

Session 2 Instructor Guide: Creating Reusable Components

Learning Outcomes

By the end of Session 2, students will be able to:

1. **Explain the architectural differences** between React and vanilla JavaScript approaches to DOM updates
2. **Create custom React components** that combine markup, styling, and logic using JSX
3. **Use props** to pass data and behavior from parent to child components
4. **Apply JSX syntax rules** including curly braces for dynamic expressions and className for styling
5. **Structure components effectively** using imports, function declarations, destructuring, and return statements
6. **Style components dynamically** using template literals and variant-based class names
7. **Implement interactivity** by passing functions as props to handle events like clicks
8. **Use default parameters** to provide fallback values for props
9. **Leverage VS Code built-in features** to accelerate component development
10. **Inspect component structure and props** using React DevTools for debugging
11. **Compose components together** to build scalable, maintainable UIs
12. **Follow a smart development workflow** including incremental testing and Hot Module Replacement

Instruction

Instructor introduces key concepts students need to succeed:

1. **Why React Feels Like Magic** - Connect Session 1 component swap experience to React's architectural advantages using relatable language that emphasizes the smooth, magical feeling of React development
2. **React Components: Your First Custom Tags** - Define components as the building blocks of React apps and show how they encapsulate markup, styling, and logic
3. **JSX and Curly Braces** - Explain JSX syntax and how {} enables dynamic content, styling, and behavior inside components

4. **Props: Data and Behavior Flow** - Demonstrate how props allow parent components to pass information and actions to children
 5. **Component Anatomy** - Break down a functional component into imports, function declaration, props destructuring, and JSX return
 6. **Styling with Variants** - Use template literals and dynamic class names to style components based on props
 7. **Functions as Props** - Show how components can trigger actions by receiving functions as props (e.g., onClick)
 8. **Default Parameters and Destructuring** - Introduce fallback values and cleaner syntax for handling props
 9. **VS Code Built-in Features** - Reinforce modern tooling with IntelliSense, auto-completion, and formatting
 10. **React DevTools: Inspect Like a Pro** - Install and use DevTools to explore component trees and props in real time
 11. **Component Composition** - Illustrate how small components combine to form complex UIs, reinforcing the LEGO analogy
 12. **Development Workflow** - Emphasize incremental development, Hot Module Replacement, and debugging best practices
 13. **Let's Build!** - Kick off the hands-on coding mission: create, style, and test your own GameButton component
-

Slide Deck Outline

Slide 1: Creating Reusable Components

- **Title:** “Session 2: Creating Reusable Components — Building Game Components”
- **Session 1 Recap:** “Last time: You set up your trivia game and experienced React’s component system”
- **Hook:** “You’ve experienced React’s magic — now let’s build your own custom components!”
- **Today’s Mission:**
 - **Create** your first reusable React component
 - **Master** props for component communication
 - **Style** components with variants
 - **Install** essential developer tools

- **Experience** component composition in action
- **Visual:** LEGO blocks assembling into a complex structure
- **Connection:** “You’ve experienced React’s component magic — now let’s understand why it’s so powerful and build your own!”

Slide 2: Understanding React’s Approach

- **Title:** “Understanding React’s Approach”
- **Teaching Focus:** Connect Session 1 experience to React’s architectural advantages using relatable language
- **Session 1 Connection:** “Why did swapping `<StartHere />` for `<SplashScreen />` feel so effortless? It’s all about React’s approach to building UIs.”
- **Visual:** The side-by-side flowchart diagram

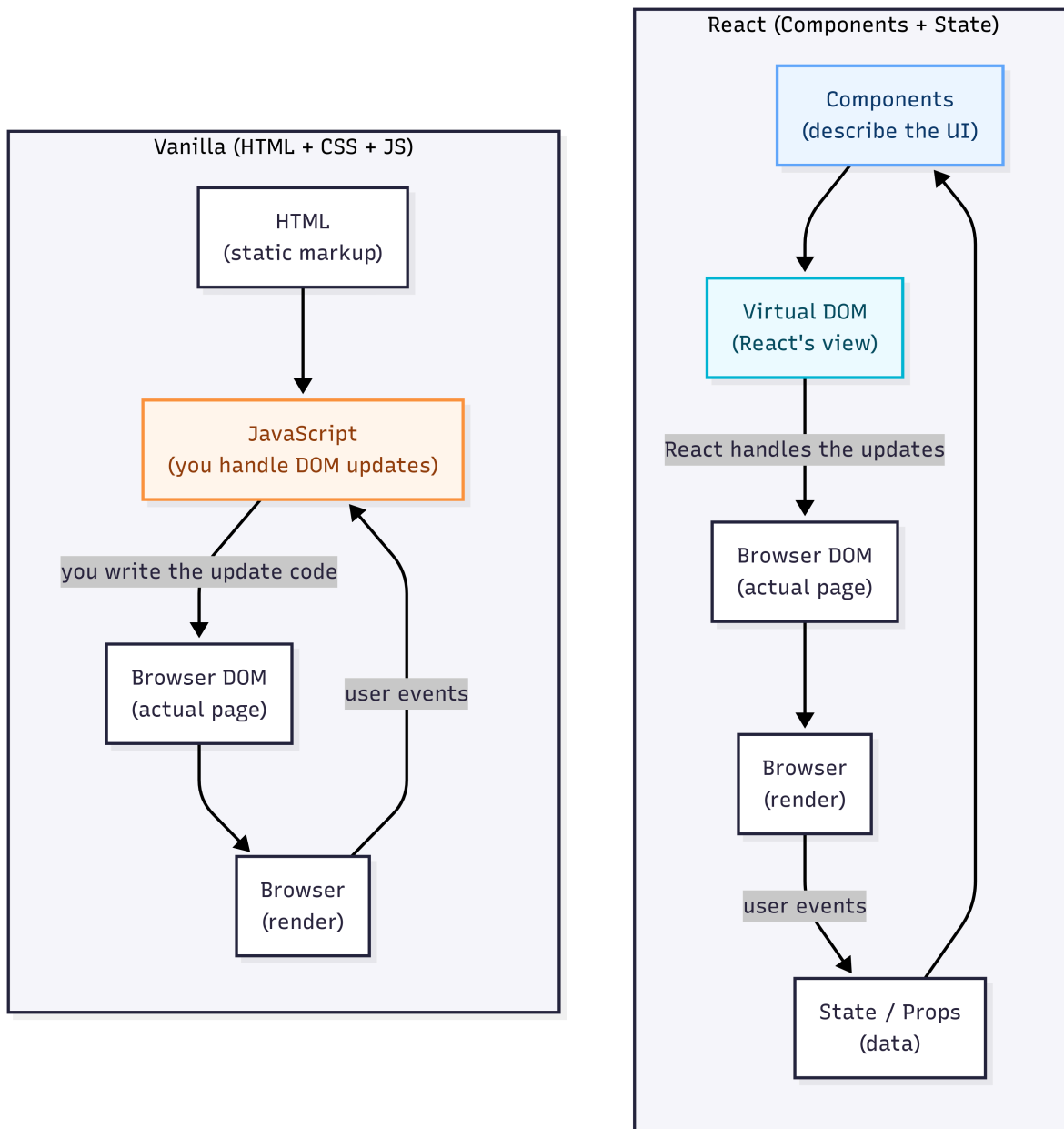


Figure: Vanilla JavaScript vs React — Why Components Make Development Easier

- **Key Teaching Points:**
 - **Vanilla JavaScript:** "You write lots of repetitive code to update the page"
 - **React:** "React works differently: you build self-contained components, and React handles all the messy details of getting them on screen and keeping them updated"
 - **The Connection:** "That's why swapping components felt so smooth. You weren't just editing code—you were shaping the UI with reusable building blocks."
- **Student Motivation:** Use the "magic" framing to make React's advantages feel exciting and accessible
- **Transition:** "Now let's build your first custom component and see that power in action."

Slide 3: Components Are Digital LEGO Blocks 🧩

- **Title:** “What Makes Components So Powerful?”
- **Key Points:**
 - **Reusable** - Write once, use everywhere
 - **Composable** - Small pieces build complex UIs
 - **Maintainable** - Change in one place, updates everywhere
 - **Testable** - Isolated pieces are easier to debug
- **Why Components Matter:**
 - **Scalability** - Apps with hundreds of components stay organized
 - **Team collaboration** - Different developers can work on different components
 - **Code quality** - Smaller pieces are easier to understand and debug
- **Real Example:** “A Button component can be used for ‘Start Game’, ‘Credits’, ‘Submit Answer’, etc.”
- **Visual:** Component tree showing GameButton used in multiple places
- **Student Connection:** “You’ll build a GameButton that works everywhere in your trivia game”
- **Transition:** “Now that you understand React’s advantages, let’s see what makes components so powerful...”

Slide 4: JSX Mastery - React’s Special Language 🌟

- **Title:** “JSX: The Language of React Components”
- **What is JSX?**
 - **JSX** = JavaScript XML - React’s HTML-like syntax
 - Looks familiar but is actually JavaScript
 - Every React component returns JSX to describe its UI
- **File Extensions:** `.jsx` files clearly indicate JSX syntax
- **The Magic of Curly Braces {}:**

```
const name = "Alice";  
return <h1>Hello, {name}!</h1>; // Dynamic content!
```

- **JSX vs HTML - Key Differences:**

- `className` instead of `class` (JavaScript reserved word)
- `onClick` instead of `onclick` (camelCase convention)
- `{expression}` for dynamic content
- Self-closing tags required: `` not ``
- **JSX Gotchas to Remember:**
 - Always close tags: `
` not `
`
 - Use camelCase for event handlers: `onClick`, `onChange`
 - Wrap multi-line JSX in parentheses
- **Curly Brace Power in Action:**
 - `{text}` — displays variable content
 - `{onClick}` — passes function references
 - Dynamic CSS classes:

```
className={`game-button ${variant}`}
```

- **Student Connection:** “JSX + {} = unlimited UI possibilities!”

Slide 4: Anatomy of a React Component

- **Title:** “Component Blueprint: What You’ll Build Today”
- **Visual:** Annotated GameButton component with labeled parts

```
// 1. Import statements (if needed)

// 2. Function declaration with props destructuring
export default function GameButton({ text, onClick, variant = "primary" }) {
  // 3. Logic/variables (optional)
  const buttonClass = `game-button ${variant}`;

  // 4. Return JSX - this is what gets rendered!
  return (
    <button className={buttonClass} onClick={onClick}>
      {text}
    </button>
  );
}
```

- **Usage Example:**

```
<GameButton
  text="Start Adventure"
  onClick={() => alert('Game starting!')}
  variant="primary"
/>
```

- **Component Blueprint:**
 - **Props in** - text, onClick, variant
 - **Logic** - Dynamic class name creation
 - **JSX out** - Styled, interactive button
- **Student Focus:** “This is exactly what you’ll build in the next 20 minutes!”

Slide 5: Props: The Component Communication System 📦

- **Title:** “How Components Talk to Each Other”
- **Analogy:** “Props are like function parameters, but for React components”
- **Visual:** Parent-child diagram showing data flow
- **Key Rules:**
 - **One-way flow** - Parent to child only
 - **Read-only** - Child can’t modify props
 - **Any data type** - strings, numbers, functions, objects
- **Common Prop Types:**
 - **string** - `text="Start Game"`
 - **function** - `onClick={() => alert('Hi!')}`
 - **boolean** - `isDisabled={false}`
 - **number** - `count={42}`
- **Props in Action Preview:** “Watch props flow from SplashScreen to GameButton in real-time”
- **Student Preview:** “You’ll pass text, onClick, and variant as props to your GameButton!”

Slide 6: VS Code Built-in Features - Accelerate Your Development ⚡

- **Title:** “React Developers Use Powerful Tools”

- **Core Concept:** “VS Code has powerful built-in features for React development — IntelliSense, auto-completion, and formatting”
- **Key Features You’ll Use Today:**
 - **IntelliSense** - Intelligent code completion and suggestions
 - **Auto-formatting** - Consistent code style on save
 - **Error detection** - Real-time syntax checking
 - **Import assistance** - Automatic import suggestions
 - **Bracket matching** - Visual code structure helpers
- **Live Demo:** Show IntelliSense suggesting React component structure
- **Benefits:**
 - **Speed** - Built-in suggestions accelerate coding
 - **Consistency** - Auto-formatting ensures clean code
 - **Less errors** - Real-time error detection
 - **Modern workflow** - Widely-used editor
- **Student Encouragement:** “VS Code’s built-in features are powerful — explore extensions later to customize your workflow!”

Slide 7: Functions as Props - Passing Behavior 🎯

- **Title:** “Components That Do Things”
- **Concept:** “Props aren’t just data — they can be functions too!”
- **Example:** `onClick={() => alert('Start Game!')}`
- **Benefits:**
 - **Flexible components** - Same button, different actions
 - **Separation of concerns** - Component handles UI, parent handles logic
 - **Reusability** - One component, many behaviors
- **Visual:** Same GameButton with different onClick behaviors
- **Student Preview:** “Your buttons will show alerts now, navigate screens later”

Slide 8: React DevTools - X-Ray Vision for Your App 🔍

- **Title:** “Essential Debugging Tools”
- **Installation Demo:** Show browser extension installation

- **Key Features:**
 - **Component tree** - See your app's structure
 - **Props inspection** - View data flowing between components
 - **State monitoring** - Watch values change in real-time
 - **Performance profiling** - Optimize your app
- **Debug Like a Pro Checklist:**
 - ☒ Install React DevTools extension
 - ☒ Open Components tab (not Elements)
 - ☒ Click on components to see props
 - ☒ Watch props update in real-time
- **Live Demo:** Inspect GameButton props in DevTools
- **Real-World Context:** "Every React developer uses this daily"

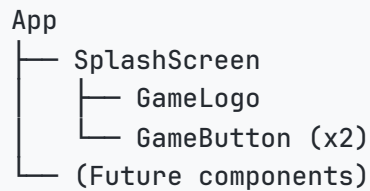
Slide 9: Build Your First Custom Component! 🚀

- **Today's Coding Journey:**
 1. **Create** GameButton.jsx with export default function structure
 2. **Add** text prop for customizable content
 3. **Include** onClick prop for interactivity
 4. **Implement** variant prop for styling
 5. **Import** and use in SplashScreen
 6. **Install** React DevTools for inspection
- **Success Criteria:** "Two styled buttons with working click handlers"
- **Incremental Approach:** "We'll build step by step — test after each change"

[HANDS-ON WORK HAPPENS HERE]

Slide 10: Component Composition - The Big Picture 🌟

- **Title:** "How Small Pieces Build Big Apps"
- **Visual:** Component hierarchy diagram



- **Key Insight:** “Your entire trivia game is just components all the way down”
- **Real-World Perspective:** “Large apps have hundreds of components working together”
- **Student Motivation:** “You’re building the foundation for your entire game”

Slide 11: Development Workflow

- **Title:** “How React Developers Build Components”
- **Workflow Steps:**
 1. **Plan** the component’s purpose and props
 2. **Create** the basic structure with VS Code IntelliSense
 3. **Add** props incrementally
 4. **Test** with Hot Module Replacement
 5. **Style** with CSS classes
 6. **Debug** with React DevTools
- **Best Practices:**
 - **Start simple** - Add complexity gradually
 - **Test often** - Catch issues early
 - **Use tools** - Snippets, DevTools, HMR
- **Student Empowerment:** “You’re learning real development practices”

Slide 12: What’s Next - Shared State with Context

- **Title:** “Preview of Session 3”
- **Today’s Foundation:** “You built reusable components with props”
- **Next Challenge:** “Make components remember things and navigate between screens”
- **Concepts Coming:**
 - **State** - Components that remember data
 - **Context** - Sharing data across the entire app

- **Navigation** - Moving between game screens
- **Motivation:** "Your buttons will actually start the game instead of showing alerts!"
- **Visual:** Preview of game navigation flow