**When building a React-Redux application with accessibility in mind, there are several key points to consider.**

**Semantic HTML:** Use appropriate HTML elements to convey the meaning and structure of your application. For example, use <nav> for navigation, <button> for interactive elements, <input> for form inputs, etc. This helps screen readers and other assistive technologies understand the content and functionality.

**ARIA Attributes:** Augment your HTML with ARIA (Accessible Rich Internet Applications) attributes to provide additional information and context for accessibility. For instance, you can use the aria-label attribute to provide a descriptive label for elements that lack visible text

Example:

<button aria-label="Close modal" onClick={handleClose}>

<span aria-hidden="true">&times;</span>

</button>

**Focus Management:** Manage focus properly to guide users through your application and provide clear visual cues. When opening a modal or a dropdown, for example, you can programmatically shift focus to the appropriate element.

Example:

function openModal() {

setModalOpen(true);

}

useEffect(() => {

if (isModalOpen) {

// Shift focus to the modal when it opens

modalRef.current.focus();

}

}, [isModalOpen]);

**Alternative Text for Images:** Include descriptive alternative text for images using the alt attribute. This text is read aloud by screen readers to provide a meaningful description of the image

Example:

<img src="path/to/image.jpg" alt="A red apple on a white background" />

**Color Contrast:** Ensure that your application has sufficient color contrast between foreground and background elements. This helps users with visual impairments or color deficiencies to read and understand the content. You can use online tools, such as WebAIM's Contrast Checker, to verify color contrast compliance.