Modals in React

- A modal is a dialog box or a pop-up window that is displayed on top of the current page.
- Modals are often used to grab the user's attention and display critical information or actions that require their interaction.
- In React, modals are commonly implemented as components that can be conditionally rendered based on the application state.

Why Modals are used in React?

1. **Focused User Interaction**: Modals are designed to focus the user's attention on a specific task or piece of information while dimming the rest of the application interface.
2. **Contextual Actions**: They allow developers to present contextual actions like confirmation, alerts, or detailed views without navigating away from the main page.
3. **Improved User Experience**: Modals prevent unnecessary page reloads or navigation, maintaining a seamless user experience.
4. **Dynamic Content**: They are a great way to show dynamic content like forms, alerts, or additional information without cluttering the main UI.

Use Cases of Modals

1. **Form Submissions**: Collect user input for tasks like login, sign-up, or feedback.
2. **Confirmation Dialogs**: Confirm user actions like deleting a file or completing a transaction.
3. **Notifications and Alerts**: Display critical messages to users, such as error alerts or updates.
4. **Displaying Additional Information**: Show content like image previews, video playback, or product details without navigating away from the page.
5. **Embedded Workflows**: Handle multistep workflows, like onboarding processes or wizards.

A modal is typically used for tasks that require focused user input or attention.
Example: A login form that blocks the user from accessing the rest of the site until they log in.

User Interaction Patterns Using Modals

1. **Blocking UI**: Modals often block the underlying UI, ensuring users complete the required action before interacting with other parts of the application.
2. **Focus Management**: Modals should focus on the first interactive element when opened and return focus to the trigger element when closed.
3. **Keyboard Accessibility**: Support for Escape key to close modals, tab navigation within the modal, and screen reader support.
4. **Background Interactions**: Typically, the background is dimmed and made non-interactive to ensure focus stays on the modal.
5. **Responsive Design**: Modals should adapt gracefully to different screen sizes and orientations.

Example: Basic Modal Implementation

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

const App = () => {
  const [isOpen, setIsOpen] = useState(false);

  return (
    <div>
      <button onClick={() => setIsOpen(true)}>Open Modal</button>
      {isOpen && (
        <div style={modalStyle}>
          <div style={modalContentStyle}>
            <h2>Basic Modal</h2>
            <p>This is a simple modal!</p>
            <button onClick={() => setIsOpen(false)}>Close</button>
          </div>
        </div>
      )}
    </div>
  );
};

const modalStyle = {
  position: 'fixed',
  top: 0,
  left: 0,
  width: '100%',
  height: '100%',
  backgroundColor: 'rgba(0, 0, 0, 0.5)',
  display: 'flex',
  justifyContent: 'center',
  alignItems: 'center',
};

const modalContentStyle = {
  background: '#fff',
  padding: '20px',
  borderRadius: '5px',
  width: '300px',
  textAlign: 'center',
};

export default App;
```

When you press the Escape key, the modal will not close.

The issue occurs because the code does not have an event listener for detecting when the Escape key is pressed. You need to add logic to handle the Escape key and close the modal programmatically when it is pressed.

Example: Code with Escape Key Support

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

const App = () => {
  const [isOpen, setIsOpen] = useState(false);

  // Function to handle Escape key press
  const handleKeyDown = (event) => {
    if (event.key === 'Escape') {
      setIsOpen(false);
    }
  };
  useEffect(() => {
    // Add event listener for keydown when the modal is open
    if (isOpen) {
      window.addEventListener('keydown', handleKeyDown);
    }

    // Clean up the event listener when the modal is closed or the component
unmounts
    return () => {
      window.removeEventListener('keydown', handleKeyDown);
    };
  }, [isOpen]);

  return (
    <div>
      <button onClick={() => setIsOpen(true)}>Open Modal</button>
      {isOpen && (
        <div style={modalStyle}>
          <div style={modalContentStyle}>
            <h2>Basic Modal</h2>
            <p>This is a simple modal!</p>
            <button onClick={() => setIsOpen(false)}>Close</button>
          </div>
        </div>
      )}
    </div>
  );
};

const modalStyle = {
  position: 'fixed',
  top: 0,
  left: 0,
  width: '100%',
  height: '100%',
  backgroundColor: 'rgba(0, 0, 0, 0.5)',
  display: 'flex',
  justifyContent: 'center',
  alignItems: 'center',
};

const modalContentStyle = {
  background: '#fff',
  padding: '20px',
  borderRadius: '5px',
  width: '300px',
  textAlign: 'center',
};

export default App;
```

note: In the above program, the event listener is efficiently managed, ensuring there are no unnecessary listeners attached to the window object when the modal is not open.

Example: Modal with Contextual Actions

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

const App = () => {
  const [isOpen, setIsOpen] = useState(false);

  const handleDelete = () => {
    console.log('Item Deleted');
    setIsOpen(false);
  };

  return (
    <div>
      <button onClick={() => setIsOpen(true)}>Delete Item</button>
      {isOpen && (
        <div style={modalStyle}>
          <div style={modalContentStyle}>
          <h2>Confirm Deletion</h2>
          <p>Are you sure you want to delete this item?</p>
          <button onClick={handleDelete}>Yes</button>
          <button onClick={() => {console.log("no item deleted");
setIsOpen(false)}}>No</button>
        </div>
      </div>
    )}
  </div>
 );
};

const modalStyle = {
 position: 'fixed',
 top: 0,
 left: 0,
 width: '100%',
 height: '100%',
 backgroundColor: 'rgba(0, 0, 0, 0.5)',
 display: 'flex',
 justifyContent: 'center',
 alignItems: 'center',
};

const modalContentStyle = {
 background: '#fff',
 padding: '20px',
 borderRadius: '5px',
 width: '300px',
 textAlign: 'center',
};

export default App;
```

Portals in React

- Portals provide a way to render a child component into a DOM node that exists outside the DOM hierarchy of the parent component.
- By default, React components are rendered within the DOM tree of their parent components. However, sometimes we may need to render a component outside of this hierarchy. Portals make this possible.
- Portals are created using the ReactDOM.createPortal method.
- Portals are a powerful tool in React for rendering UI elements like modals, tooltips, or dropdowns outside the parent DOM hierarchy.
- They solve common CSS and rendering issues while maintaining React's event-handling structure.
- By using ReactDOM.createPortal, you can cleanly and efficiently render complex components like modals in real-world applications.

Relation Between Portals and Modals

Modals

- Modals are UI elements that appear on top of the current page and block interaction with the rest of the application until the modal is closed.
- Since modals often need to be visually and interactively distinct from their parent components, they are good candidates for being rendered using portals.

Why use Portals for Modals

- Modals may face z-index or CSS overflow issues when rendered inside the parent component's DOM hierarchy.
- By using portals, you can render the modal outside the normal DOM hierarchy, ensuring that it behaves and appears as expected.

Advantages of Portals for Modals

- **CSS Isolation:** Modals won't be affected by the parent's CSS rules, like overflow or z-index.
- **Cleaner Code:** Keeps modal logic isolated from the main application components.
- **Better Debugging:** Separating modal rendering into a dedicated DOM node makes debugging easier.

Example : Modal Using Portals

```
import React, { useState } from 'react';
import ReactDOM from 'react-dom';

const PortalModal = ({ isOpen, onClose }) => {
  if (!isOpen) return null;

  return ReactDOM.createPortal(
    <div style={modalStyle}>
      <div style={modalContentStyle}>
        <h2>Portal Modal</h2>
        <p>This modal is rendered using React Portals.</p>
        <button onClick={onClose}>Close</button>
      </div>
    </div>,
    document.getElementById('modal-root')
  );
};

const App = () => {
  const [isOpen, setIsOpen] = useState(false);

  return (
    <div>
      <button onClick={() => setIsOpen(true)}>Open Modal</button>
      <PortalModal isOpen={isOpen} onClose={() => setIsOpen(false)} />
    </div>
  );
};

const modalStyle = {
  position: 'fixed',
  top: 0,
  left: 0,
  width: '100%',
  height: '100%',
  backgroundColor: 'rgba(0, 0, 0, 0.5)',
  display: 'flex',
  justifyContent: 'center',
  alignItems: 'center',
};

const modalContentStyle = {
  background: '#fff',
  padding: '20px',
  borderRadius: '5px',
  width: '300px',
  textAlign: 'center',
};


export default App;
```

Example: Rendering a Modal using Portals

HTML Structure: Ensure your public/index.html includes a separate container for the portal:

```
<div id="root"></div>
<div id="modal-root"></div>
```

App.js

```
import React from "react";
import ReactDOM from "react-dom";

const Modal = ({ isOpen, onClose, children }) => {
  if (!isOpen) return null;

  return ReactDOM.createPortal(
    <div style={modalStyle}>
      <div style={modalContentStyle}>
        <button onClick={onClose} style={closeButtonStyle}>
```

```
          X
        </button>
        {children}
      </div>
    </div>,
    document.getElementById("modal-root") // Render in a separate DOM node
  );
};

const App = () => {
  const [isModalOpen, setModalOpen] = React.useState(false);

  return (
    <div>
      <h1>React Portals Example</h1>
      <button onClick={() => setModalOpen(true)}>Open Modal</button>
      <Modal isOpen={isModalOpen} onClose={() => setModalOpen(false)}>
        <h2>Modal Content</h2>
        <p>This modal is rendered using portals!</p>
      </Modal>
    </div>
  );
};

const modalStyle = {
  position: "fixed",
  top: 0,
  left: 0,
  width: "100%",
  height: "100%",
  backgroundColor: "rgba(0, 0, 0, 0.5)",
  display: "flex",
  justifyContent: "center",
  alignItems: "center",
};

const modalContentStyle = {
  background: "#fff",
  padding: "20px",
  borderRadius: "5px",
  width: "300px",
  textAlign: "center",
};

const closeButtonStyle = {
  position: "absolute",
  top: "10px",
  right: "10px",
};

export default App;
```

Let's consider a scenario:

Suppose you are building an e-commerce application where clicking on a product displays its details in a modal. The modal will be rendered outside the main app's DOM hierarchy using portals.

HTML Structure

Ensure your public/index.html file has a separate container for the modal:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>React Portal Example</title>
  </head>
  <body>
    <div id="root"></div>
    <div id="modal-root"></div>
  </body>
</html>
```

Modal Component : The Modal component will render its content in the #modal-root using portals.

Modal.js

```
import React from "react";
import ReactDOM from "react-dom";

const Modal = ({ isOpen, onClose, children }) => {
  if (!isOpen) return null;

  return ReactDOM.createPortal(
    <div style={overlayStyle} onClick={onClose}>
      <div style={modalStyle} onClick={(e) => e.stopPropagation()}>
        <button style={closeButtonStyle} onClick={onClose}>
          &times;
        </button>
        {children}
      </div>
    </div>,
    document.getElementById("modal-root")
  );
};

const overlayStyle = {
  position: "fixed",
  top: 0,
  left: 0,
  width: "100%",
  height: "100%",
  backgroundColor: "rgba(0, 0, 0, 0.7)",
  display: "flex",
  justifyContent: "center",
  alignItems: "center",
};

const modalStyle = {
  backgroundColor: "#fff",
  padding: "20px",
  borderRadius: "5px",
  position: "relative",
  width: "400px",
  textAlign: "center",
};

const closeButtonStyle = {
  position: "absolute",
  top: "10px",
  right: "10px",
  background: "none",
  border: "none",
  fontSize: "18px",
  cursor: "pointer",
};

export default Modal;
```

Main App : The main app will handle when to show the modal and pass data to it.
App.js

```
import React, { useState } from "react";
import Modal from "./Modal";
const App = () => {
  const [isModalOpen, setModalOpen] = useState(false);
  const [product, setProduct] = useState(null);
  const products = [
    { id: 1, name: "Laptop", description: "High performance laptop" },
    { id: 2, name: "Phone", description: "Latest smartphone" },
    { id: 3, name: "Headphones", description: "Noise-cancelling headphones" },
  ];

  const openModal = (product) => {
    setProduct(product);
    setModalOpen(true);
  };
```

```
 const closeModal = () => {
  setModalOpen(false);
  setProduct(null);
 };
 return (
  <div>
   <h1>Product List</h1>
   <ul>
    {products.map((prod) => (
     <li key={prod.id}>
      {prod.name}{" "}
      <button onClick={() => openModal(prod)}>View Details</button>
     </li>
    ))}
   </ul>
   <Modal isOpen={isModalOpen} onClose={closeModal}>
    {product && (
     <>
      <h2>{product.name}</h2>
      <p>{product.description}</p>
      <button onClick={closeModal}>Close</button>
     </>
    )}
   </Modal>
  </div>
 );
};

export default App;
```

Advantages in the Example

- **Z-index Management:** The modal is rendered on top of all content, even if the parent components have complex z-index styles.
- **Reusability:** The Modal component can be reused anywhere in the app for different purposes.
- **Event Propagation:** Clicking outside the modal closes it (overlay click), while clicking inside does not (using stopPropagation).

Difference Between Modals and Portals

Modals

- A modal is a UI design pattern used to create a dialog or popup that overlays the rest of the UI, blocking user interaction with the main application until the modal is dismissed.
- A modal is a component that is part of your React application and can be implemented without portals.

Portals

- A portal is a React feature used to render a component (like a modal) outside its parent DOM hierarchy, into a different DOM node (usually at the root level of the application).
- Portals are used to solve rendering issues related to modals, such as z-index or CSS overflow problems, ensuring the modal is rendered in the correct visual context.

Example: Using Modal Without Portals

The example shows a modal without using portals. It renders inside the parent DOM hierarchy.

```
import React, { useState } from "react";
const ModalWithoutPortal = ({ isOpen, onClose, children }) => {
 if (!isOpen) return null;
 return (
   <div style={overlayStyle} onClick={onClose}>
     <div style={modalStyle} onClick={(e) => e.stopPropagation()}>
       <button style={closeButtonStyle} onClick={onClose}>
         &times;        </button>
       {children}
     </div>
   </div>
 );};
const App = () => {
 const [isModalOpen, setModalOpen] = useState(false);
 return (
   <div style={{ position: "relative", zIndex: 1 }}>
```

```
      <h1>Modal Without Portals</h1>
      <button onClick={() => setModalOpen(true)}>Open Modal</button>
      <ModalWithoutPortal        isOpen={isModalOpen}        onClose={()        =>
setModalOpen(false)}>
        <h2>Modal Content</h2>
        <p>This modal is rendered without portals.</p>
      </ModalWithoutPortal>
    </div>
  );};
const overlayStyle = {
  position: "fixed",
  top: 0,
  left: 0,
  width: "100%",
  height: "100%",
  backgroundColor: "rgba(0, 0, 0, 0.7)",
  display: "flex",
  justifyContent: "center",
  alignItems: "center",
  zIndex: 2, // Must be higher than the parent container's z-index
};
const modalStyle = {
  backgroundColor: "#fff",
  padding: "20px",
  borderRadius: "5px",
  position: "relative",          };
const closeButtonStyle = {
  position: "absolute",
  top: "10px",
  right: "10px",          };

export default App;
```

note:

- The modal is part of the parent DOM hierarchy.
- It works, but if the parent has z-index issues or overflow: hidden, the modal might be clipped.

Example: Using Modal With Portals

The example renders the same modal but uses portals to render it outside the parent DOM hierarchy.

```
import React, { useState } from "react";
import ReactDOM from "react-dom";

const ModalWithPortal = ({ isOpen, onClose, children }) => {
  if (!isOpen) return null;

  return ReactDOM.createPortal(
    <div style={overlayStyle} onClick={onClose}>
      <div style={modalStyle} onClick={(e) => e.stopPropagation()}>
        <button style={closeButtonStyle} onClick={onClose}>
          &times;
        </button>
        {children}
      </div>
    </div>,
    document.getElementById("modal-root") // Render outside parent hierarchy
  );
};

const App = () => {
  const [isModalOpen, setModalOpen] = useState(false);

  return (
    <div style={{ position: "relative", zIndex: 1 }}>
      <h1>Modal With Portals</h1>
      <button onClick={() => setModalOpen(true)}>Open Modal</button>
      <ModalWithPortal isOpen={isModalOpen} onClose={() =>
setModalOpen(false)}>
        <h2>Modal Content</h2>
        <p>This modal is rendered with portals.</p>
      </ModalWithPortal>
    </div>
  );
};

const overlayStyle = {
  position: "fixed",
  top: 0,
  left: 0,
  width: "100%",
  height: "100%",
  backgroundColor: "rgba(0, 0, 0, 0.7)",
  display: "flex",
  justifyContent: "center",
  alignItems: "center",
  zIndex: 2, // z-index still applies, but it's independent of parent
};

const modalStyle = {
  backgroundColor: "#fff",
  padding: "20px",
  borderRadius: "5px",
  position: "relative",
};

const closeButtonStyle = {
  position: "absolute",
  top: "10px",
  right: "10px",
};

export default App;
```

HTML File: Add a modal-root for the portal

```
<div id="root"></div>
<div id="modal-root"></div>
```

Note:

- The modal is rendered outside the parent DOM hierarchy (inside #modal-root).
- It solves issues like clipping (overflow: hidden) or improper z-index stacking.

Points to Remember

- Use modal without portals for simple UIs where rendering within the parent DOM doesn't cause issues.
- Use portals for modals when you need to isolate the modal's rendering context from the parent DOM to handle CSS complexities and ensure correct positioning.

Concluded Comparison

Feature	**Modal Without Portal**	**Modal With Portal**
Rendering Context	Part of the parent DOM hierarchy.	Outside the parent DOM hierarchy.
Use Cases	Works fine in simple scenarios.	Handles complex cases (e.g., z-index, overflow).
Implementation Complexity	Easier to implement.	Requires ReactDOM.createPortal
Clipping Issues	May face issues with parent styles.	No clipping issues.
Reusability	Reusable, but context-dependent.	Fully reusable, independent of parent.

Demonstrating Complex Cases

Where a modal (rendered without a portal) will face issues due to CSS properties like z-index or overflow, and how these issues are resolved when the modal is rendered using portals.

Scenario: You have a parent container with the following styles:

- **overflow: hidden:** This can clip any child elements rendered outside the visible area.
- **z-index issues:** The modal might not appear above other elements because it's part of the parent's DOM hierarchy.

Observations

Feature	**Modal without Portal**	**Modal with Portal**
Parent Overflow	Clipped due to overflow: hidden	No clipping; rendered outside parent
Z-index Issues	Dependent on parent's stacking context	Independent of parent's stacking context
Visual Behavior	Modal may appear behind other elements	Modal always appears on top

Example: Modal Without Portal

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

const ModalWithoutPortal = ({ isOpen, onClose, children }) => {
  if (!isOpen) return null;
  return (
    <div style={overlayStyle} onClick={onClose}>
      <div style={modalStyle} onClick={(e) => e.stopPropagation()}>
        <button style={closeButtonStyle} onClick={onClose}>
          &times;
        </button>
        {children}
      </div>     </div>
  );};

const App = () => {
  const [isModalOpen, setModalOpen] = useState(false);
  return (
    <div style={parentStyle}>
      <h1>Modal Without Portals</h1>
      <button onClick={() => setModalOpen(true)}>Open Modal</button>
      <div style={otherContentStyle}>
        This is another section with higher z-index.
      </div>
      <ModalWithoutPortal isOpen={isModalOpen} onClose={() =>
setModalOpen(false)}>
        <h2>Modal Content</h2>
        <p>This modal is rendered without portals.</p>
      </ModalWithoutPortal>
    </div>
  );};
const parentStyle = {
  position: "relative",
  zIndex: 1,
  overflow: "hidden",
  height: "200px",
  border: "1px solid black",  };
```

```
const overlayStyle = {
  position: "fixed",
  top: 0,
  left: 0,
  width: "100%",
  height: "100%",
  backgroundColor: "rgba(0, 0, 0, 0.7)",
  display: "flex",
  justifyContent: "center",
  alignItems: "center",
  zIndex: 2, // Relies on parent z-index
};
const modalStyle = {
  backgroundColor: "#fff",
  padding: "20px",
  borderRadius: "5px",
  position: "relative",        };
const closeButtonStyle = {
  position: "absolute",
  top: "10px",
  right: "10px",          };
const otherContentStyle = {
  position: "absolute",
  top: "50px",
  left: "50px",
  zIndex: 10, // Higher than the modal
  backgroundColor: "#f8d7da",
  padding: "10px",
  color: "#721c24",    };

export default App;
```

Points to consider

- Since the parent has overflow: hidden, the modal might get clipped if it extends outside the parent's visible area.
- The modal’s z-index is scoped to the parent’s stacking context, and other content with a higher z-index (like the "other content" div) may appear above the modal.

Example: Modal With Portal

```
import React, { useState } from "react";
import ReactDOM from "react-dom";

const ModalWithPortal = ({ isOpen, onClose, children }) => {
  if (!isOpen) return null;
  return ReactDOM.createPortal(
    <div style={overlayStyle} onClick={onClose}>
      <div style={modalStyle} onClick={(e) => e.stopPropagation()}>
        <button style={closeButtonStyle} onClick={onClose}>
          &times;
        </button>
        {children}
      </div>
    </div>,
    document.getElementById("modal-root")
  );};

const App = () => {
  const [isModalOpen, setModalOpen] = useState(false);
  return (
    <div style={parentStyle}>
      <h1>Modal With Portals</h1>
      <button onClick={() => setModalOpen(true)}>Open Modal</button>
      <div style={otherContentStyle}>
        This is another section with higher z-index.      </div>
      <ModalWithPortal          isOpen={isModalOpen}          onClose={()          =>
setModalOpen(false)}>
        <h2>Modal Content</h2>
        <p>This modal is rendered with portals.</p>
      </ModalWithPortal>     </div>
  );      };
const parentStyle = {
  position: "relative",
  zIndex: 1,
  overflow: "hidden",
  height: "200px",
  border: "1px solid black",  };
const overlayStyle = {
  position: "fixed",
  top: 0,
  left: 0,
  width: "100%",
  height: "100%",
  backgroundColor: "rgba(0, 0, 0, 0.7)",
  display: "flex",
  justifyContent: "center",
  alignItems: "center",
  zIndex: 1000, };       // Independent of parent z-index
const modalStyle = {
  backgroundColor: "#fff",
  padding: "20px",
  borderRadius: "5px",
  position: "relative",        };
const closeButtonStyle = {
  position: "absolute",
  top: "10px",
  right: "10px",          };
const otherContentStyle = {
  position: "absolute",
  top: "50px",
  left: "50px",
  zIndex: 10, // Higher than parent but irrelevant to the portal
  backgroundColor: "#f8d7da",
  padding: "10px",
  color: "#721c24", };

export default App;
```

Points to consider

- The modal is rendered directly into #modal-root in the public/index.html, outside the parent container with overflow: hidden. This ensures no clipping.
- The modal is not constrained by the parent container's z-index. The portal allows it to have a z-index independent of its parent.

Example: Multi-Step Modal Workflow

```
import React, { useState } from 'react';
const App = () => {
  const [step, setStep] = useState(1);
  const nextStep = () => setStep(step + 1);
  const prevStep = () => setStep(step - 1);
  return (
    <div>
      <button onClick={() => setStep(1)}>Open Modal</button>
      {step > 0 && (
        <div style={modalStyle}>
          <div style={modalContentStyle}>
            {step === 1 && <p>Step 1: Enter Personal Details</p>}
            {step === 2 && <p>Step 2: Enter Professional Details</p>}
            {step === 3 && <p>Step 3: Confirm Submission</p>}
            <div>
              {step > 1 && <button onClick={prevStep}>Back</button>}
              {step < 3 ? <button onClick={nextStep}>Next</button> : <button
onClick={() => setStep(0)}>Close</button>}
            </div>
          </div>
        </div>
      )}
    </div>
  );
};
const modalStyle = {
  position: 'fixed',
  top: 0,
  left: 0,
  width: '100%',
  height: '100%',
  backgroundColor: 'rgba(0, 0, 0, 0.5)',
  display: 'flex',
  justifyContent: 'center',
  alignItems: 'center',
};
```

```
const modalContentStyle = {
  background: '#fff',
  padding: '20px',
  borderRadius: '5px',
  width: '300px',
  textAlign: 'center',
};
export default App;
```

Example: Modal with Animation

App.js

```
import React, { useState } from 'react';
import './App.css';

const App = () => {
  const [isOpen, setIsOpen] = useState(false);
  const [isClosing, setIsClosing] = useState(false);

  const closeModal = () => {
    setIsClosing(true);
    setTimeout(() => {
      setIsClosing(false);
      setIsOpen(false);
    }, 300); // Match the duration of fadeOut animation
  };

  return (
    <div>
      <button onClick={() => setIsOpen(true)}>Open Modal</button>
      {isOpen && (
        <div
          className={`modal-overlay ${isClosing ? 'fade-out' : ''}`}
          onClick={closeModal}
        >
```

```
          <div
            className={`modal-content ${isClosing ? 'slide-out' : ''}`}
            onClick={(e) => e.stopPropagation()}
          >
            <h2>Animated Modal</h2>
            <p>This modal has entrance and exit animations!</p>
            <button onClick={closeModal}>Close</button>
          </div>
        </div>
      )}
    </div>
  );
};

export default App;
```

App.css

```
/* Modal Overlay */
.modal-overlay {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background-color: rgba(0, 0, 0, 0.7);
  display: flex;
  justify-content: center;
  align-items: center;
  opacity: 0;
  visibility: hidden;
  animation: fadeIn 0.3s forwards;
}

/* Modal Content */
.modal-content {
  background: #fff;
  padding: 20px;
  border-radius: 10px;
  width: 400px;
  text-align: center;
  transform: scale(0.8);
  animation: slideIn 0.3s forwards;
}

/* Button Style */
button {
  background-color: #007bff;
  color: white;
  border: none;
  padding: 10px 20px;
  border-radius: 5px;
  cursor: pointer;
  font-size: 16px;
}

button:hover {
  background-color: #0056b3;
}

/* Animations */
@keyframes fadeIn {
  from {
    opacity: 0;
    visibility: hidden;
  }
  to {
    opacity: 1;
    visibility: visible;
  }
}

@keyframes fadeOut {
  from {
    opacity: 1;
    visibility: visible;
  }
  to {
    opacity: 0;
    visibility: hidden;
  }
}

@keyframes slideIn {
  from {
    transform: scale(0.8);
  }
  to {
    transform: scale(1);
  }
}

@keyframes slideOut {
  from {
    transform: scale(1);
  }
  to {
    transform: scale(0.8);
  }
}
/* Add fade-out animation class */
.modal-overlay.fade-out {
  animation: fadeOut 0.3s forwards;
}

/* Add slide-out animation class */
.modal-content.slide-out {
  animation: slideOut 0.3s forwards;
}
```