Solution: Implementing scroller position with render props

Here is the completed solution code for the App.js file:

```javascript
import "./App.css";
import { useEffect, useState } from "react";

const MousePosition = ({ render }) => {
  const [mousePosition, setMousePosition] = useState({
    x: 0,
    y: 0,
  });

  useEffect(() => {
    const handleMousePositionChange = (e) => {
      setMousePosition({
        x: e.clientX,
        y: e.clientY,
      });
    };

    window.addEventListener("mousemove", handleMousePositionChange);

    return () => {
      window.removeEventListener("mousemove", handleMousePositionChange);
    };
  }, []);

  return render({ mousePosition });
};

const PanelMouseLogger = () => {
  return (
    <div className="BasicTracker">
      <p>Mouse position:</p>
      <MousePosition
        render={({ mousePosition }) => (
          <div className="Row">
            <span>x: {mousePosition.x}</span>
            <span>y: {mousePosition.y}</span>
          </div>
        )}
      />
    </div>
```

1. Implement the body of `handleMousePositionChange`

The `mousemove` handler function receives an event as parameter that contains the mouse coordinates as `clientX` and `clientY` properties. Therefore you can provide a position update by calling the state setter `setMousePosition` with the new values.

```javascript
const handleMousePositionChange = (e) => {
 setMousePosition({
   x: e.clientX,
   y: e.clientY,
 });
};

```

2. Implement the `return` statement of the component

The `MousePosition` component receives a `render` prop, which is the special prop name designed by convention to specify a function that returns some JSX. Since the `MousePosition` component does not take care of any visualization logic, but rather encapsulates cross-cutting concerns, it should return the result of calling the render function with the `mousePosition` as an argument. In other words, it's up to the components that consume `MousePosition` to specify what sort of UI they want to display when they receive a new value of the mouse position on the screen.

```
return render({ mousePosition })
```

Step 2

The `PanelMouseLogger` component should not receive any props. Hence, the early return from the previous implementation if no props were provided is no longer needed.

Instead, the `mousePosition` is now injected as the first argument of the render function prop that `MousePosition` uses. It's in this render function body where the previous JSX should be extracted and returned.

```
const PanelMouseLogger = () => {
 return (
   <div className="BasicTracker">
     <p>Mouse position:</p>
     <MousePosition
       render={({ mousePosition }) => (
         <div className="Row">
           <span>x: {mousePosition.x}</span>
           <span>y: {mousePosition.y}</span>
         </div>
       )}
     />
   </div>
 );
};

```

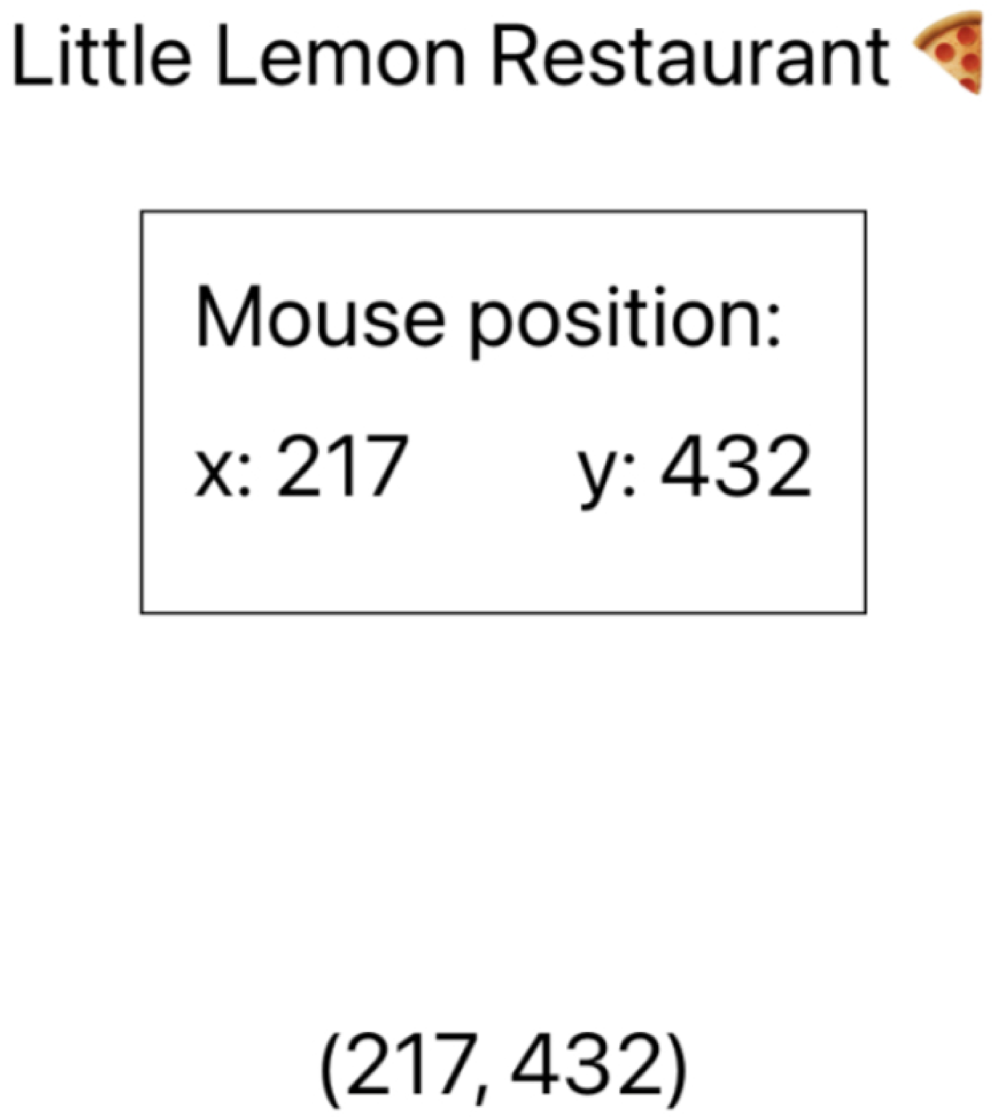
Step 3

Similarly, as in step 2, the component should not receive any props and the early if statement should be removed. The particular UI for this component is provided as part of the render prop as well.

```
const PointMouseLogger = () => {
 return (
   <MousePosition
     render={({ mousePosition }) => (
       <p>
         ({mousePosition.x}, {mousePosition.y})
       </p>
     )}
   />
 );
};

```

At this point, the implementation has been completed and you should see the following result when you run the app in the browser:



Little Lemon Restaurant 🍕

Mouse position:

x: 217 y: 432

(217, 432)