

FEDF Assignment 3 – Matrix Sum Calculator Using ReactJS (Props and State)

Name: M. Venkata Avinash

Roll Number: 2420030458

Section: 6

Overview

This document explains the complete implementation and working of the **Matrix Sum Calculator** application developed using **ReactJS**.

The project demonstrates the use of **React State Hooks** and **Props** to take matrix inputs and calculate their sum dynamically.

Solution Overview

The application allows users to:

- Define the number of rows and columns for two matrices.
- Enter individual matrix values.
- Compute and display the **sum** of both matrices dynamically.

React **state management** is used to handle matrix values, and **conditional rendering** ensures proper display of matrices and results.

Commands Executed in Terminal

```
npx create-react-app matrix-sum
```

```
cd matrix-sum
```

```
npm install
```

```
npm start
```

MatrixSumCalculator.jsx Code

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

```
const MatrixSumCalculator = () => {
```

```
const [rowsA, setRowsA] = useState(2);
const [colsA, setColsA] = useState(2);
const [rowsB, setRowsB] = useState(2);
const [colsB, setColsB] = useState(2);
const [matrixA, setMatrixA] = useState([]);
const [matrixB, setMatrixB] = useState([]);
const [resultMatrix, setResultMatrix] = useState([]);
```

```
const handleMatrixASetup = () => {
  const emptyMatrix = Array.from({ length: rowsA }, () =>
    Array.from({ length: colsA }, () => 0)
  );
  setMatrixA(emptyMatrix);
};
```

```
const handleMatrixBSetup = () => {
  const emptyMatrix = Array.from({ length: rowsB }, () =>
    Array.from({ length: colsB }, () => 0)
  );
  setMatrixB(emptyMatrix);
};
```

```
const handleMatrixChange = (matrixSetter, rowIndex, colIndex, value) => {
  matrixSetter((prevMatrix) => {
    const updatedMatrix = [...prevMatrix];
    updatedMatrix[rowIndex][colIndex] = parseInt(value) || 0;
    return updatedMatrix;
  });
};
```

```
};
```

```
const calculateSum = () => {  
  if (  
    matrixA.length !== matrixB.length ||  
    matrixA[0].length !== matrixB[0].length  
  ) {  
    alert('Matrix A and B must have the same dimensions!');  
    return;  
  }  
}
```

```
const result = matrixA.map((row, i) =>  
  row.map((val, j) => val + matrixB[i][j])  
);  
setResultMatrix(result);  
};
```

```
const renderMatrix = (matrix, setMatrix) => (  
  <table>  
    <tbody>  
      {matrix.map((row, rowIndex) => (  
        <tr key={rowIndex}>  
          {row.map((col, colIndex) => (  
            <td key={colIndex}>  
              <input  
                type="number"  
                value={col}  
                onChange={(e) =>
```

```

        handleMatrixChange(setMatrix, rowIndex, colIndex, e.target.value)
    }
    style={{ width: '50px', textAlign: 'center' }}
  />
</td>
  )}
</tr>
  )}
</tbody>
</table>
);

```

```

return (
  <div>
    <h2>Matrix Sum Calculator</h2>

    <div>
      <h3>Matrix A Dimensions</h3>
      <label>
        Rows:
        <input
          type="number"
          value={rowsA}
          onChange={(e) => setRowsA(parseInt(e.target.value) || 0)}
          style={{ width: '50px', marginLeft: '5px', marginRight: '10px' }}
        />
      </label>
      <label>

```

Columns:

```
<input
  type="number"
  value={colsA}
  onChange={(e) => setColsA(parseInt(e.target.value) || 0)}
  style={{ width: '50px', marginLeft: '5px', marginRight: '10px' }}
/>
</label>
<button onClick={handleMatrixASetup}>Set Matrix A</button>
</div>
```

```
{matrixA.length > 0 && (
  <>
    <h3>Matrix A</h3>
    {renderMatrix(matrixA, setMatrixA)}
  </>
)}
```

```
<div>
  <h3>Matrix B Dimensions</h3>
  <label>
    Rows:
    <input
      type="number"
      value={rowsB}
      onChange={(e) => setRowsB(parseInt(e.target.value) || 0)}
      style={{ width: '50px', marginLeft: '5px', marginRight: '10px' }}
    />
```

```

</label>

<label>

  Columns:

  <input

    type="number"

    value={colsB}

    onChange={(e) => setColsB(parseInt(e.target.value) || 0)}

    style={{ width: '50px', marginLeft: '5px', marginRight: '10px' }}

  />

</label>

<button onClick={handleMatrixBSetup}>Set Matrix B</button>

</div>

```

```

{matrixB.length > 0 && (
  <>
    <h3>Matrix B</h3>

    {renderMatrix(matrixB, setMatrixB)}

  </>
)}

```

```

{matrixA.length > 0 && matrixB.length > 0 && (
  <button onClick={calculateSum} style={{ marginTop: '10px' }}>

    Calculate Sum

  </button>

)}

```

```

{resultMatrix.length > 0 && (
  <>

```

```

    <h3>Result Matrix</h3>

    <table>

    <tbody>

      {resultMatrix.map((row, rowIndex) => (

        <tr key={rowIndex}>

          {row.map((col, colIndex) => (

            <td key={colIndex}>{col}</td>

          ))}

        </tr>

      ))}

    </tbody>

  </table>

</>

)}

</div>

);

};

```

```
export default MatrixSumCalculator;
```

App.jsx Code

```

import React from "react";

import MatrixSumCalculator from "../MatrixSumCalculator";

const App = () => {

  return (

    <div className="container">

      <h2>Matrix Sum Calculator</h2>

```

```
    <MatrixSumCalculator />
  </div>

);
};
```

```
export default App;
```

index.js Code

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";

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

Outputs:

Matrix Sum Calculator

Matrix Sum Calculator

Matrix A Dimensions

Rows: Columns: Set Matrix A

Matrix A

<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>
<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>
<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>

Matrix B Dimensions

Rows: Columns: Set Matrix B

Matrix B

<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>
<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>
<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>
<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>

```
[eslint]
src/App.js
  Line 12:16:  eval can be harmful  no-eval

Search for the keywords to learn more about each warning.
To ignore, add // eslint-disable-next-line to the line before.

#WARNING in [eslint]
src/App.js
  Line 12:16:  eval can be harmful  no-eval

webpack compiled with 1 warning
^C
yashwanthshaga@Yashwanths-MacBook-Air calculator % history
996 cd ~/Desktop\n
997 git clone https://github.com/Yashwanth410/FEDF_ASSIGNMENTS.git\n
998 cp -r ~/matrix-sum ~/Desktop/FEDF_ASSIGNMENTS/Assignment_3_MatrixSum\n
999 cd ~/Desktop/FEDF_ASSIGNMENTS\n
1000 git add .\ngit commit -m "Added Assignment 3 - Matrix Sum Calculator using React props and state"\ngit push origin main\n
1001 npx create-react-app calculator\ncd calculator\n
1002 ls ~/Downloads\n
1003 cd ~/calculator_using_reactJS/Calculate\n
1004 ls ~\n
1005 cd ~/calculator_using_reactJS\n
1006 pwd\n
1007 npm install\n
1008 npm run dev\n
1009 npm start\n
1010 nano src/App.js\n
1011 npm start\n
yashwanthshaga@Yashwanths-MacBook-Air calculator %
```

GitHub Repository

<https://github.com/Avinash-2007-M/FEDF-Assignments>

