Session 2 — Building Game Components

Building Reusable UI Elements 🧩

You're about to build your first custom React component and unlock the power of reusable UI building blocks — the secret to fast, scalable development in React. This guide walks you through creating a GameButton component, understanding props, and using professional developer tools. Ready to build like a pro? Let's go!

Table of Contents

- Access Your Codespace
- Create Your First Component
- Understanding Props
- Adding Click Functionality
- Styling with Variants
- Reusing Your Component
- Install React DevTools
- Essential Terms
- Ask the AI

☁️ Access Your Codespace

Visit github.com/codespaces to relaunch your Codespace from Session 1.

🧩 Create Your First Component

Let's build a GameButton component for starting your trivia game. Components are like digital LEGO blocks — custom, reusable UI elements.

When you create a component, export it with `export default` so it can be shared across your project. Then bring it into other files with `import`.

1. **Create the file**: Right-click `src/components` → New File → name it `GameButton.jsx`
2. **Type the component structure**:

```
export default function GameButton() {
  return <button>Start Adventure</button>;
}
```

3. **Import into SplashScreen**: Add `import GameButton from "./GameButton"` at the top
4. **Add your button**: Place your button within `div.splash-buttons`:

```
<GameButton />
```

5. **Test it**: Run `npm run dev` and you should see your custom button!

💡 Why This Matters

Components are the heart of React — reusable UI elements that combine markup, styling, and logic. Think of them as your own custom HTML tags. The `.jsx` file extension means you're writing **JSX**, a special syntax that looks like HTML but is actually JavaScript. JSX lets you describe what the UI should look like using readable, expressive code.

🏆 Bonus Challenge

Try changing the button text in GameButton.jsx and watch it update instantly thanks to Hot Module Replacement!

📦 Understanding Props

Props are how you pass data from parent components to child components. They're like function parameters but for React components.

1. **Add text prop to GameButton**:

```
export default function GameButton({ text }) {
  return <button>{text}</button>;
}
```

2. **Update SplashScreen to pass text**:

```
<GameButton text="Start Adventure" />
```

3. **Watch the magic**: Your button now shows custom text!

💡 Why This Matters

Props let parent components pass data to child components — just like function parameters. This makes your components flexible and reusable. The `{ text }` syntax is called **destructuring** — it pulls out just the values you need from the props object, keeping your code clean and readable.

🖱️ Adding Click Functionality

Let's make your buttons actually do something when clicked. In React, you can pass functions as props just like any other data.

1. **Add onClick prop to GameButton**:

```
export default function GameButton({ text, onClick }) {
  return <button onClick={onClick}>{text}</button>;
}
```

2. **Update SplashScreen with click handler**:

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

3. **Test it**: Click your button and see the alert!

💡 Why This Matters

Functions as props are like giving your components different personalities. Your GameButton can do different things depending on where you use it — same button, different actions. It's a key pattern in React for building interactive apps.

🎨 Styling with Variants

Let's add visual variety to your buttons using CSS classes, default parameters, and a clean variable approach.

1. **Add variant prop with default value and create buttonClass variable**:

```
export default function GameButton({ text, onClick, variant = "primary" }) {
  const buttonClass = `game-button ${variant}`;

  return (
    <button className={buttonClass} onClick={onClick}>
      {text}
    </button>
  );
}
```

2. **Update SplashScreen with variant**:

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

3. **Admire your styled button**: Your button now has the primary styling!

💡 Why This Matters

className is React's version of the HTML `class` attribute. We use a **template literal** to build a dynamic class name like `game-button primary`. This matches the styles already defined in your project. The `variant` prop lets you switch between styles like `primary` and `secondary`, and **default parameters** like `variant = "primary"` ensure your component still works even if no variant is passed.

🔄 Reusing Your Component

Now that you've built a complete, fully-featured GameButton component, let's experience the power of reusability by adding a second button for the game's credits.

1. **Add a second button**: Below your existing GameButton in SplashScreen, add one that will show credits when clicked:

```
<GameButton
  text="Credits"
  onClick={() => alert('Show Credits')}
  variant="secondary"
/>
```

2. **Admire your work**: You now have two different buttons using the same component!

💡 Why This Matters

Component reusability is React's superpower. You wrote the GameButton code once, but now you can use it anywhere in your app with different props. Thanks to your stylesheet, each variant (`primary`, `secondary`) automatically applies the right look — no extra styling needed.

🏆 Bonus Challenge

Try adding a third GameButton with `variant="primary"` and `text="Instructions"` to see how easy it is to scale your UI!

🔍 Install React DevTools

React DevTools is like X-ray vision for your React app — see component structure, props, and state in real-time.

Browser Installation

Browser	Installation Link	Notes
Chrome	Chrome Web Store	Most popular choice
Firefox	Firefox Add-ons	Great alternative
Edge	Edge Add-ons	Windows default
Safari	Manual installation required	Advanced users only

Using DevTools

1. **Open DevTools**: Press F12 or right-click → Inspect

2. **Find Components tab**: Look for “Components” next to Console, Network, etc.
3. **Explore your app**: Click on components in the tree to see their props
4. **Inspect GameButton**: Find your GameButton component and see the text, onClick, and variant props!

💡 Why This Matters

React DevTools gives you X-ray vision into your app. You can inspect components, props, and state in real time — just like a pro. It’s one of the most important tools for debugging and understanding how your app works under the hood.

📚 Essential Terms

Quick reference for all the React concepts you just learned:

Term	Definition	Why it matters
🧩 component	A reusable piece of UI that can include markup, styles, and logic (example: `<SplashScreen />`).	You’ll build your entire app by composing components together — they’re React’s building blocks.
📦 props	Data passed from parent to child components.	Props let you customize components and pass data around your app — essential for reusable components.
✨ JSX	JavaScript syntax that looks like HTML — used to describe UI in React components (`.jsx`).	You’ll write JSX in your GameButton component to describe what the button should look like.
🎨 className	React’s version of the HTML `class` attribute for applying CSS styles.	Use className instead of class because class is a reserved word in JavaScript.

📥 destructuring	Extracting values from objects/arrays into variables, like `{ text, onClick }` from props.	Makes your code cleaner by avoiding repetitive `props.text`, `props.onClick` syntax.
🔤 template literals	String interpolation using backticks and `${}` for dynamic strings.	Perfect for creating dynamic CSS classes like `` `game-button ${variant}` ``.
⚙️ default parameters	Fallback values for function parameters, like `variant = "primary"`.	Ensures your components work even when some props aren't provided.
🔍 React DevTools	Browser extension for inspecting React component trees, props, and state.	Essential debugging tool — like X-ray vision for your React app.

🤖 Ask the AI — Component Mastery

You just created your first reusable React component with props, styling, and click handlers — excellent work!

Now let's deepen your understanding of components, props, and the React development workflow. Here are the most impactful questions to ask your AI assistant about today's session:

- **What makes React components reusable and why is that important?**
- **How do props work in React and why are they read-only?**
- **Why do we use className instead of class in React?**
- **How do template literals work and why are they perfect for dynamic CSS classes?**
- **What is interpolation in JSX and can you show me examples?**
- **How does JSX let me write HTML-like code inside JavaScript?**
- **Can I pass functions as props? How does that work and why is it powerful?**
- **What can I do with React DevTools that I can't do with regular browser DevTools?**

💬 Pro Tip:

Components are everywhere in React — every piece of UI you see is a component. Start thinking in components: “What reusable pieces could I break this UI into?” This mindset will make you a better React developer.