1a. Using MongoDB, create a collection called transactions in database usermanaged (drop if it already exists) and bulk load the data from a json file, **transactions.json**
1b. Upsert the record from the new file called **transactions_upsert.json** in Mongodb.

Transactions.json

```
[
        {
        "_id": 1,
        "item": "book",
        "price": 10
        },
        {
        "_id": 2,
        "item": "pen",
        "price": 2
        },
        {
        "_id": 3,
        "item": "notebook",
        "price": 5
        }
]
```

Using MongoDB, create a collection called transactions in database usermanaged (drop if it already exists) and bulk load the data from a json file, transactions.json

```
mongoimport --db usermanaged --collection transactions --file transactions.json --jsonArray –drop
```

-db usermanaged : creates a database named usermanaged

–collection **transaction :** name of transaction(filename).

–file *transactions.json*: file name

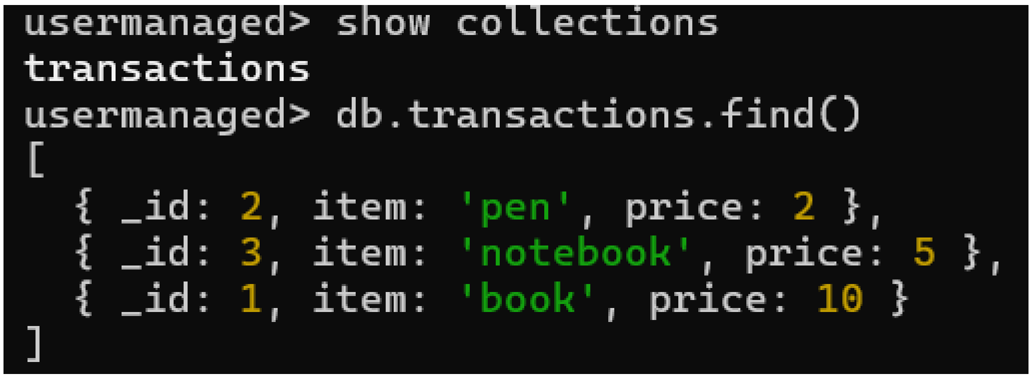
–jsonArray: Type of file.

–drop: if the file exist drop the file.

Open a new terminal in command prompt

mongosh → show dbs → use *database_name* → show collections → db.collection_name.find()

Output:



```
usermanaged> show collections
transactions
usermanaged> db.transactions.find()
[
  { _id: 2, item: 'pen', price: 2 },
  { _id: 3, item: 'notebook', price: 5 },
  { _id: 1, item: 'book', price: 10 }
]
```

Upsert the record from the new file called transactions_upsert.json in Mongodb.

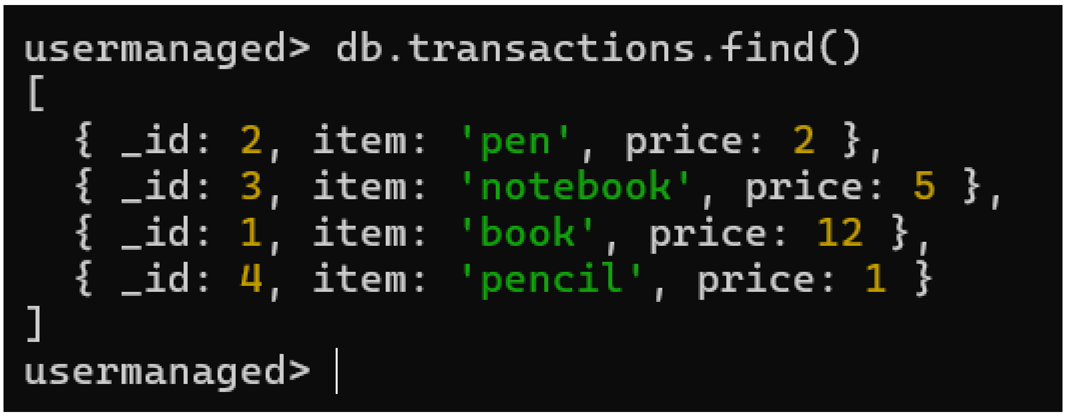
```
mongoimport --db usermanaged --collection transactions --file transactions_upsert.json --jsonArray –upsert
```

–upsert: Updates the file if the document value already exist in file or adds the value if value is not present in document.

Transaction_upsert.json

```
[
        {
        "_id": 1,
        "item": "book",
        "price": 12
        },
        {
        "_id": 4,
        "item": "pencil",
        "price": 1
        }
]
```

Output:



```
usermanaged> db.transactions.find()
[
  { _id: 2, item: 'pen', price: 2 },
  { _id: 3, item: 'notebook', price: 5 },
  { _id: 1, item: 'book', price: 12 },
  { _id: 4, item: 'pencil', price: 1 }
]
usermanaged>
```

Query MongoDB with Conditions: [Create appropriate collection with necessary documents to answer the

query]

a. Find any record where Name is Somu

b. Find any record where total payment amount (Payment.Total) is 600.

c. Find any record where price (Transaction.price) is between 300 to 500.

d. Calculate the total transaction amount by adding up Payment.Total in all records

- use *Database_Name(Give your usn_2ndprog)*
- Syntax:
  db.createCollection("*Collection_name*")

Create a collection named Customers

- db.createCollection("customers")

Insert 3-4 values to the collection

- db.customers.insertMany([

```
        {
            "Name": "Somu",
            "Payment": { "Total": 600 },
            "Transaction": { "price": 450 }
        },
        {
            "Name": "Ravi",
            "Payment": { "Total": 300 },
            "Transaction": { "price": 350 }
        },
        {
            "Name": "John",
            "Payment": { "Total": 200 },
            "Transaction": { "price": 150 }
        },
        {
            "Name": "Sara",
            "Payment": { "Total": 700 },
            "Transaction": { "price": 400 }
        },
        {
            "Name": "Nina",
            "Payment": { "Total": 600 },
            "Transaction": { "price": 500 }
        }
    ])
```

- Find any record where Name is Somu
```
db.customers.find({ "Name": "Somu" })
```

- Find any record where total payment amount (Payment.Total) is 600
```
db.customers.find({ "Payment.Total": 600 })
```

- Find any record where price (Transaction.price) is between 300 to 500
```
db.customers.find({ "Transaction.price": { $gte: 300, $lte: 500 } }
```

- Calculate the total transaction amount by adding up Payment.Total in all records
```
db.customers.aggregate([
   { $group: { _id: null, totalTransactionAmount: { $sum: "$Payment.Total" } } }])
```

a. Write a program to check request header for cookies.
b. Write node.js program to print the a car object properties, delete the second property and get length of the object.

Note: The line in red color is comments

Write a program to check request header for cookies.

```
// importing the modules

const express = require('express'); //imports express module(web framework) for Nodejs

const cookieParser = require('cookie-parser'); // imports cookieParser middleware which is
used to parse the cookies to client req


// creating the instances and setting port for the same.

const app = express(); //instance of express appl

const port = 3000; //setting the app in port 3000


//telling the instance to use cookieParser module

app.use(cookieParser()); // tells the application to use cookieparser module


// Defining route for the root URL

app.get('/', (req, res) => { //defines a route handler for GET requests

  console.log('Cookies:', req.cookies); // Logs cookies

  res.send('Check the console for cookies.'); // Send a response to the client

});


// Start the server

app.listen(port, () => {

  console.log(`Server running at http://localhost:${port}`);

});
```

Write node.js program to print the a car object properties, delete the second property and get length of the object.

```
// Define a car object with properties

const car = {

  brand: ‘lamborghini’,

  model: ‘Sian’,

  year: 2020,

  color: 'red'

 };


 // Print all properties of the car object

 console.log('Car properties:', car);


 // Delete the second property (in this case, 'model')

 const keys = Object.keys(car);

 if (keys.length > 1) {

  delete car[keys[1]];

 }


 // Print the car object after deletion

 console.log('Car properties after deletion:', car);


 // Get the length of the car object

 const length = Object.keys(car).length;

 console.log('Number of properties in the car object:', length);
```

Implement all CRUD operations on a File System using Node JS

Step1:
Open your USN folder in VS Code → Create a new folder called (5th_prog) → Open terminal in VS Code and write the following commands

```
npm init -y

npm install fs
```

Step2: Create a new file called **script.js** in 5th_prog folder

```
const fs = require('fs'); //import fs package

const path = './data'; // directory/folder where files will be stored while creating the file



// checking whether the directory/folder exist if yes ignore else creating the directory/folder

if (!fs.existsSync(path)) {

  fs.mkdirSync(path);

}



// Create function

const createFile = (filename, content) => {

    // If the file does exist, its content will be replaced with the new content. If file exist content will be
replaced with new content.

//err--> This is a callback function that gets called when the write operation completes. It takes one
parameter, err, which will contain an error object if an error occurred, or null if the operation was successful

  fs.writeFile(`${path}/${filename}`, content, (err) => {

        // If there was an error during the file write operation, this block will execute, and it will log an error
message to the console along with the error object.

    if (err) console.error('Error creating file:', err);

    else console.log('File created:', filename);

  });

};



// Read function
```

```
const readFile = (filename) => {
    // utf-8 This specifies the encoding to use when reading the file. takes 2 parameter: 1--> err(error object)
and 2--> data(content)
  fs.readFile(`${path}/${filename}`, 'utf8', (err, data) => {
    if (err) console.error('Error reading file:', err);
    else console.log('File content:', data);
  });
};


// Update function
const updateFile = (filename, content) => {
  fs.writeFile(`${path}/${filename}`, content, (err) => {
    if (err) console.error('Error updating file:', err);
    else console.log('File updated:', filename);
  });
};


// Delete function
const deleteFile = (filename) => {
    // fs.unlink is used to delete a file.
  fs.unlink(`${path}/${filename}`, (err) => {
    if (err) console.error('Error deleting file:', err);
    else console.log('File deleted:', filename);
  });
};


// filename given in variable which will be called in function call
const filename = 'sample.txt';


// file content written
// const content = 'This is a sample content';    // call this variable in function call


// Creating the file

createFile(filename, "Sample file created using node js");


// Reading the  file

readFile(filename);


// Updating the file and updating its content {you can create a new varibale and call that variable in fucntion
call just like const content present above}

updateFile(filename, 'Sample file is updated using node js');


// Deleting the file

deleteFile(filename)
```

Step3: To run the script open terminal and write the below command

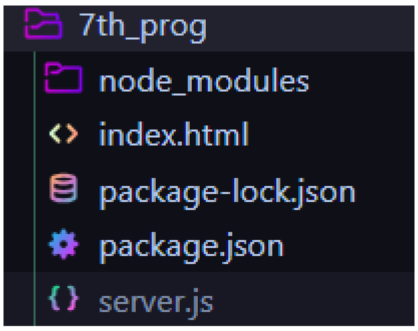
```
node script.js
```

Develop an authentication mechanism with email_id and password using HTML and Express JS (POST method)

STEPS TO FOLLOW:

- Open your USN folder in VS Code
- Create a new directory or folder called 7th_prog (else open this folder in VS Code)
- Get inside that folder (*If you have opened the 7th_prog folder in VS Code ignore this step*)
- Open Terminal in VS Code and write the following commands
    1. npm init -y
    2. npm install express body-parser
- Create 2 files
    1. Server.js
    2. Index.html

STRUCTURE OF THE FILE'S



```
7th_prog
  node_modules
  index.html
  package-lock.json
  package.json
  server.js
```

.

Server.js

```
const express = require('express');
const bodyParser = require('body-parser');

const app = express();
const PORT = 3000; //change the port if you get error

// Middleware to parse the body of POST requests
app.use(bodyParser.urlencoded({ extended: true }));

// Route to serve the login page
app.get('/', (req, res) => {
  res.sendFile(__dirname + '/index.html');
});

// Route to handle login
app.post('/login', (req, res) => {
  const { email, password } = req.body;


  // assigning random email id and password for testing {keep your usn@example.com as
email id}
```

```
  const validEmail = 'user@example.com'; {4mt22ci001@example.com}
  const validPassword = 'pass1234';

  if (email === validEmail && password === validPassword) {
    res.send('Login successful!');
  } else {
    res.send('Invalid email or password.');
  }
});

//starting the server
app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

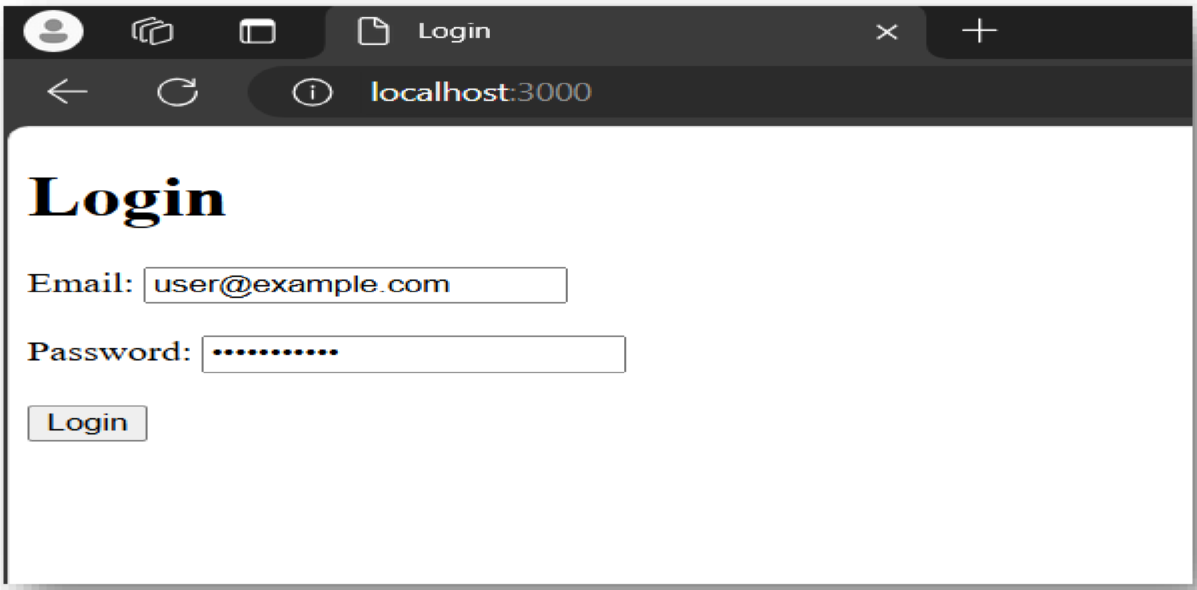
Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Login</title>
</head>
<body>
  <h1>Login</h1>
  <form action="/login" method="POST">
    <label for="email">Email:</label>
    <input type="email" id="email" name="email" required>
    <br><br>
    <label for="password">Password:</label>
    <input type="password" id="password" name="password" required>
    <br><br>
    <button type="submit">Login</button>
  </form>
</body>
</html>
```

How to run the script

- Node server.js

OUTPUT

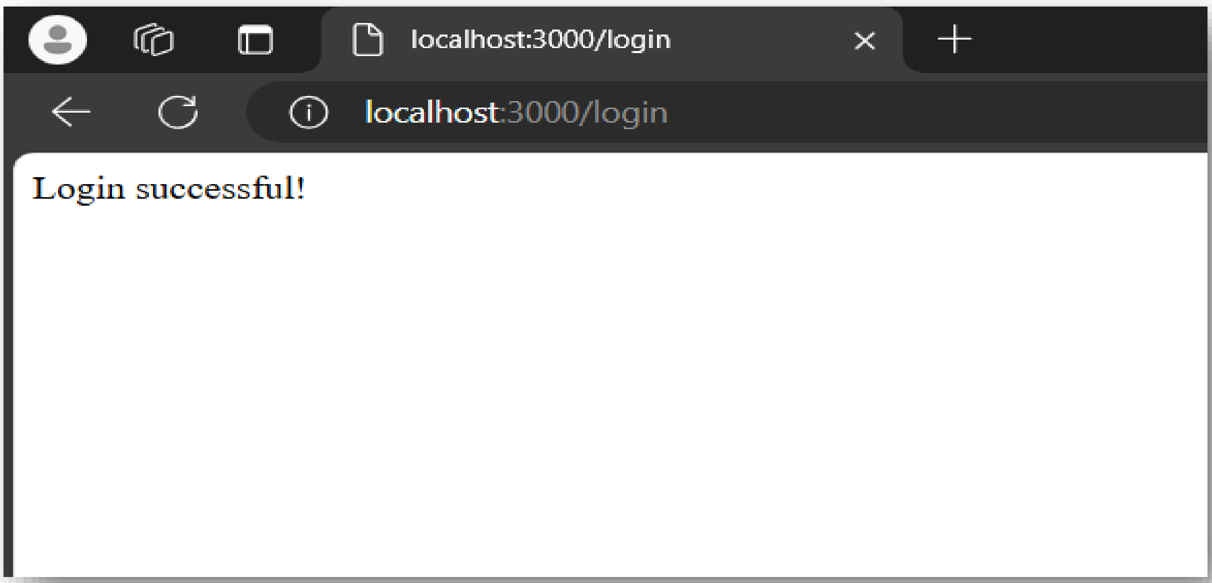


Login

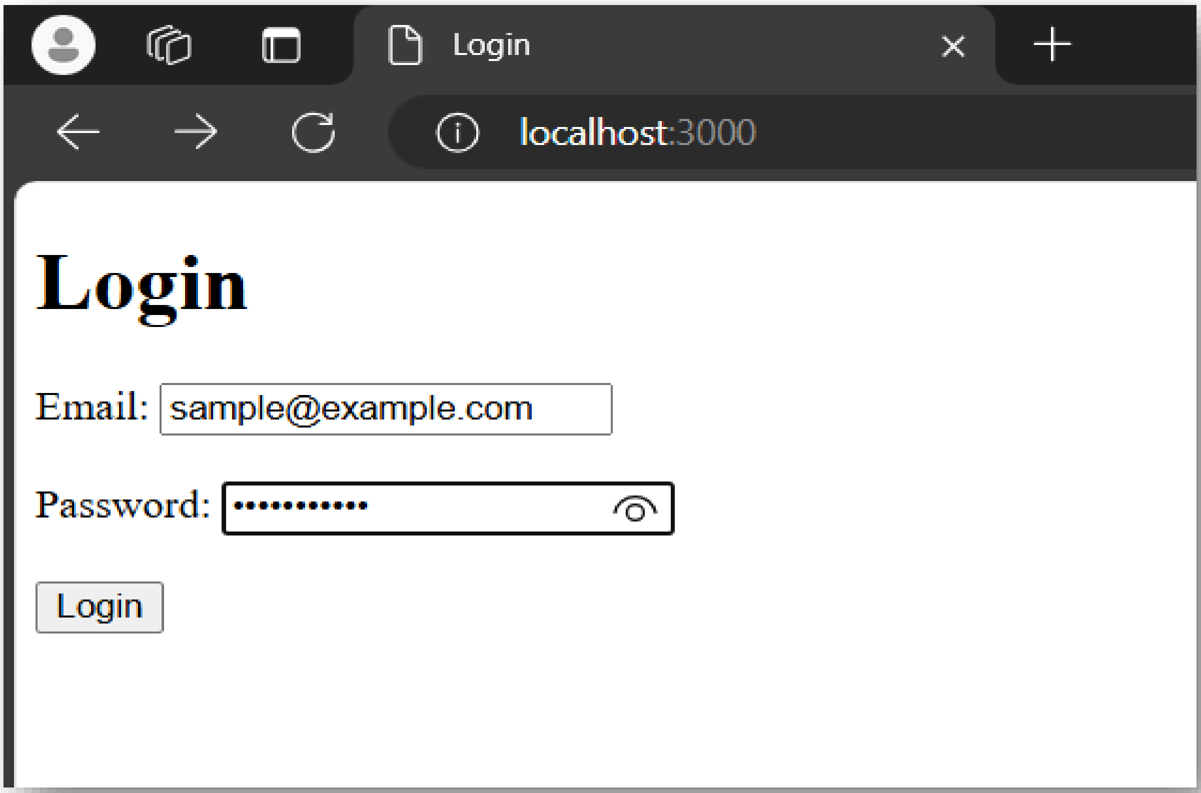
Email: user@example.com

Password: ••••••••••••

Login



Login successful!

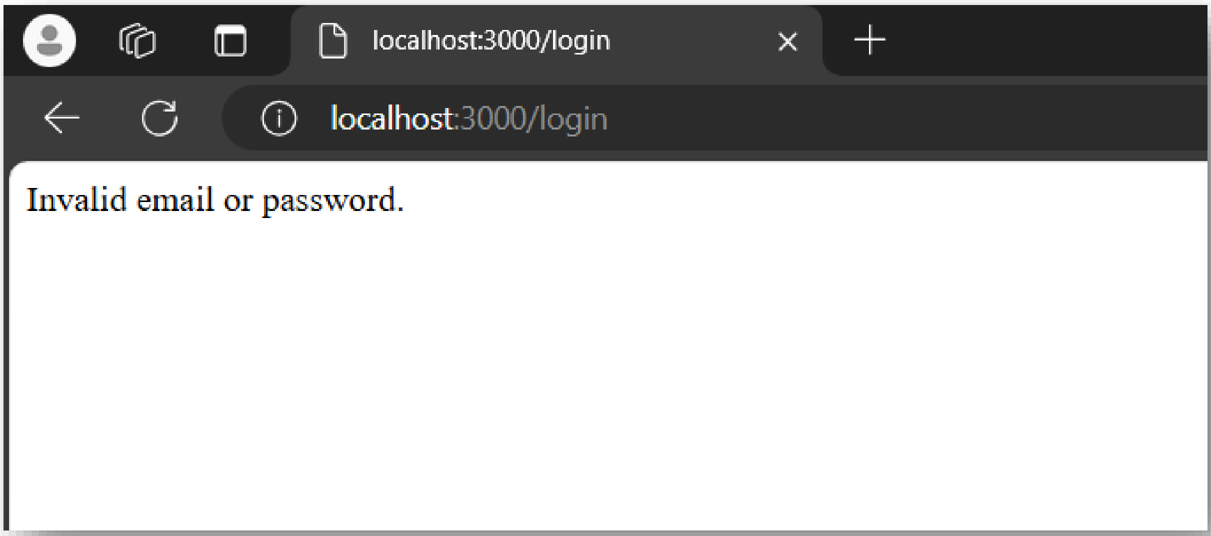


Login

Email: sample@example.com

Password: ••••••••••••

Login



Invalid email or password.

Develop two routes: find_prime_100 and find_cube_100 which prints prime numbers less than 100 and cubes less than 100 using Express JS routing mechanism

```
const express = require('express');
const app = express();
const port = 3000;

// Function to find prime numbers less than 100
const findPrimes = () => {
  const primes = [];
  for (let i = 2; i < 100; i++) {
    let isPrime = true;
    for (let j = 2; j <= Math.sqrt(i); j++) {
      if (i % j === 0) {
        isPrime = false;
        break;
      }
    }
    if (isPrime) primes.push(i);
  }
  return primes;
};

// Function to find cubes less than 100
const findCubes = () => {
  const cubes = [];
  for (let i = 1; i ** 3 < 100; i++) {
    cubes.push(i ** 3);
  }
  return cubes;
};

// Middleware to log requests
app.use((req, res, next) => {
  console.log(`${req.method} request for '${req.url}'`);
  next();
});

// Route to get prime numbers less than 100
app.get('/prime', (req, res) => {
  const primes = findPrimes();
  res.json(primes);
});

// Route to get cubes less than 100
app.get('/cube', (req, res) => {
  const cubes = findCubes();
  res.json(cubes);
});

// Start the server
app.listen(port, () => {
  console.log(`Server is running on http://localhost:${port}/find_prime_100`);
  console.log(`Server is running on http://localhost:${port}/find_cube_100`);

});
```

Develop a React code to build a simple search filter functionality to display a filtered list based on the search query entered by the user.

Steps

1. Open your USN folder/9th_prog in Vs Code
2. Open terminal and write following command
    a. npx create-react-app search-filter {*app_name*}
    b. cd search-filter
3. Replace the code in "src/App.js" with the script given below.
4. To run the program type the command
    a. npm start

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

// Sample data to search
const items = ['Apple','Banana','Cherry','Dragonfruit','Mango','Orange','Grape'];

function App() {
 // State to manage the search query
 const [query, setQuery] = useState('');

 // Function to handle input change
 const handleInputChange = (event) => {
  setQuery(event.target.value);
 };

 // Filtered list based on the search query
 const filteredItems = items.filter((item) =>
  item.toLowerCase().includes(query.toLowerCase())
 );

 return (
  <div style={{ padding: '20px' }}>
   <h1>Search Filter</h1>
   <input
    type="text"
    placeholder="Search..."
    value={query}
    onChange={handleInputChange}
    style={{ padding: '10px', fontSize: '16px', width: '300px' }}
   />
   <ul>
    {filteredItems.map((item, index) => (
     <li key={index} style={{ listStyle: 'none', padding: '5px 0' }}>
      {item}
     </li>
    ))}
   </ul>
  </div>
 );
}

export default App;
```

Develop a React code to collect data from rest API.

Steps

1. Open your USN folder/10th_prog in Vs Code
2. Open terminal and write following command
    a. npx create-react-app rest_api {*app_name*}
    b. cd search-filter
3. Create a new file called FetchData.js inside src folder
4. Replace the code in “src/App.js” with the script given below.
5. To run the program, type the command
    a. npm start

FetchData.js

```
// src/FetchData.js
import React, { useState, useEffect } from 'react';

const FetchData = () => {
  const [data, setData] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    // Fetch data from a REST API
    const fetchData = async () => {
      try {
        const response = await fetch('https://jsonplaceholder.typicode.com/posts');
        if (!response.ok) {
          throw new Error('Network response was not ok');
        }
        const data = await response.json();
        setData(data);
        setLoading(false);
      } catch (error) {
        setError(error);
        setLoading(false);
      }
    };

    fetchData();
  }, []);

  if (loading) return <div>Loading...</div>;
  if (error) return <div>Error: {error.message}</div>;

  return (
    <div>
      <h1>Fetched Data</h1>
      <ul>
        {data.map(item => (
          <li key={item.id}>{item.title}</li>
      ))}
    </ul>
  </div>
 );
};

export default FetchData;
```

/src/App.js

```
// src/App.js
import React from 'react';
import FetchData from './FetchData';

function App() {
 return (
  <div className="App">
   <header className="App-header">
    <FetchData />
   </header>
  </div>
 );
}

export default App;
```