Conclusion

45

Gaurav Singhal

45

How to Create a Right-click Menu in React

Gaurav Singhal

Jun 12, 2020 • 8 Min read • 15,110 Views

Jun 12, 2020 • 8 Min read • 15,110 Views

Web Development Front End Web Dev... Client-side Framew... React

Introduction

React is an excellent library for building component-based UI with reusability in mind. You can create tons of components and reuse them across multiple pages. Occasionally, for styling purposes, you may want to override the default right-click menu or the context menu in your React app.

You can easily create a context menu by creating a custom component for handling right clicks and displaying the menu. In this ide, you will learn how to create a reusable and flexible context nu component in your React app.

45

Implementation Overview

To override the default browser right-click menu, the first step is to prevent the default right-click behavior. This can be done by listening to the `contextmenu` event and using the `preventDefault()` method on the event.

```js
document.addEventListener("contextmenu", (event) => {
  event.preventDefault();
});
```

Next, capture the current (x,y) position, and render your component at that coordinate on the page. Get the x and y position from the `pageX` and `pageY` properties from the event object.

```js
document.addEventListener("contextmenu", (event) => {
  event.preventDefault();
  const xPos = event.pageX + "px";
```

```jsx
        const yPos = event.pageY + "px";
        //
    });
```

45

nen, pass the coordinates as the position style attribute to the component. The critical point to note here is that the custom menu should have the `position` style set to `absolute` for the menu to open at the correct position.

jsx

```jsx
<ContextMenu style={{ top: xPos, left: yPos }} />
```

That's a general overview of implementation.

Creating a Custom ContextMenu Component

Start by defining the component and the state for the component.

jsx

```jsx
class ContextMenu extends Component {
    state = {
        xPos: "0px",
        yPos: "0px:,
        showMenu: false
    }

    render() {
            // ...
        }
    }
```

45

The state will contain the (x, y) position and a boolean state to show or hide the menu.

Next, add the event listeners in the `componentDidMount` lifecycle method in the component.

jsx

```jsx
class ContextMenu extends Component {
    state = {
        xPos: "0px",
        yPos: "0px:,
        showMenu: false
    }

    componentDidMount() {
        document.addEventListener("click", this.handleClick);
        document.addEventListener("contextmenu", this.handleContextMenu);
    }

    componentWillUnmount() {
        document.removeEventListener("click", this.handleClick);
        document.removeEventListener("contextmenu", this.handleContextMer
    }

    handleClick = (e) => {
        // ...
    }

```

Conclusion

45

45

```
        handleContextMenu = (e) => {
            e.preventDefault();

            // ...
        }

        render() {
            // ...
        }
    }
```

Add two event listeners for the click and context menu event in the `componentDidMount` method. Don't forget to remove the event listeners in the `componentWillUnmount` method to avoid memory leaks from your component.

In the `handleClick` method, you need to close the menu if it's currently open in the page.

jsx

```jsx
handleClick = (e) => {
  if (this.state.showMenu) this.setState({ showMenu: false });
};
```

In the `handleContextMenu` method, all you have to do is prevent the default event from triggering, set the current (x,y) position of the right-click in the state, and display the menu.

jsx

```jsx
handleContextMenu = (e) => {
  e.preventDefault();

  this.setState({
    xPos: `${e.pageX}px`,
    yPos: `${e.pageY}px`,
    showMenu: true,
  });
};
```

Conclusion

45

45

In the `render` method, create a list with the menu items and display it when the `showMenu` state is set to true.

jsx

```jsx
class ContextMenu extends Component {
  // ...
  render() {
    const { showMenu, xPos, yPos } = this.state;

    if (showMenu)
      return (
        <ul
          className="menu"
          style={{
            top: yPos,
            left: xPos,
          }}
        >
          <li>Login</li>
          <li>Register</li>
          <li>Open Profile</li>
```

```
            </ul>
          );
        else return null;
      }
    }
```

45

If you check out the results now, you should be able to see your custom menu when you right-click on the page. You got the desired results, but this component still needs some refactoring. The menu component is hardcoded here; it's best if it comes in as a prop. That way, you can show a different context menu on different pages. Also, add some fade animation to the menu when it opens up using the `react-motion` library.

jsx

```jsx
class ContextMenu extends Component {
  // ...
  render() {
    const { showMenu, yPos, xPos } = this.state;
    return (
      <Motion
        defaultStyle={{ opacity: 0 }}
        style={{ opacity: !showMenu ? spring(0) : spring(1) }}
      >
        {(interpolatedStyle) => (
          <>
            {showMenu ? (
              <div
                className="menu-container"
                style={{
```

Conclusion

45

45

```jsx
                  top: yPos,
                  left: xPos,
                  opacity: interpolatedStyle.opacity,
                }}
              >
                {this.props.menu}
              </div>
            ) : (
              <></>
            )}
          </>
        )}
      </Motion>
    );
  }
}
```

And now you can use the `ContextMenu` component from your root or the entry component as shown below.

jsx

```jsx
const CustomMenu = () => (
  <ul className="menu">
    <li>Login</li>
    <li>Register</li>
    <li>Open Profile</li>
  </ul>
);

const App = () => (
  <div className="App">
    {/* ... */}
    <ContextMenu menu={() => <CustomMenu>}>
```

```jsx
      </div>
    )
```

45

Creating a Custom Context Menu Using Hooks

The previous section outlined how to create a component to render a custom context menu. But that was a class component; you could do the same with functional components using hooks.

All state variables can be created and managed with the `useState` hook. Handle the callbacks using the `useCallback` hook and pass a dependency array to it so that it can memoize the results and only change when the dependencies get updated. Wrap all the logic in a separate custom hook and return the `xPos`, `yPos` and `showMenu` state variables from the hook.

jsx

```jsx
const useContextMenu = () => {
  const [xPos, setXPos] = useState("0px");
  const [yPos, setYPos] = useState("0px");
  const [showMenu, setShowMenu] = useState(false);

  const handleContextMenu = useCallback(
    (e) => {
      e.preventDefault();

```

Conclusion

45

45

```
        setXPos(`${e.pageX}px`);
        setYPos(`${e.pageY}px`);
        setShowMenu(true);
      },
      [setXPos, setYPos]
    );

    const handleClick = useCallback(() => {
      showMenu && setShowMenu(false);
    }, [showMenu]);

    useEffect(() => {
      document.addEventListener("click", handleClick);
      document.addEventListener("contextmenu", handleContextMenu);
      return () => {
        document.addEventListener("click", handleClick);
        document.removeEventListener("contextmenu", handleContextMenu);
      };
    });

    return { xPos, yPos, showMenu };
  };
```

jsx

```
  const ContextMenu = ({ menu }) => {
    const { xPos, yPos, showMenu } = useContextMenu();
    return (
      <Motion
        defaultStyle={{ opacity: 0 }}
        style={{ opacity: !showMenu ? spring(0) : spring(1) }}
      >
        {(interpolatedStyle) => (
          <>
            {showMenu ? (
            <div
              className="menu-container"
              style={{
                top: yPos,
                left: xPos,
                opacity: interpolatedStyle.opacity,
              }}
            >
              {menu}
            </div>
          ) : (
            <></>
          )}
        </>
      )}
    </Motion>
  );
};
```

Conclusion

Consider this guide as a gateway to exploring more advanced use cases, such as creating custom hooks in functional components, overriding default browser UI components like context menu, etc. You will get better at developing custom UI components if you understand the implementation logic behind them. In any case, to override default behavior, the first thing you need to do is prevent the event by using `preventDefault()`.

45

Conclusion

45