CN5006 Portfolio

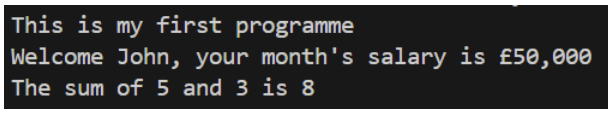
Antony Buck – U2223218

Table of Contents

Week 1

Exercise 1:

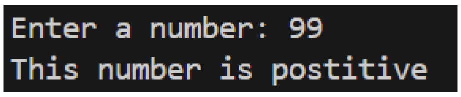
Output:



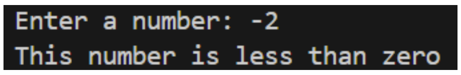
```
This is my first programme
Welcome John, your month's salary is £50,000
The sum of 5 and 3 is 8
```

Exercise 2:

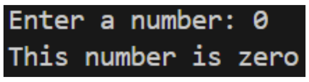
Output:



```
Enter a number: 99
This number is postitive
```



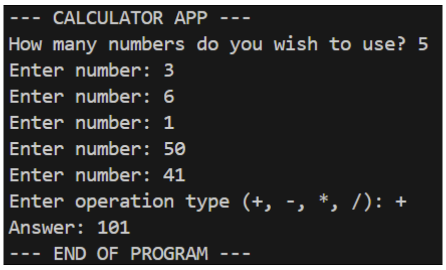
```
Enter a number: -2
This number is less than zero
```



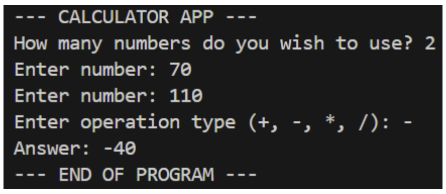
```
Enter a number: 0
This number is zero
```

Exercise 3 – Calculator Program:

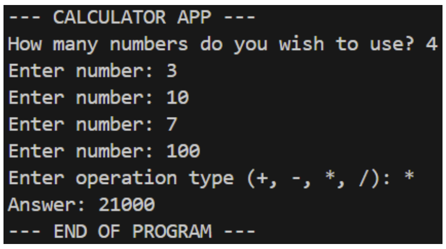
Output:



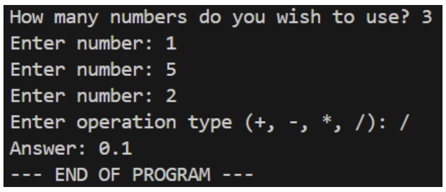
```
--- CALCULATOR APP ---
How many numbers do you wish to use? 5
Enter number: 3
Enter number: 6
Enter number: 1
Enter number: 50
Enter number: 41
Enter operation type (+, -, *, /): +
Answer: 101
--- END OF PROGRAM ---
```



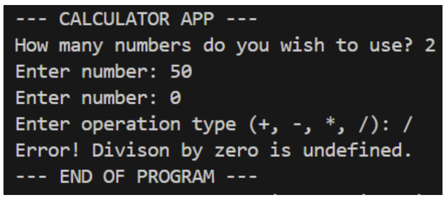
```
--- CALCULATOR APP ---
How many numbers do you wish to use? 2
Enter number: 70
Enter number: 110
Enter operation type (+, -, *, /): -
Answer: -40
--- END OF PROGRAM ---
```



```
--- CALCULATOR APP ---
How many numbers do you wish to use? 4
Enter number: 3
Enter number: 10
Enter number: 7
Enter number: 100
Enter operation type (+, -, *, /): *
Answer: 21000
--- END OF PROGRAM ---
```



```
How many numbers do you wish to use? 3
Enter number: 1
Enter number: 5
Enter number: 2
Enter operation type (+, -, *, /): /
Answer: 0.1
--- END OF PROGRAM ---
```



```
--- CALCULATOR APP ---
How many numbers do you wish to use? 2
Enter number: 50
Enter number: 0
Enter operation type (+, -, *, /): /
Error! Divison by zero is undefined.
--- END OF PROGRAM ---
```

Reflection

Exercise 1 taught me how to print text to a terminal using console.log, how to declare, initialize, and assign integer values to constants, and how to use basic addition. Exercise 2 taught me how to get integer input from a user using a prompt, and how to use conditionals – if, else if, and else.

For exercise 3, I decided to go beyond the basic requirements and create a calculator that can take as many operands as the user desires. I was able to achieve this with the use of an array. I learned that in JavaScript an array's length is not fixed and can thus take as many number of values as is needed. I was able to assign the user's input to each element space in the array using a for loop.

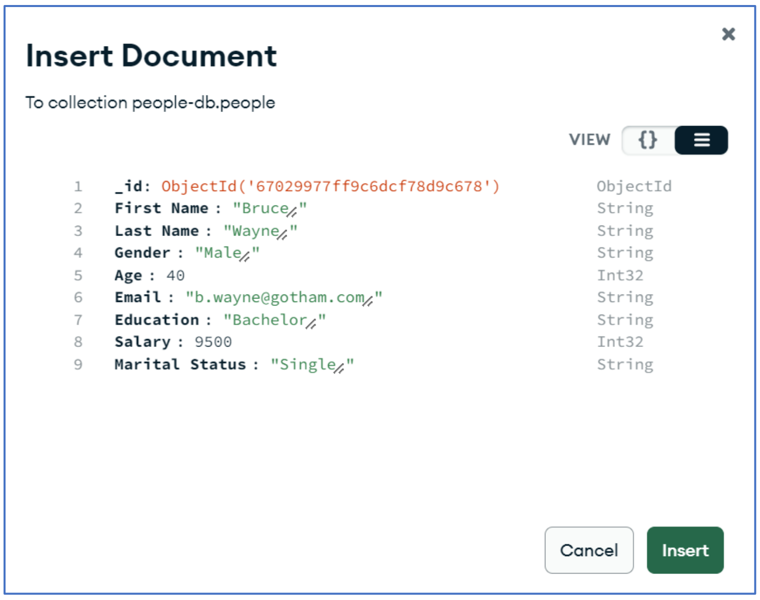
With the user choosing an operator, the best solution to applying said operator to the user's chosen numbers was with a switch statement. I had to pay special attention to each case as the initial answer value had to be different depending on the operator chosen. For example, the multiplicative case required that the first value was set to 1; and the subtraction case required that the first value was set to the first index value in the array. I also implemented an escape encase of the division-by-zero scenario.

Lastly, I was pleasantly surprised to find out that the basic syntax of JavaScript is similar to that of Java, which made it much easier to learn.

[NEXT PAGE]

Week 2

MongoDB Compass: Inserting a Document:

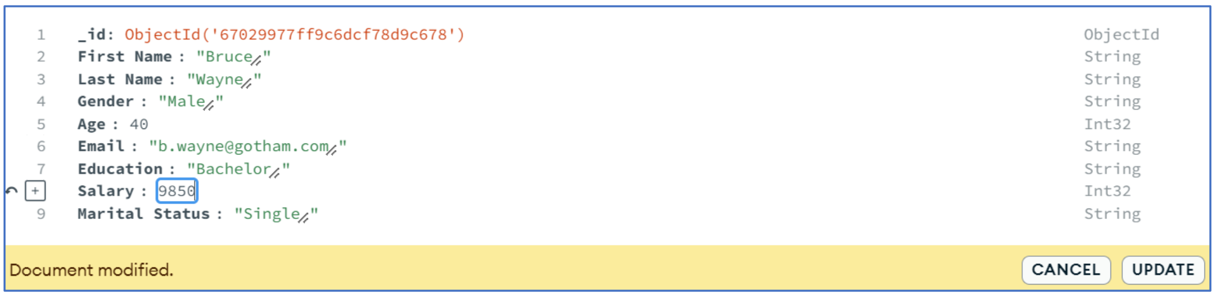


```
_id: ObjectId('67029977ff9c6dcf78d9c678')          ObjectId
First Name : "Bruce"                               String
Last Name : "Wayne"                                String
Gender : "Male"                                    String
Age : 40                                           Int32
Email : "b.wayne@gotham.com"                       String
Education : "Bachelor"                             String
Salary : 9500                                      Int32
Marital Status : "Single"                          String
```

Explanation:

I used the MongoDB Compass GUI to insert a document. The list view also further simplified the process of adding new fields and values.

MongoDB Compass: Updating a Document:

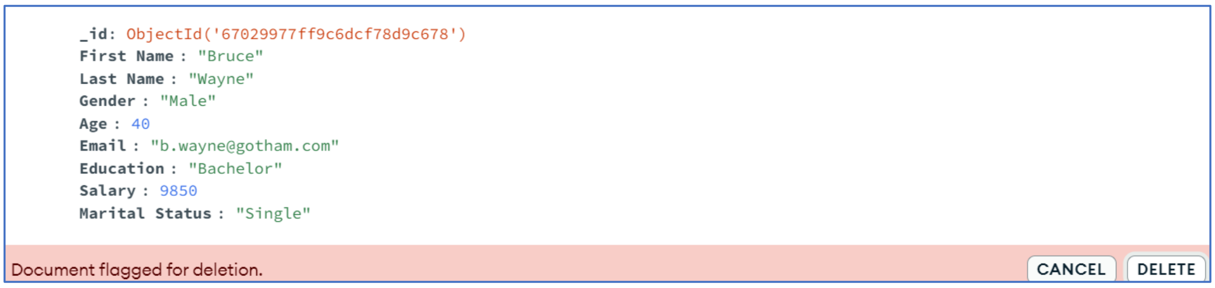


```
_id: ObjectId('67029977ff9c6dcf78d9c678')          ObjectId
First Name : "Bruce"                               String
Last Name : "Wayne"                                String
Gender : "Male"                                    String
Age : 40                                           Int32
Email : "b.wayne@gotham.com"                       String
Education : "Bachelor"                             String
Salary : 9850                                      Int32
Marital Status : "Single"                          String
```

Explanation:

I updated the salary field of one record.

MongoDB Compass: Deleting a Document:

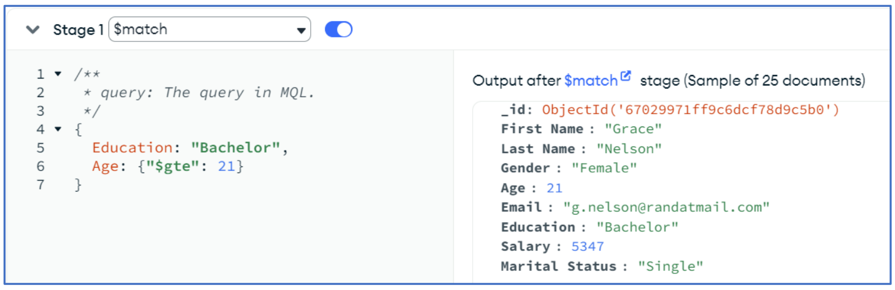


```
_id: ObjectId('67029977ff9c6dcf78d9c678')
First Name : "Bruce"
Last Name : "Wayne"
Gender : "Male"
Age : 40
Email : "b.wayne@gotham.com"
Education : "Bachelor"
Salary : 9850
Marital Status : "Single"
```

Explanation:

I deleted one record from the database.

MongoDB Compass: Aggregate Pipe Line:

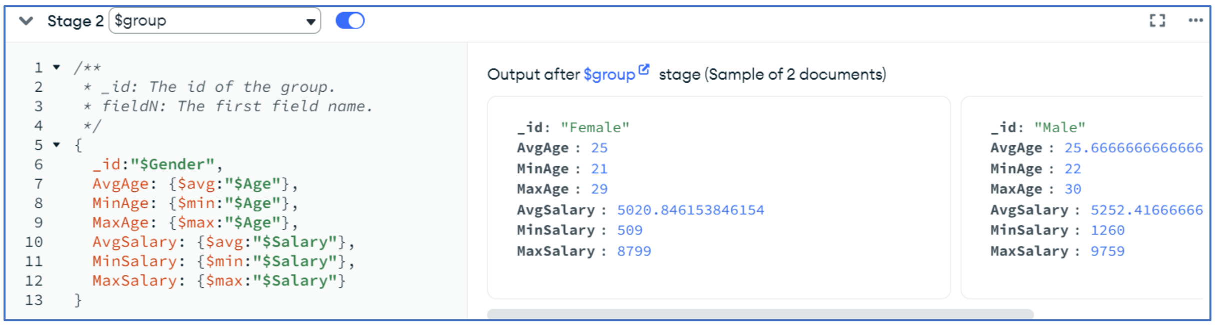


Stage 1 $match

```
/**
 * query: The query in MQL.
 */
{
  Education: "Bachelor",
  Age: {"$gte": 21}
}
```

Output after $match stage (Sample of 25 documents)

```
_id: ObjectId('67029971ff9c6dcf78d9c5b0')
First Name : "Grace"
Last Name : "Nelson"
Gender : "Female"
Age : 21
Email : "g.nelson@randatmail.com"
Education : "Bachelor"
Salary : 5347
Marital Status : "Single"
```



Stage 2 $group

```
/**
 * _id: The id of the group.
 * fieldN: The first field name.
 */
{
  _id:"$Gender",
  AvgAge: {$avg:"$Age"},
  MinAge: {$min:"$Age"},
  MaxAge: {$max:"$Age"},
  AvgSalary: {$avg:"$Salary"},
  MinSalary: {$min:"$Salary"},
  MaxSalary: {$max:"$Salary"}
}
```

Output after $group stage (Sample of 2 documents)

```
_id: "Female"
AvgAge : 25
MinAge : 21
MaxAge : 29
AvgSalary : 5020.846153846154
MinSalary : 509
MaxSalary : 8799
```

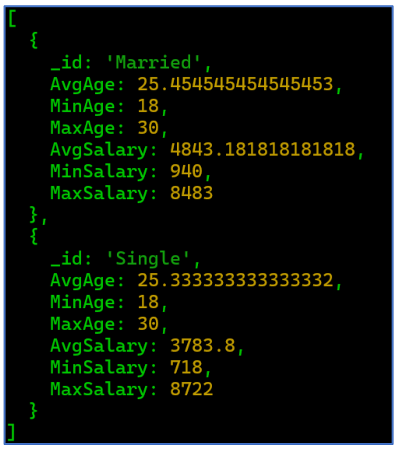
```
_id: "Male"
AvgAge : 25.6666666666666
MinAge : 22
MaxAge : 30
AvgSalary : 5252.41666666
MinSalary : 1260
MaxSalary : 9759
```

Explanation:

I performed the aggregate pipeline on the collection of records to produce the desired results. I now understand that the pipeline refers to the stages involved in producing the aggregation. The first stage required that I filter records based on bachelor education and an age of 21 or greater. The second stage required that I perform a series of functions to produce the averages, minimum values, and maximum values of age and salary within a gender grouping.

MongoDB Shell Task 1:

Output:



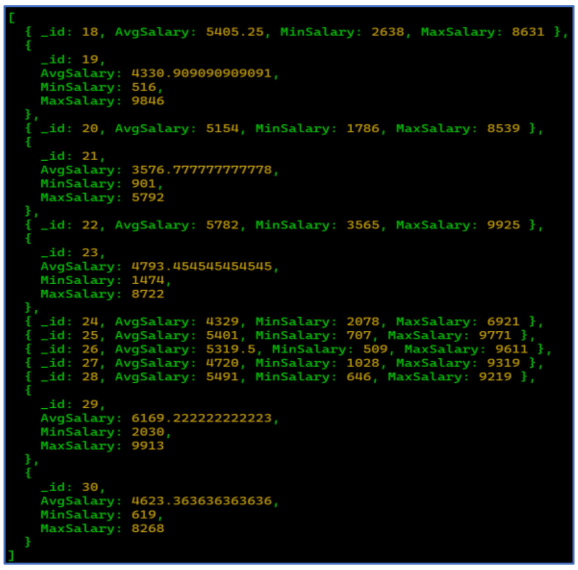
```
[
  {
    _id: 'Married',
    AvgAge: 25.454545454545453,
    MinAge: 18,
    MaxAge: 30,
    AvgSalary: 4843.181818181818,
    MinSalary: 940,
    MaxSalary: 8483
  },
  {
    _id: 'Single',
    AvgAge: 25.333333333333332,
    MinAge: 18,
    MaxAge: 30,
    AvgSalary: 3783.8,
    MinSalary: 718,
    MaxSalary: 8722
  }
]
```

Explanation:

I first filtered the records based on a master education using the $match operation. Next I used $group operation to produce two documents from the values of marital status. I performed the various averages, minimum values, and maximum values functions on the values of age and salary.

MongoDB Shell Task 2:

Output:



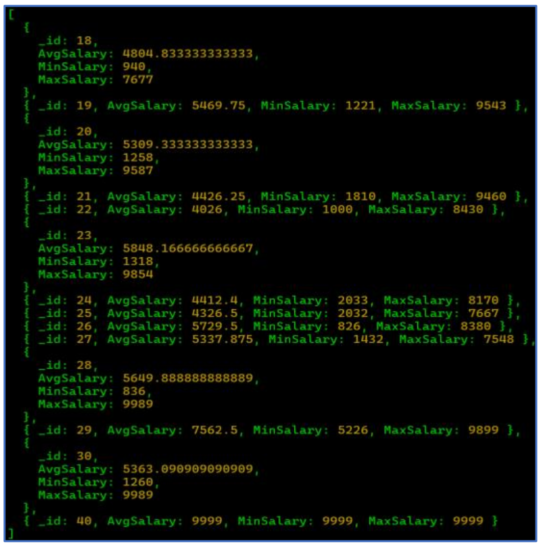
```
[
  { _id: 18, AvgSalary: 5405.25, MinSalary: 2638, MaxSalary: 8631 },
  {
    _id: 19,
    AvgSalary: 4330.909090909091,
    MinSalary: 516,
    MaxSalary: 9846
  },
  { _id: 20, AvgSalary: 5154, MinSalary: 1786, MaxSalary: 8539 },
  {
    _id: 21,
    AvgSalary: 3576.777777777778,
    MinSalary: 901,
    MaxSalary: 5792
  },
  { _id: 22, AvgSalary: 5782, MinSalary: 3565, MaxSalary: 9925 },
  {
    _id: 23,
    AvgSalary: 4793.454545454545,
    MinSalary: 1474,
    MaxSalary: 8722
  },
  { _id: 24, AvgSalary: 4329, MinSalary: 2078, MaxSalary: 6921 },
  { _id: 25, AvgSalary: 5401, MinSalary: 707, MaxSalary: 9771 },
  { _id: 26, AvgSalary: 5319.5, MinSalary: 509, MaxSalary: 9611 },
  { _id: 27, AvgSalary: 4720, MinSalary: 1028, MaxSalary: 9319 },
  { _id: 28, AvgSalary: 5491, MinSalary: 646, MaxSalary: 9219 },
  {
    _id: 29,
    AvgSalary: 6169.222222222223,
    MinSalary: 2030,
    MaxSalary: 9913
  },
  {
    _id: 30,
    AvgSalary: 4623.363636363636,
    MinSalary: 619,
    MaxSalary: 8268
  }
]
```

Explanation:

I first filtered records by the female gender, next I grouped by age value and performed average, minimum, and maximum functions on salary, and lastly I sorted by age in ascending order.

MongoDB Shell Task 3:

Output:



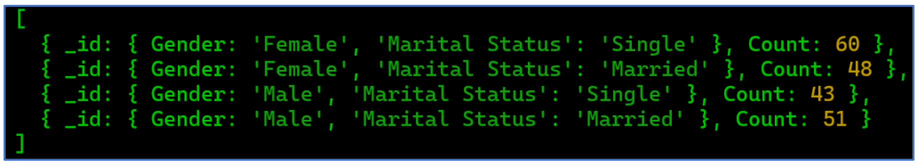
```
[
  {
    _id: 18,
    AvgSalary: 4804.833333333333,
    MinSalary: 940,
    MaxSalary: 7677
  },
  { _id: 19, AvgSalary: 5469.75, MinSalary: 1221, MaxSalary: 9543 },
  {
    _id: 20,
    AvgSalary: 5309.333333333333,
    MinSalary: 1258,
    MaxSalary: 9587
  },
  { _id: 21, AvgSalary: 4426.25, MinSalary: 1810, MaxSalary: 9460 },
  { _id: 22, AvgSalary: 4026, MinSalary: 1000, MaxSalary: 8430 },
  {
    _id: 23,
    AvgSalary: 5848.166666666667,
    MinSalary: 1318,
    MaxSalary: 9854
  },
  { _id: 24, AvgSalary: 4412.4, MinSalary: 2033, MaxSalary: 8170 },
  { _id: 25, AvgSalary: 4326.5, MinSalary: 2032, MaxSalary: 7667 },
  { _id: 26, AvgSalary: 5729.5, MinSalary: 826, MaxSalary: 8380 },
  { _id: 27, AvgSalary: 5337.875, MinSalary: 1432, MaxSalary: 7548 },
  {
    _id: 28,
    AvgSalary: 5649.888888888889,
    MinSalary: 836,
    MaxSalary: 9989
  },
  { _id: 29, AvgSalary: 7562.5, MinSalary: 5226, MaxSalary: 9899 },
  {
    _id: 30,
    AvgSalary: 5363.090909090909,
    MinSalary: 1260,
    MaxSalary: 9989
  },
  { _id: 40, AvgSalary: 9999, MinSalary: 9999, MaxSalary: 9999 }
]
```

Explanation:

This task was the exact same as the last except I filtered by male instead of female.

MongoDB Shell Task 4:

Output:



```
[
  { _id: { Gender: 'Female', 'Marital Status': 'Single' }, Count: 60 },
  { _id: { Gender: 'Female', 'Marital Status': 'Married' }, Count: 48 },
  { _id: { Gender: 'Male', 'Marital Status': 'Single' }, Count: 43 },
  { _id: { Gender: 'Male', 'Marital Status': 'Married' }, Count: 51 }
]
```

Explanation:

I first grouped by two IDs of gender and marital status, followed by a count using the sum function. The result was four distinct outputs with a total.

Reflection

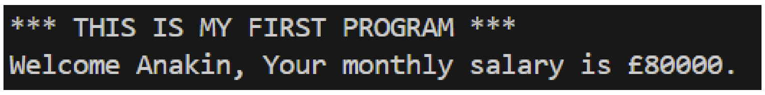
This week gave me a basic understanding of non-SQL document-based databases. I gained some proficiency in MongoDB and it’s respective software – MongoDB Compass and Mongo shell. I learned how data is structed within non-SQL using collections and key-value records, and how the data is in the form of JSON and is thus less constrained than relational data. I learned how to import, add, update, delete, find, and aggregate records using MongoDB Compass and shell. The shell required that I learn various commands to achieve the desired results.

[NEXT PAGE]

Week 3

Exercise 1:

Output:



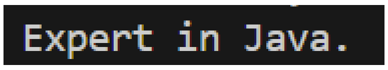
```
*** THIS IS MY FIRST PROGRAM ***
Welcome Anakin, Your monthly salary is £80000.
```

Explanation:

I defined a function with two parameters that printed a message and then called it.

Exercise 2:

Output:



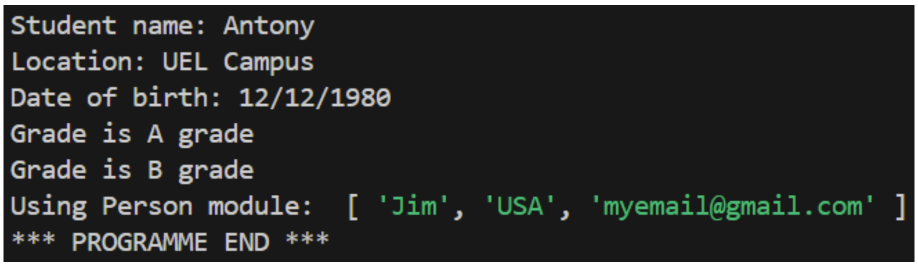
```
Expert in Java.
```

Explanation:

I created an arrow function with one parameter that printed a message and then called it.

Exercise 3:

Output:



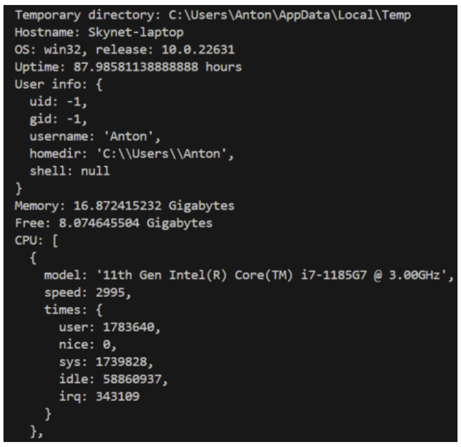
```
Student name: Antony
Location: UEL Campus
Date of birth: 12/12/1980
Grade is A grade
Grade is B grade
Using Person module:  [ 'Jim', 'USA', 'myemail@gmail.com' ]
*** PROGRAMME END ***
```

Explanation:

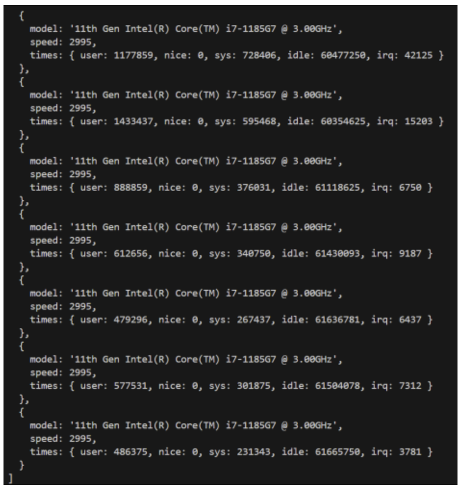
I created two additional JS files, exported the various functions and class within, and then imported them into the index file. I called the various functions and printed the results to the console.

Exercise 4:

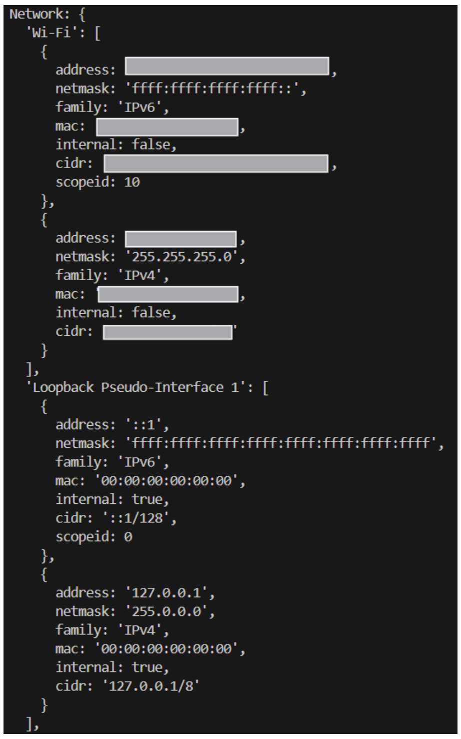
Output:



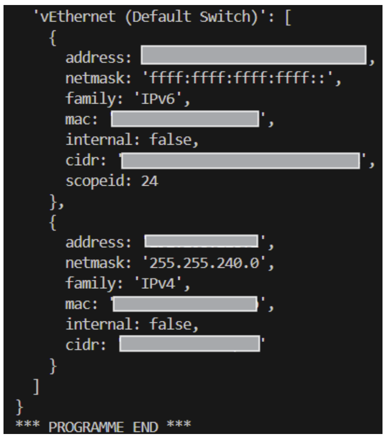
```
Temporary directory: C:\Users\Anton\AppData\Local\Temp
Hostname: Skynet-laptop
OS: win32, release: 10.0.22631
Uptime: 87.98581138888888 hours
User info: {
  uid: -1,
  gid: -1,
  username: 'Anton',
  homedir: 'C:\\Users\\Anton',
  shell: null
}
Memory: 16.872415232 Gigabytes
Free: 8.074645504 Gigabytes
CPU: [
  {
    model: '11th Gen Intel(R) Core(TM) i7-1185G7 @ 3.00GHz',
    speed: 2995,
    times: {
      user: 1783640,
      nice: 0,
      sys: 1739828,
      idle: 58860937,
      irq: 343109
    }
  },
```



```
  {
    model: '11th Gen Intel(R) Core(TM) i7-1185G7 @ 3.00GHz',
    speed: 2995,
    times: { user: 1177859, nice: 0, sys: 728406, idle: 60477250, irq: 42125 }
  },
  {
    model: '11th Gen Intel(R) Core(TM) i7-1185G7 @ 3.00GHz',
    speed: 2995,
    times: { user: 1433437, nice: 0, sys: 595468, idle: 60354625, irq: 15203 }
  },
  {
    model: '11th Gen Intel(R) Core(TM) i7-1185G7 @ 3.00GHz',
    speed: 2995,
    times: { user: 888859, nice: 0, sys: 376031, idle: 61118625, irq: 6750 }
  },
  {
    model: '11th Gen Intel(R) Core(TM) i7-1185G7 @ 3.00GHz',
    speed: 2995,
    times: { user: 612656, nice: 0, sys: 340750, idle: 61430093, irq: 9187 }
  },
  {
    model: '11th Gen Intel(R) Core(TM) i7-1185G7 @ 3.00GHz',
    speed: 2995,
    times: { user: 479296, nice: 0, sys: 267437, idle: 61636781, irq: 6437 }
  },
  {
    model: '11th Gen Intel(R) Core(TM) i7-1185G7 @ 3.00GHz',
    speed: 2995,
    times: { user: 577531, nice: 0, sys: 301875, idle: 61504078, irq: 7312 }
  },
  {
    model: '11th Gen Intel(R) Core(TM) i7-1185G7 @ 3.00GHz',
    speed: 2995,
    times: { user: 486375, nice: 0, sys: 231343, idle: 61665750, irq: 3781 }
  }
]
```



```
Network: {
  'Wi-Fi': [
    {
      address: [redacted],
      netmask: 'ffff:ffff:ffff:ffff::',
      family: 'IPv6',
      mac: [redacted],
      internal: false,
      cidr: [redacted],
      scopeid: 10
    },
    {
      address: [redacted],
      netmask: '255.255.255.0',
      family: 'IPv4',
      mac: [redacted],
      internal: false,
      cidr: [redacted]'
    }
  ],
  'Loopback Pseudo-Interface 1': [
    {
      address: '::1',
      netmask: 'ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff',
      family: 'IPv6',
      mac: '00:00:00:00:00:00',
      internal: true,
      cidr: '::1/128',
      scopeid: 0
    },
    {
      address: '127.0.0.1',
      netmask: '255.0.0.0',
      family: 'IPv4',
      mac: '00:00:00:00:00:00',
      internal: true,
      cidr: '127.0.0.1/8'
    }
  ],
```



```
  'vEthernet (Default Switch)': [
    {
      address: [redacted],
      netmask: 'ffff:ffff:ffff:ffff::',
      family: 'IPv6',
      mac: [redacted]',
      internal: false,
      cidr: '[redacted]',
      scopeid: 24
    },
    {
      address: [redacted]',
      netmask: '255.255.240.0',
      family: 'IPv4',
      mac: '[redacted]',
      internal: false,
      cidr: [redacted]'
    }
  ]
}
*** PROGRAMME END ***
```

Explanation:

For this exercise, I used the core modules os and util to gather and print system information to the console.

Reflection:

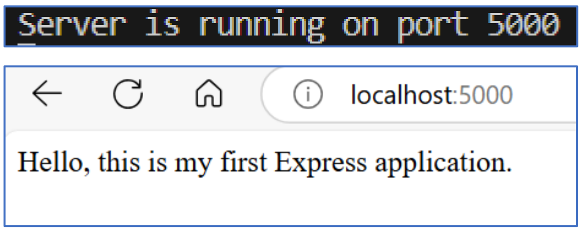
This week I learned how to construct and call both regular and arrow functions in JavaScript. I also learned about modules: how to create local ones and access core ones. In particular, the keyword export is used to define which code is to be accessible outside of the native file. I acquired knowledge on the various methods of the core os module which I then put into practice to print various system information about my computer. Overall, I believe this week has helped me to acquire a better understanding of how JavaScript code is organised, managed, and distributed across many applications.

[NEXT PAGE]

Week 4

Exercise 1:

Output:



```
Server is running on port 5000
```

localhost:5000

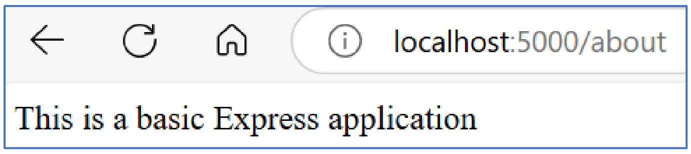
Hello, this is my first Express application.

Explanation:

I established the default route and sent a string message using the .get route method and its parameter res.

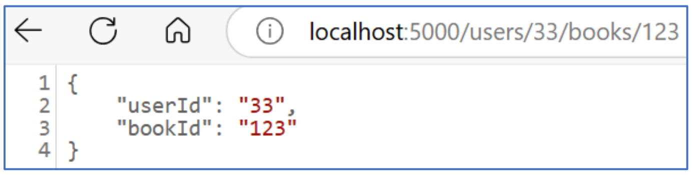
Exercise 2:

Output:



localhost:5000/about

This is a basic Express application



localhost:5000/users/33/books/123

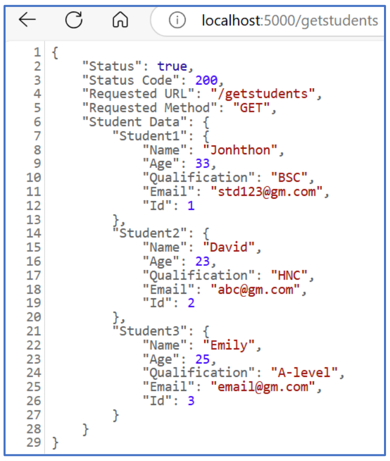
```
{
    "userId": "33",
    "bookId": "123"
}
```

Explanation:

I created additional routes: /about and /users… The second route included parameters which I accessed with the use of req.params.

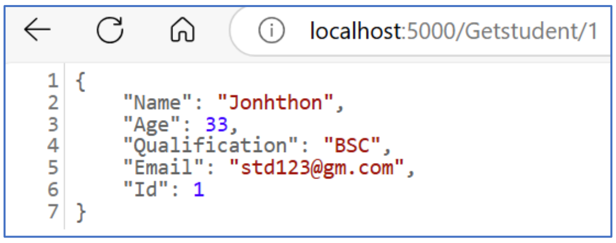
Exercise 3:

Output:

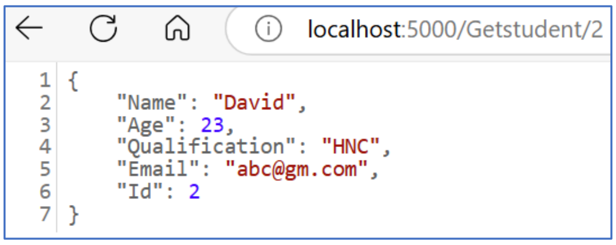


localhost:5000/getstudents

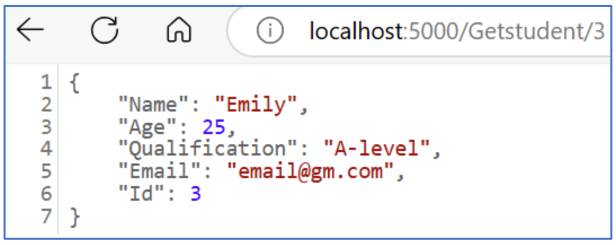
```
{
    "Status": true,
    "Status Code": 200,
    "Requested URL": "/getstudents",
    "Requested Method": "GET",
    "Student Data": {
        "Student1": {
            "Name": "Jonhthon",
            "Age": 33,
            "Qualification": "BSC",
            "Email": "std123@gm.com",
            "Id": 1
        },
        "Student2": {
            "Name": "David",
            "Age": 23,
            "Qualification": "HNC",
            "Email": "abc@gm.com",
            "Id": 2
        },
        "Student3": {
            "Name": "Emily",
            "Age": 25,
            "Qualification": "A-level",
            "Email": "email@gm.com",
            "Id": 3
        }
    }
}
```


localhost:5000/Getstudent/1

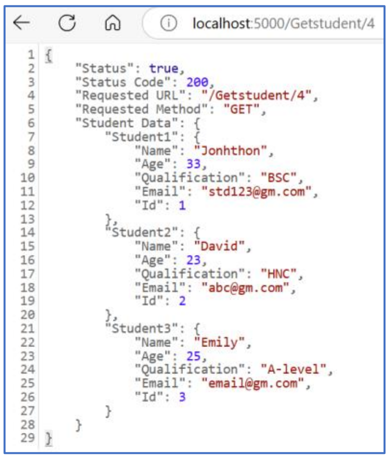
```
{
    "Name": "Jonhthon",
    "Age": 33,
    "Qualification": "BSC",
    "Email": "std123@gm.com",
    "Id": 1
}
```


localhost:5000/Getstudent/2

```
{
    "Name": "David",
    "Age": 23,
    "Qualification": "HNC",
    "Email": "abc@gm.com",
    "Id": 2
}
```


localhost:5000/Getstudent/3

```
{
    "Name": "Emily",
    "Age": 25,
    "Qualification": "A-level",
    "Email": "email@gm.com",
    "Id": 3
}
```


localhost:5000/Getstudent/4

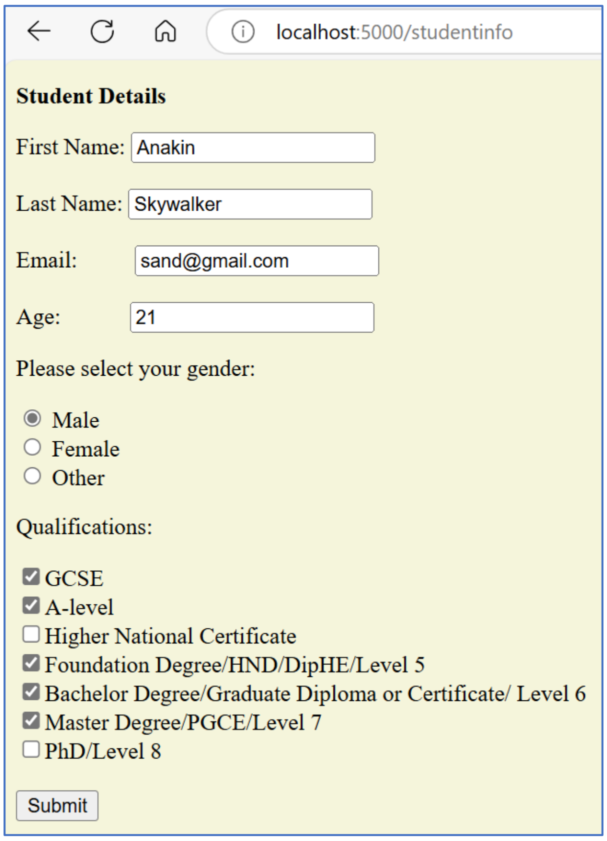
```
{
    "Status": true,
    "Status Code": 200,
    "Requested URL": "/Getstudent/4",
    "Requested Method": "GET",
    "Student Data": {
        "Student1": {
            "Name": "Jonhthon",
            "Age": 33,
            "Qualification": "BSC",
            "Email": "std123@gm.com",
            "Id": 1
        },
        "Student2": {
            "Name": "David",
            "Age": 23,
            "Qualification": "HNC",
            "Email": "abc@gm.com",
            "Id": 2
        },
        "Student3": {
            "Name": "Emily",
            "Age": 25,
            "Qualification": "A-level",
            "Email": "email@gm.com",
            "Id": 3
        }
    }
}
```

Explanation:

I created another route that reads and parses a JSON file and sends it to the browser using the response object. I then created another route with an id parameter which filtered out the record with the matching id. The last screenshot shows all records if that particular id does not match an existing record.

Exercise 4:

Output:



localhost:5000/studentinfo

Student Details

First Name: Anakin

Last Name: Skywalker

Email: sand@gmail.com

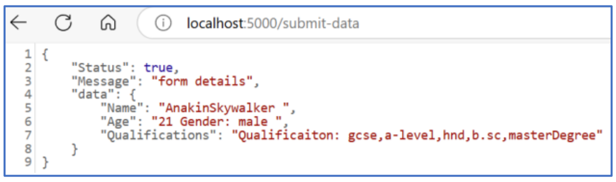
Age: 21

Please select your gender:

- [✓ Male]
- Female
- Other

Qualifications:

- [✓ GCSE]
- [✓ A-level]
- Higher National Certificate
- [✓ Foundation Degree/HND/DipHE/Level 5]
- [✓ Bachelor Degree/Graduate Diploma or Certificate/ Level 6]
- [✓ Master Degree/PGCE/Level 7]
- PhD/Level 8



localhost:5000/submit-data

```
{
    "Status": true,
    "Message": "form details",
    "data": {
        "Name": "AnakinSkywalker ",
        "Age": "21 Gender: male ",
        "Qualifications": "Qualificaiton: gcse,a-level,hnd,b.sc,masterDegree"
    }
}
```

Explanation:

I created an html document, sent it to a route, and enabled data submission with the use of the post method.

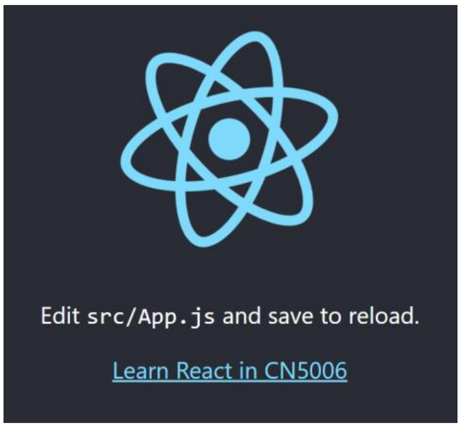
Reflection:

This week, I learned the theory behind creating an Express server, including the concepts of middleware, request/response cycle, and HTTP headers. I was able to create an Express server using an instance of the express module which I called app. I used .use, .get, and .post to set routes, filter results by URL parameters, send and manipulate response and request objects, and submit JSON data via a HTTP form.

Week 5

Exercise 1:

Output:



Edit `src/App.js` and save to reload.

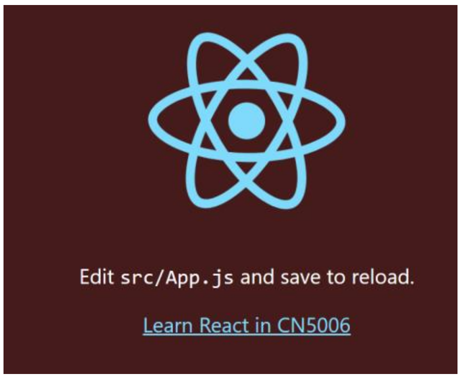
Learn React in CN5006

Explanation:

I modified the anchor HTML element in the App.js file to display a new message.

Exercise 2:

Output:



Edit `src/App.js` and save to reload.

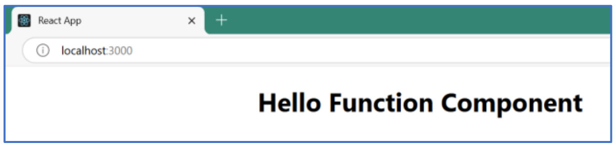
Learn React in CN5006

Explanation:

I changed the background colour within the App.css file.

Exercise 3:

Output:



React App

localhost:3000

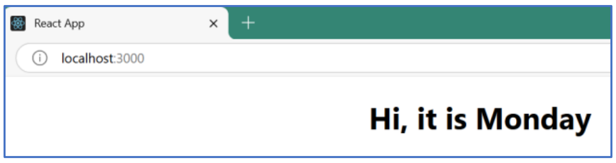
Hello Function Component

Explanation:

I created my own React component and imported it into index.js file. The component's contents were rendered in the .render() method.

Exercise 4:

Output:



React App

localhost:3000

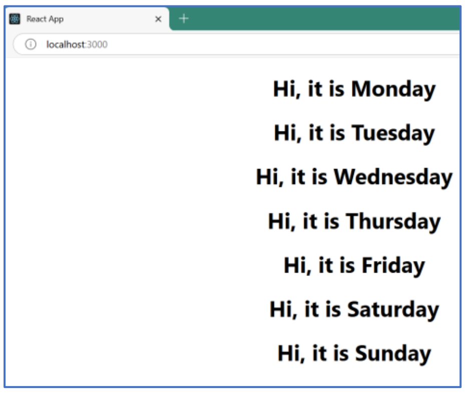
Hi, it is Monday

Explanation:

I created another React component, however, this one included a prop. When the component is called in the .render() method, the prop is passed as an argument and a String message is displayed.

Exercise 5:

Output:



React App

localhost:3000

Hi, it is Monday

Hi, it is Tuesday

Hi, it is Wednesday

Hi, it is Thursday

Hi, it is Friday

Hi, it is Saturday

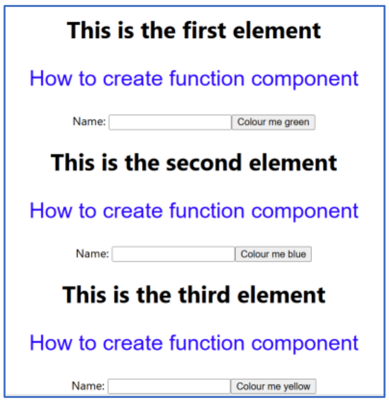
Hi, it is Sunday

Explanation:

I simply added an additional six lines of code and, as a result, called the functional component six additional times in the .render() method to display each day of the week.

Exercise 6:

Output:



This is the first element

How to create function component

Name: Colour me green

This is the second element

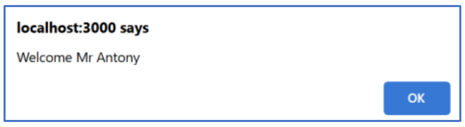
How to create function component

Name: Colour me blue

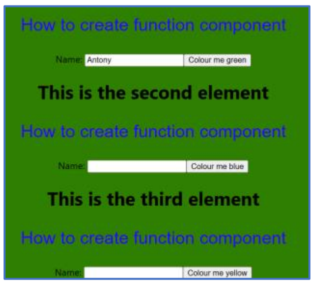
This is the third element

How to create function component

Name: Colour me yellow


localhost:3000 says

Welcome Mr Antony


How to create function component

Name: Antony

This is the second element

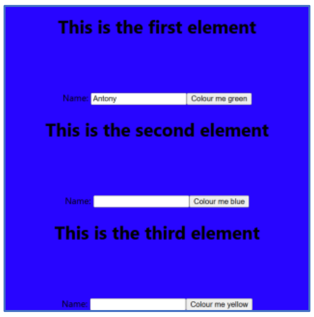
How to create function component

Name:

This is the third element

How to create function component

Name:


This is the first element

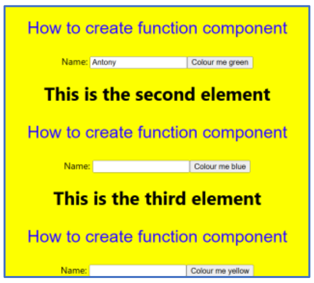
Name: Antony

This is the second element

Name:

This is the third element

Name:


How to create function component

Name: Antony

This is the second element

How to create function component

Name:

This is the third element

How to create function component

Name:

Explanation:

I created a new component which displays HTML text and input elements. The properties heading, name, and color were passed as various arguments within the .render() method to add functionality to the interface. Clicking the button calls a function that both alerts the user with a concatenation of "Mr " and the user's name and changes the background colour of the page.

Questions:

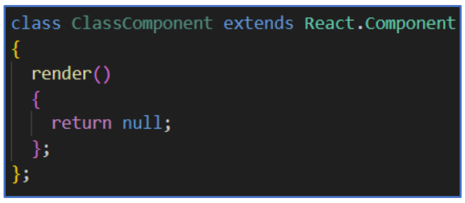
1. What is React?

   Ans: React is a JavaScript library used to develop responsive user interfaces for single-page applications. It uses a virtual DOM in memory to enable quick and responsive rendering of elements.

2. What do you understand about React components and what command must you use to create a React component with or without properties?

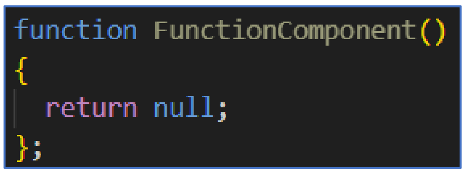
   Ans: React components are functions that return HTML elements. There are two ways to create a component: class and function.

Class:



```
class ClassComponent extends React.Component
{
  render()
  {
    return null;
  };
};
```

Function:

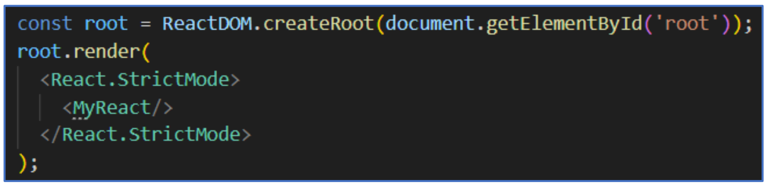


```
function FunctionComponent()
{
  return null;
};
```

Properties are defined in the parentheses.

3. What command do you use to render the newly created component named MyReact?

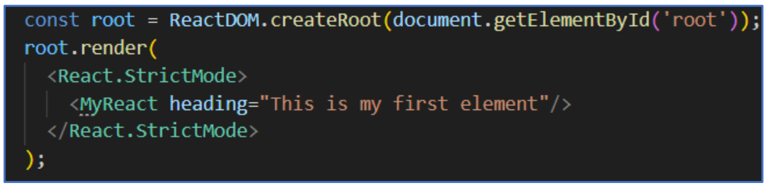
   Ans:



```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <MyReact/>
  </React.StrictMode>
);
```

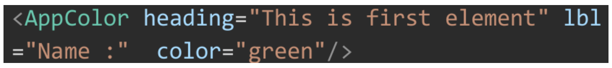
4. Suppose the MyReact component has a property heading, write down the code that could be used to render the MyReact component, and pass the message to the property heading as “This is my first element”.

   Ans:



```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <MyReact heading="This is my first element"/>
  </React.StrictMode>
);
```

5. Observe the code below and answer the questions:



```
<AppColor heading="This is first element" lbl
="Name :"  color="green"/>
```

   a. What is the name of the React component?

      Ans: AppColor.

   b. How many properties does this component use?

      Ans: Three.

6. Look at the following code:

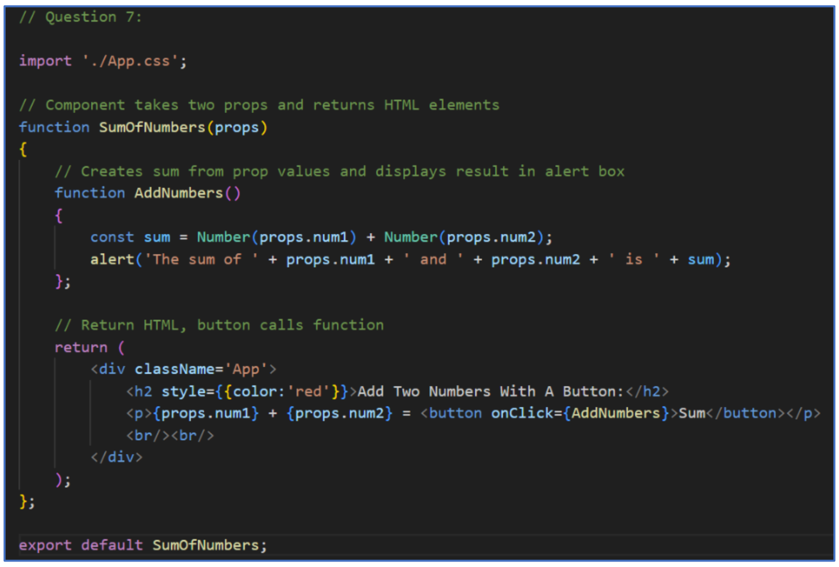
```
function GreetingElementwithProp(props) {

  return (
    <div className="App">
      <h1>Wellcome , {props.studentname}</h1>;
    </div>
  );
}
  export default ??????
```

What can you write to make the exporting of this function correct?

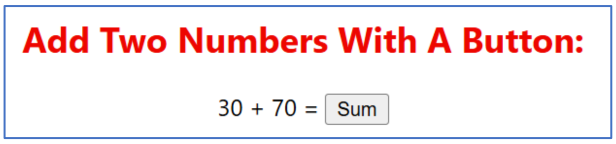
Ans: GreetingElementWithProp.

7. Add a function that takes two properties as numbers, adds these numbers on the click event of a button, and then displays the sum.

Ans:

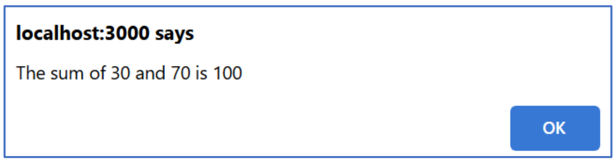


```
// Question 7:

import './App.css';

// Component takes two props and returns HTML elements
function SumOfNumbers(props)
{
    // Creates sum from prop values and displays result in alert box
    function AddNumbers()
    {
        const sum = Number(props.num1) + Number(props.num2);
        alert('The sum of ' + props.num1 + ' and ' + props.num2 + ' is ' + sum);
    };

    // Return HTML, button calls function
    return (
        <div className='App'>
            <h2 style={{color:'red'}}>Add Two Numbers With A Button:</h2>
            <p>{props.num1} + {props.num2} = <button onClick={AddNumbers}>Sum</button></p>
            <br/><br/>
        </div>
    );
};

export default SumOfNumbers;
```



Add Two Numbers With A Button:

30 + 70 = Sum



localhost:3000 says

The sum of 30 and 70 is 100

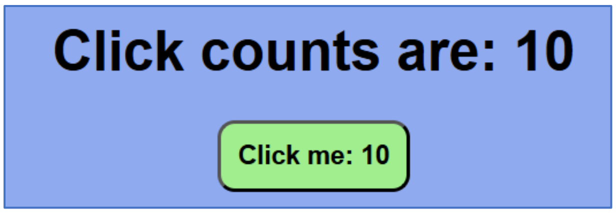
Reflection:

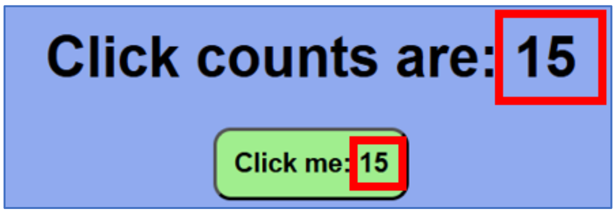
This week I learned the basics of the React library and JSX. React uses a virtual DOM in memory and compares it with the actual DOM to determine which elements must be re-rendered; This is the key to React's responsiveness. I learned how to develop a simple React interface using components, props, JSX, HTML, CSS, and the .render() method. Lastly, I was able to familiarise myself with the structure of React, including the rendered index.html file in the public folder and the various files within the src folder.

Week 7

Task 1:

Output:



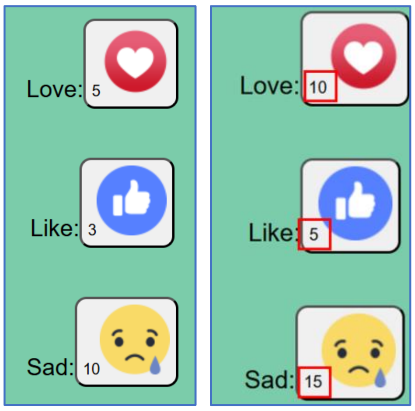


Explanation:

I used Hook API's useState method to keep track of the number of clicks. A button calls an event handler which executes the counter's function updater by an increment of one.

Task 2:

Output:



Explanation:

I used Hook API's useState and useEffect to keep track of and update both the counter's state and emoji's state. UseEffect activates whenever a change to the value of 'props.pic' happens. A conditional checks a string value and compares it to 'props.pic'. if a match is true, use the emoji's updater function to set the emoji's value. A button calls an event handler which executes the counter's function updater by an increment of one.

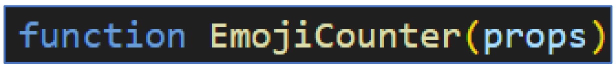
Question 1:

I learned that Hook API's enable state functionality within functional components. States are variables with values that are stored between function calls and are updated via updater functions. These functions are embedded within an event handler function so that they can be called when required. I worked with two methods of the Hook API: useState and useEffect. useState is used to declare and initialise a state and useEffect is used to perform a series of tasks when a particular change takes place. States are particularly important as their values either remain the same or are updated for each render/re-render.

Question 2:

<u>What is Name of the Component you have created in EmojeeCounters.js?</u>

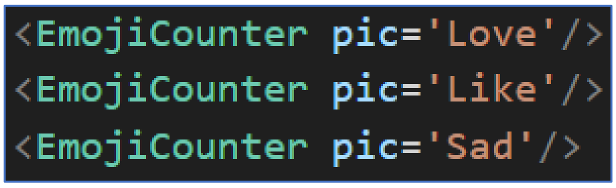
Ans:



```
function EmojiCounter(props)
```

<u>Identify the line of code that uses the EmojeeCounter in index.js:</u>

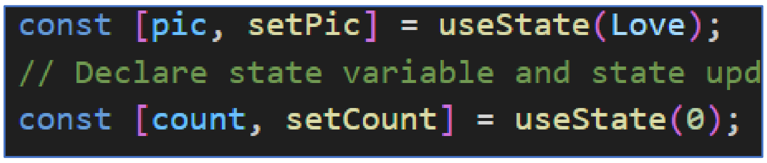
Ans:



```
<EmojiCounter pic='Love'/>
<EmojiCounter pic='Like'/>
<EmojiCounter pic='Sad'/>
```

<u>Declares the states of each of the html elements defined in the EmojeeCounters.js (identify these lines and explain only those lines):</u>

Ans:



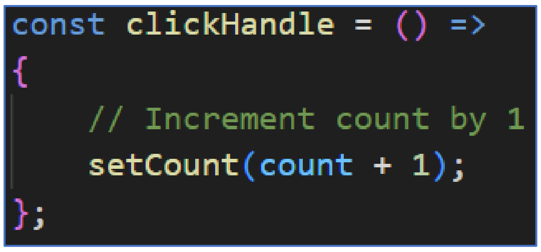
```
const [pic, setPic] = useState(Love);
// Declare state variable and state upd
const [count, setCount] = useState(0);
```

First state has variable 'pic' and updater function 'setPic'. 'pic' is initialised as 'Love' emoji. Second state has variable 'count' and updater function 'setCount'. 'count' is initialised as 0.

Lines of codes that are used to associate the event handler used:

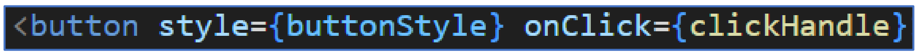
Ans:

Event handler function:



```
const clickHandle = () =>
{
    // Increment count by 1
    setCount(count + 1);
};
```

Called via button:



```
<button style={buttonStyle} onClick={clickHandle}
```

Explain the line : <EmojeeCounter pic='Love'/>, what does pic='Love' mean in this line?

Ans: 'pic' is a property of the component 'EmojeeCounter' and the argument 'Love' is passed as the property of said component. The Love emoji image is displayed.

What is useEffect and why do you think we have used it in a Component?

Ans: 'useEffect' is a method of the React Hook API which is used to perform various actions when states change, and adds additional functionality for the rendering of components.

Explain these line of the codes in functional component EmojeeCounter.js:

// Returns JSX.

return (

// Set div class to App.

<div className="App">

// Paragraph that displays the value of 'props.pic'.

<p>{props.pic}<span></span>

// Button is associated with event handler 'ClickHandle'. When the button is clicked, the event handler performs the count updater function which increments the count by 1. Count is also displayed in the button tag.

<button onClick={ClickHandle}>{count}

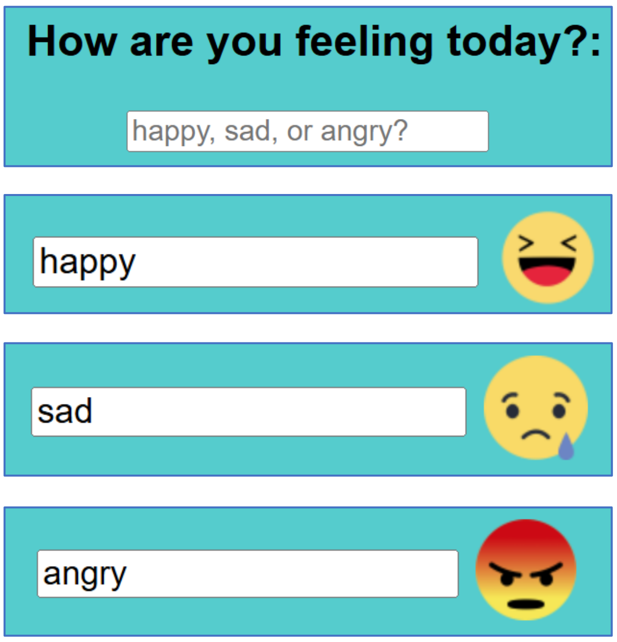
// Image is displayed within the button tag. The source of the image is set to the value of 'pic', which has been imported into the file.

<img src={pic} alt=""/>

```
// Closing tags.

</button>

</p>

// End of return method.

);
```

Question 3:

Output:



Explanation:

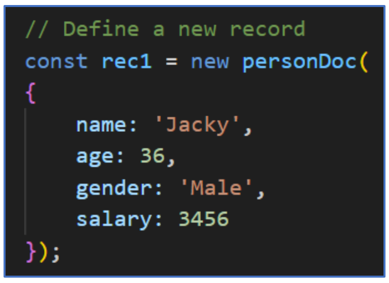
I first imported useState, useEffect, and the three emojis into my JS file. I declared a functional component called ‘TextBoxtEmoji’ that is composed of state declarations, an event handler function, useEffect with an embedded switch statement, CSS object styling, and a return method. A text state is mapped to the value of the input field, and the emoji state’s value is determined by the comparison between the text state and string values. The rendered text field calls the event handler function on change and thus sets in motion the chain of events explained above.

[NEXT PAGE]

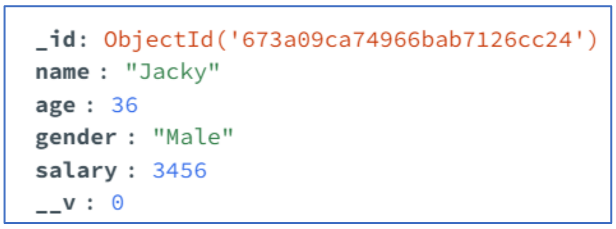
Portfolio Week 8

Task 1:

Example 1:

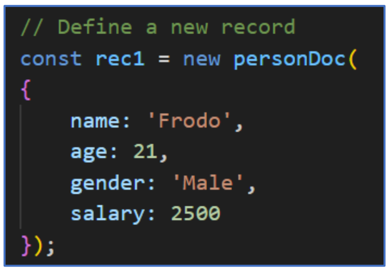


```
// Define a new record
const rec1 = new personDoc(
{
    name: 'Jacky',
    age: 36,
    gender: 'Male',
    salary: 3456
});
```

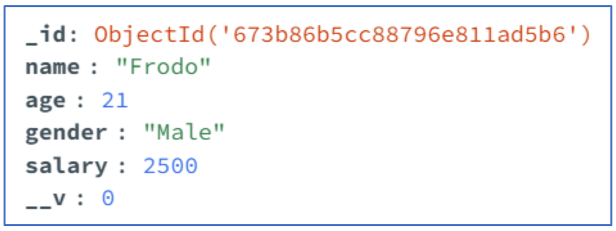


```
_id: ObjectId('673a09ca74966bab7126cc24')
name : "Jacky"
age : 36
gender : "Male"
salary : 3456
__v : 0
```

Example 2:

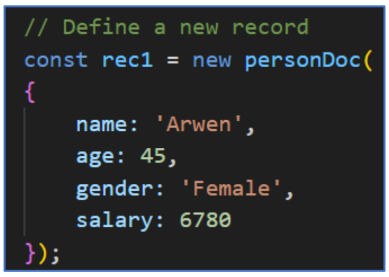


```
// Define a new record
const rec1 = new personDoc(
{
    name: 'Frodo',
    age: 21,
    gender: 'Male',
    salary: 2500
});
```

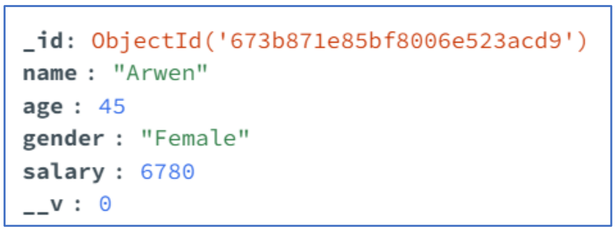


```
_id: ObjectId('673b86b5cc88796e811ad5b6')
name : "Frodo"
age : 21
gender : "Male"
salary : 2500
__v : 0
```

Example 3:



```
// Define a new record
const rec1 = new personDoc(
{
    name: 'Arwen',
    age: 45,
    gender: 'Female',
    salary: 6780
});
```

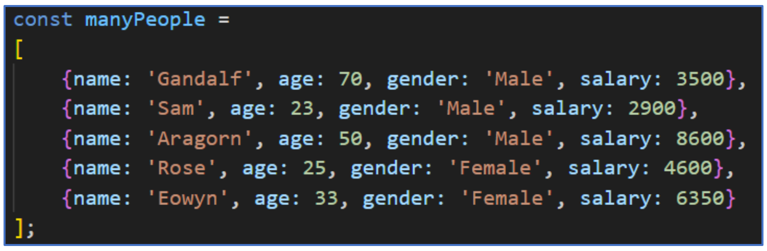


```
_id: ObjectId('673b871e85bf8006e523acd9')
name : "Arwen"
age : 45
gender : "Female"
salary : 6780
__v : 0
```

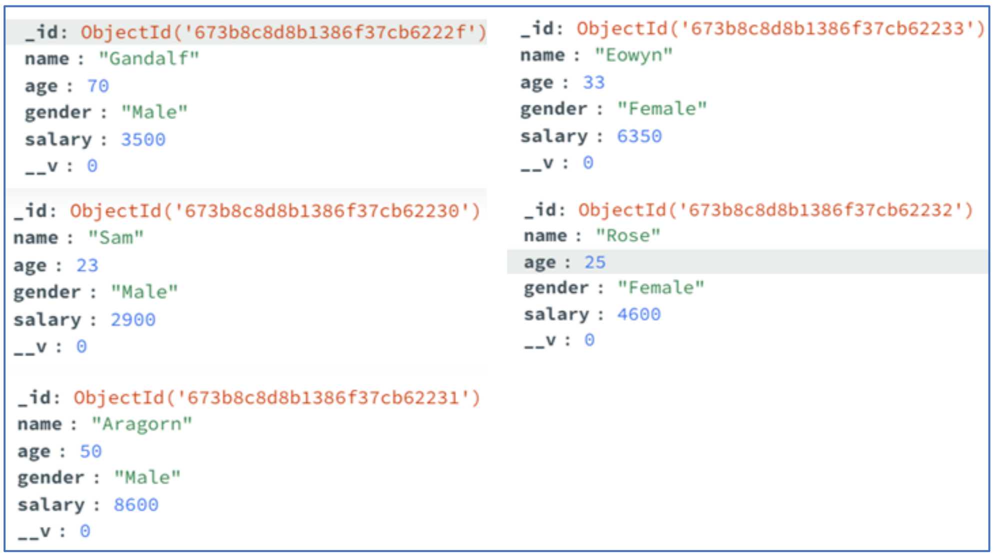
Explanation:

I created three new records within my collection using the Mongoose module. 'doc1' is an instance of the model I created. It represents the record that is to be saved into the database. '.save' is an asynchronous method which saves the record to the MongoDB database and returns a promise object. Once the record is saved, '.then' executes. However, if an error occurs, '.catch' is executed.

Task 2:



```
const manyPeople =
[
    {name: 'Gandalf', age: 70, gender: 'Male', salary: 3500},
    {name: 'Sam', age: 23, gender: 'Male', salary: 2900},
    {name: 'Aragorn', age: 50, gender: 'Male', salary: 8600},
    {name: 'Rose', age: 25, gender: 'Female', salary: 4600},
    {name: 'Eowyn', age: 33, gender: 'Female', salary: 6350}
];
```



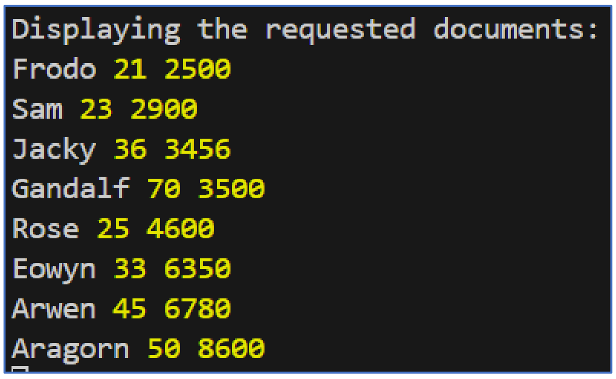
```
_id: ObjectId('673b8c8d8b1386f37cb6222f')
name : "Gandalf"
age : 70
gender : "Male"
salary : 3500
__v : 0

_id: ObjectId('673b8c8d8b1386f37cb62230')
name : "Sam"
age : 23
gender : "Male"
salary : 2900
__v : 0

_id: ObjectId('673b8c8d8b1386f37cb62231')
name : "Aragorn"
age : 50
gender : "Male"
salary : 8600
__v : 0

_id: ObjectId('673b8c8d8b1386f37cb62233')
name : "Eowyn"
age : 33
gender : "Female"
salary : 6350
__v : 0

_id: ObjectId('673b8c8d8b1386f37cb62232')
name : "Rose"
age : 25
gender : "Female"
salary : 4600
__v : 0
```

Explanation:

I used an array to hold multiple JSON records, which were then inserted into the database using the '.insertMany' method. The array was passed as an argument to this method, which returned a promise object. '.then' handled the successful execution, whilst '.catch' handled any unsuccessful execution.

Task 3:

Output:



```
Displaying the requested documents:
Frodo 21 2500
Sam 23 2900
Jacky 36 3456
Gandalf 70 3500
Rose 25 4600
Eowyn 33 6350
Arwen 45 6780
Aragorn 50 8600
```

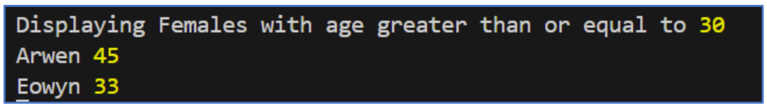
Explanation:

I used the '.sort' method to sort salary by ascending order, and used the '.select' method to display the name, age, and salary fields. The '.limit' method was used to limit the number of records printed to the console. The '.exec' method then executed the query, which was followed by the '.then' and '.catch' methods.

Task 4:

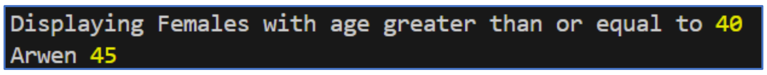
Output:

Example 1:



```
Displaying Females with age greater than or equal to 30
Arwen 45
Eowyn 33
```

Example 2:



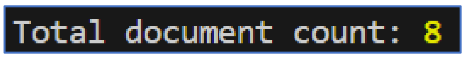
```
Displaying Females with age greater than or equal to 40
Arwen 45
```

Explanation:

Similar to task 3 except filtering was used within the '.find' method. I filtered by female gender and age, sorted by name, displayed only name and age, and limited displayed documents to 10.

Task 5:

Output:



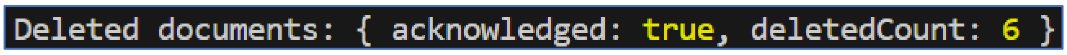
```
Total document count: 8
```

Explanation:

I used the '.countDocuments' method to count the total number of documents and print them to the console. After '.exec', the promise object is returned and either '.then' or '.catch' is executed.

Task 6:

Output:



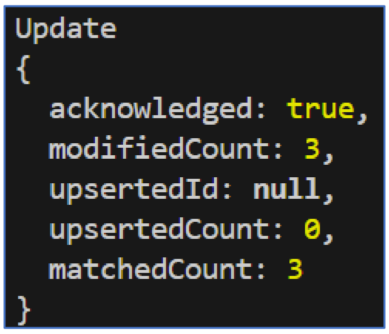
```
Deleted documents: { acknowledged: true, deletedCount: 6 }
```

Explanation:

I deleted 6 documents using the '.deleteMany' method. I filtered the delete query with ages greater than or equal to 25. In the '.then' method, the total count is printed to the console.

Task 7:

Output:



```
Update
{
  acknowledged: true,
  modifiedCount: 3,
  upsertedId: null,
  upsertedCount: 0,
  matchedCount: 3
}
```

Explanation:

I updated 3 records of gender female with a salary of 5555 using the '.updateMany' method. The '.then' method prints the results to the console.

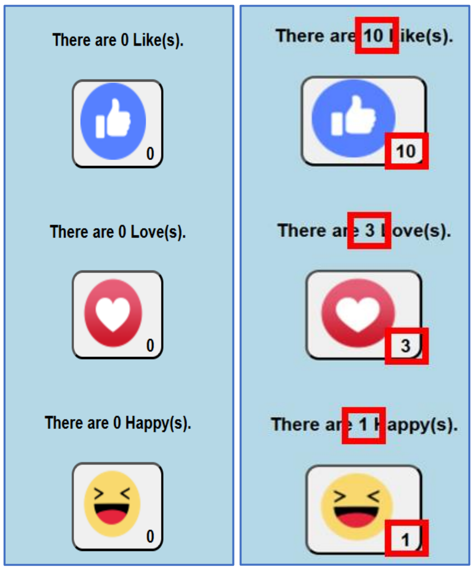
Reflection:

Overall, I learned how to connect a part of the backend to a MongoDB database. This was done using Mongoose which provided various methods for defining a connection, schema, schema wrapper and interface (model), and CRUD operations. I also learned of the importance of the promise object that is returned during the execution of a query/command; and of the success and failure of handling asynchronous operations.

Portfolio Week 9

Task 1:

Output:



There are 0 Like(s).

0

There are 0 Love(s).

0

There are 0 Happy(s).

0

There are 10 Like(s).

10

There are 3 Love(s).

3

There are 1 Happy(s).

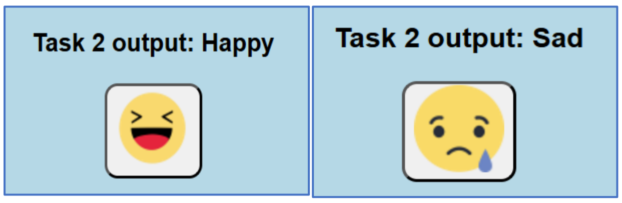
1

Explanation:

I created a class component that, via the constructor, initialises props and states with the desired default values. The state of ‘number’ is set to 0 and ‘pic’ property is set to null. A set of conditionals within the constructor then check the value of ‘type’ and compares it to a string that matches the imported emoji pictures. If a match is found, set the ‘pic’ to that particular image. An increment method updates the number’s state using ‘.setState’. The increment method is called on a button click.

Task 2:

Output:



Task 2 output: Happy

Task 2 output: Sad

Explanation:

I created a class component similar to the first task, except the managed state is the emoji image itself. Default state of ‘pic’ is set to ‘Happy’. A toggle method updates the ‘pic’ state using ‘.setState’. Conditionals compare the state of the ‘pic’ and set the new state to the opposite image (e.g. happy to sad or sad to happy). The toggle method is called on a button click.

Reflection:

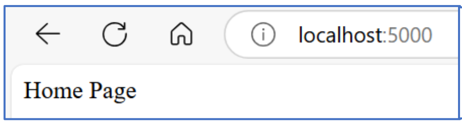
I learned how to implement class components, their props, and their states. Although the results achieved between class and functional components are the same, how they are implemented differs significantly. Class components require inheriting from the React library in order to use its various methods. One such example is the render method, which is absent from functional components. Class components also have constructors for the initialisation of states and props. States are managed differently as class components managed them with methods from the React library, such as setState, whereas functional components managed them through the Hook API library. ‘this’ is also often needed with class components so as to remove ambiguity by directly stating that particular class instance. Overall, class components appear to be harder to implement and require more code to achieve the same results.

[NEXT PAGE]

Week 10 and 11

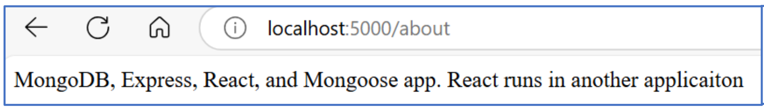
Backend:

Output:



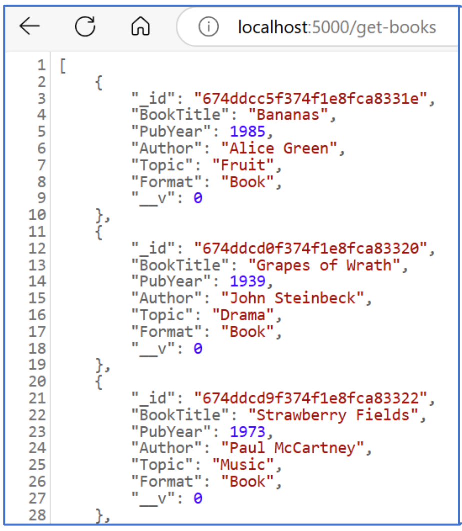
Home Page

Get method. Home page route ('/').



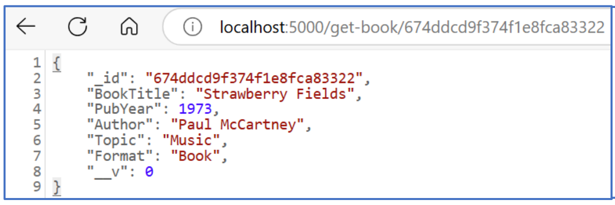
MongoDB, Express, React, and Mongoose app. React runs in another applicaiton

Get method. About route ('/about').



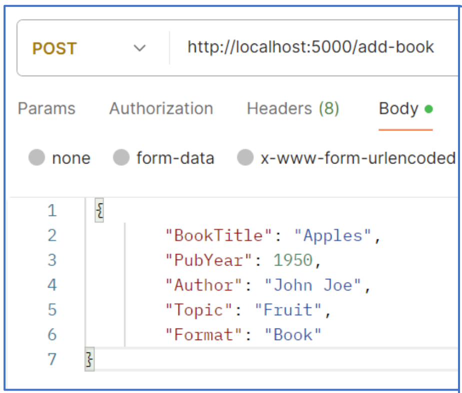
```
[
    {
        "_id": "674ddcc5f374f1e8fca8331e",
        "BookTitle": "Bananas",
        "PubYear": 1985,
        "Author": "Alice Green",
        "Topic": "Fruit",
        "Format": "Book",
        "__v": 0
    },
    {
        "_id": "674ddcd0f374f1e8fca83320",
        "BookTitle": "Grapes of Wrath",
        "PubYear": 1939,
        "Author": "John Steinbeck",
        "Topic": "Drama",
        "Format": "Book",
        "__v": 0
    },
    {
        "_id": "674ddcd9f374f1e8fca83322",
        "BookTitle": "Strawberry Fields",
        "PubYear": 1973,
        "Author": "Paul McCartney",
        "Topic": "Music",
        "Format": "Book",
        "__v": 0
    },
```

Get method and MongoDB find query. Get all books route ('/get-books).

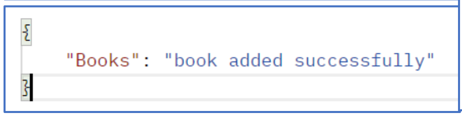


```
{
    "_id": "674ddcd9f374f1e8fca83322",
    "BookTitle": "Strawberry Fields",
    "PubYear": 1973,
    "Author": "Paul McCartney",
    "Topic": "Music",
    "Format": "Book",
    "__v": 0
}
```

Get method and MongoDB findById query. Filter by id. Get book by id route ('/get-book/:id').

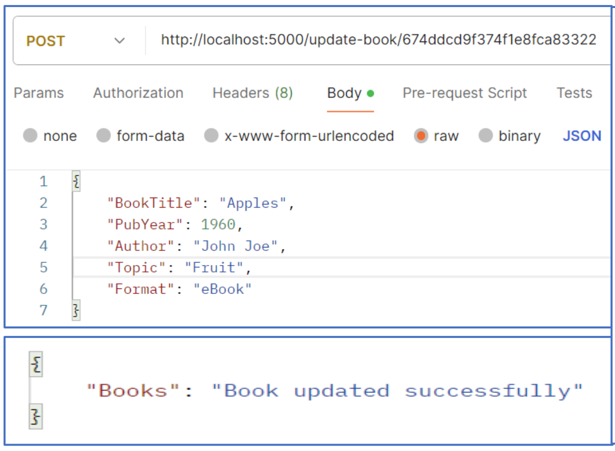


```
{
    "BookTitle": "Apples",
    "PubYear": 1950,
    "Author": "John Joe",
    "Topic": "Fruit",
    "Format": "Book"
}
```



```
{
    "Books": "book added successfully"
}
```

Post method and MongoDB save. Add book route ('/add-book').

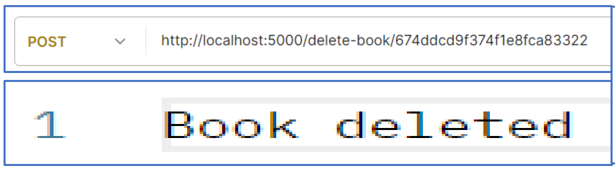


```
http://localhost:5000/update-book/674ddcd9f374f1e8fca83322
```

```
{
    "BookTitle": "Apples",
    "PubYear": 1960,
    "Author": "John Joe",
    "Topic": "Fruit",
    "Format": "eBook"
}
```

```
{
    "Books": "Book updated successfully"
}
```

Post method and MongoDB findByIdAndUpdate. Filter by id. Update book route (‘/update-book/:id’).



```
http://localhost:5000/delete-book/674ddcd9f374f1e8fca83322
```

```
Book deleted
```

Post method and MongoDB findByIdAndDelete. Filter by id. Delete book route (‘/delete-book/:id’).

Reflection:

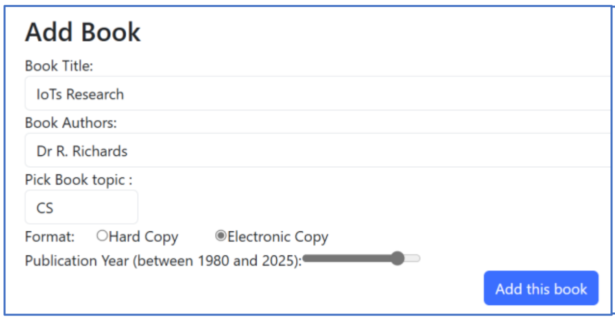
The purpose of this week was to develop the backend of an online library web application. Three files were developed to achieve this: the book model, database connection, and the express server. The model and connection were imported into the express server file. For the express server, seven routes were developed: Get methods and post methods. The various routes used MongoDB queries to filter and select the desired data. The data was access either in the request body or the request URL parameters. The asynchronous operations’ promises were handled by the .then and .catch methods. Successful and unsuccessful results returned a response status code, JSON message, and/or console message.

For the backend coursework, I have identified six RESTAPI points: a post method to add a team; a post method to update a team via an id parameter, and MongoDB findByIdAndUpdate query; a post method to delete a team via an id parameter, and MongoDB findByIdAndDelete query; a get method to show total games played, won, and drawn via a year parameter, and MongoDB aggregation pipeline (filter and group by year); a get method to show top ten wins via a greater than or equal number parameter, and MongoDB find, limit, and sort queries; a get method to show the average ‘goals for’ via a year parameter, and MongoDB aggregation pipeline (filter by year and group by team).

[NEXT PAGE]

Frontend:

Output:


Add Book

Book Title: IoTs Research

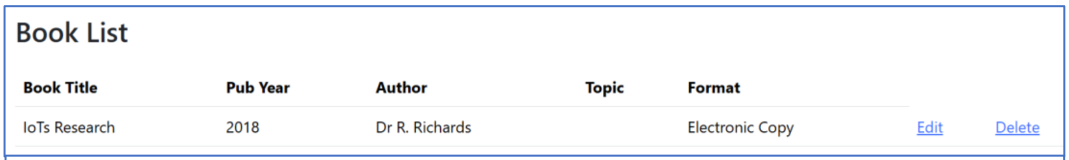
Book Authors: Dr R. Richards

Pick Book topic : CS

Format: ○Hard Copy ◉Electronic Copy

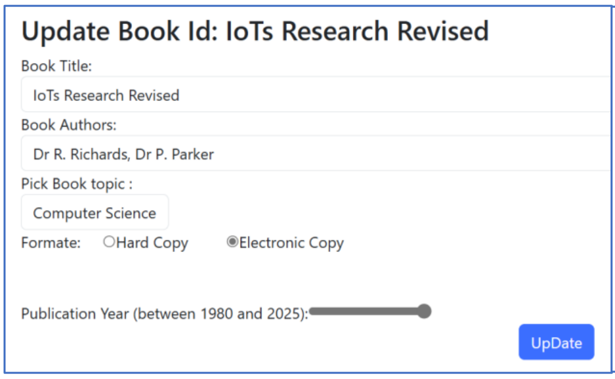
Publication Year (between 1980 and 2025):

Add book component which connects to the backend API via Axios post.


Book List

Book Title	Pub Year	Author	Topic	Format		
IoTs Research	2018	Dr R. Richards		Electronic Copy	Edit	Delete

Display all books component which connects to the backend API via Axios get.


Update Book Id: IoTs Research Revised

Book Title: IoTs Research Revised

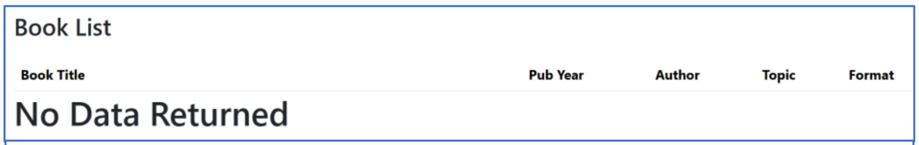
Book Authors: Dr R. Richards, Dr P. Parker

Pick Book topic : Computer Science

Formate: ○Hard Copy ◉Electronic Copy

Publication Year (between 1980 and 2025):

Update book component which connects to the backend API via Axios post.


Book List

Book Title	Pub Year	Author	Topic	Format

No Data Returned

Delete book component which connects to the backend API via Axios post.

Reflection:

The purpose of this week was to develop a front end React SPA with the use of various React components, routing of said components, and connection to the backend with Axios. The adding, updating, displaying, and deleting of books corresponds to a particular component, which in turn corresponds to the APIs. React Hook API's useState and useEffect were used to keep track of book data regardless of the rendering/re-rendering of component interfaces. An App component then consolidates everything with the use of React Router DOM's Routes and Links.

For the frontend coursework, I plan to create six components: add team form interface, update team form interface, delete team interface, show total games played, won, and drawn for a team given a year interface, show top ten wins for teams given a greater than or equal to number interface, and show all teams' average 'goals for' given a year interface. Lastly, a React router application which will consolidate and integrate all the components.

[END]