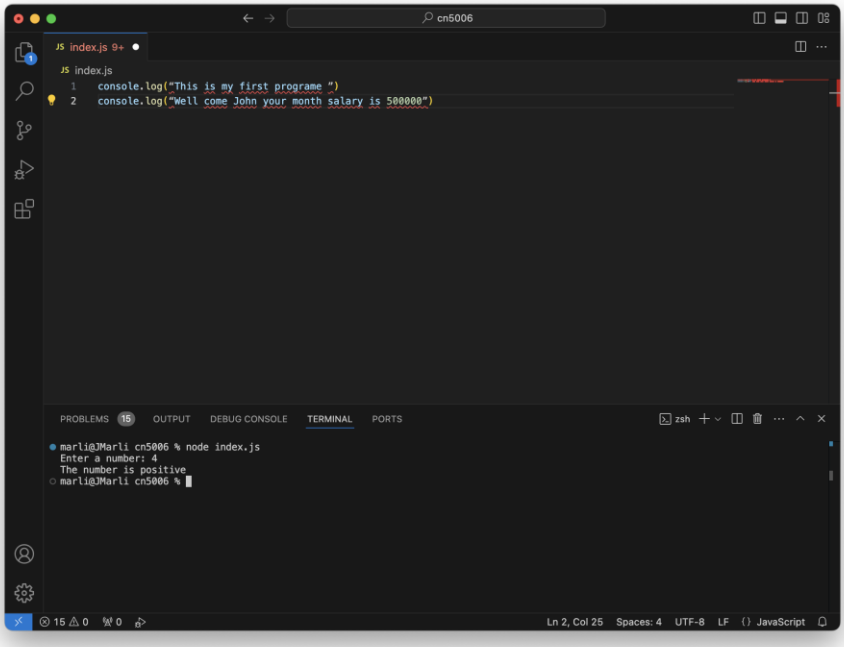
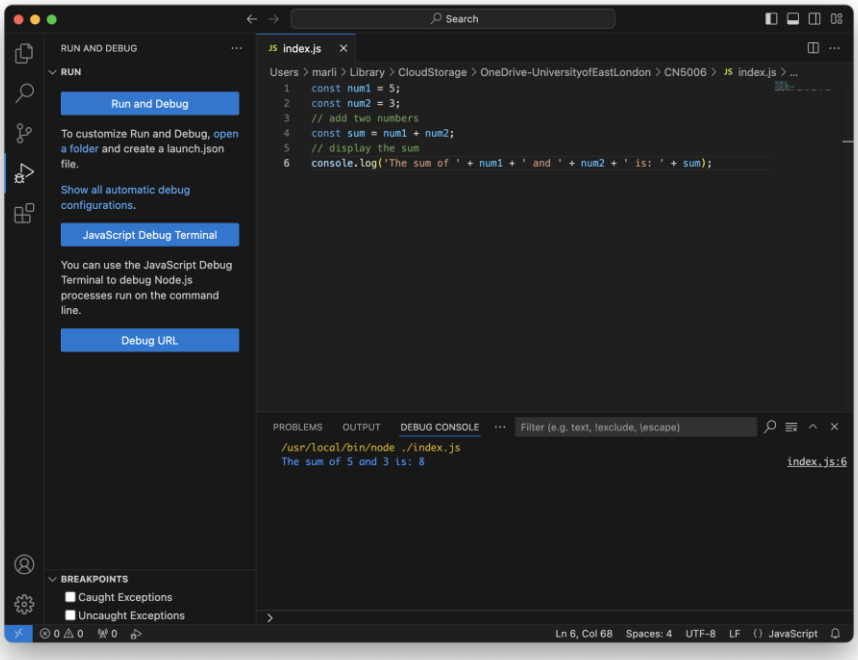
U2283556

# CN5006 PORTFOLIO:

## Week 1 code:



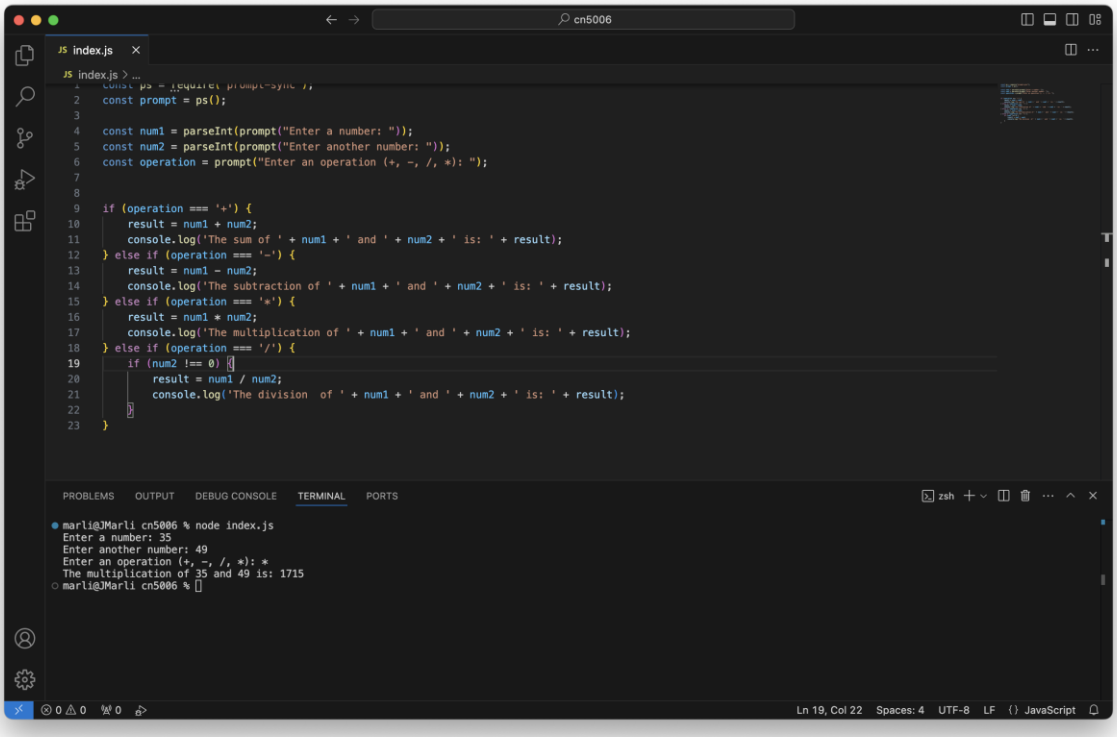




## Self-evaluation:

I have improved my java script language skills and will work more to improve them. This was a great refresher after the summer holiday.

## Exercise:

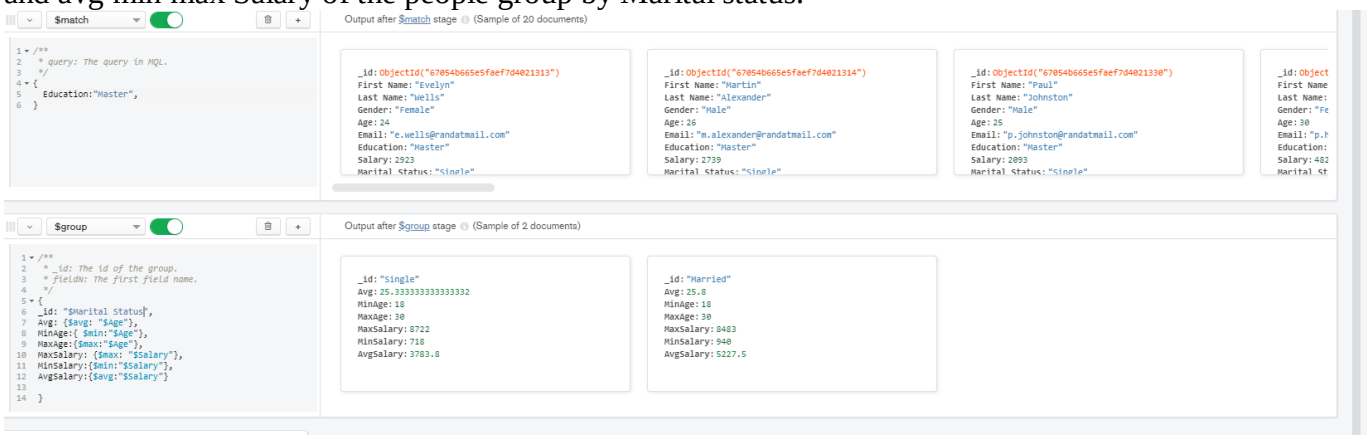
U2283556




This program was created to function as a calculator that is capable of four kinds of operations.

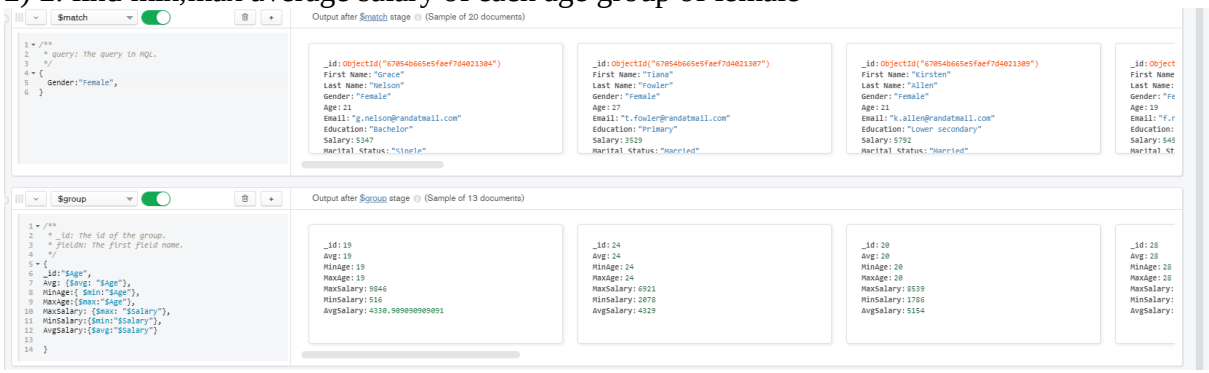
# Week 2 lab tasks:

1) Repeat the same process to search Education for Master and .Find the avg,min,max age and avg min max Salary of the people group by Marital status.

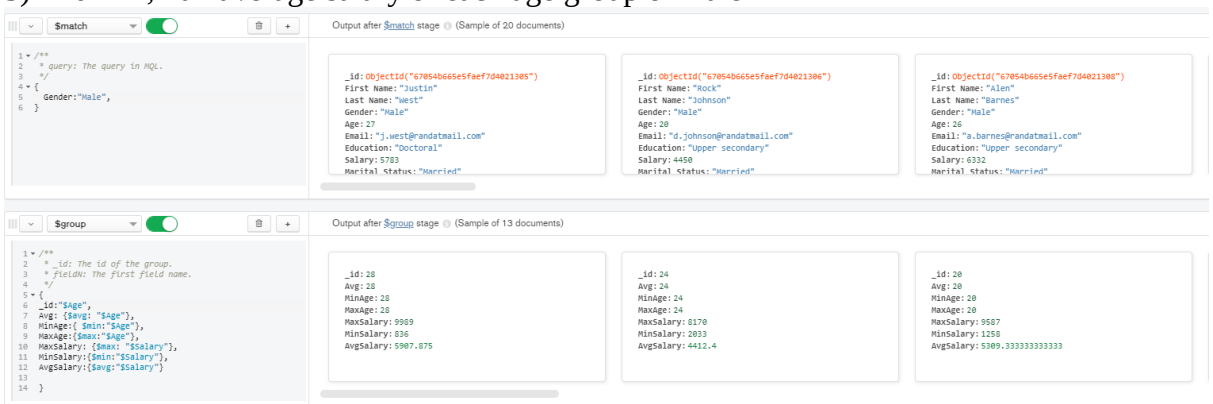


U2283556

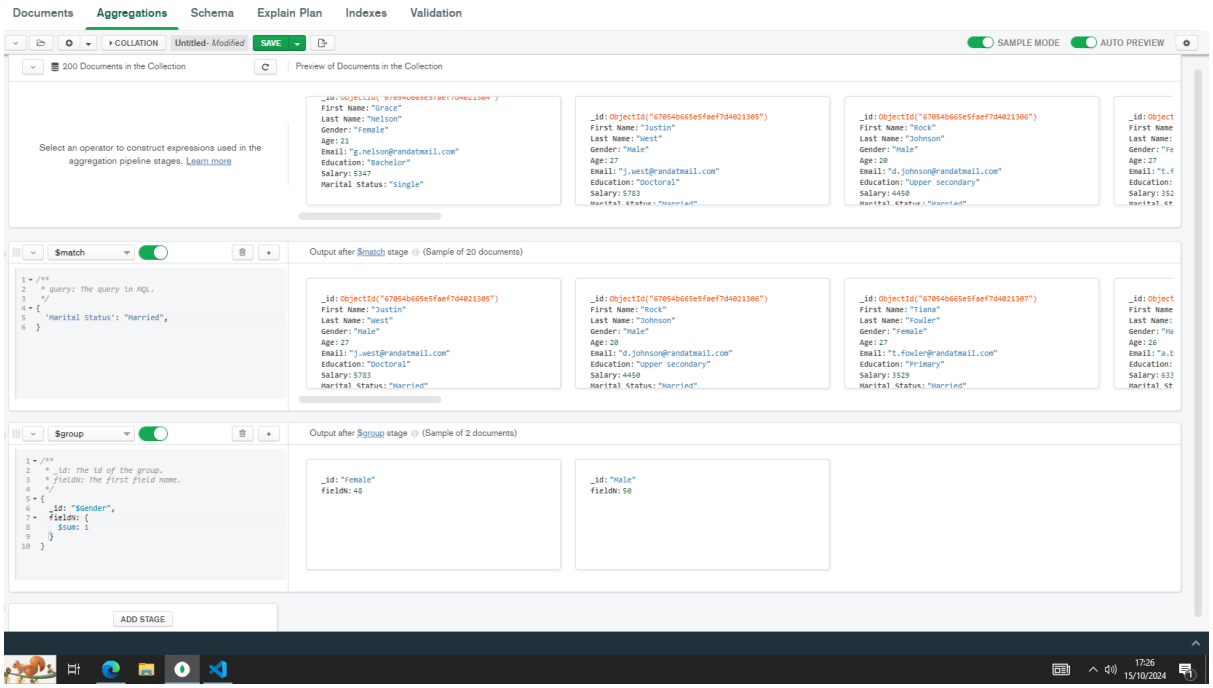
2) 2. find min,max average salary of each age group of female



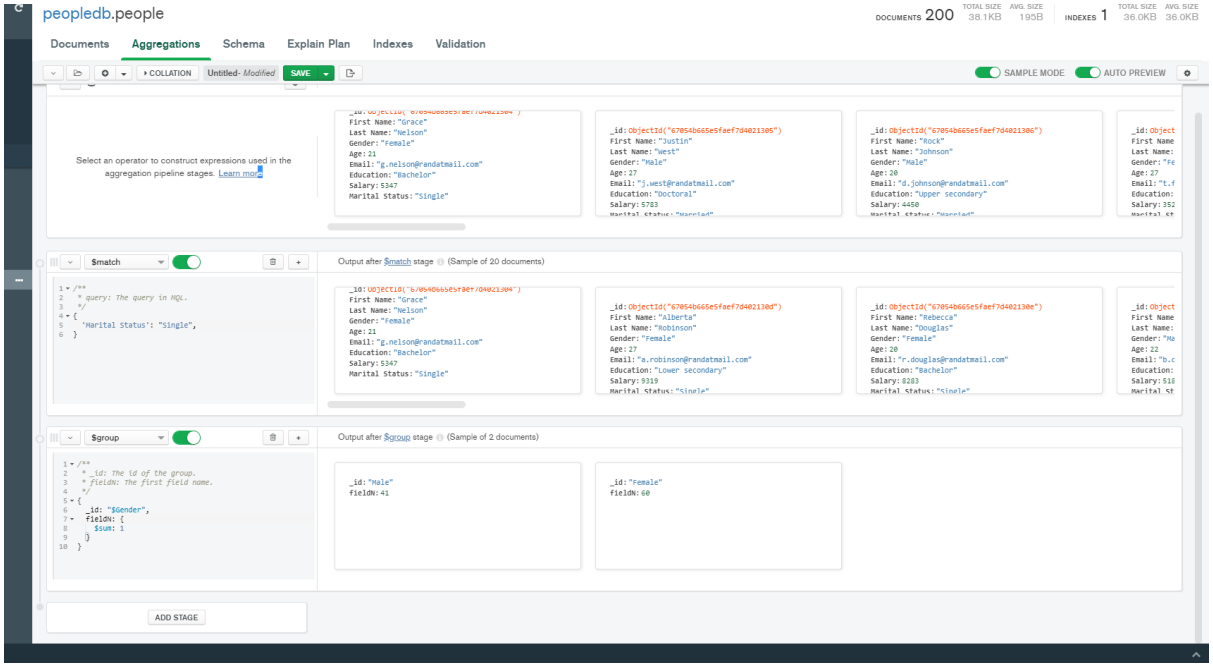

3) find min,max average salary of each age group of male




4) Count married and unmarried females and males.



U2283556



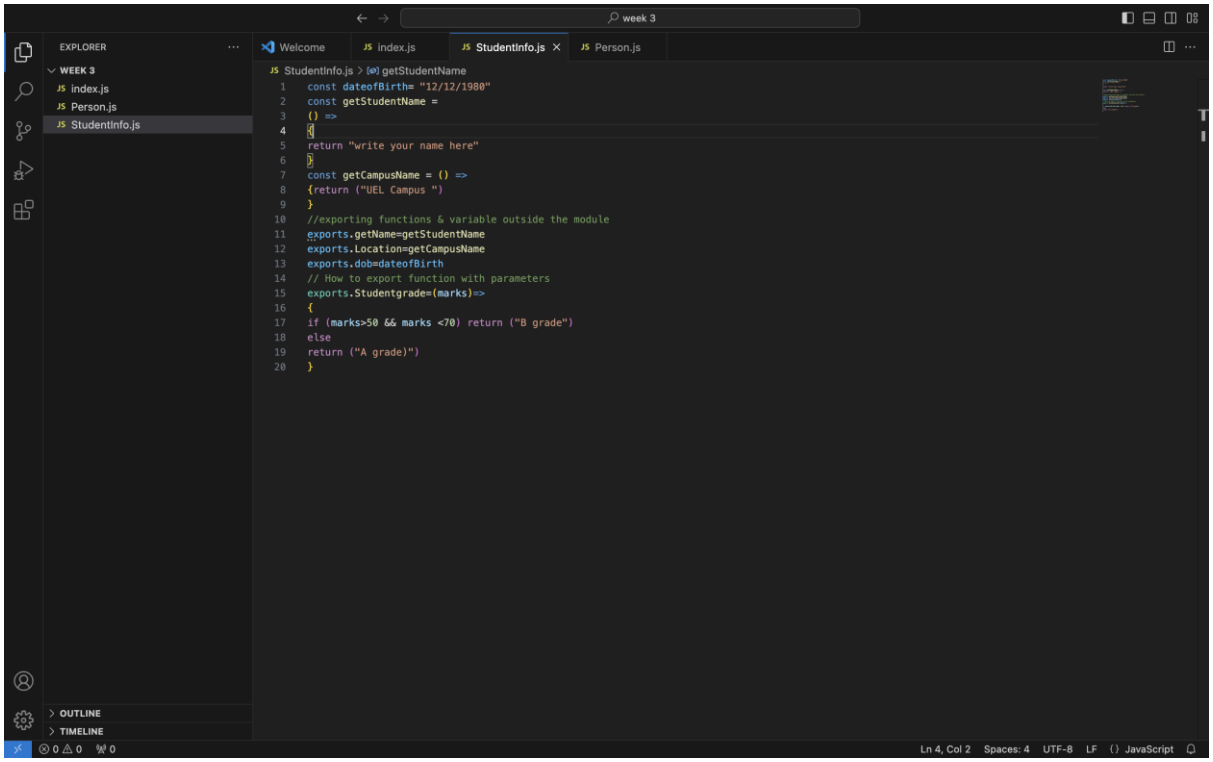

**Report:**

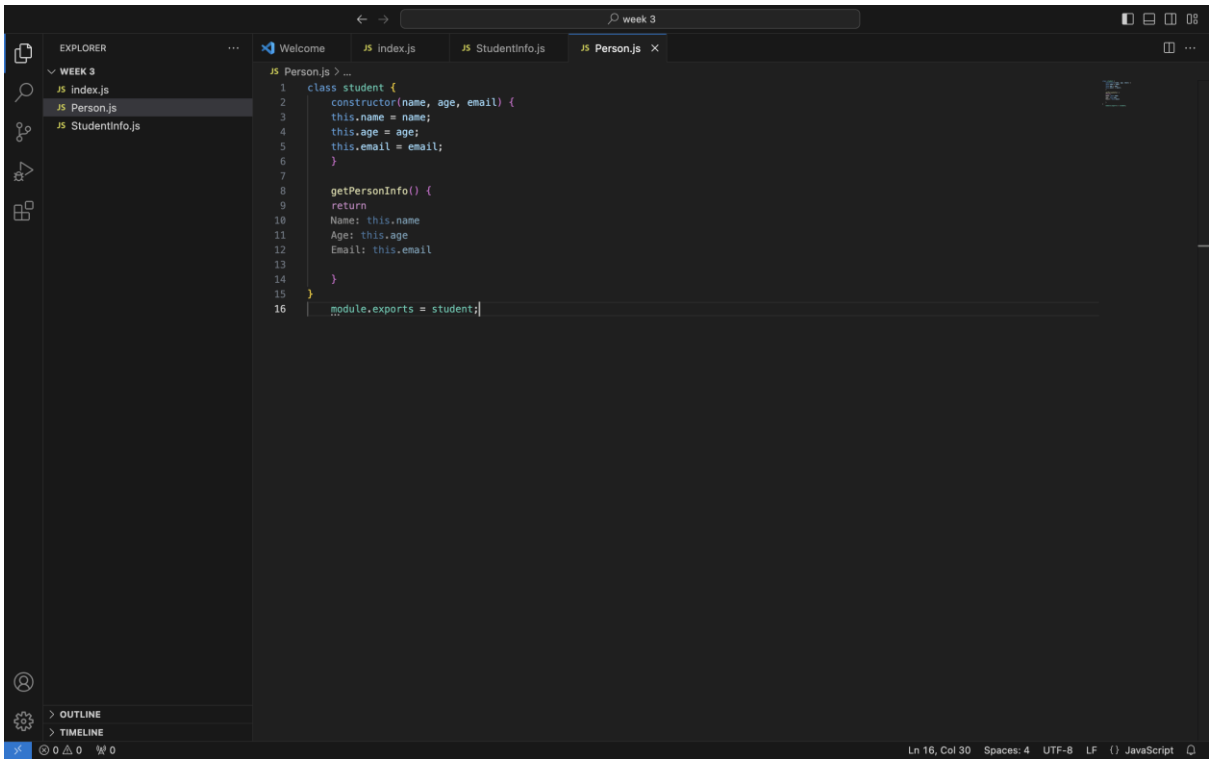
The work that been done with this code has been used to separate the data that has been given from a spread sheet then sorted into different categories depending on the need.

# WEEK 3:

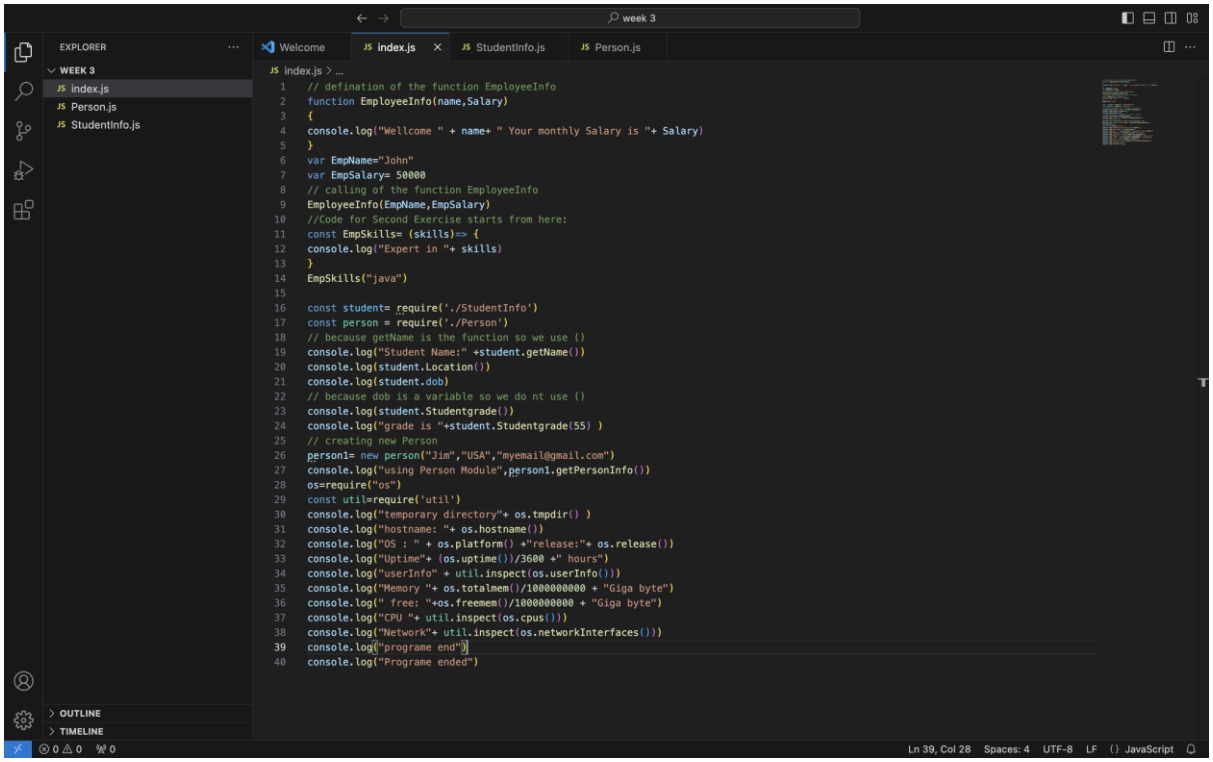
Submit the code for the completed Exercises i.e.: i) Index.js ii) Person.js iii) Employeeinfo.js iv) Exercise 4.js

U2283556


week 3
EXPLORER
WEEK 3
index.js
Person.js
StudentInfo.js
Welcome
index.js
StudentInfo.js
Person.js
StudentInfo.js > getStudentName
const dateofBirth= "12/12/1980"
const getStudentName =
() =>
{
return "write your name here"
}
const getCampusName = () =>
{return ("UEL Campus ")
}
//exporting functions & variable outside the module
exports.getName=getStudentName
exports.Location=getCampusName
exports.dob=dateofBirth
// How to export function with parameters
exports.Studentgrade=(marks)=>
{
if (marks>50 && marks <70) return ("B grade")
else
return ("A grade)")
}
OUTLINE
TIMELINE
Ln 4, Col 2
Spaces: 4
UTF-8
LF
JavaScript


week 3
EXPLORER
WEEK 3
index.js
Person.js
StudentInfo.js
Welcome
index.js
StudentInfo.js
Person.js
Person.js > ...
class student {
constructor(name, age, email) {
this.name = name;
this.age = age;
this.email = email;
}
getPersonInfo() {
return
Name: this.name
Age: this.age
Email: this.email
}
}
module.exports = student;
OUTLINE
TIMELINE
Ln 16, Col 30
Spaces: 4
UTF-8
LF
JavaScript

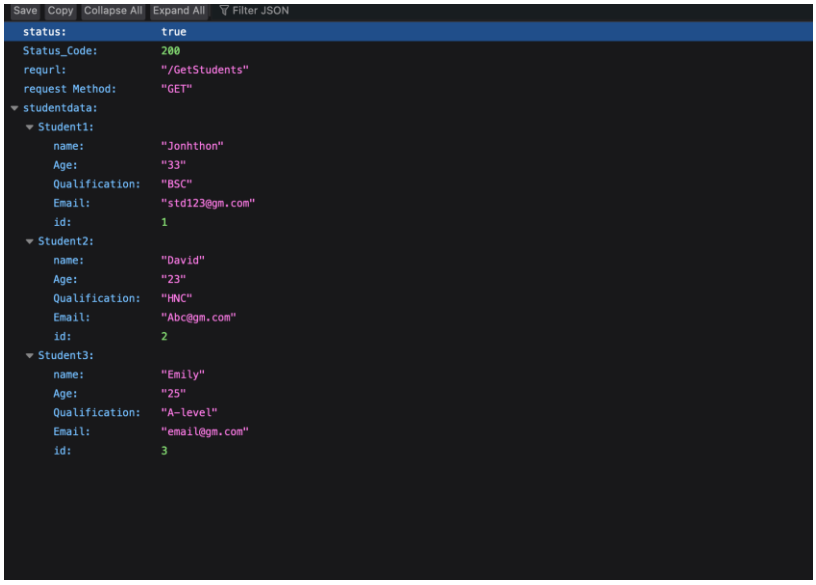
U2283556




**Report:**

The coding that I've done during this lesson displays the information of the device that is being used. It also displays the information of the student information that has been given. I learned how to link different methods together.

# WEEK 4:




{"status":true,"Status_Code":200,"requrl":"/GetStudents","request Method":"GET","studentdata":{"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/GetStudents

U2283556

```
name:            "Jonhthon"
Age:             "33"
Qualification:   "BSC"
Email:           "std123@gm.com"
id:              1
```

Localhost:5000/ GetStudentid/1

```
name:            "David"
Age:             "23"
Qualification:   "HNC"
Email:           "Abc@gm.com"
id:              2
```

Localhost:5000/ GetStudentid/2

```
name:            "Emily"
Age:             "25"
Qualification:   "A-level"
Email:           "email@gm.com"
id:              3
```

Localhost:5000/ GetStudentid/3

```
status:                 true
Status_Code:            200
requrl:                 "/GetStudentid/4"
request Method:         "GET"
studentdata:
  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/ GetStudentid/4

Localhost:5000/ studentinfo

U2283556

**Student Details**

**First Name:** jahmarli

**Last Name :** hibbert

**Email:** u2283556@uel.ac.uk

**Age :** 21

**Please select your gender:**

- (●) **Male**
- ( ) **Female**
- ( ) **Other**

**Qualifications**

- [ ] **GCSE**
- [ ] **A- level**
- [ ] **Higher National Certificate/Level 4**
- [x] **Foundation Degree/HND/DipHE/Level 5**
- [ ] **Bachelor Degree/Graduate diploma or Certificate/Level 6**
- [ ] **Master Degree/PGCE/Level7**
- [ ] **PhD/Level8**

Submit Query

```
status:          true
message:         "form Details"
data:
    name:        "jahmarli hibbert "
    age:         "21 Gender: male"
    Qualification:  " QualificationHND"
```

{"status":true,"message":"form Details","data":{"name":"jahmarli hibbert ","age":" Gender: male","Qualification":" QualificationHND"}}

## Report:

This week I learnt how to implement data that has been provided and display it in a format that separates the data into different categories. The data has also been displayed in different formats.

U2283556

# WEEK 5:

**For todays Lab submission, After you complete the lab write down a word document answering following questions for your portfolio:**

1. What is React
   React is a JavaScript library for building  a user interface.
2. What do you understand by React component and what command do you use to create a React component with or without property
   A React Component is a piece of UI logic that uses HTML, CSS, and JavaScript code to represent a part of the user interface.
   without

```
function MyComponent() {
  return <h1>Hello, World!</h1>;
}
```

   with

```
function MyComponent(props) {
  return <h1>{props.heading}</h1>;
}
```

3. What command will you use to render the the newly created component named as myREACT

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import MyREACT from './MyREACT';

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

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”

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import MyReact from './MyReact';

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

U2283556

5. Observe this code and answer the questions below
   <AppColor heading="This is first element" lbl
   ="Name :" color="green"/>
   What is the name of the React Component
   AppColor
   How many properties this component uses
   heading, lbl, and color.
6. Look at the following Code:
   function GreetingElementwithProp(props) {

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

what will you write to make this export this function correctly?
Hint you need to replace ?????? with the correct word.

Add a function that takes two properties as numbers ,add these numbers on the click event of the button and display the sum.

Hint you will be using in jsx

```
<button value={props.color} onClick={Namdofyourfunc
tion}
```

```
import React, { useState } from 'react';

function GreetingElementwithProp(props) {
  const [sum, setSum] = useState(null); // State to store the sum

  // Function to add num1 and num2
  const handleAddition = () => {
    const result = props.num1 + props.num2;
    setSum(result); // Update the sum state
  };

  return (
    <div className="App">
      <h1>Welcome, {props.studentname}</h1>
      <button value={props.color} onClick={handleAddition}>
        Add Numbers
      </button>
      {sum !== null && <p>The sum is: {sum}</p>}
    </div>
  );
}

export default GreetingElementwithProp;
```

# Week 7:

Write one page reflective what did you learn about React Hook API during this week
Q2. Study the code in EmojeeCounters.js, Please note, You Do not need to submit the full code rather you need to answer the following questions for your this week portfolio
• What is Name of the Component you have created in EmojeeCounters.js
**EmojeeCounter**
• Identify the line of code that uses the EmojeeCounter in index.js
import EmojeeCounter from './ EmojiCounter';

• Declares the states of each of the html elements defined in the EmojeeCounters.js ( identify these lines and explain only those lines )
const [count, setCount] = useState(0);
• Lines of codes that are used to associate the event handler used.
<button onClick={ClickHandle}>{count }

• Explain the line : <EmojeeCounter pic='Love'/> , what is pic=’Love’ means in this line.
This means that the component can access the image called ‘love’.
• What is useEffect and why you think we have used it in the Component.

U2283556

It can be used for tasks like data fetching.

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

```
return (
<div className="App">
<p>{props.pic} <span></span>
<button onClick={ClickHandle}>{count }
<img src={pic} alt=""/>
</button>
</p>
</div>
)
}
```

{props.pic}: Shows the picture prop that was supplied to the component.

This button's onClick={ClickHandle}>{count}: A button element that, when clicked, launches the ClickHandle function and displays the current count.

<img src={photo} alt=""/>: Shows an image in which the source is the pic state or prop

Q3

Create a code for a Component that takes two HTML one text box and one label. Label will be used to display the images. So it should be like this

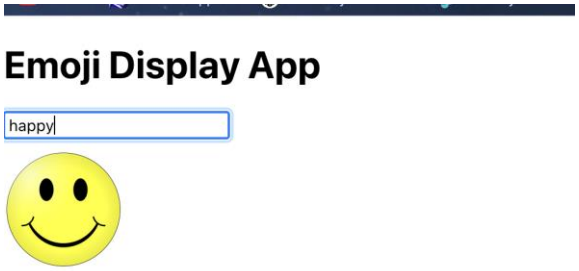
If I write “Happy” in the text box the label should show happy face (You can use any image)

If I write “Like” in the text box the label should show Like icon

If I write “sad “ the label should show sad emoji.

Run this component take the screen shot of your newly run component and write a

U2283556




paragraph how did you develop this component.



Using React and useState to control the input text state, I created the EmojiDisplayComponent. To enable fast lookup, I made an imageMap object that links particular words ("happy," "like," and "sad") to image URLs. When the user types, the component updates the state and shows the matching image if the input contains a keyword.

# Week 8:

U2283556

```
 removed in the next major version
Connected to mongodb://localhost:27017/Week8
New document has been added into your database: {
  name: 'JACK',
  age: 20,
  Gender: 'Male',
  Salary: 3456,
  _id: new ObjectId('673cc97f616a8a136ff8382a'),
  __v: 0
}
Data inserted successfully!
```

```
_id: ObjectId('673cc8568b623cc87abb31ad')
name : "Simon"
age : 42
Gender : "Male"
Salary : 3456
__v : 0
```

```
_id: ObjectId('673cc8568b623cc87abb31ae')
name : "Neesha"
age : 23
Gender : "Female"
Salary : 1000
__v : 0
```

```
_id: ObjectId('673cc8568b623cc87abb31af')
name : "Mary"
age : 27
Gender : "Female"
Salary : 5402
__v : 0
```

```
_id: ObjectId('673cc8568b623cc87abb31b0')
name : "Mike"
age : 40
Gender : "Male"
Salary : 4519
__v : 0
```

U2283556

```
Showing 5 documents with all information:
{
  _id: new ObjectId('673cc8568b623cc87abb31ac')
  name: 'Yousuf',
  age: 44,
  Gender: 'Male',
  Salary: 3456,
  __v: 0
}
{
  _id: new ObjectId('673cc8568b623cc87abb31ad')
  name: 'Simon',
  age: 42,
  Gender: 'Male',
  Salary: 3456,
  __v: 0
}
{
  _id: new ObjectId('673cc8568b623cc87abb31ae')
  name: 'Neesha',
  age: 23,
  Gender: 'Female',
  Salary: 1000,
  __v: 0
}
{
  _id: new ObjectId('673cc8568b623cc87abb31af')
  name: 'Mary',
  age: 27,
  Gender: 'Female',
  Salary: 5402,
  __v: 0
}
{
  _id: new ObjectId('673cc8568b623cc87abb31b0')
  name: 'Mike',
  age: 40,
  Gender: 'Male',
  Salary: 4519,
  __v: 0
}
New document has been added into your database:
```

```
Showing unique names with Salary greater than 3000:
JACK
Mary
Mike
Simon
Yousuf
Data inserted successfully!
```

U2283556

https://github.com/Marli1741/WorkSchool

**1. mongoose.connect(uri, options)**

- **Purpose**: Establishes a connection to the MongoDB database.
- **Parameters**:
  - uri: The MongoDB connection string (e.g., mongodb://localhost:27017/Week8).
  - options: An object specifying additional options, such as:
    - useUnifiedTopology: Enables the new MongoDB driver's unified topology layer for better connection management.
    - useNewUrlParser: Ensures the new URL string parser is used.
- **Returns**: A Promise that resolves when the connection is successful.

**2. db.on(event, callback) and db.once(event, callback)**

- **Purpose**: Listen for specific events on the database connection object.
- **Events**:
  - error: Triggered when a connection error occurs.
  - connected: Triggered when the database connection is successfully established.
- **Usage**:
  - on: Attaches a listener that runs every time the specified event occurs.
  - once: Attaches a listener that runs only the first time the specified event occurs.

**3. mongoose.Schema**

- **Purpose**: Defines the structure and constraints of documents in a collection.
- **Parameters**:
  - An object representing the fields (e.g., name, age, Gender, Salary) and their configurations.
- **Example Configurations**:
  - type: Specifies the data type (e.g., String, Number).
  - required: Marks the field as mandatory.
  - default: Specifies a default value for the field.
  - unique: Ensures the field's values are unique across the collection.

**4. mongoose.model(name, schema, collection)**

- **Purpose**: Creates a model for interacting with the database collection.
- **Parameters**:
  - name: The name of the model (e.g., Person).
  - schema: The schema to apply (e.g., PersonSchema).
  - collection (optional): The name of the MongoDB collection (e.g., personCollection).

**5. document.save()**

- **Purpose**: Saves a single document (instance of a model) to the database.
- **Returns**: A Promise that resolves with the saved document or rejects with an error.
- **Example Usage**:

**6. Model.insertMany(documents)**

- **Purpose**: Inserts multiple documents into the database in a single operation.
- **Parameters**:
  - documents: An array of objects, where each object represents a document.
- **Returns**: A Promise that resolves when all documents are successfully inserted or rejects with an error.

**7. Model.find(query)**

- **Purpose**: Retrieves documents from the collection that match the query criteria.
- **Parameters**:
  - query: An object defining the conditions for retrieval (e.g., { Gender: "Female", age: { $gte: 30 } }).
- **Returns**: A Query object that can be further refined (e.g., sorted, limited).

**8. Query Modifiers**

Modifiers are chained to refine the results of a query. Here are the ones used in your script:

- **.sort(fields)**:
  - Sorts the results based on the specified fields.
  - Example: { Salary: 1 } sorts by Salary in ascending order; { Salary: -1 } for descending.
- **.select(fields)**:
  - Specifies the fields to include or exclude in the result.
  - Example: 'name Salary age' includes only name, Salary, and age.
- **.limit(number)**:
  - Limits the number of documents returned.
  - Example: 10 restricts the output to 10 documents.
- **.exec()**:
  - Executes the query and returns a Promise.
  - Commonly used to handle asynchronous results.

**9. .then() and .catch()**

- **Purpose**: Handle the resolved or rejected state of Promises.
- **Usage**:
  - .then(callback): Runs the callback function if the Promise resolves successfully.
  - .catch(callback): Runs the callback function if the Promise is rejected (error occurs).

# WEEK 9:

**It is 5 Like.**


**It is 2 Love.**

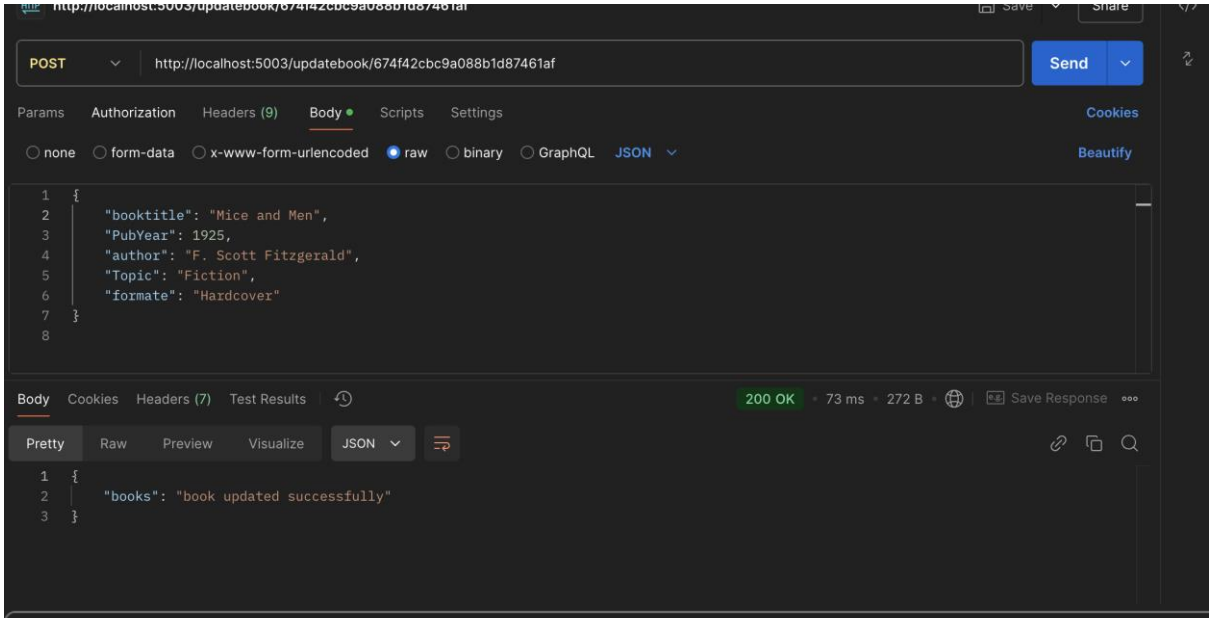

**This is output of Task 2: sad**

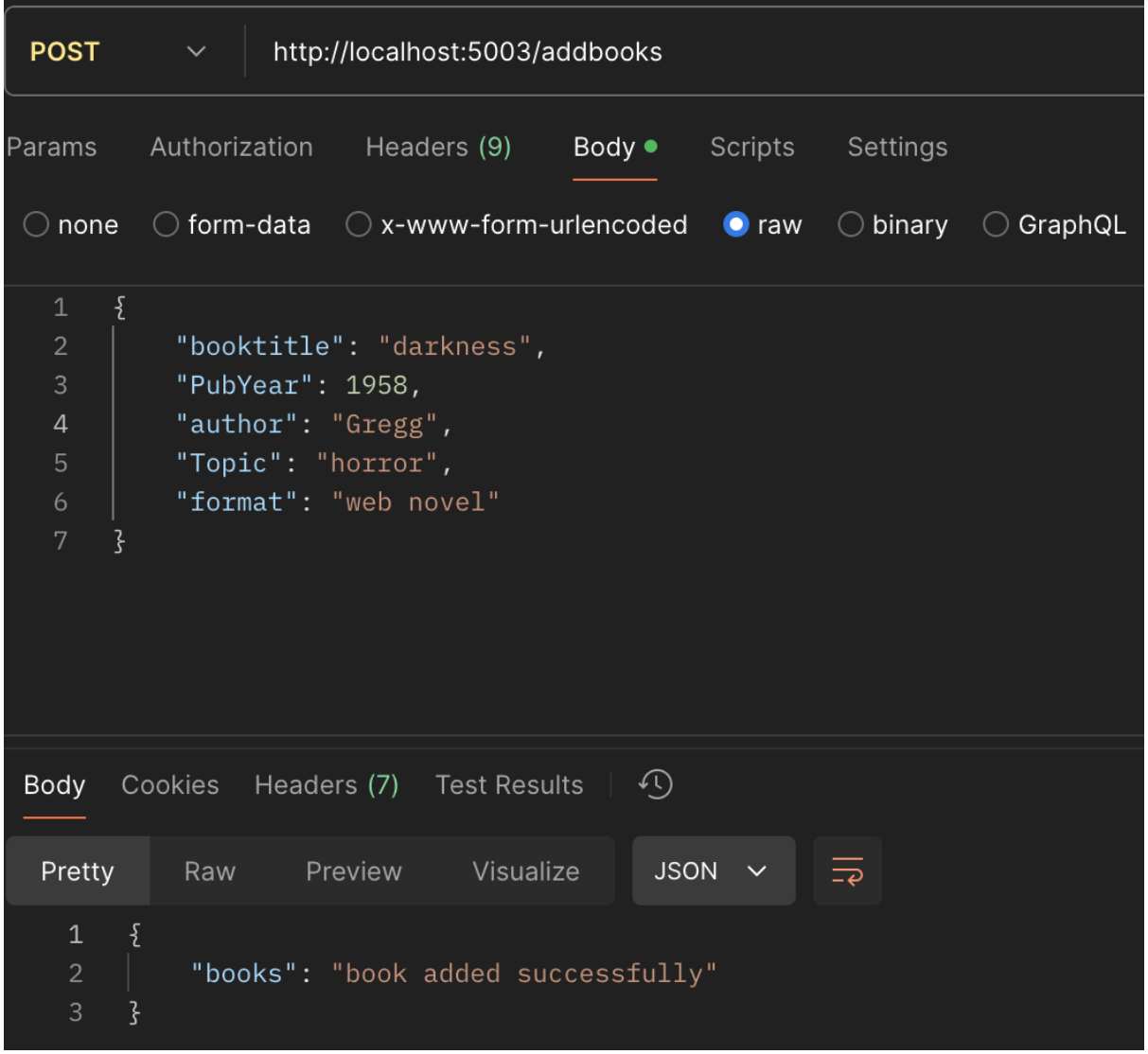
https://github.com/Marli1741/WorkSchool

Functional components are simpler and more concise than class components. They use the useState hook to manage state, eliminating the need for constructors and this.setState(). This makes them easier to understand and maintain. Additionally, hooks like useEffect allow functional components to handle side effects, making them more versatile. For these reasons, functional components are often the preferred choice for new React projects.

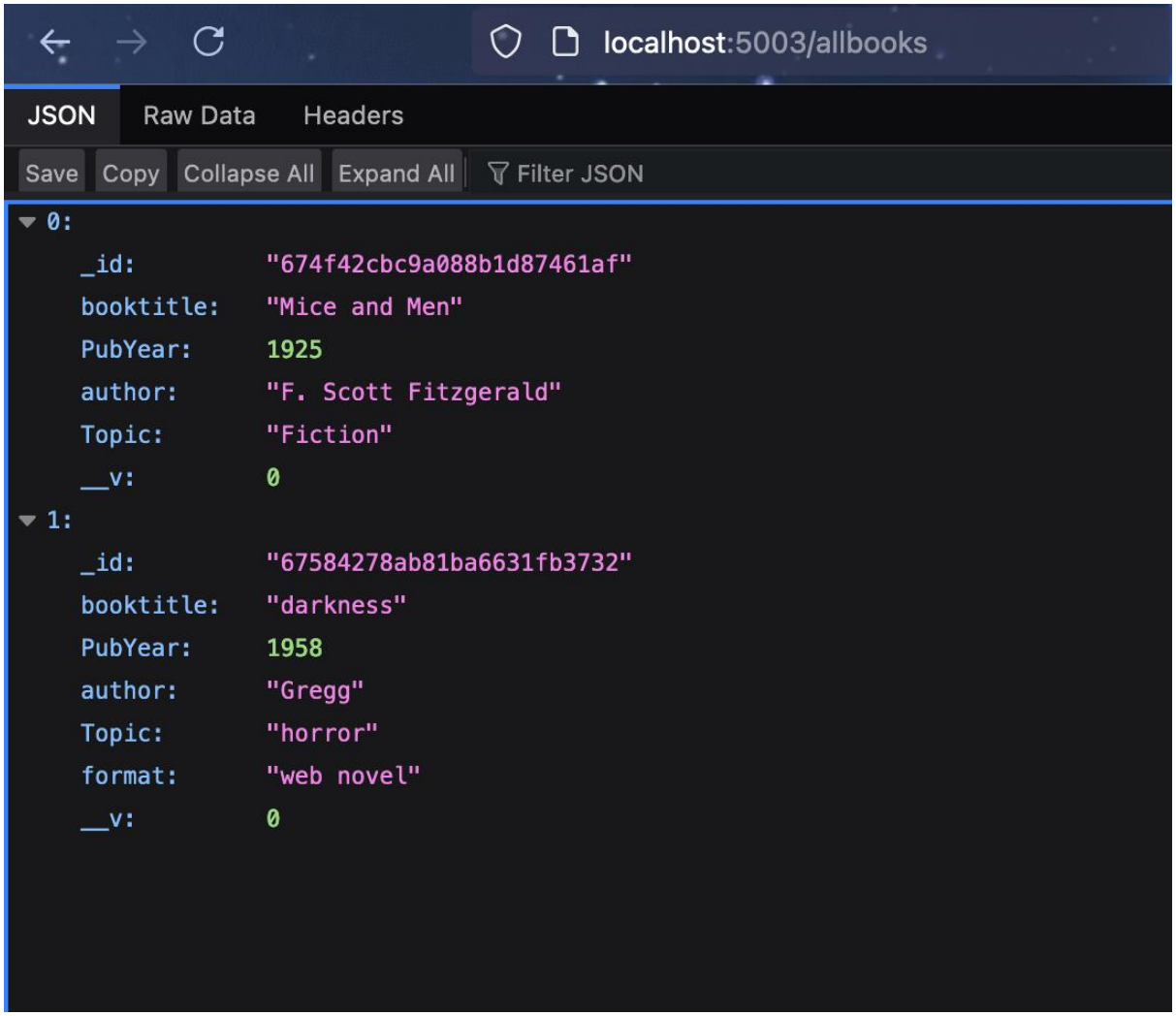
# Week 10:

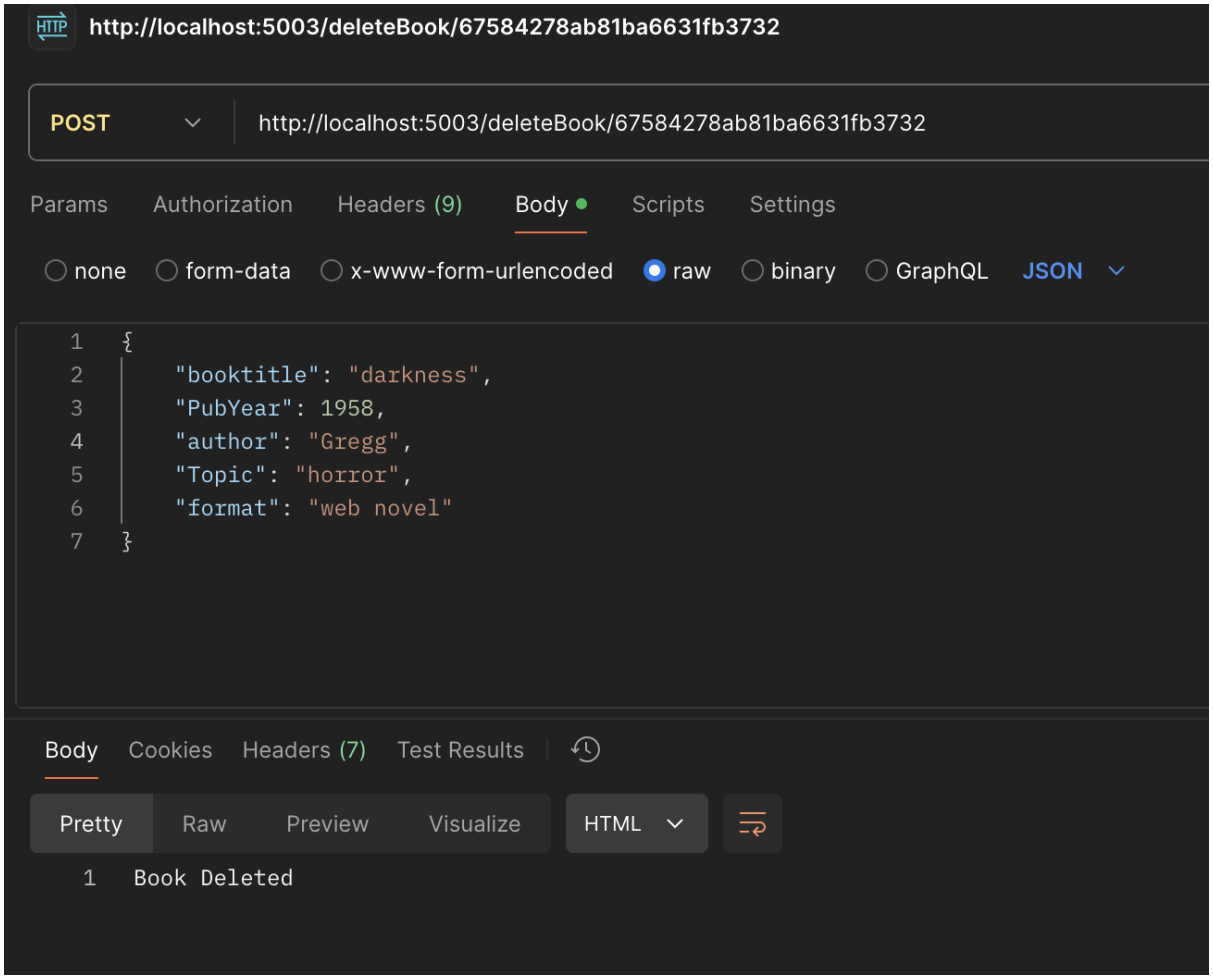
U2283556


POST
http://localhost:5003/updatebook/674f42cbc9a088b1d87461af
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"booktitle": "Mice and Men",
"PubYear": 1925,
"author": "F. Scott Fitzgerald",
"Topic": "Fiction",
"formate": "Hardcover"
}
Body
Cookies
Headers (7)
Test Results
200 OK
73 ms
272 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"books": "book updated successfully"
}


POST
http://localhost:5003/addbooks
Params
Authorization
Headers (9)
Body
Scripts
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
{
"booktitle": "darkness",
"PubYear": 1958,
"author": "Gregg",
"Topic": "horror",
"format": "web novel"
}
Body
Cookies
Headers (7)
Test Results
Pretty
Raw
Preview
Visualize
JSON
{
"books": "book added successfully"
}

U2283556


localhost:5003/allbooks
JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON
0:
_id: "674f42cbc9a088b1d87461af"
booktitle: "Mice and Men"
PubYear: 1925
author: "F. Scott Fitzgerald"
Topic: "Fiction"
__v: 0
1:
_id: "67584278ab81ba6631fb3732"
booktitle: "darkness"
PubYear: 1958
author: "Gregg"
Topic: "horror"
format: "web novel"
__v: 0




This coursework's backend is a Node.js application using Express.js for routing and MongoDB as the database. It follows RESTful principles for easy integration with the frontend.

The backend has five main API endpoints:

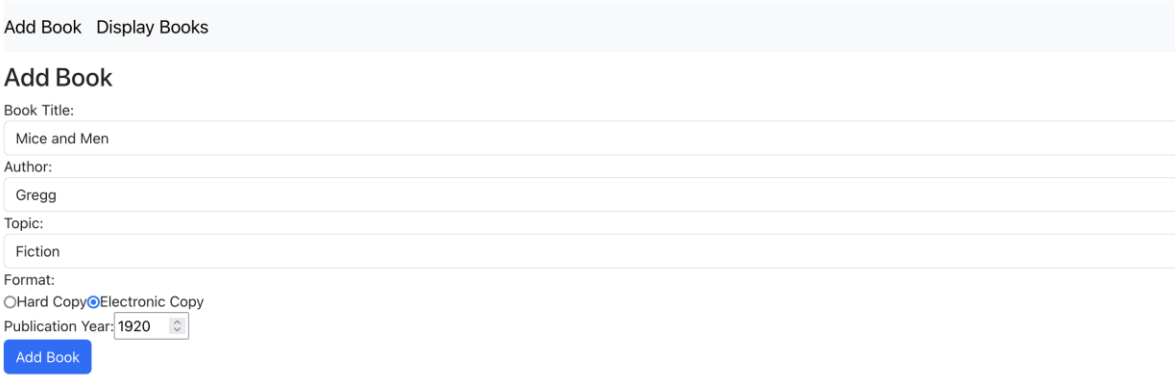
- **GET /books:** Retrieves all books using the find query.
- **GET /books/:id:** Fetches a specific book by ID using the findOne query.
- **POST /addbooks:** Adds a new book to the database using the insertOne query.
- **PUT /books/:id:** Updates an existing book's details using the updateOne query.
- **DELETE /books/:id:** Removes a book using the deleteOne query.

The backend supports all basic CRUD operations (Create, Read, Update, Delete) and includes input validation middleware. It follows a modular MVC architecture, making it easy to maintain and scale. This design provides a strong foundation for future additions like user authentication or more book-related features.

https://github.com/Marli1741/WorkSchool

U2283556

# Week 11:




I built the front-end of this project using React.js. This framework's component-based structure and efficient state management helped me create a user-friendly interface. To connect with the backend, I used Axios to send HTTP requests to the REST API. For example, I sent form data as JSON to the /addbooks endpoint to add books to the MongoDB database and fetched book data from the /books endpoint to populate the interface. I managed form inputs and application state with React's useState hook. I also implemented error handling with .then() and .catch() blocks to provide feedback to the user during operations like adding or updating books.

I planned to create five main components: a form for adding books, a list to display all books, a detailed view for individual books, a form for editing book details, and a function for deleting books.

Working on this project improved my skills in React and Axios. I learned communication between the front-end and backend in building interactive web applications.

https://github.com/Marli1741/WorkSchool