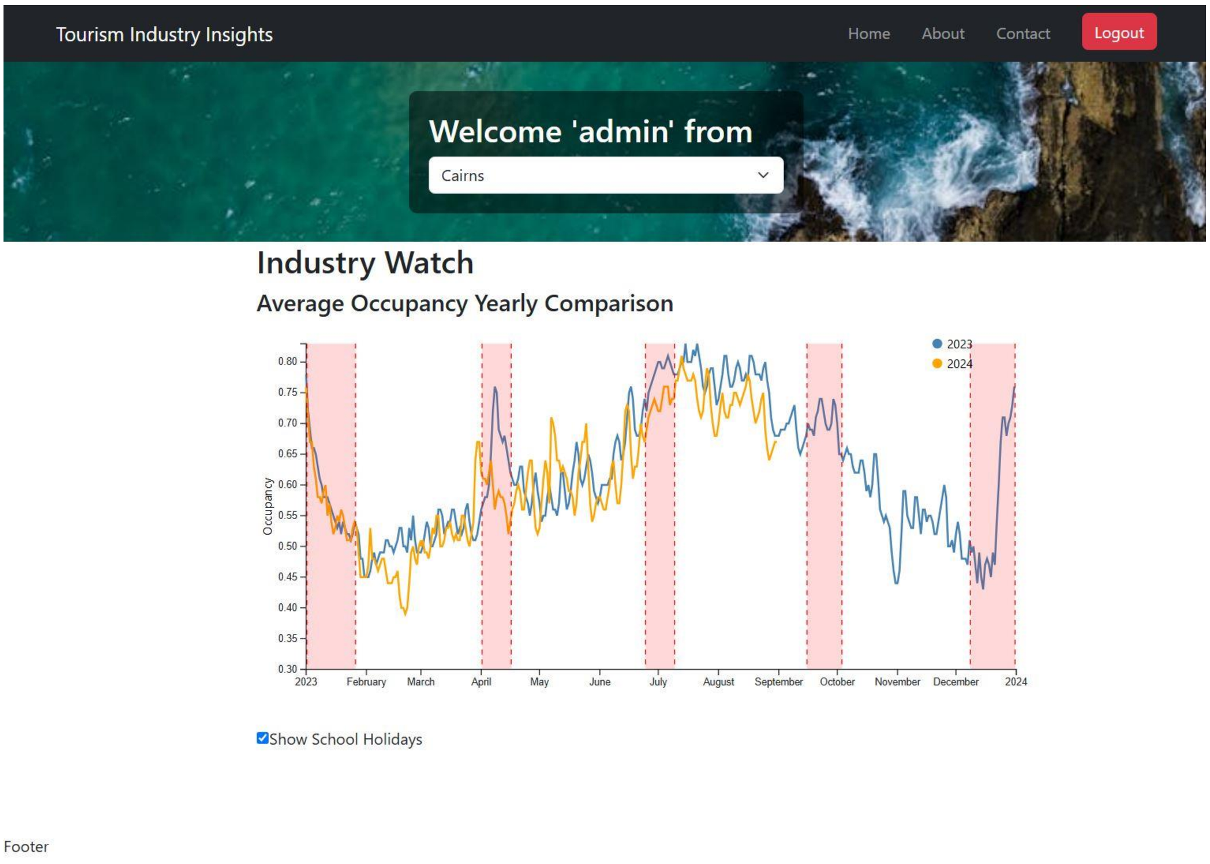

Tourism Industry Insights
Home
About
Contact
Register
Sign In
Welcome
Gain data-based insights into the tourism industry in your local area
Sign In
Username
Enter username
Password
Password
Sign In

FIGURE 1: HOMEPAGE

The main feature of the app will be an interactive chart to display data relevant to the users location. The location is set by default to the local government area the users organisation resides in or represents.


Tourism Industry Insights

Home About Contact Logout

Welcome 'admin' from

Cairns

Industry Watch

Average Occupancy Yearly Comparison

☑ Show School Holidays

Footer

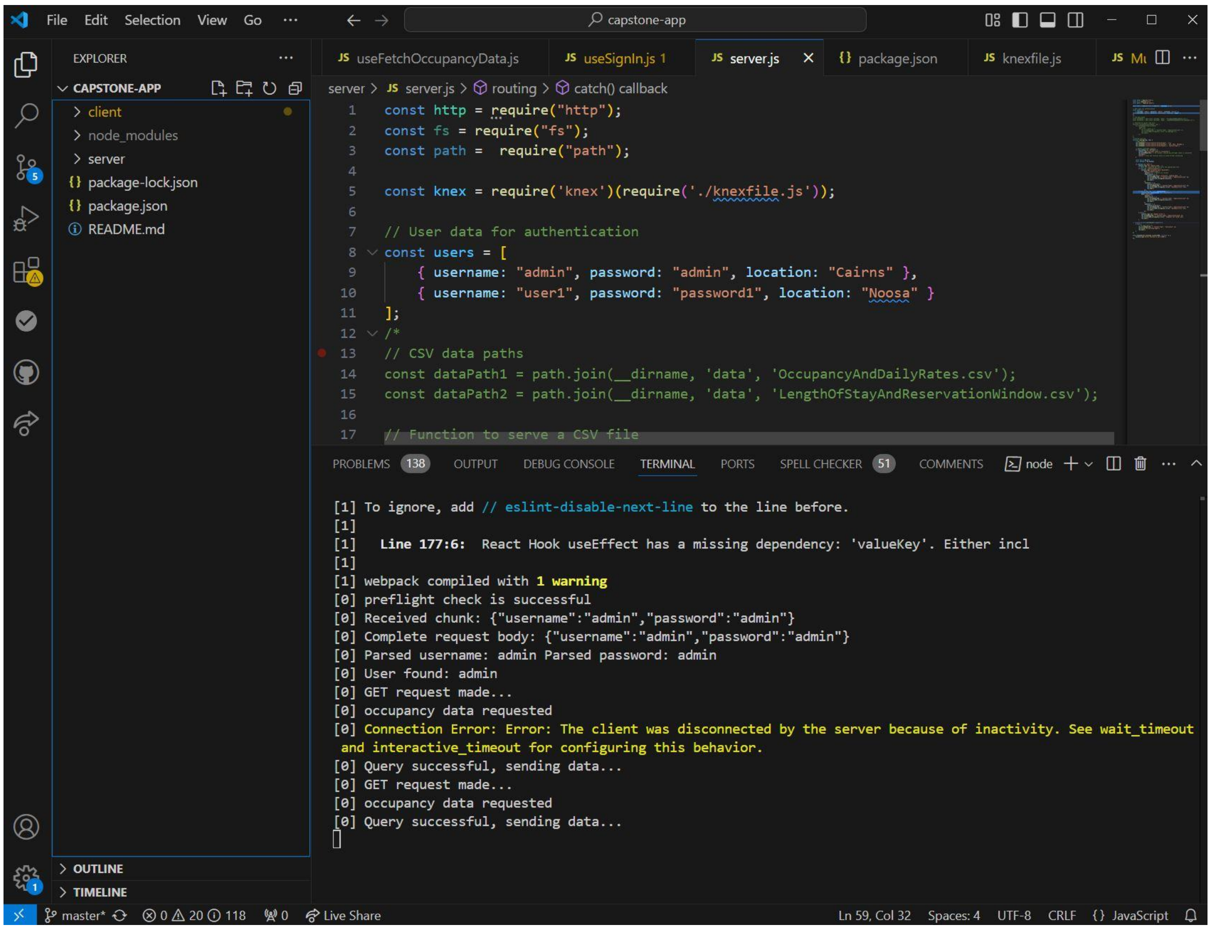
FIGURE 2: HOMEPAGE FOR LOGGED IN USER

The homepage for the logged in user displays a chart compare yearly average occupancy rates for hotels in the user's location. Check boxes allow the user to hide or display important holiday periods. Drop down selector allows the user to change the location data the chart uses. It was thought that a user focused on one location might like to view the data from other locations, this might help them understand how their locations is doing in comparison to others.

It was intended for the app to have more charts and more options to alter the charts but this was not completed.

Application Architecture

The application utilises the third-party frameworks and libraries React, Node.js, MySQL, Knex, and D3. It is being backed up on a GitHub repository. The project is split into a 'server' side and a 'client' side.

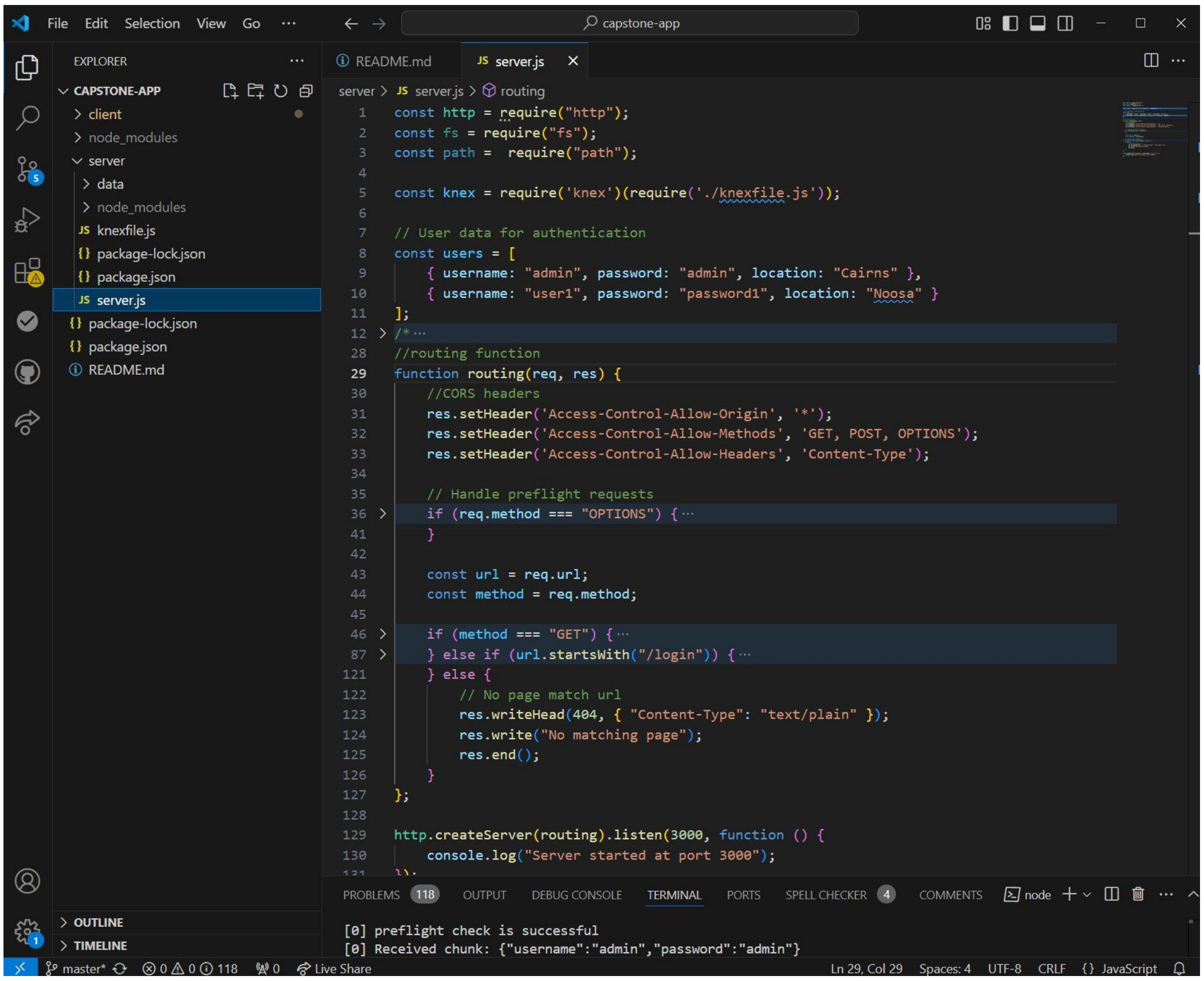


```
const http = require("http");
const fs = require("fs");
const path =  require("path");

const knex = require('knex')(require('./knexfile.js'));

// User data for authentication
const users = [
    { username: "admin", password: "admin", location: "Cairns" },
    { username: "user1", password: "password1", location: "Noosa" }
];
/*
// CSV data paths
const dataPath1 = path.join(__dirname, 'data', 'OccupancyAndDailyRates.csv');
const dataPath2 = path.join(__dirname, 'data', 'LengthOfStayAndReservationWindow.csv');

// Function to serve a CSV file
```

```
[1] To ignore, add // eslint-disable-next-line to the line before.
[1]
[1]   Line 177:6:  React Hook useEffect has a missing dependency: 'valueKey'. Either incl
[1]
[1] webpack compiled with 1 warning
[0] preflight check is successful
[0] Received chunk: {"username":"admin","password":"admin"}
[0] Complete request body: {"username":"admin","password":"admin"}
[0] Parsed username: admin Parsed password: admin
[0] User found: admin
[0] GET request made...
[0] occupancy data requested
[0] Connection Error: Error: The client was disconnected by the server because of inactivity. See wait_timeout
 and interactive_timeout for configuring this behavior.
[0] Query successful, sending data...
[0] GET request made...
[0] occupancy data requested
[0] Query successful, sending data...
```

FIGURE 3: CLIENT AND SERVER FOLDERS

Folders for 'node_modules' are present in the route folder and each of the server and client folders. This is because changes were made to the folder structure during sprint 1 that resulted in time consuming GitHub problems (node_modules were not designated as 'ignore' in GitHub config).



```javascript
const http = require("http");
const fs = require("fs");
const path =  require("path");

const knex = require('knex')(require('./knexfile.js'));

// User data for authentication
const users = [
    { username: "admin", password: "admin", location: "Cairns" },
    { username: "user1", password: "password1", location: "Noosa" }
];
/*···
//routing function
function routing(req, res) {
    //CORS headers
    res.setHeader('Access-Control-Allow-Origin', '*');
    res.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS');
    res.setHeader('Access-Control-Allow-Headers', 'Content-Type');

    // Handle preflight requests
    if (req.method === "OPTIONS") {···
    }

    const url = req.url;
    const method = req.method;

    if (method === "GET") {···
    } else if (url.startsWith("/login")) {···
    } else {
        // No page match url
        res.writeHead(404, { "Content-Type": "text/plain" });
        res.write("No matching page");
        res.end();
    }
};

http.createServer(routing).listen(3000, function () {
    console.log("Server started at port 3000");
});
```

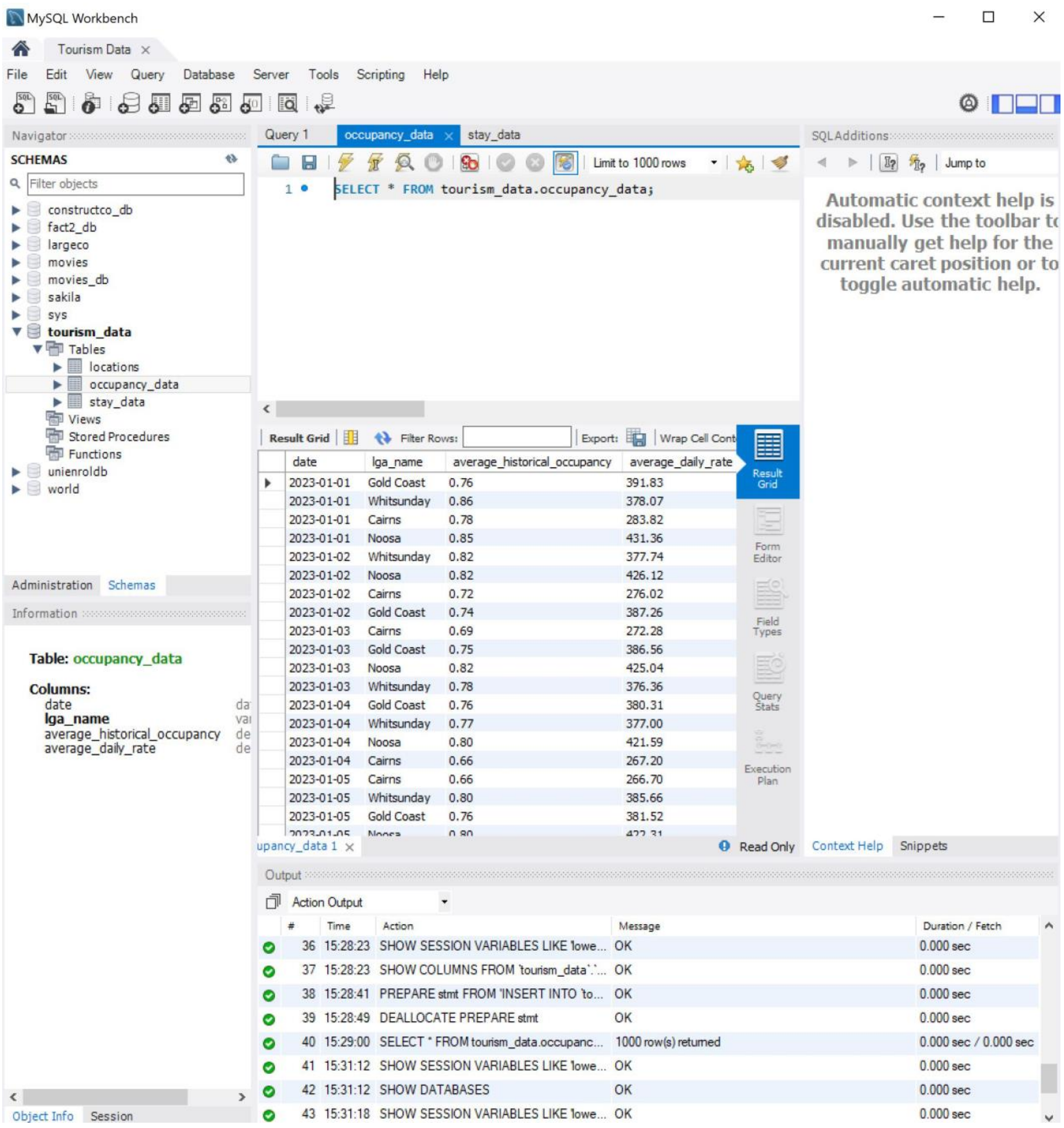
```
[0] preflight check is successful
[0] Received chunk: {"username":"admin","password":"admin"}
```

FIGURE 4: SERVER.JS

The server for this project runs from a single file; server.js. This is adequate for now as the server only has a few routing options. When the project expands this routing function will have to be split into multiple files each corresponding to a single route.

The routing function is the main way the client communicates with the server. It communicates its requests mainly through the use of URLs.

In sprint 1 the data was stored in the ‘data’ folder. For sprint 2 it has been moved to a MySQL database, this is accessed via knex in the ‘knexfile.js’ file.



Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

```
constructco_db
fact2_db
largeco
movies
movies_db
sakila
sys
tourism_data
  Tables
    locations
    occupancy_data
    stay_data
  Views
  Stored Procedures
  Functions
unienroldb
world
```

Table: occupancy_data

Columns:
```
date                          da
lga_name                      va
average_historical_occupancy  de
average_daily_rate            de
```

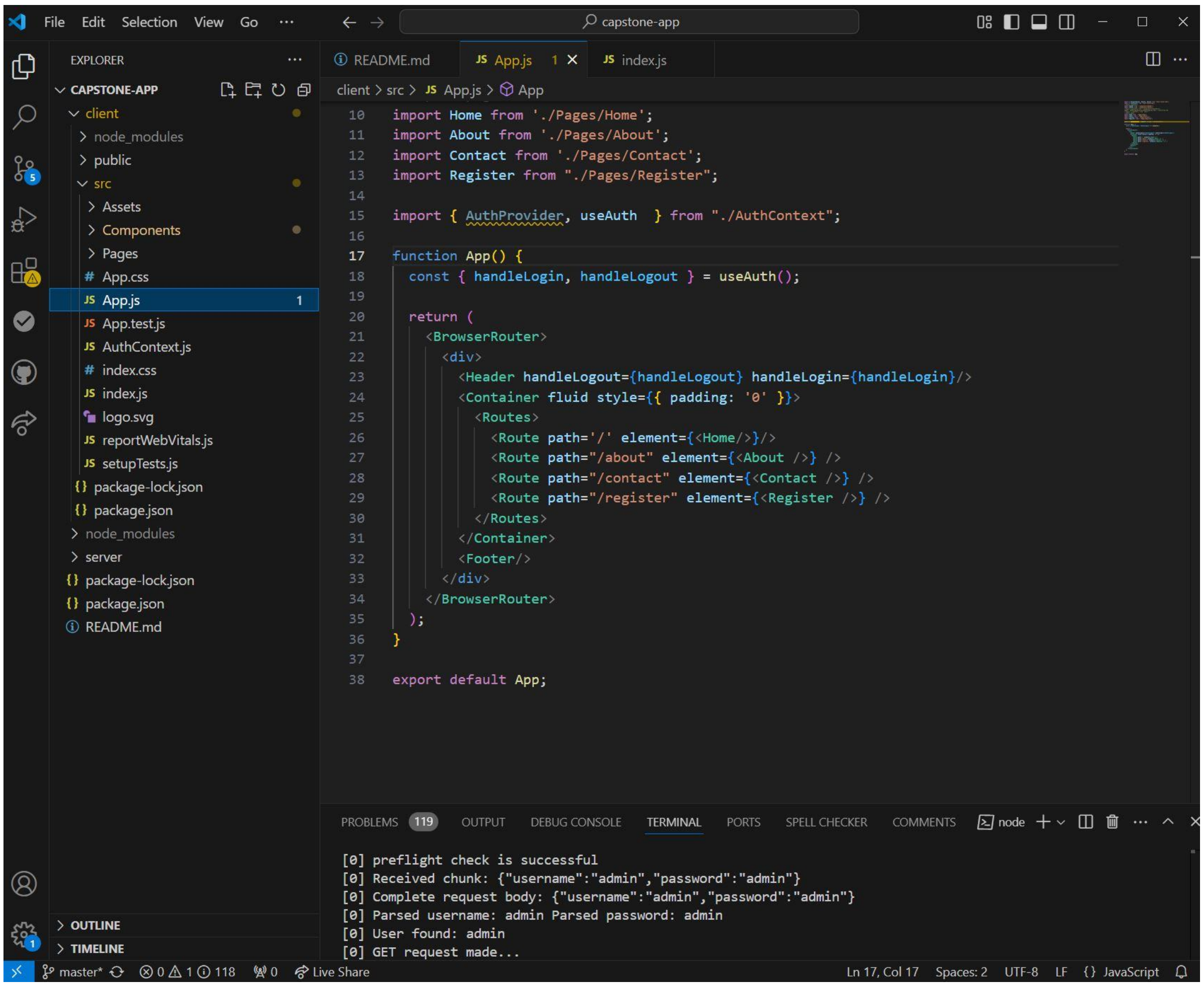
```sql
SELECT * FROM tourism_data.occupancy_data;
```

date	lga_name	average_historical_occupancy	average_daily_rate
2023-01-01	Gold Coast	0.76	391.83
2023-01-01	Whitsunday	0.86	378.07
2023-01-01	Cairns	0.78	283.82
2023-01-01	Noosa	0.85	431.36
2023-01-02	Whitsunday	0.82	377.74
2023-01-02	Noosa	0.82	426.12
2023-01-02	Cairns	0.72	276.02
2023-01-02	Gold Coast	0.74	387.26
2023-01-03	Cairns	0.69	272.28
2023-01-03	Gold Coast	0.75	386.56
2023-01-03	Noosa	0.82	425.04
2023-01-03	Whitsunday	0.78	376.36
2023-01-04	Gold Coast	0.76	380.31
2023-01-04	Whitsunday	0.77	377.00
2023-01-04	Noosa	0.80	421.59
2023-01-04	Cairns	0.66	267.20
2023-01-05	Cairns	0.66	266.70
2023-01-05	Whitsunday	0.80	385.66
2023-01-05	Gold Coast	0.76	381.52
2023-01-05	Noosa	0.80	422.31

#	Time	Action	Message	Duration / Fetch
36	15:28:23	SHOW SESSION VARIABLES LIKE 'lowe...	OK	0.000 sec
37	15:28:23	SHOW COLUMNS FROM `tourism_data`.`...	OK	0.000 sec
38	15:28:41	PREPARE stmt FROM 'INSERT INTO `to...	OK	0.000 sec
39	15:28:49	DEALLOCATE PREPARE stmt	OK	0.000 sec
40	15:29:00	SELECT * FROM tourism_data.occupanc...	1000 row(s) returned	0.000 sec / 0.000 sec
41	15:31:12	SHOW SESSION VARIABLES LIKE 'lowe...	OK	0.000 sec
42	15:31:12	SHOW DATABASES	OK	0.000 sec
43	15:31:18	SHOW SESSION VARIABLES LIKE 'lowe...	OK	0.000 sec

FIGURE 5: MYSQL DATABASE

The is potential to use the MySQL query language to make the application code less complicated. The query language can be used to easily find meta data like max, min, or provide a unique subset of the data. This may result in more database requests from the client, and it is unclear if that is a problem or not.



```javascript
import Home from './Pages/Home';
import About from './Pages/About';
import Contact from './Pages/Contact';
import Register from "./Pages/Register";

import { AuthProvider, useAuth  } from "./AuthContext";

function App() {
  const { handleLogin, handleLogout } = useAuth();

  return (
    <BrowserRouter>
      <div>
        <Header handleLogout={handleLogout} handleLogin={handleLogin}/>
        <Container fluid style={{ padding: '0' }}>
          <Routes>
            <Route path='/' element={<Home/>}/>
            <Route path="/about" element={<About />} />
            <Route path="/contact" element={<Contact />} />
            <Route path="/register" element={<Register />} />
          </Routes>
        </Container>
        <Footer/>
      </div>
    </BrowserRouter>
  );
}

export default App;
```

```
[0] preflight check is successful
[0] Received chunk: {"username":"admin","password":"admin"}
[0] Complete request body: {"username":"admin","password":"admin"}
[0] Parsed username: admin Parsed password: admin
[0] User found: admin
[0] GET request made...
```

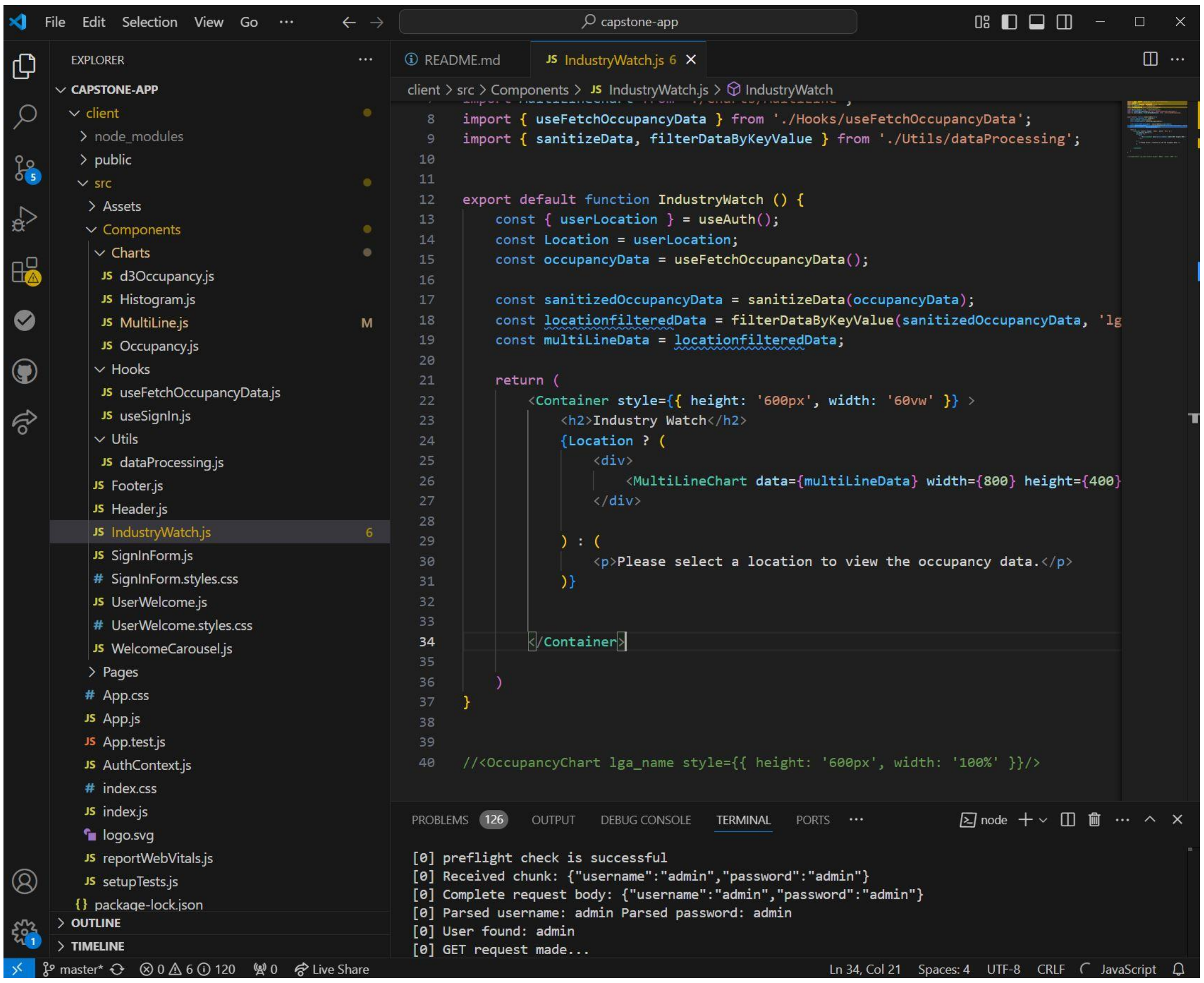
Figure 6: App.js

The client side of the application is the most complicated. It is mainly run from the App.js file under the 'src' folder. It calls on page files from the Pages folder, which call on component functions from the 'Components' folder.

The 'Assets' folder contains the images used in the app.

'AuthContext.js' contains the functions and state management related to user login.

'Index.js' the root of the application. It is called first. It then calls AuthContext.js, AuthContext wraps the entire app before App.js is called. This was necessary to provide authorisation related data and functions to the rest of the app.



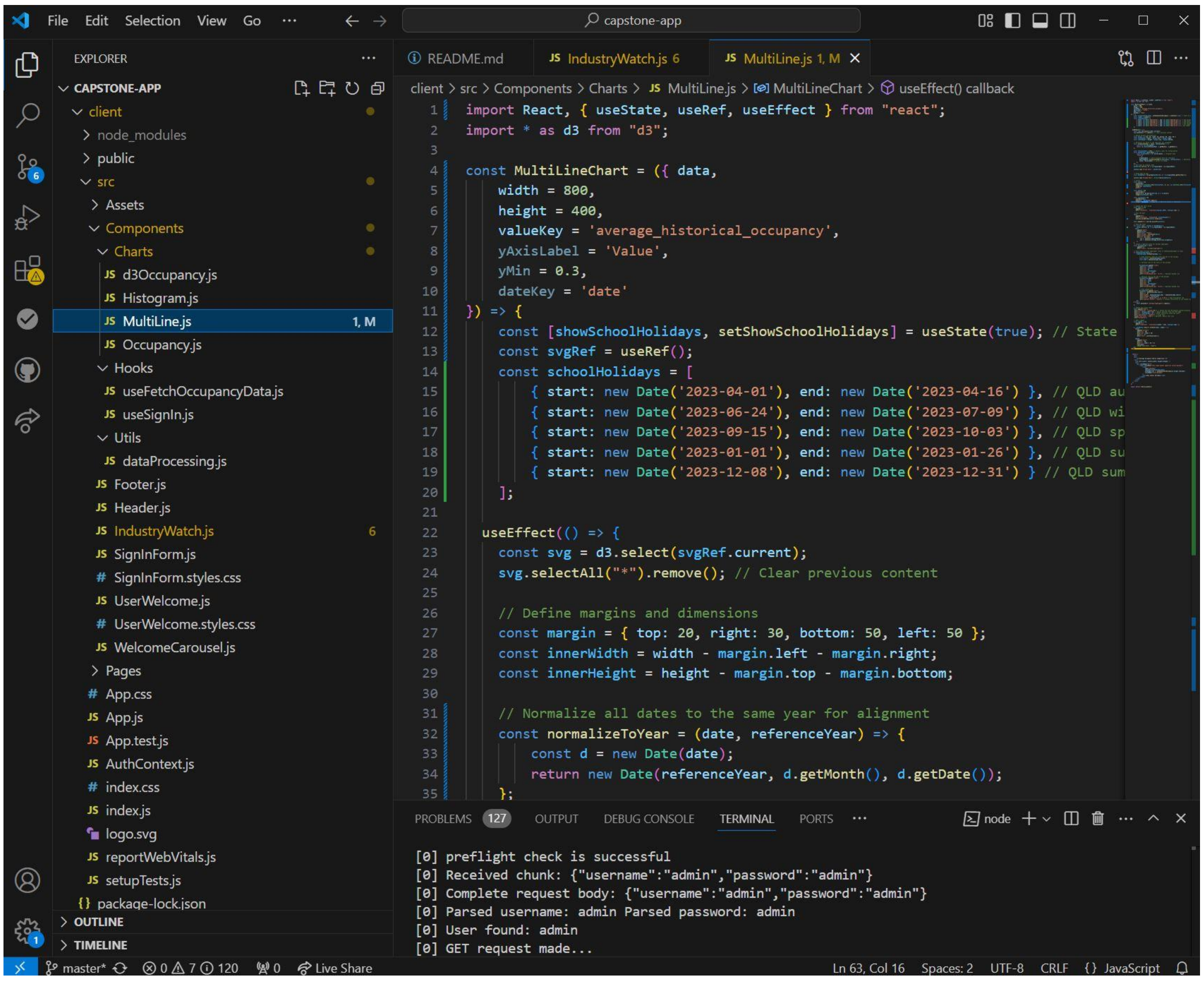
```
import { useFetchOccupancyData } from './Hooks/useFetchOccupancyData';
import { sanitizeData, filterDataByKeyValue } from './Utils/dataProcessing';


export default function IndustryWatch () {
    const { userLocation } = useAuth();
    const Location = userLocation;
    const occupancyData = useFetchOccupancyData();

    const sanitizedOccupancyData = sanitizeData(occupancyData);
    const locationfilteredData = filterDataByKeyValue(sanitizedOccupancyData, 'lg
    const multiLineData = locationfilteredData;

    return (
        <Container style={{ height: '600px', width: '60vw' }} >
            <h2>Industry Watch</h2>
            {Location ? (
                <div>
                    <MultiLineChart data={multiLineData} width={800} height={400}
                </div>

            ) : (
                <p>Please select a location to view the occupancy data.</p>
            )}


        </Container>

    )
}


//<OccupancyChart lga_name style={{ height: '600px', width: '100%' }}/>
```

```
[0] preflight check is successful
[0] Received chunk: {"username":"admin","password":"admin"}
[0] Complete request body: {"username":"admin","password":"admin"}
[0] Parsed username: admin Parsed password: admin
[0] User found: admin
[0] GET request made...
```

FIGURE 7: INDUSTRY WATCH AND COMPONENTS

The main feature of the app is centred on the 'IndustryWatch.js' file. It calls the 'MultiLineChart.js' which draws the chart on the users' homepage. The structure of this approach confusing for the developer as there a multiple nested layers of functions that return html/jsx code.

In sprint 1 'IndustryWatch' called 'Occupancy.js', this file was responsible for rendering the chart. Since the migration to D3 for sprint 2 'MultilineChart.js' is responsible for drawing the chart.



```
import React, { useState, useRef, useEffect } from "react";
import * as d3 from "d3";

const MultiLineChart = ({ data,
    width = 800,
    height = 400,
    valueKey = 'average_historical_occupancy',
    yAxisLabel = 'Value',
    yMin = 0.3,
    dateKey = 'date'
}) => {
    const [showSchoolHolidays, setShowSchoolHolidays] = useState(true); // State
    const svgRef = useRef();
    const schoolHolidays = [
        { start: new Date('2023-04-01'), end: new Date('2023-04-16') }, // QLD au
        { start: new Date('2023-06-24'), end: new Date('2023-07-09') }, // QLD wi
        { start: new Date('2023-09-15'), end: new Date('2023-10-03') }, // QLD sp
        { start: new Date('2023-01-01'), end: new Date('2023-01-26') }, // QLD su
        { start: new Date('2023-12-08'), end: new Date('2023-12-31') } // QLD sum
    ];

  useEffect(() => {
    const svg = d3.select(svgRef.current);
    svg.selectAll("*").remove(); // Clear previous content

    // Define margins and dimensions
    const margin = { top: 20, right: 30, bottom: 50, left: 50 };
    const innerWidth = width - margin.left - margin.right;
    const innerHeight = height - margin.top - margin.bottom;

    // Normalize all dates to the same year for alignment
    const normalizeToYear = (date, referenceYear) => {
        const d = new Date(date);
        return new Date(referenceYear, d.getMonth(), d.getDate());
    };
```

```
[0] preflight check is successful
[0] Received chunk: {"username":"admin","password":"admin"}
[0] Complete request body: {"username":"admin","password":"admin"}
[0] Parsed username: admin Parsed password: admin
[0] User found: admin
[0] GET request made...
```

FIGURE 8: MULTILINE CHART

'MultiLine.js' is responsible for drawing the chart with the third party D3 library. It is the largest function in the project. Currently the values for school holiday dates are written into the function, this is unideal and they should instead be passed into the function via a prop. These dates and others should be stored on the MySQL database, and fetch functions should be written for their retrieval.

MuliLine.js also returns html/jsx code. This can make adjusting the DOM confusing.

It would be possible, with skill and time, to write a reusable MuliLineChart function that can have the attribute data and highlight dates passed into it. This would be desirable as it would reduce the number of charting functions.

Currently the MultLineChart function handles most of the data processing, this maybe typical industry practice, but results in data processing occurring in multiple places. It would be better if different concerns were separated. Where the data processing should ideally occur is unclear.

Test Plan and Results

Manuel testing of the prototype is required. Below are the results.

Test cases	Expected Outcomes	Test Result
App start-up	Web page should be unsigned-in homepage, with nav bar, welcome carousal, and sign-in form present.	Pass
Welcome Carousal	Clicking to the left side or right of the image should change the image and the welcome message. 4 images and messages about the site should be available.	Pass
Sign In Form: Correct input credentials	Entering 'admin' to both username and password fields and clicking the sign in button should result in the user signing in. The page will change to display a personalised welcome message, 'IndustryWatch' feature and the blue Sign In tab in the top right should change to a red 'Logout' button.	Pass
Sign In Form: Incorrect input	A red message saying 'invalid credentials' should appear under the sign in button.	Pass
Users Homepage: Logout button	Once the user is logged in, clicking on the red logout button in the top right should log out the user returning them to the unsigned in Home page.	Pass
Admin default location	Welcome page location should default to Cairns for the user 'admin'	Pass
Location drop-down selector	Clicking on the drop down selector to should display a list of 4 locations: Noosa, GoldCoast, Whitsundays, Cairns. Clicking on one of the options should change the location to the selected location. The chart below should change to match the location. The default value of 'Cairns' for user 'admin' should remain the same upon logging	Pass

	out and in again.	
‘Show Holidays’ check box	Checking the check box under the chart should toggle the display of the holiday period highlights.	Pass
Sign In button top right	Clicking this button should move the sign in form to the top of the page.	Fail, feature not implemented.

More features were planned for Sprint 2 but were not implemented.

Project Development Reflection

Aspects of the project that went well include the correct functioning of features that were implemented. The main problem with the project was the lack of features implemented. The project team was only one person, and that person had little experience in web development, an unfortunate bout of illness made progress more difficult. Lack of experience resulted in some things taking longer than expected, leaving less time to develop other features.

The decision to change the charting library to D3, and redoing the main chart, has undoubtedly taken up time for little apparent change to the end product. It was concluded that it would be better to implement a charting system that could grow to meet the needs of the project rather than producing more charts with limited features and usefulness.

Similarly, the move to MySQL for database storage didn't change the outward appearance of the final app, but it did greatly improve the quality of its internal workings. MySQL is a much more professional solution to data storage than simply storying it in the server's folder. If the project was to expand its databases size and complexity, this will be easily accommodated by MySQL.

It was disappointing to not implement the storage of user credentials to the MySQL platform. This along with story details on user marketing campaign dates, would have showcased the utility of the database. Building server routes to access this data would have been good application of techniques learnt in 'Advanced Web Development'.

One of the most difficult aspects of the project was its folder structure. Multiple nested functions returned html/jsx code. The resultant web page did not appear complex, but was difficult to navigate and understand throughout development. In future, projects that have limited scope should try to only have one html per page.

Other aspects like hooks(useEffect) and promises(.then) are not confidently understood. As such they are probably not being used to their full potential.

Limited progress was made with styling and design layout. This was mainly because limited features were implemented and there was indecision about what features would be useful for the user. It maybe better to view the main features/feature set of the app as a 'dash board' of sorts and take lessons from finance and stock market applications for clues in how to layout a detail rich application.

One of the major benefits of being in a one-person team is the full range of experience you gain from doing all parts of the project. More progress could have been made with more team members, but this would result less exposure and coding practice across a wide variety of fields.

If this project was to continue, I would further study the possibilities of D3 and then showcase its charting features with mock data. Storing this data with MySQL and fetching it with a selection of server endpoints would be the preferred outcome.

References

[1] "Installation - Tailwind CSS," Tailwind CSS. [Online]. Available: https://tailwindcss.com/docs/installation [Accessed: Dec. 13, 2024].

[2] "Introduction - Primer," Primer Style. [Online]. Available: https://primer.style/guides/introduction [Accessed: Dec. 13, 2024].

[3] "JavaScript Graphing Library | Plotly," Plotly. [Online]. Available: https://plotly.com/javascript/ [Accessed: Dec. 13, 2024].

[4] "Chart.js Documentation," Chart.js. [Online]. Available: https://www.chartjs.org/docs/latest/ [Accessed: Dec. 13, 2024].

[5] "D3 Gallery," Observable. [Online]. Available: https://observablehq.com/@d3/gallery [Accessed: Dec. 13, 2024].

[6] "AG Grid Charts Overview," AG Grid. [Online]. Available: https://ag-grid.com/charts/ [Accessed: Dec. 13, 2024].

[7] "Welcome to React-Vis," React-Vis. [Online]. Available: https://uber.github.io/react-vis/documentation/welcome-to-react-vis [Accessed: Dec. 13, 2024].

[8] "Recharts," Recharts. [Online]. Available: https://recharts.org/en-US [Accessed: Dec. 13, 2024].

[9] "Apache ECharts," Apache ECharts. [Online]. Available: https://echarts.apache.org/en/index.html [Accessed: Dec. 13, 2024].

[10] "Graphing Libraries | Plotly," Plotly. [Online]. Available: https://plotly.com/graphing-libraries/ [Accessed: Dec. 20, 2024].