Session 9 Instructor Guide: Custom Hooks & Browser APIs

Learning Outcomes

By the end of Session 9, students will be able to:

1. **Define and implement custom hooks** that bundle reusable logic and follow React naming conventions
2. **Understand the DRY principle** (Don't Repeat Yourself) and explain how custom hooks eliminate code duplication
3. **Distinguish built-in hooks from custom hooks** and identify common use cases for each type
4. **Describe the HTMLAudioElement interface** and its role in programmatic audio control
5. **Compare refs and state** to understand when to use each approach
6. **Use the useRef hook** to create persistent references and access ref values using the current property
7. **Apply lazy initialization** patterns to create resources only when first needed
8. **Connect browser APIs** with React hooks to build interactive features
9. **Implement error handling** in custom hooks to create robust, polished code
10. **Use AI assistance** effectively to learn developer workflows and prompt engineering techniques
11. **Build interactive components** that leverage HTMLAudioElement through custom hooks
12. **Apply cleanup patterns** using useEffect to prevent memory leaks and resource conflicts

Instruction

Instructor introduces key concepts students need to succeed:

1. **Custom Hooks Philosophy** - Define custom hooks as React's solution for bundling reusable logic, emphasizing the DRY principle and "write once, use often" approach
2. **Web Audio Interface** - Describe HTMLAudioElement as part of the Web API for audio control, demonstrating basic audio playback methods
3. **Refs vs State Distinction** - Compare when to use refs versus state for different data management approaches

4. **useRef Hook Mechanics** - Show how useRef creates persistent storage with the current property pattern
5. **Lazy Initialization Pattern** - Introduce lazy initialization as a resource management strategy
6. **Component Integration** - Show how to build MusicToggle component as testing interface before implementing functionality
7. **Audio Reference Implementation** - Demonstrate adding useRef storage to useAudio hook for persistent audio elements
8. **Audio Playback Creation** - Guide implementation of play function with HTMLAudioElement and lazy initialization
9. **GitHub Copilot Workflow** - Introduce GitHub Copilot to teach developer workflows and prompt engineering techniques
10. **Solo Mission Preparation** - Set up independent challenge for pause functionality, error handling, and cleanup with AI assistance
11. **Memory Management** - Explain cleanup patterns to prevent audio conflicts and memory leaks
12. **Let's Add Music!** - Launch the hands-on mission: build a complete audio system using custom hooks, Web APIs, and AI assistance

---

Slide Deck Outline

Slide 1: Custom Hooks & Browser APIs 🎵

- **Title:** "Session 9: Custom Hooks & Browser APIs — Adding Theme Music"
- **Session 8 Recap:** "Last time: You implemented scoring and victory with complex state management"
- **Hook:** "Your game tracks progress — now let's make it sound amazing!"
- **Today's Mission:** Build custom audio hooks with browser API integration and AI-assisted development
- **Visual:** Audio waveform with React hook icons
- **Connection:** "From interactive components to immersive audio experiences!"

Slide 2: Custom Hooks 🪝

- **Teaching Focus:** Why custom hooks make coding easier and more fun

- **Key Concepts:**
  - **Custom hooks** as functions starting with “use” that extract component logic
  - **DRY principle** - “Don’t Repeat Yourself” through reusable solutions
  - **“Write once, use often”** approach to eliminate code duplication
- **Built-in vs Custom Hooks Comparison:**
  - Built-in: `useState`, `useEffect`, `useRef` (provided by React)
  - Custom: `useGame`, `useAudio` (created by developers)
- **Discussion Questions:**
  - “What happens when you copy audio logic into every component?”
  - “How do custom hooks solve code duplication?”
- **Real-World Context:** Show examples from popular React libraries
- **Student Preparation:** “You’ll create useAudio to bundle all audio complexity”

Slide 3: Browser APIs 🔊

- **Teaching Focus:** A perfect example of what custom hook use case, i.e., adding audio functionality to a web-based game
- **Key Concepts:**
  - **HTMLAudioElement** as browser’s built-in interface for audio control
  - **Programmatic control** through JavaScript vs HTML audio tags
  - **Core methods:** `play()`, `pause()`, volume, loop properties
  - **Promise-based play()** method for error handling
- **Demonstration:** Live audio control with HTMLAudioElement
- **Hook Use Case:** “HTMLAudioElement is exactly what useAudio will wrap in a reusable way”
- **Common Misconceptions:** Address when students think they need external audio libraries
- **Preview:** “Next: How do we store browser objects in React components?”

Slide 4: Refs and State 🔗

- **Teaching Focus:** Decision-making criteria for data storage
- **Key Concepts:**
  - **State:** Triggers re-renders, for UI-affecting data, accessed with `value`

    - **Refs:** No re-renders, for non-UI data, accessed with `ref.current`
    - **Refs as bookmarks** that remember information without affecting rendering
- **Interactive Exercise:** Students categorize different data types
- **Key Question:** "Does changing this data affect what users see?"
- **Audio Example:** Audio object (ref) vs playback state (state)
- **Key Skill:** "This decision-making process applies to all React development"

Slide 5: useRef Hook 📍

- **Teaching Focus:** Resource management strategies and common useRef patterns
- **Key Concepts:**
    - **useRef mechanics:** Container for mutable data, persistent across updates
    - **Access via current:** `ref.current` holds the actual value
    - **Mutable data** that can be changed without triggering re-renders
    - **Lazy initialization:** Creating resources only when first needed
- **Common useRef Patterns:**
    - **Storing Mutable Values (Audio Use Case):** `audioRef.current = new Audio(src)` - bookmark pattern for browser objects
    - **Accessing DOM Elements:** `inputRef.current.focus()` - direct connection to HTML elements
    - **Key Pattern:** `if (!ref.current)` for lazy initialization and null checks
- **Audio Example Walkthrough:**

```
const audioRef = useRef(null); // Bookmark starts empty
if (!audioRef.current) {       // Check if audio exists
  audioRef.current = new Audio(src); // Create and bookmark
}
audioRef.current.play();       // Use bookmarked audio
```

- **Performance Discussion:** Why create resources only when needed?
- **Memory Management:** How refs persist across re-renders without causing them
- **Student Application:** "You'll use both patterns - mutable values for audio storage and the bookmark concept for persistent references"

Slide 6: Component Integration 🎛️

- **Teaching Focus:** Development workflow
- **Key Concepts:**
  - **Conditional rendering** with dynamic images and tooltips
  - **State-driven UI** where appearance changes based on data
  - **Visual feedback** through different icons for play/pause states
- **Strategy:** Build testing interface before functionality
- **Benefits:** Immediate visual feedback during development
- **Student Guidance:** “MusicToggle gives you a way to test useAudio as you build it”
- **Key Strategy:** “This is how developers build complex features step by step”

Slide 7: Audio Reference 📻

- **Teaching Focus:** Connecting useRef with audio storage
- **Instructor Demonstration:** Add audioRef to useAudio hook
- **Key Concepts:** Persistent storage, null initialization
- **Student Checkpoint:** “Now your hook can remember audio elements”
- **Next Step Preview:** “Ready to create and control audio playback”

Slide 8: Audio Playback 🎵

- **Teaching Focus:** Efficient resource creation patterns
- **Instructor Demonstration:** Implement play function with lazy initialization
- **Key Insight:** Create audio elements only when needed
- **Performance Benefits:** Discuss resource efficiency
- **Student Milestone:** “Music toggle now controls actual audio playback”

Slide 9: AI Assistance 🤖

- **Teaching Focus:** How to work effectively with AI tools
- **Key Concepts:**
  - **GitHub Copilot Chat workflow:** Commands, prompts, review, apply, test
  - **Prompt Engineering Tips:** Be specific, include context, specify behavior
  - **High-quality code** development with AI assistance

- **Demonstration:** Live GitHub Copilot workflow
- **Key Skills:** Prompt engineering, code review, testing
- **When to Use AI:** Complex error handling, cleanup patterns, edge cases
- **Student Preparation:** “You’ll use this workflow for your Solo Mission”

Slide 10: Solo Mission 🎖️

- **Teaching Focus:** Independent problem-solving with AI support
- **Mission Overview:** Complete pause, error handling, and cleanup
- **Key Concepts:**
  - **Error handling** with promise-based audio play method
  - **Graceful degradation** when audio fails to load
  - **Memory management** through useEffect cleanup functions
  - **Resource cleanup** to prevent memory leaks and audio conflicts
- **Success Criteria:** Clear requirements and testing strategies
- **AI Guidance:** When and how to use Copilot effectively
- **Instructor Role:** Available for guidance, not implementation

[HANDS-ON WORK HAPPENS HERE]

Slide 11: Hook Architecture Review 🏗️

- **Teaching Focus:** Code quality and maintainability
- **Key Concepts:**
  - **Hook responsibilities:** State management, resource management, API integration
  - **Simple interface design:** Clean API that hides complexity
  - **Reusable patterns:** Any component can add audio with one line
  - **Testable and maintainable:** Logic centralized in one place
- **Discussion:** What makes useAudio well-designed?
- **Key Principles:** Simple interface, robust error handling, proper cleanup
- **Real-World Application:** How this pattern scales to complex applications
- **Student Reflection:** “You’ve built high-quality React code”

Slide 12: What's Next - Version Control & Deployment 🚀

- **Today's Achievement:** "You built custom hooks with browser API integration"
- **Key Skills Gained:** Custom hooks, refs, lazy initialization, AI assistance
- **Next Challenge:** "Deploy your complete game to the internet"
- **Concepts Coming:**
  - **Git workflow** - Version control and collaboration
  - **GitHub Pages** - Free hosting for your React app
  - **Deployment pipeline** - Automated build and deploy process
  - **Production optimization** - Performance and SEO considerations
- **Motivation:** "Your trivia game will soon be live for the world to play!"