

PROJECT REPORT

**CALCULATOR**

**NAME : RIYA**

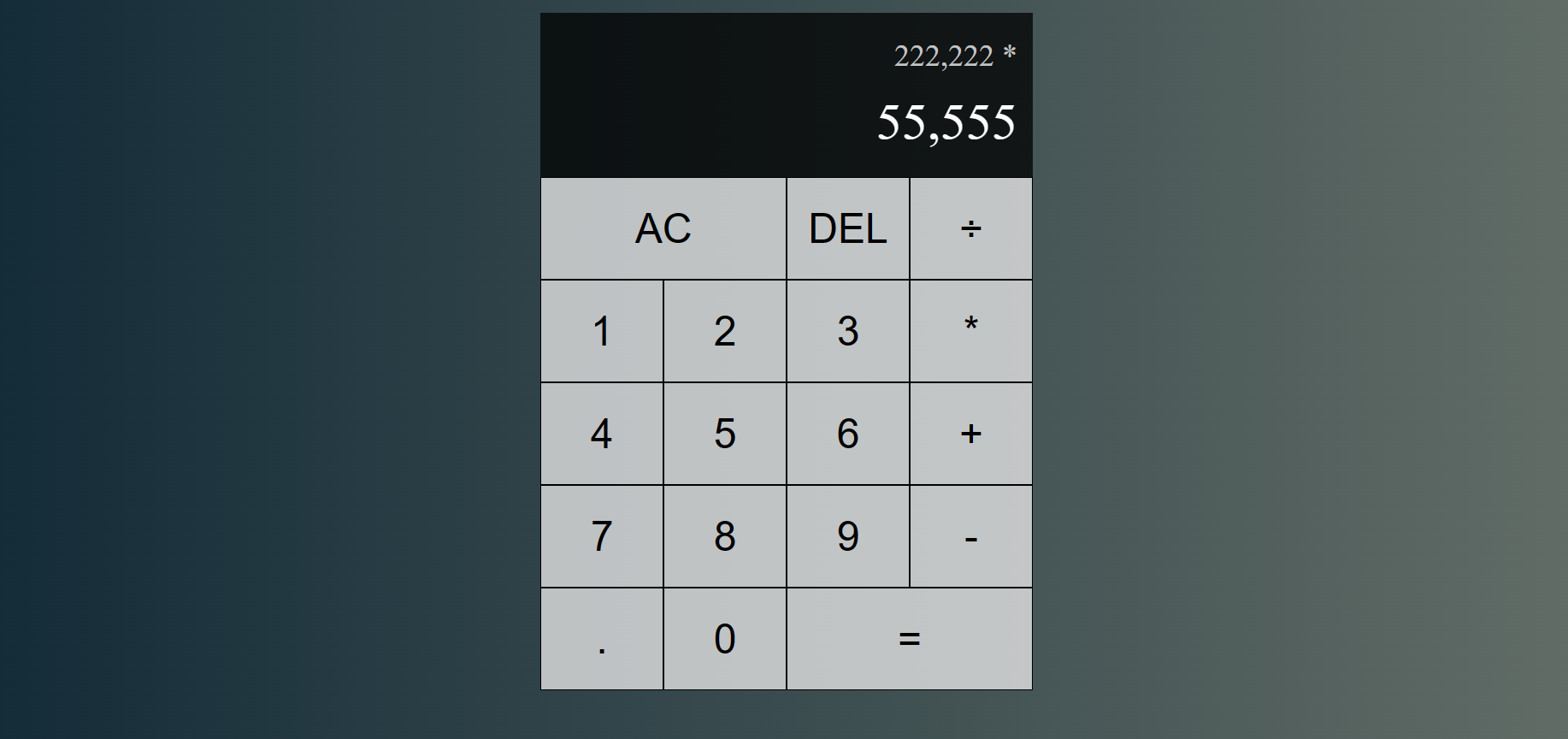
**ROLL NO. : 2110991186**

**GROUP : G19**

**SUBMITTED TO :**

**PRITPAL MAM**

**INTRODUCTION**

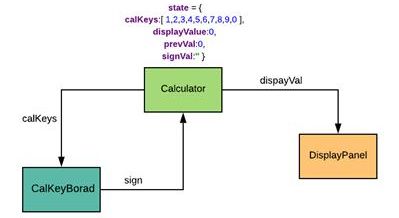


The JavaScript React calculator project implements a modern and user-friendly calculator application. It leverages React for the user interface, providing a dynamic and responsive design. Users can perform basic arithmetic operations, utilize advanced scientific functions, and customize the calculator's appearance. The two-line display layout shows the previous operand and operation at the top, with the current operand displayed below, ensuring clear and organized calculation tracking. This calculator employs JavaScript for logic, CSS for styling, and `Intl.NumberFormat` for numeric value formatting, offering a comprehensive and aesthetically pleasing calculator experience.

**OBJECTIVES**

* React-Based User Interface: Utilize React to design a responsive and dynamic user interface, providing an intuitive and efficient environment for users to perform mathematical calculations.
* Basic Arithmetic Operations: Implement standard arithmetic operations, such as addition, subtraction, multiplication, and division, using React's component-based approach for clear and organized code.
* Scientific Functions: Leverage React components to incorporate advanced scientific functions, ensuring that the calculator meets the needs of scientific and engineering professionals.
* State Management: Harness React's state management capabilities to handle user inputs and the calculator's current state, enabling real-time updates of calculations.
* Memory Functions: Utilize React's state and component architecture to create memory functions for storing and retrieving values during calculations.
* Cross-Platform Compatibility: Ensure cross-platform compatibility by utilizing React to build a web-based application that can be accessed on various devices and operating systems.

**FLOWCHART**

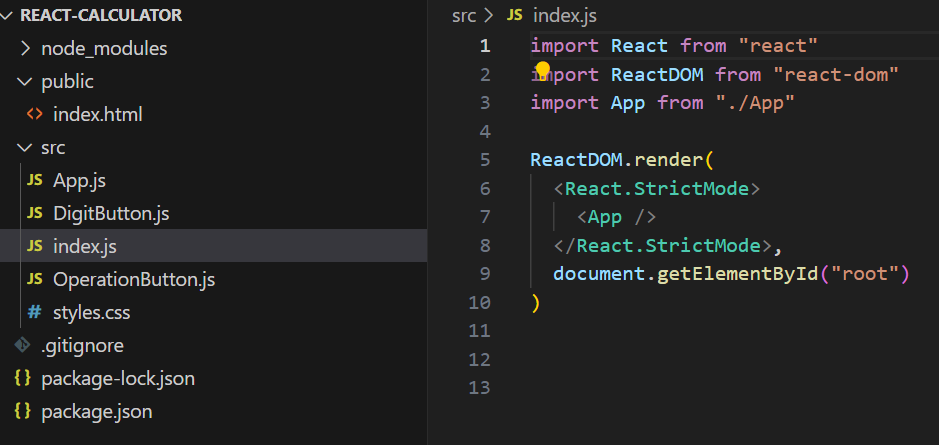


**TECHNOLOGIES USED**

* React: The core technology used to build the user interface and manage component-based rendering. React enables the development of a dynamic and responsive user interface, making it an ideal choice for creating a modern calculator application. The component-based architecture of React allows for organized and efficient code development, making it easier to handle user inputs and state management.
* JavaScript: The programming language for building the calculator's logic and functionality.
* CSS: Used for styling and formatting the user interface elements of the calculator.
* Intl.NumberFormat: An internationalization feature in JavaScript used to format numbers, allowing the calculator to display numeric values appropriately, including thousands separators.

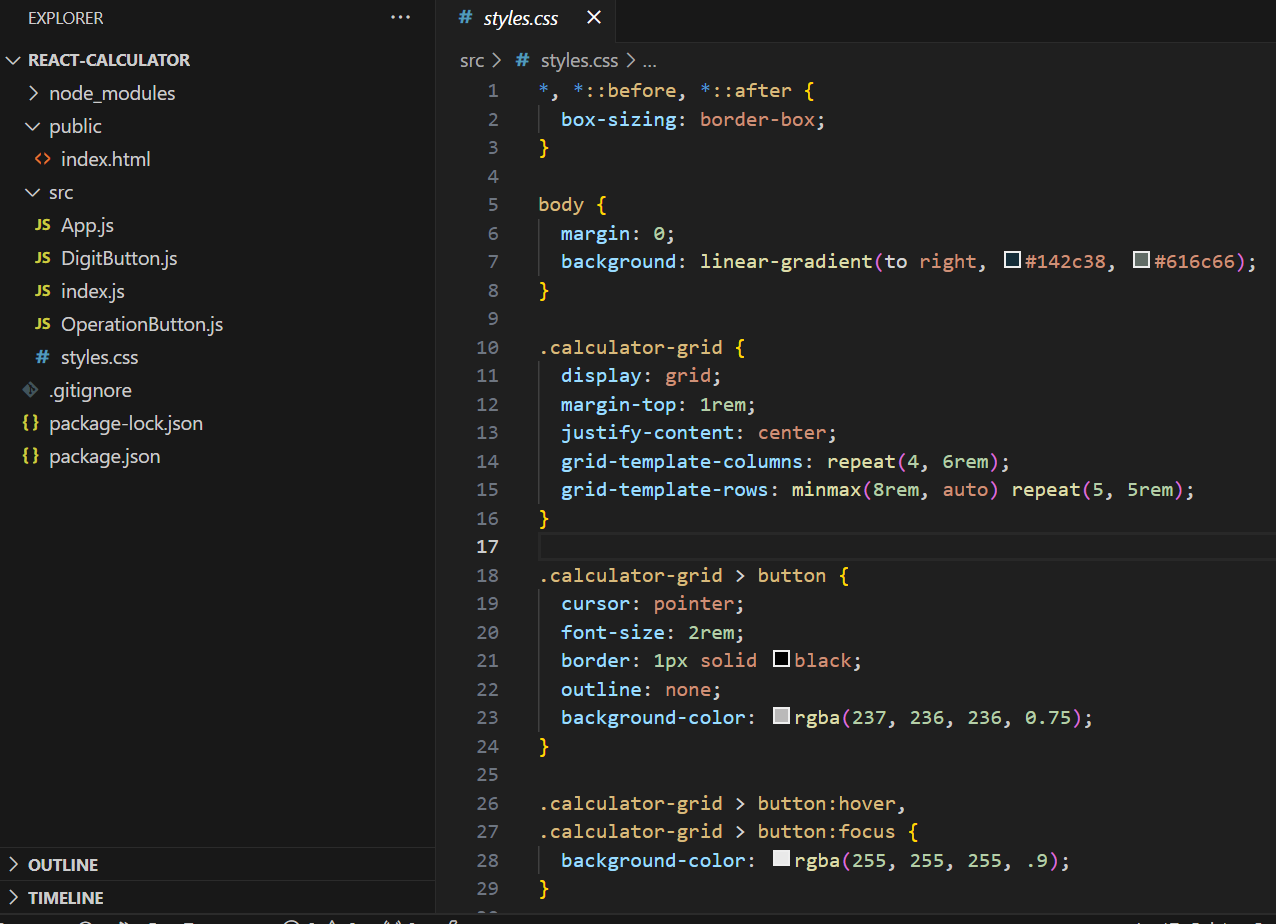
**EXPLANATION**

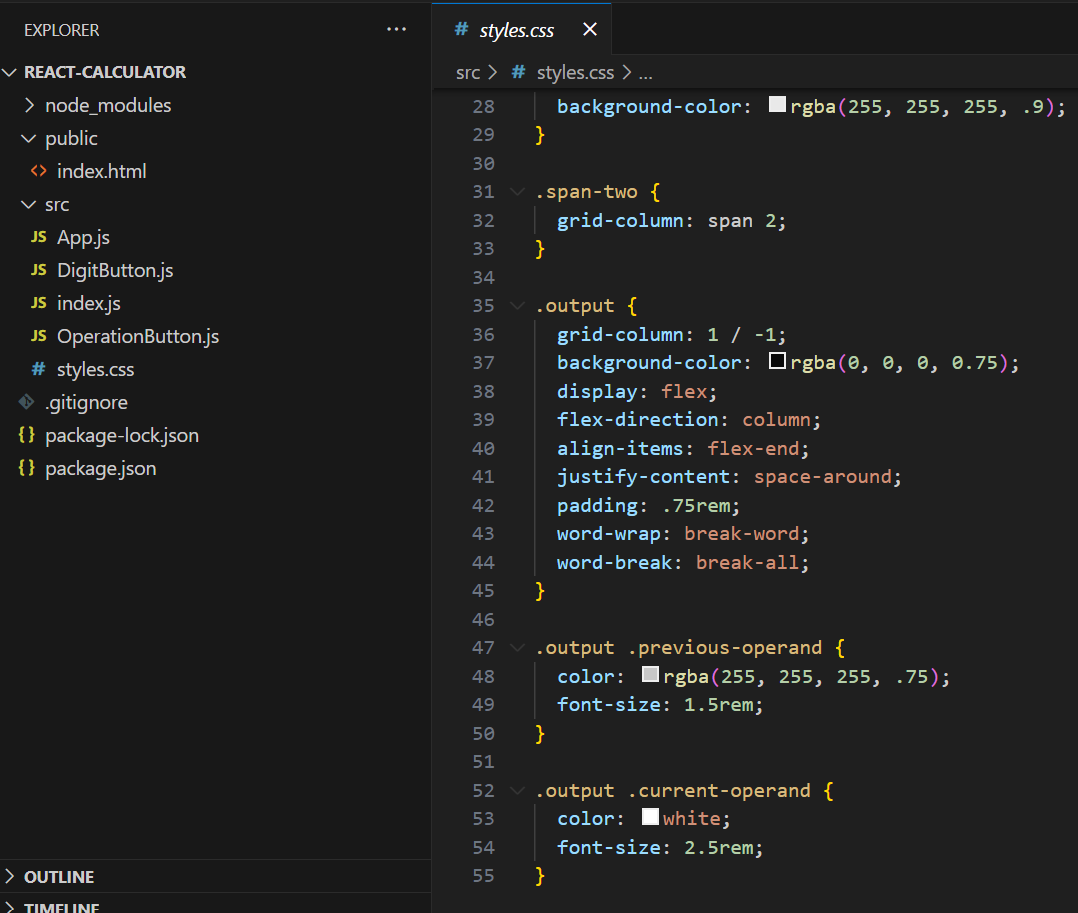
**INDEX.JS**



This code serves as the starting point for a React application. It imports necessary React and ReactDOM libraries. The `ReactDOM.render()` function renders the main `App` component within a DOM element identified as "root." React's `<React.StrictMode>` wrapper is used to detect and highlight potential development issues. In summary, this code initializes the application by rendering the root component, creating the user interface, and connecting it to the HTML element with the ID "root," effectively launching the React application.

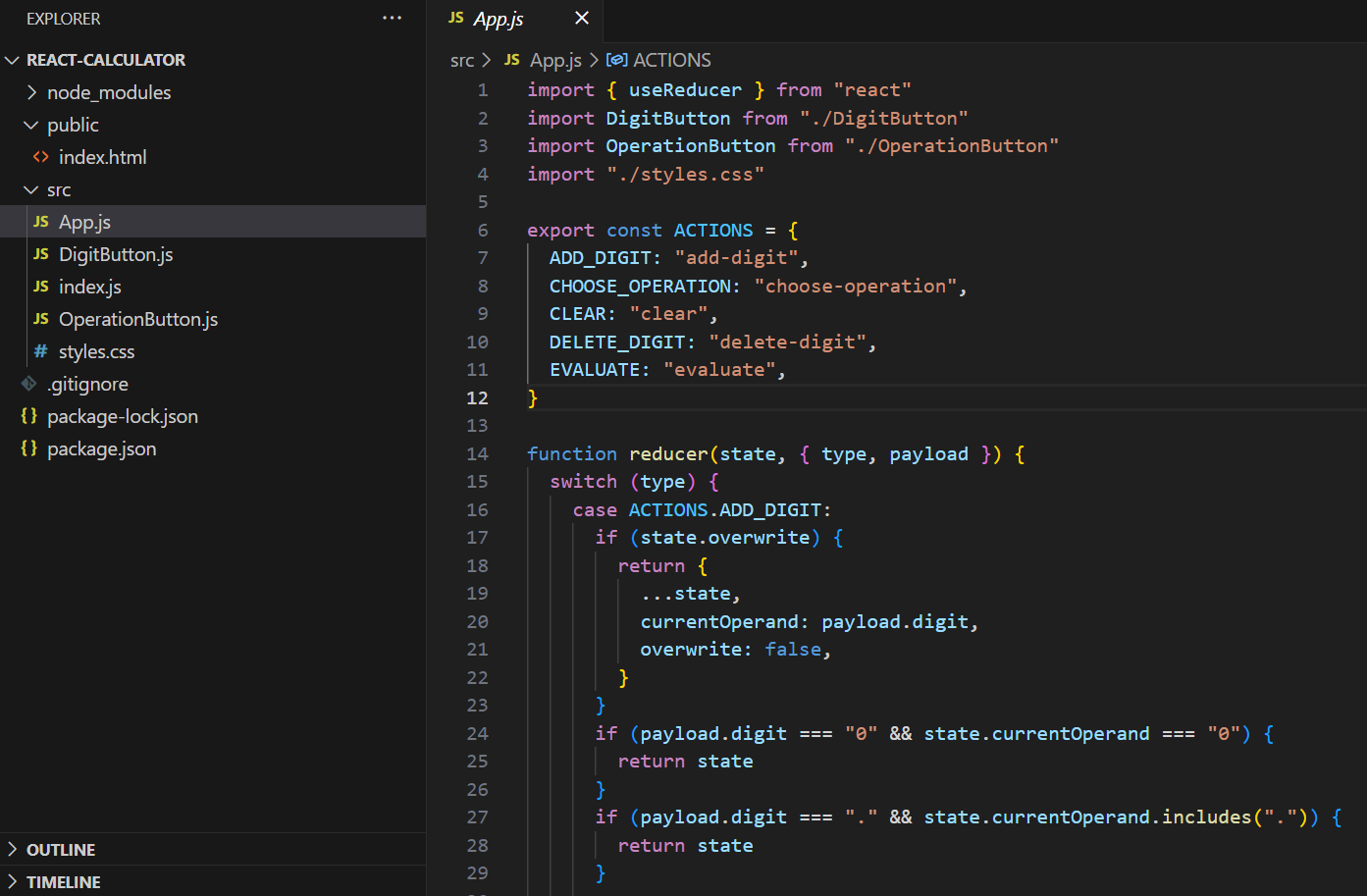
**STYLES.CSS**



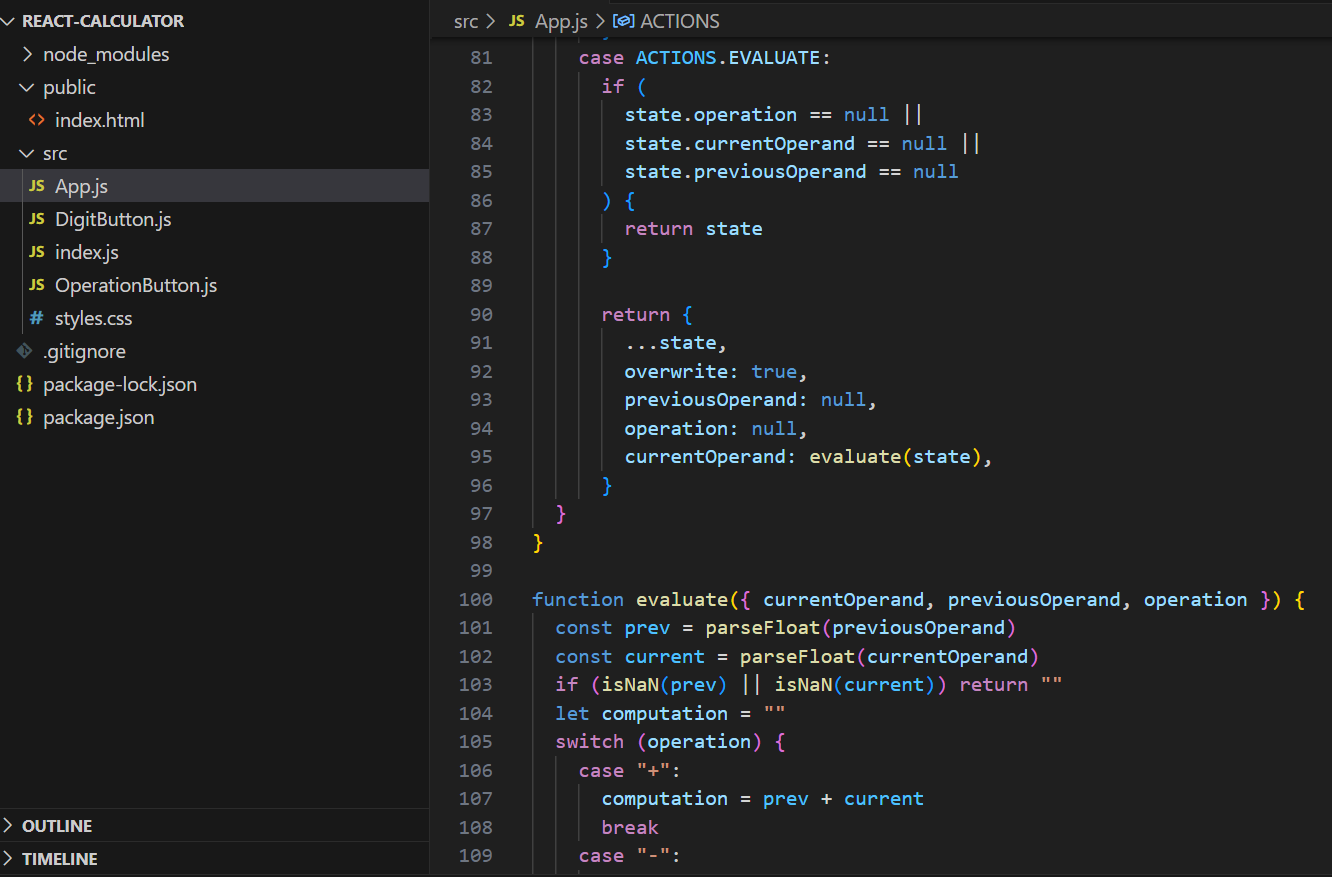


This CSS code styles a calculator user interface. It ensures responsive design using the `box-sizing` property and sets the background with a horizontal gradient. The calculator layout is defined as a grid with centered content, featuring buttons styled with a border, background color, and hover effects for user interaction. The "span-two" class spans two grid columns for specific buttons. The output display, extending across all columns, is semi-transparent with a dark background, showing previous and current operands. Previous operands have a muted color, while current operands are displayed prominently in white with appropriate font sizes, ensuring a visually appealing and functional calculator interface.

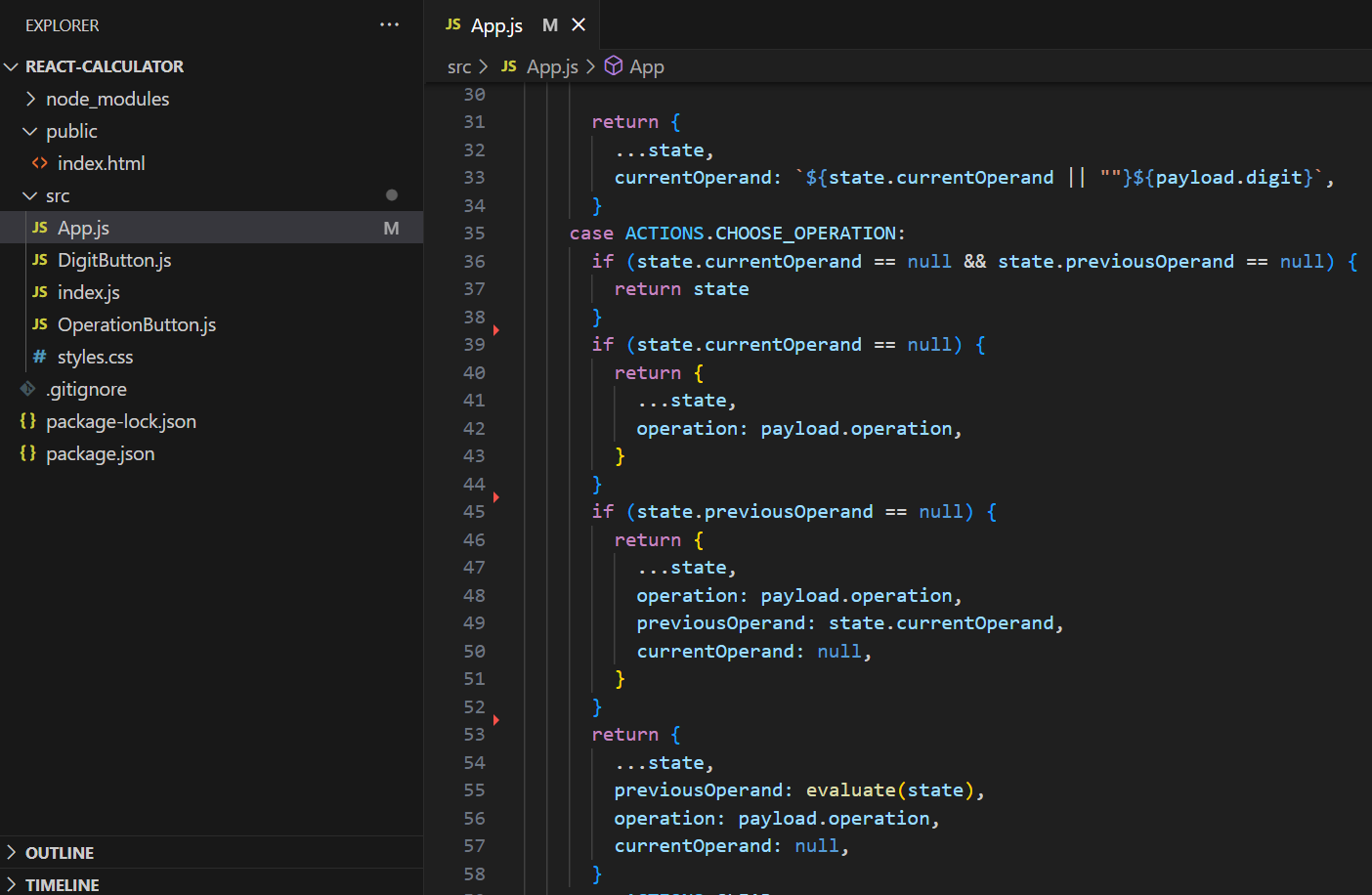
**APP.JS**

****

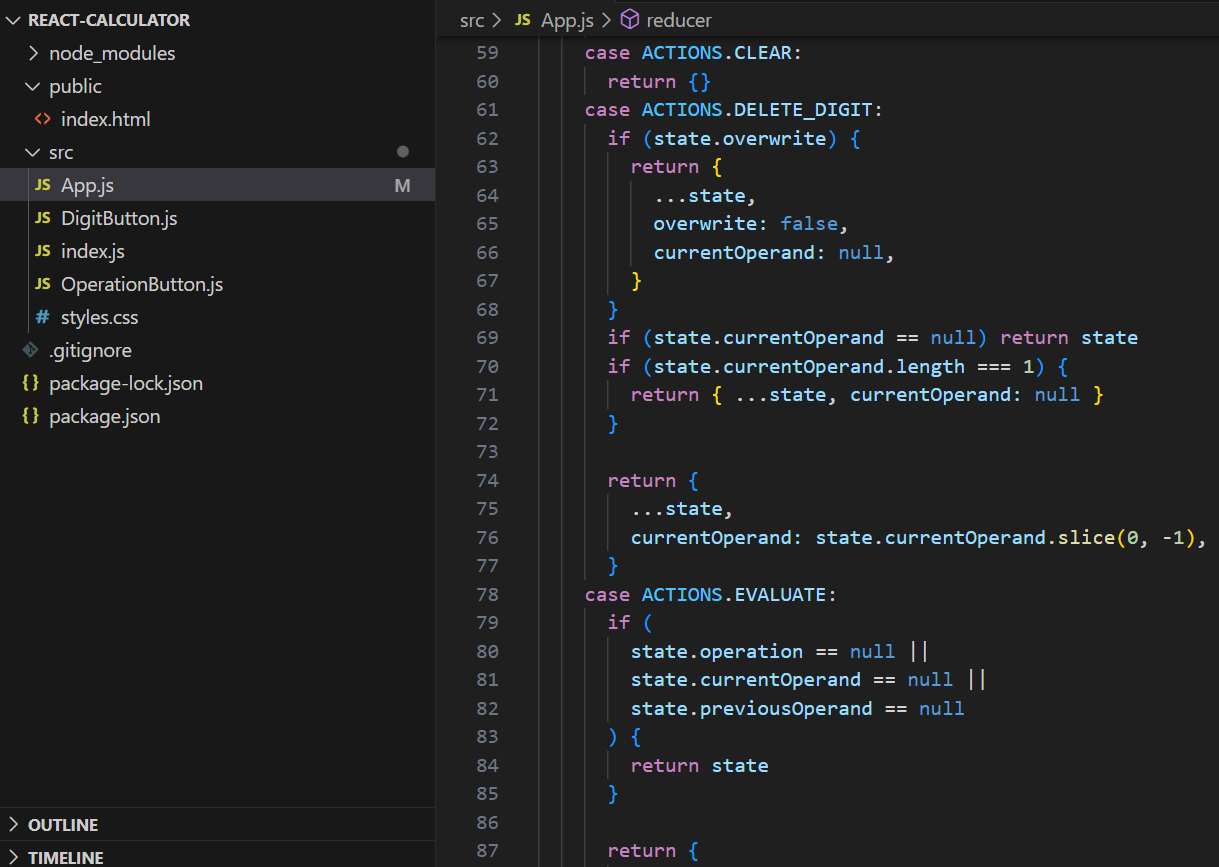
This code employs the `useReducer` hook in a React calculator to manage its state. It defines actions within the `ACTIONS` object, such as adding digits and choosing operations. The `reducer` function processes these actions by examining the current state, enabling users to interact with the calculator. For instance, it manages cases like overwriting the current operand, handling repeated zeroes, and ensuring only one decimal point. This code establishes the foundation for user input, state management, and accurate mathematical computations within the calculator.

****

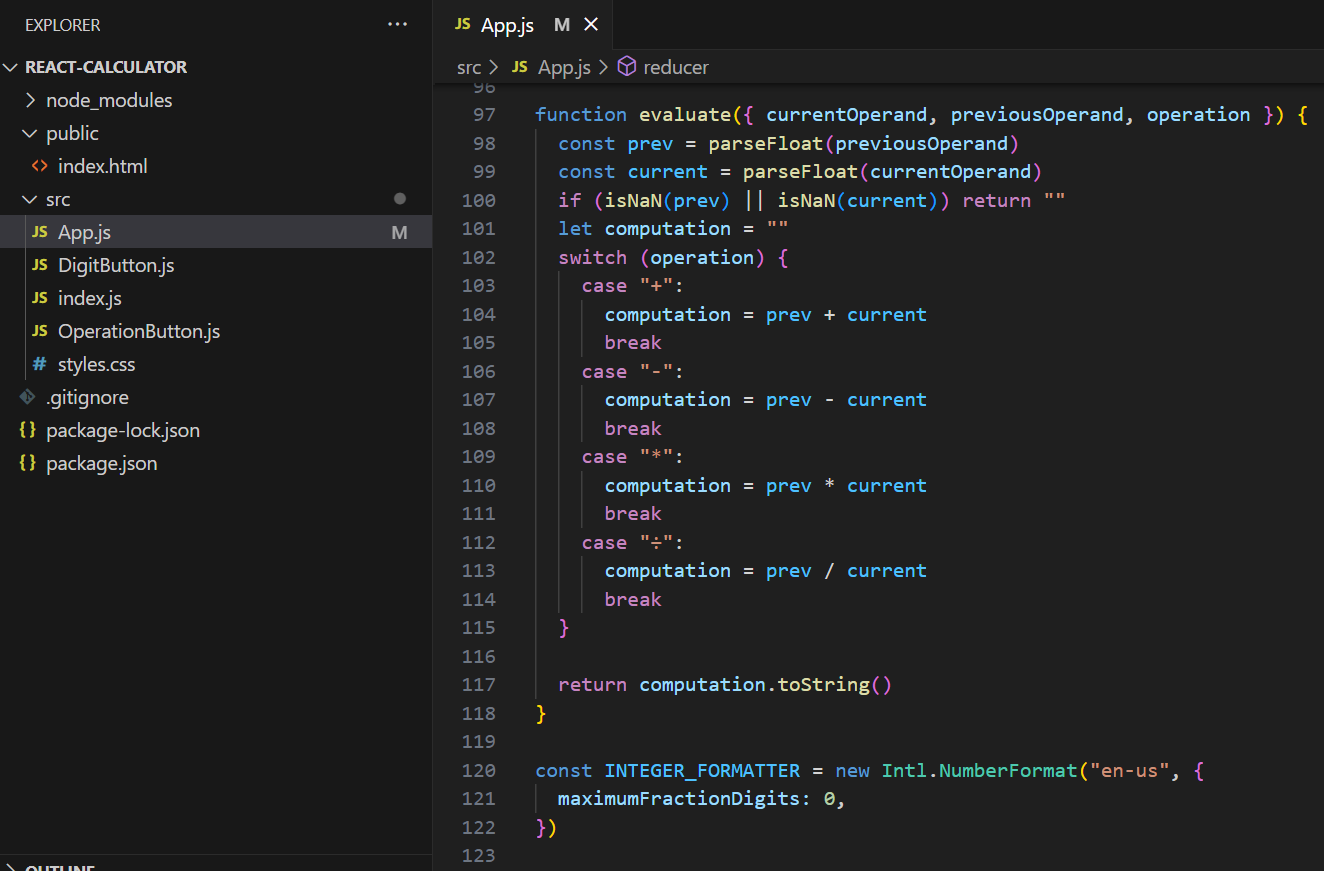
It uses React's state management through a useReducer hook. It initializes three state variables: currentOperand, previousOperand, and operation. These variables store the current and previous operands and the selected operation for the calculator. Click events are handled using dispatch to trigger the corresponding actions.The code handles various actions like adding digits, choosing operations, clearing, deleting digits, and evaluating expressions. It maintains two operands (previous and current) and an operation, displayed in a user interface featuring a two-line layout. Numeric formatting is ensured for a neat display. When users perform calculations, the code evaluates expressions based on the selected operation and displays the result. This code combines React's dynamic rendering with JavaScript logic for building a feature-rich and user-friendly calculator application.



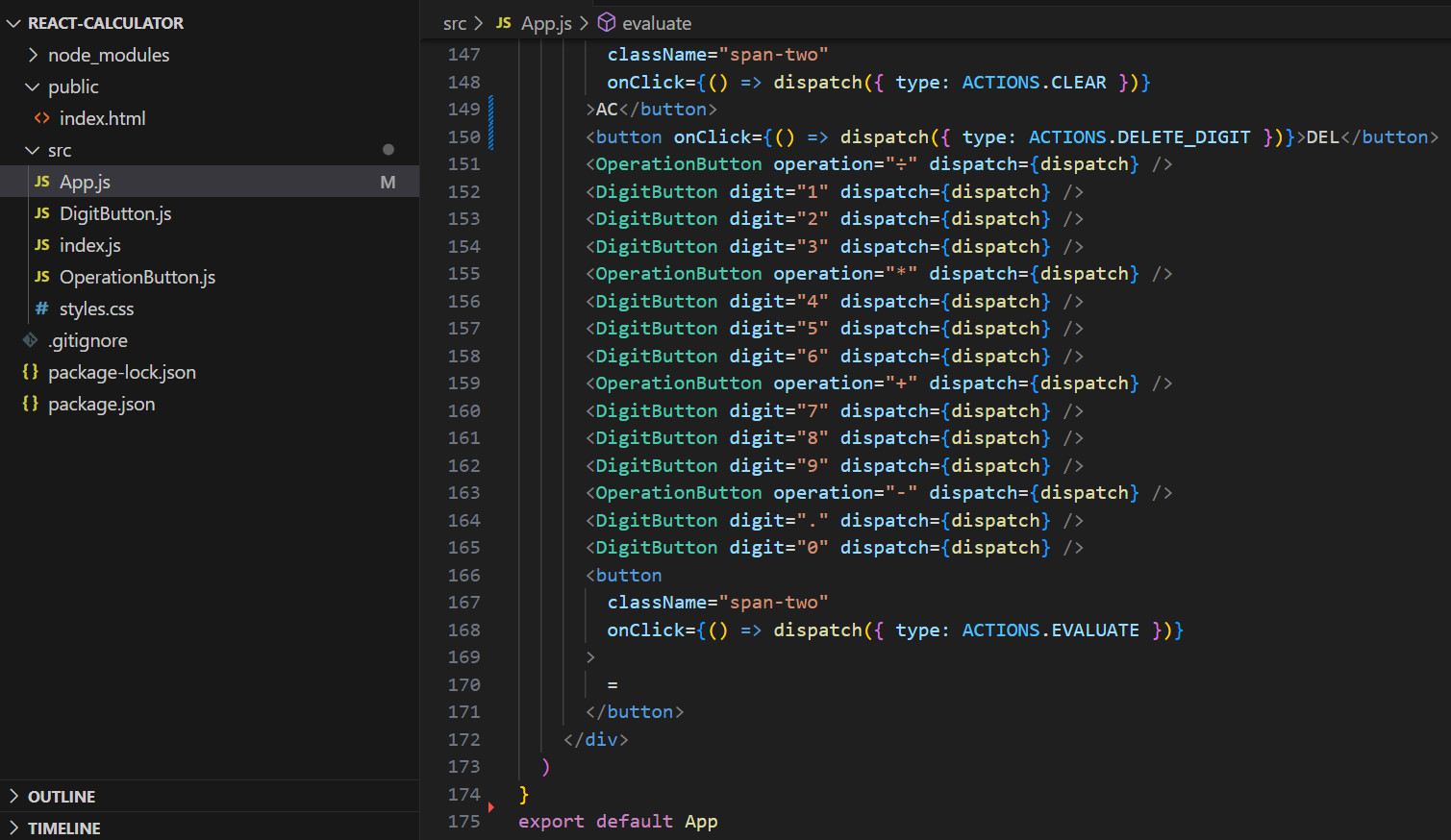
In the code, when the "ADD\_DIGIT" action is triggered, it checks if the calculator is in "overwrite" mode; if so, it sets the current operand to the new digit. If the current operand is empty or zero, it preserves the digit. When "CHOOSE\_OPERATION" is invoked, it handles different scenarios. If neither the current nor previous operands are set, it keeps the state unchanged. If only the current operand is empty, it sets the chosen operation. When both operands exist, it evaluates the expression, updates the previous operand, and sets the new operation while resetting the current operand for the next calculation.



The code handles additional actions in the calculator. When the "CLEAR" action is invoked, it resets the calculator's state to an empty object. For "DELETE\_DIGIT," it checks if the calculator is in overwrite mode; if so, it clears the current operand. If the current operand is null or contains only one character, it resets the current operand. When "EVALUATE" is triggered, it ensures that all the necessary components for a valid calculation, including an operation, current operand, and previous operand, are in place. If not, it keeps the state unchanged, ensuring accurate calculations.

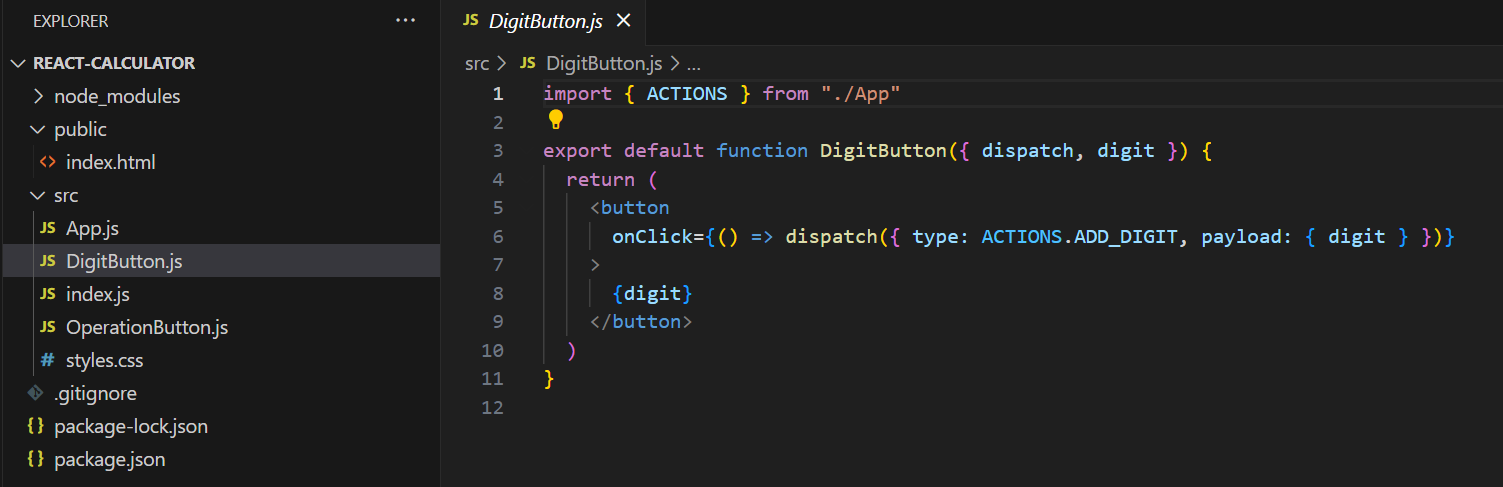


This code segment defines the `evaluate` function responsible for performing mathematical calculations. It takes the current and previous operands, along with the operation, and converts them to floating-point numbers. It ensures the validity of the numeric values to avoid errors. Using a `switch` statement, it executes the corresponding arithmetic operation based on the chosen operator (+, -, \*, ÷). The result is then converted to a string. Additionally, it initializes an `INTEGER\_FORMATTER` to format numbers, specifying a maximum of zero decimal places for integer-like presentation. This function and formatter contribute to the calculator's precise mathematical calculations and proper number formatting.

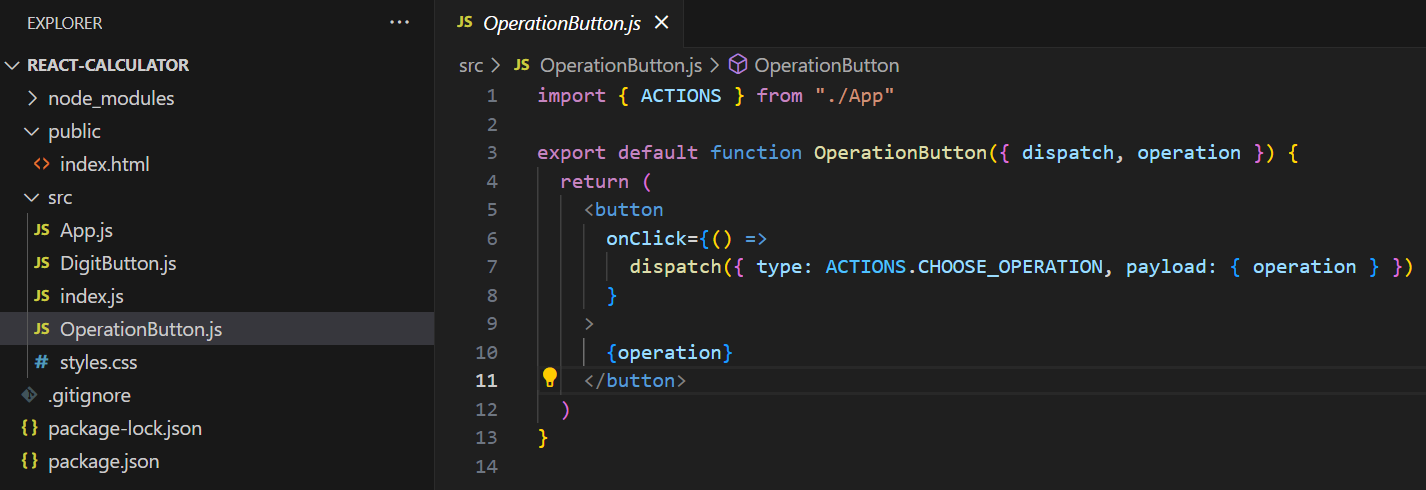


The provided JSX code segment represents the layout and functionality of a calculator in a React application. It consists of various buttons for numeric digits (0-9), mathematical operations (+, -, \*, ÷), and special functions like "AC" (clear) and "DEL" (delete). These buttons trigger actions through the `dispatch` function based on user interactions, following the `ACTIONS` defined earlier. The "span-two" class extends the width of specific buttons to span two grid columns for a clear visual distinction. When the user clicks the "=" button, it triggers the "EVALUATE" action, calculating and displaying the result. Overall, this JSX code creates a user-friendly and functional calculator interface for performing arithmetic calculations.

**DIGITBUTTON.JS**

****

This code defines a React component, ‘DigitButton’, which handles the display and functionality of digit buttons in the calculator. It receives two props: `dispatch` and `digit`. When a digit button is clicked, it triggers the `ADD\_DIGIT` action with the specific digit as payload, updating the calculator's state. This enables users to input numeric values by clicking the digit buttons, facilitating numerical input for calculations.



This code defines a React component, `OperationButton`, responsible for handling operation buttons (e.g., +, -, \*, ÷) in the calculator. It receives two props: `dispatch` and `operation`. When an operation button is clicked, it triggers the `CHOOSE\_OPERATION` action with the specific operation as payload, allowing users to select mathematical operations. This functionality plays a crucial role in building complex expressions and enabling mathematical calculations within the calculator application.