FSD Lab 5

Aim: Design and develop an interactive user interface using React.

Objectives:

1. Articulate what React is and why it is useful.
2. Explore the basic architecture of a React application.
3. Use React components to build interactive interfaces

Theory:

(1) What is React? Steps to run React app using create-react-app.

React is a JavaScript library for building user interfaces, developed by Facebook. It allows developers to create reusable UI components and efficiently update the user interface in response to changes in data. React follows a component-based architecture, where the UI is broken down into modular components, making it easier to manage and maintain.

To run a React app using create-react-app, follow these steps:

1. Install Node.js and npm (Node Package Manager) if not already installed.
2. Open a terminal and run the following command to install create-react-app globally:

```
npm install -g create-react-app
```

3. Create a new React app by running:

```
npx create-react-app my-react-app
```

4. Change to the app's directory:

```
cd my-react-app
```

5. Start the development server:

```
npm start
```

This will open the app in your default browser.

Now, you have a basic React app running locally, and you can start building your components and features.

(2) Passing data through props (Small example)

In React, props (short for properties) allow you to pass data from a parent component to a child component. This is essential for building dynamic and reusable components. Here's a small example:

```
// ParentComponent.js
import React from 'react';
import ChildComponent from './ChildComponent';

const ParentComponent = () => {
  const dataToSend = 'Hello from Parent!';

  return (
    <div>
      <h1>Parent Component</h1>
      <ChildComponent data={dataToSend} />
    </div>
  );
};

// ChildComponent.js
import React from 'react';

const ChildComponent = (props) => {
  return (
    <div>
      <h2>Child Component</h2>
      <p>Data received from parent: {props.data}</p>
    </div>
  );
};

export default ChildComponent;
```

In this example, `ParentComponent` passes the `dataToSend` variable as a prop to `ChildComponent`, which then displays the received data. Props facilitate communication between components, allowing for a flexible and modular design in React applications.

FAQ:

(1) What are React states and hooks?

Ans.

- React States:

In React, a state is a fundamental concept that allows components to manage and store local, mutable data. Unlike props, which are external inputs passed to a component, state is internal and provides a way for components to keep track of dynamic information and respond to user interactions.

In class components, state is typically initialized in the constructor using `this.state`, and updates are managed through the `this.setState()` method. For instance:

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

class TemperatureConverter extends Component {
  constructor(props) {
    super(props);
    this.state = {
      celsius: 0,
      fahrenheit: 32
    };
  }

  handleCelsiusChange = (e) => {
    const celsiusValue = e.target.value;
    const fahrenheitValue = (celsiusValue * 9) / 5 + 32;
    this.setState({ celsius: celsiusValue, fahrenheit: fahrenheitValue });
  };

  render() {
    return (
      <div>
        <p>Celsius: {this.state.celsius}</p>
        <p>Fahrenheit: {this.state.fahrenheit}</p>
        <input type="number" value={this.state.celsius} onChange={this.handleCelsiusChange} />
      </div>
    );
  }
}

export default TemperatureConverter;
```

With the advent of functional components, React introduced Hooks to allow the use of state and other features in functional components.

- React Hooks:

React Hooks provide a way to use state and other React features in functional components. The most commonly used hooks include `useState` and `useEffect`.

- `useState`: This hook allows functional components to declare state variables. It returns an array containing the current state value and a function to update it.

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

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}

export default Counter;
```

- `useEffect`: This hook is used for side effects in functional components, such as data fetching or subscribing to external events.

React Hooks simplify state management and side effect handling in functional components, enhancing the overall developer experience in building React applications.

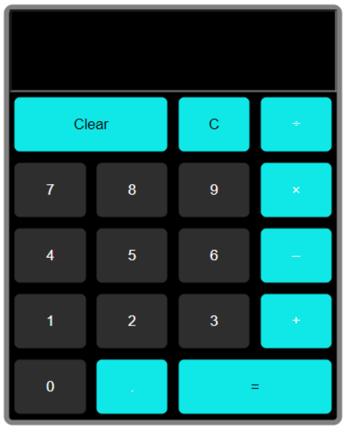
Screenshots -

REACT TICTACTOE

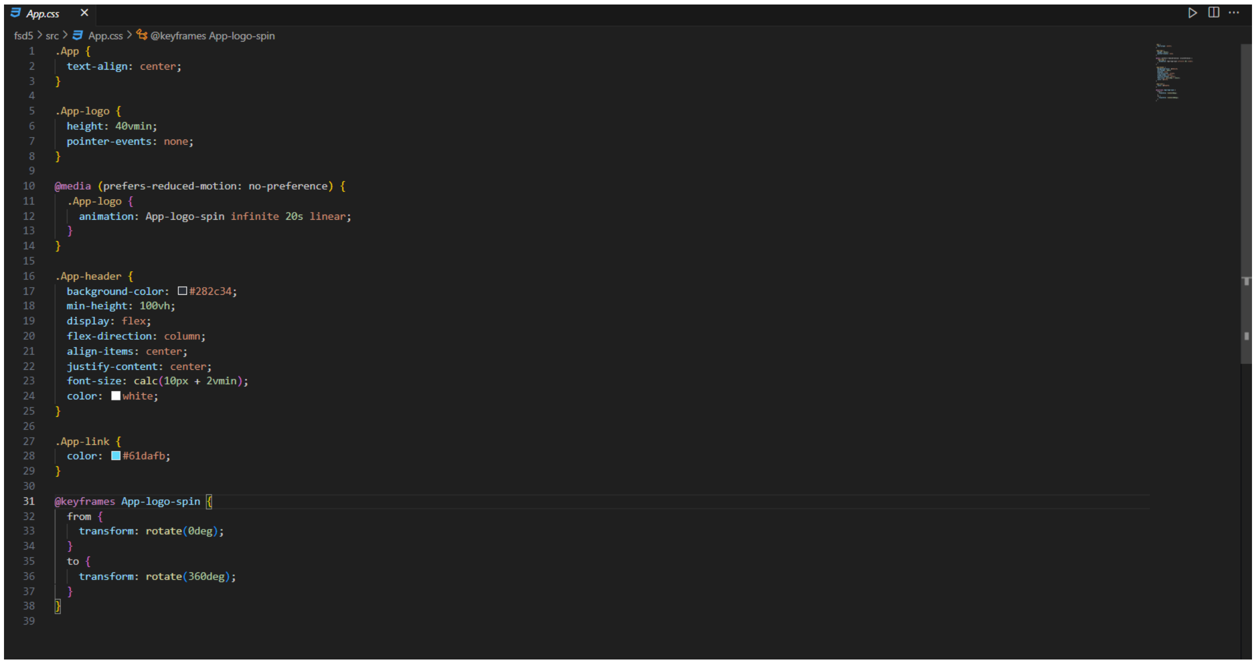
Next player: X

```
_ _ _
_ _ _
_ _ _
```

Calculator



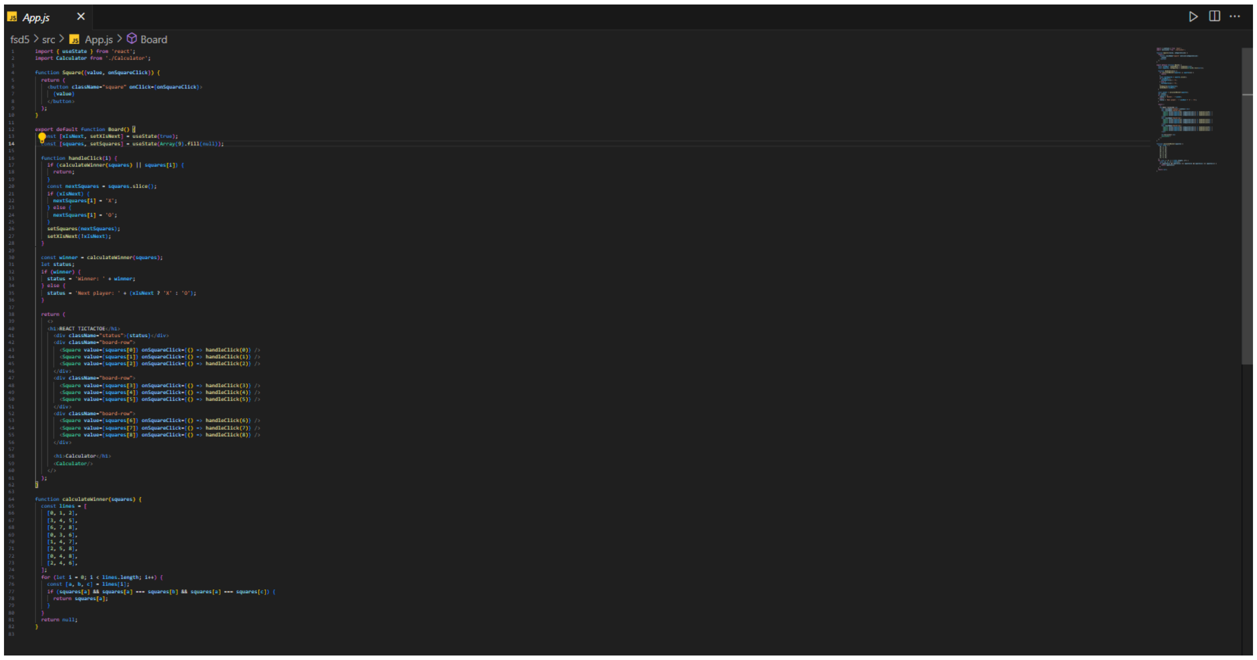
CODE -



App.css

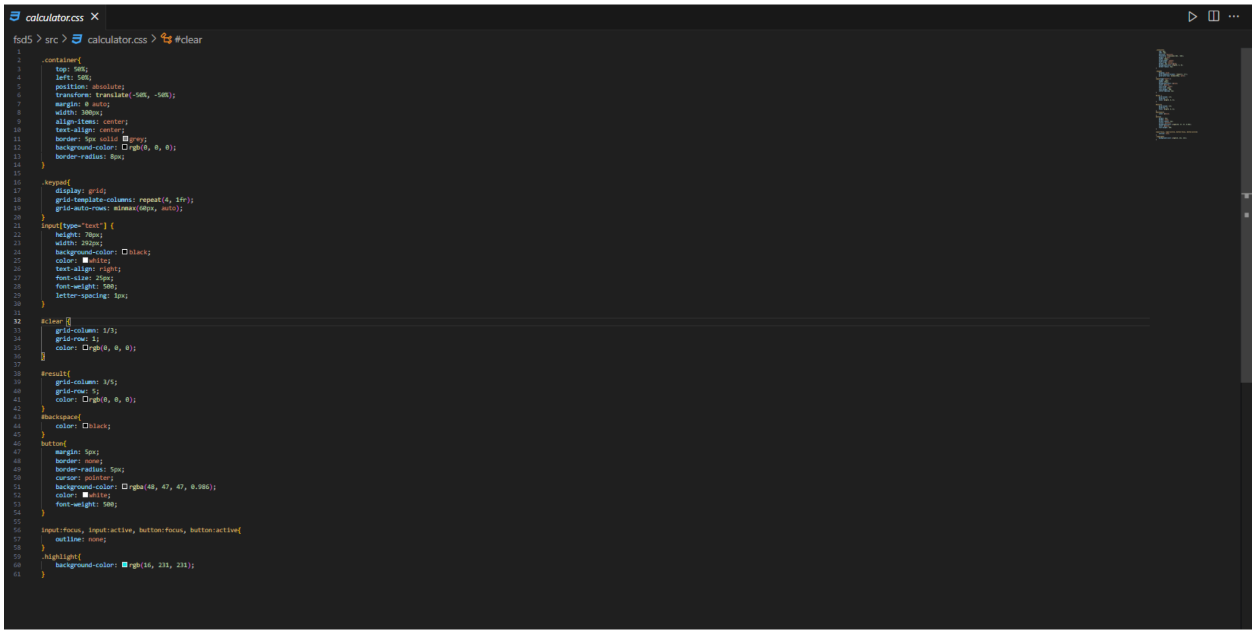
fsd5 > src > App.css > @keyframes App-logo-spin

```css
.App {
  text-align: center;
}

.App-logo {
  height: 40vmin;
  pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}

.App-link {
  color: #61dafb;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
```


App.js

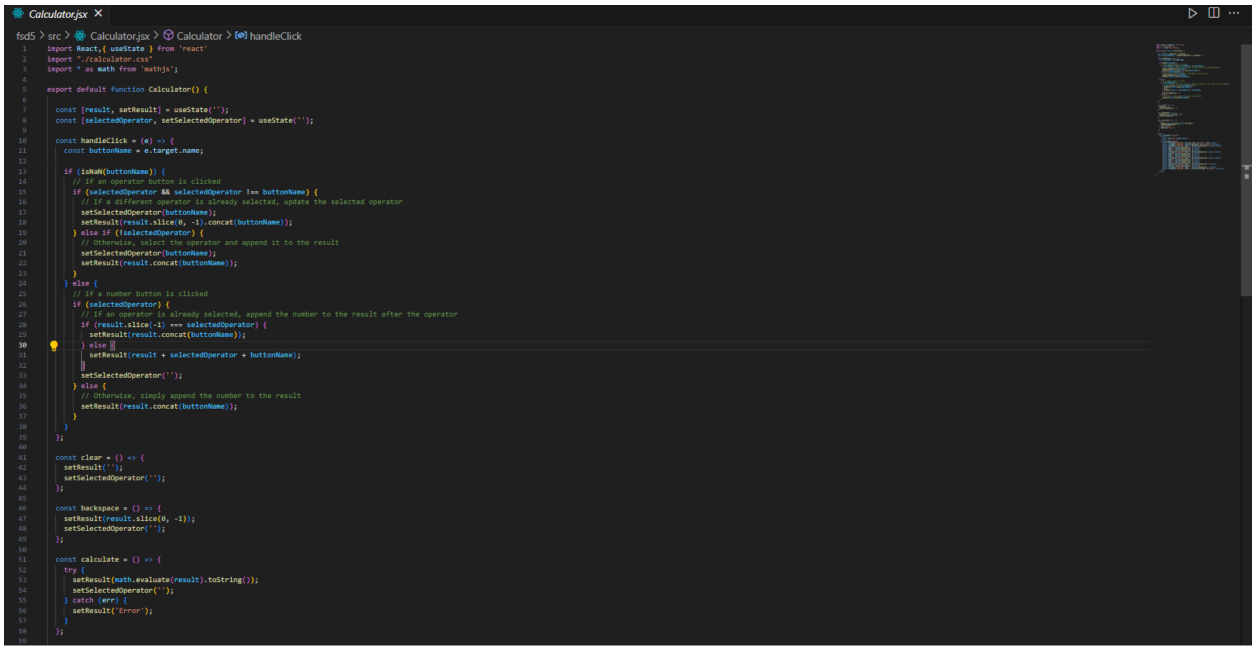
fsd5 > src > App.js > Board

```
import { useState } from 'react';
import Calculator from './Calculator';

function Square({value, onSquareClick}) {
  return (
    <button className="square" onClick={onSquareClick}>
      {value}
    </button>
  );
}

export default function Board() {
  const [xIsNext, setXIsNext] = useState(true);
  const [squares, setSquares] = useState(Array(9).fill(null));

  function handleClick(i) {
    if (calculateWinner(squares) || squares[i]) {
      return;
    }
    const nextSquares = squares.slice();
    if (xIsNext) {
      nextSquares[i] = 'X';
    } else {
      nextSquares[i] = 'O';
    }
    setSquares(nextSquares);
    setXIsNext(!xIsNext);
  }

  const winner = calculateWinner(squares);
  let status;
  if (winner) {
    status = 'Winner: ' + winner;
  } else {
    status = 'Next player: ' + (xIsNext ? 'X' : 'O');
  }

  return (
    <>
    <h1>REACT TICTACTOE</h1>
      <div className="status">{status}</div>
      <div className="board-row">
        <Square value={squares[0]} onSquareClick={() => handleClick(0)} />
        <Square value={squares[1]} onSquareClick={() => handleClick(1)} />
        <Square value={squares[2]} onSquareClick={() => handleClick(2)} />
      </div>
      <div className="board-row">
        <Square value={squares[3]} onSquareClick={() => handleClick(3)} />
        <Square value={squares[4]} onSquareClick={() => handleClick(4)} />
        <Square value={squares[5]} onSquareClick={() => handleClick(5)} />
      </div>
      <div className="board-row">
        <Square value={squares[6]} onSquareClick={() => handleClick(6)} />
        <Square value={squares[7]} onSquareClick={() => handleClick(7)} />
        <Square value={squares[8]} onSquareClick={() => handleClick(8)} />
      </div>

      <h1>Calculator</h1>
      <Calculator/>
    </>
  );
}

function calculateWinner(squares) {
  const lines = [
    [0, 1, 2],
    [3, 4, 5],
    [6, 7, 8],
    [0, 3, 6],
    [1, 4, 7],
    [2, 5, 8],
    [0, 4, 8],
    [2, 4, 6],
  ];
  for (let i = 0; i < lines.length; i++) {
    const [a, b, c] = lines[i];
    if (squares[a] && squares[a] === squares[b] && squares[a] === squares[c]) {
      return squares[a];
    }
  }
  return null;
}
```


calculator.css

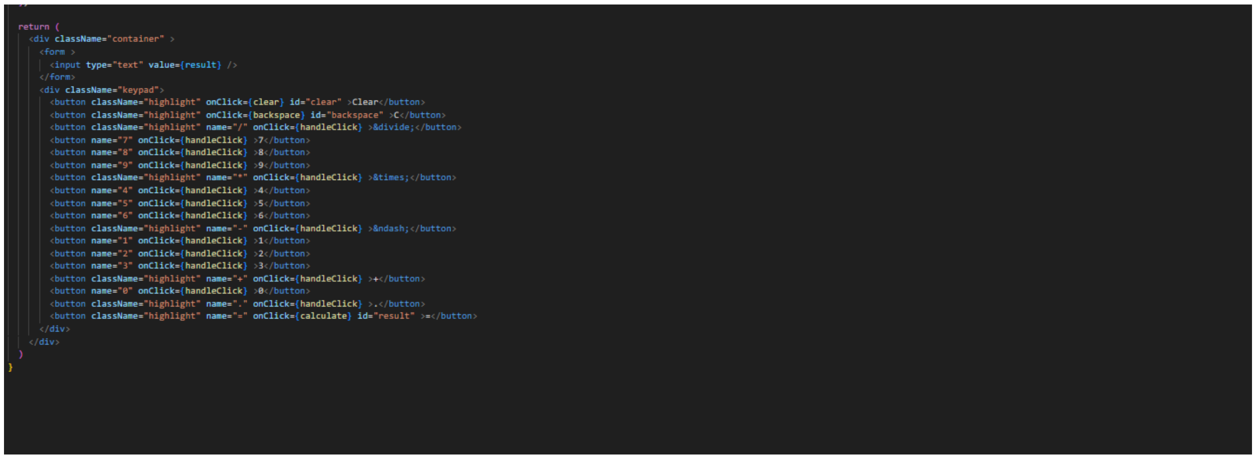
fsd5 > src > calculator.css > #clear

```css

.container{
    top: 50%;
    left: 50%;
    position: absolute;
    transform: translate(-50%, -50%);
    margin: 0 auto;
    width: 300px;
    align-items: center;
    text-align: center;
    border: 5px solid grey;
    background-color: rgb(0, 0, 0);
    border-radius: 8px;
}

.keypad{
    display: grid;
    grid-template-columns: repeat(4, 1fr);
    grid-auto-rows: minmax(60px, auto);
}
input[type="text"] {
    height: 70px;
    width: 292px;
    background-color: black;
    color: white;
    text-align: right;
    font-size: 25px;
    font-weight: 500;
    letter-spacing: 1px;
}

#clear {
    grid-column: 1/3;
    grid-row: 1;
    color: rgb(0, 0, 0);
}

#result{
    grid-column: 3/5;
    grid-row: 5;
    color: rgb(0, 0, 0);
}
#backspace{
    color: black;
}
button{
    margin: 5px;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    background-color: rgba(48, 47, 47, 0.986);
    color: white;
    font-weight: 500;
}

input:focus, input:active, button:focus, button:active{
    outline: none;
}
.highlight{
    background-color: rgb(16, 231, 231);
}
```


Calculator.jsx

fsd5 > src > Calculator.jsx > Calculator > handleClick

```jsx
import React,{ useState } from 'react'
import "./calculator.css"
import * as math from 'mathjs';

export default function Calculator() {

  const [result, setResult] = useState('');
  const [selectedOperator, setSelectedOperator] = useState('');

  const handleClick = (e) => {
    const buttonName = e.target.name;

    if (isNaN(buttonName)) {
      // If an operator button is clicked
      if (selectedOperator && selectedOperator !== buttonName) {
        // If a different operator is already selected, update the selected operator
        setSelectedOperator(buttonName);
        setResult(result.slice(0, -1).concat(buttonName));
      } else if (!selectedOperator) {
        // Otherwise, select the operator and append it to the result
        setSelectedOperator(buttonName);
        setResult(result.concat(buttonName));
      }
    } else {
      // If a number button is clicked
      if (selectedOperator) {
        // If an operator is already selected, append the number to the result after the operator
        if (result.slice(-1) === selectedOperator) {
          setResult(result.concat(buttonName));
        } else {
          setResult(result + selectedOperator + buttonName);
        }
        setSelectedOperator('');
      } else {
        // Otherwise, simply append the number to the result
        setResult(result.concat(buttonName));
      }
    }
  };

  const clear = () => {
    setResult('');
    setSelectedOperator('');
  };

  const backspace = () => {
    setResult(result.slice(0, -1));
    setSelectedOperator('');
  };

  const calculate = () => {
    try {
      setResult(math.evaluate(result).toString());
      setSelectedOperator('');
    } catch (err) {
      setResult('Error');
    }
  };

```



```
  return (
    <div className="container" >
      <form >
        <input type="text" value={result} />
      </form>
      <div className="keypad">
        <button className="highlight" onClick={clear} id="clear" >Clear</button>
        <button className="highlight" onClick={backspace} id="backspace" >C</button>
        <button className="highlight" name="/" onClick={handleClick} >&divide;</button>
        <button name="7" onClick={handleClick} >7</button>
        <button name="8" onClick={handleClick} >8</button>
        <button name="9" onClick={handleClick} >9</button>
        <button className="highlight" name="*" onClick={handleClick} >&times;</button>
        <button name="4" onClick={handleClick} >4</button>
        <button name="5" onClick={handleClick} >5</button>
        <button name="6" onClick={handleClick} >6</button>
        <button className="highlight" name="-" onClick={handleClick} >&ndash;</button>
        <button name="1" onClick={handleClick} >1</button>
        <button name="2" onClick={handleClick} >2</button>
        <button name="3" onClick={handleClick} >3</button>
        <button className="highlight" name="+" onClick={handleClick} >+</button>
        <button name="0" onClick={handleClick} >0</button>
        <button className="highlight" name="." onClick={handleClick} >.</button>
        <button className="highlight" name="=" onClick={calculate} id="result" >=</button>
      </div>
    </div>
  )
}
```