Chapter 3 - Laying the foundation

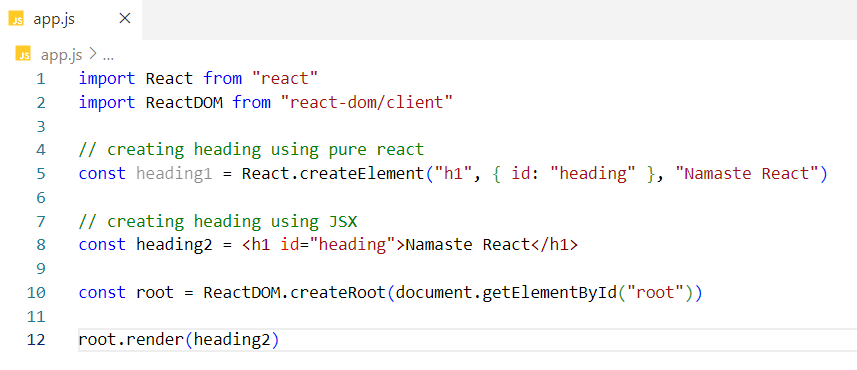
In the previous chapter, we used the command npx parcel index.html to start our app, and npx parcel build index.html to build it. Now, let's configure these commands in the package.json file so we can run them using npm.



Now, we can run our app using npm start or npm run start instead of npx parcel index.html. Similarly, we can build the app using npm run build instead of npx parcel build index.html.

When dealing with deeply nested elements in the DOM, using React's createElement API extensively can lead to code that is difficult to read and maintain. To solve this, Facebook introduced JSX - a syntax extension that significantly improves code readability and clarity.

Below is an example of a React element created with pure React code, followed by the same element written in JSX.



heading1 and heading2 are JavaScript objects when logged to the console.

In JSX syntax, it might look like we’re writing HTML inside JavaScript, but it’s important to understand that this is **not** HTML- it's **JSX,** which only **resembles** HTML.

There is a clear difference between actual HTML syntax and **HTML-like** syntax used in JSX.

What is JSX?

JSX is a syntax extension for JavaScript that lets you write **HTML-like code inside JavaScript.**

* Makes React code **cleaner and more readable**
* Helps create React **elements & components** easily
* Gets converted into plain JavaScript under the hood (via Babel)

In short: **JSX looks like HTML but works as JavaScript**, making React development faster and more intuitive.

Common misconceptions about JSX

1. **"JSX is part of React"** – Wrong
   * JSX does not come by default with React
   * It’s just a **syntax extension** for JavaScript.
   * React can be written **with or without JSX**.
   * JSX only makes React code **shorter and easier to read**.
2. **"JSX is HTML inside JavaScript"** – Wrong
   * JSX only **looks like HTML**, but it is **not HTML**.
   * It is **XML-like syntax** that gets compiled into plain JavaScript (React.createElement).
   * So JSX is more like **JavaScript sugar for writing UI**, not HTML embedded inside JS.

### Difference between HTML and JSX

1. **Expressions and Functions:** JSX allows you to include JavaScript expressions and functions inside {}. HTML, on the other hand, only permits static content.
2. **Transpilation:** JSX must be transpiled into regular JavaScript using tools like Babel. HTML does not require this process.
3. **Attribute Naming**: In HTML, you use class to assign CSS classes. In JSX, you must use className to avoid conflicts with JavaScript’s reserved keyword class.

## More Key Points on JSX

1. **Not Plain JavaScript**
   * JSX looks like HTML, but it’s not plain JavaScript.
   * Browsers can only understand JavaScript, so JSX **must be converted** before execution.
2. **How JSX Reaches the Browser**
   * Browsers **cannot read JSX directly**.
   * Tools like **Parcel (bundler)** and **Babel (transpiler)** work together to turn JSX into browser-readable JavaScript.
3. **Role of Parcel vs Babel**
   * **Parcel** - Acts as the **bundler** (packages everything).
   * **Babel** - The actual **transpiler** that converts JSX → JavaScript.
   * Parcel just **delegates transpilation** to Babel.
4. **Dependency Handling**
   * Parcel comes with **Babel as a transitive dependency.**
   * Meaning you don’t install Babel separately; Parcel already pulls it in behind the scenes to handle JSX conversion.

What is Babel?

Babel is the tool that makes modern JS + JSX compatible with older browsers by transpiling it into plain JavaScript that browsers can understand.

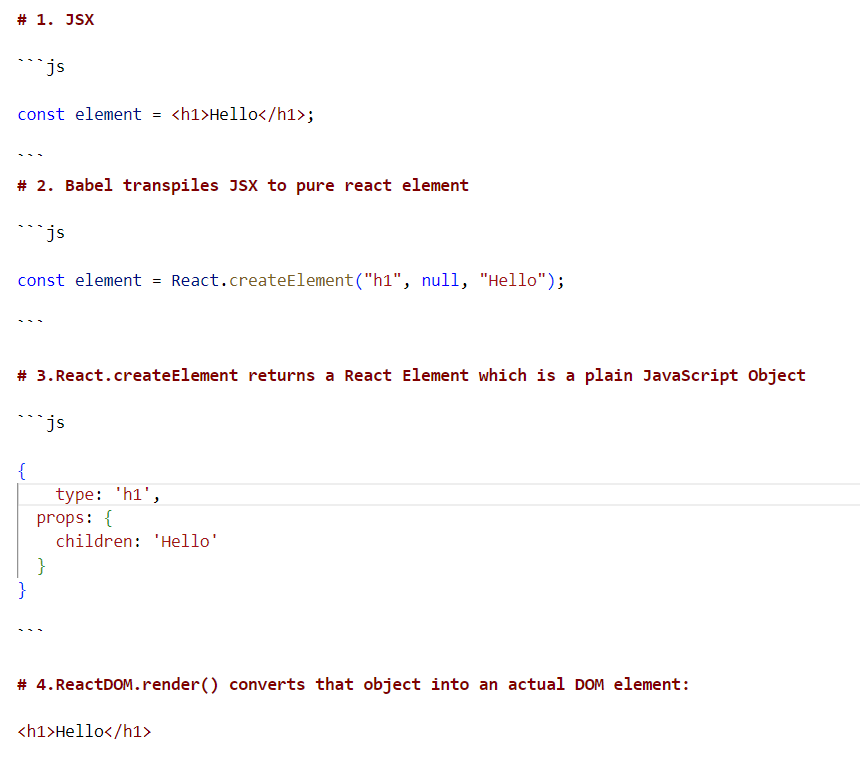
At its core, react code is just JavaScript, and it can even be loaded directly into a project using CDN links.

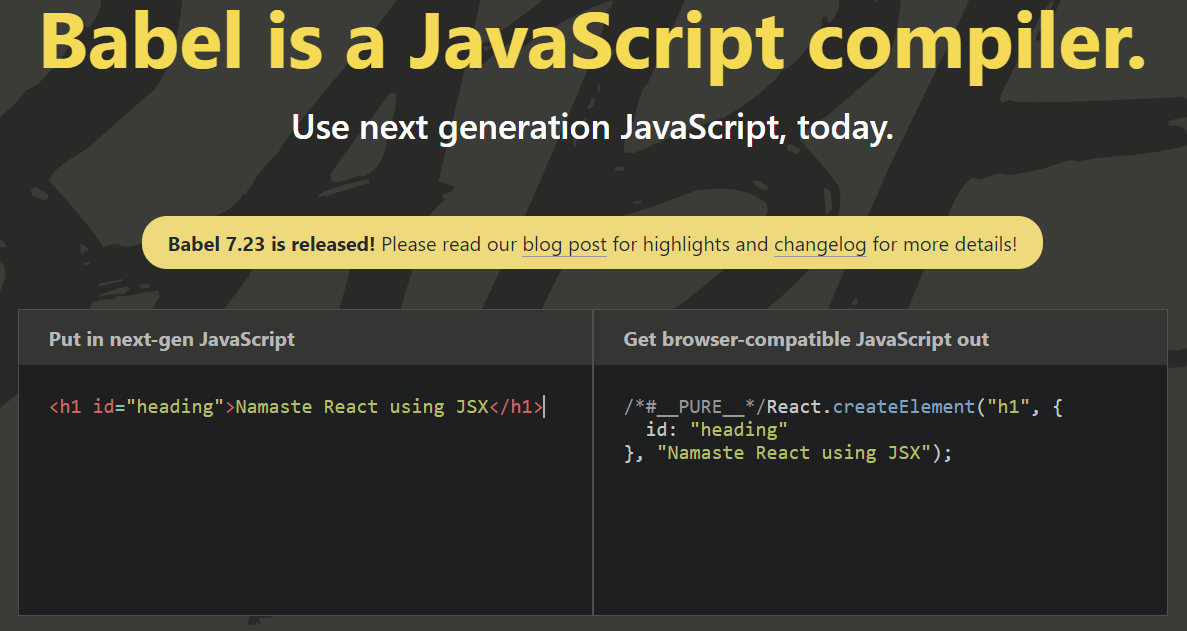
### Why Do We Need Babel?

* **Browsers don’t understand JSX** - Browsers can only read plain JavaScript, not JSX syntax.
* **JSX is just syntactic sugar** - JSX is a cleaner way of writing React.createElement() calls.
* **Babel transpiles JSX → JavaScript → HTML (via ReactDOM) -** Babel converts JSX into browser-compatible JavaScript, which ReactDOM then renders into actual HTML elements.

### JSX to HTML: The Flow

Here's a simplified step-by-step process of what happens





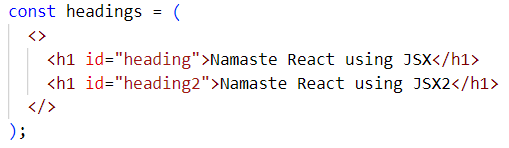
Note: Parcel does not do the transpiling itself. It delegates the job to Babel (via a transitive dependency).

### JSX Syntax Tip -

✅ **Single JSX expression -** No need for parentheses.

const element = <h1>Hello</h1>;

✅ **Multiple JSX elements -** Wrap them in a <div> or React.Fragment, and enclose in parentheses:



## React Components -

In React, there are two main types of components:

**Class-based Components:** The traditional way of writing components in React.

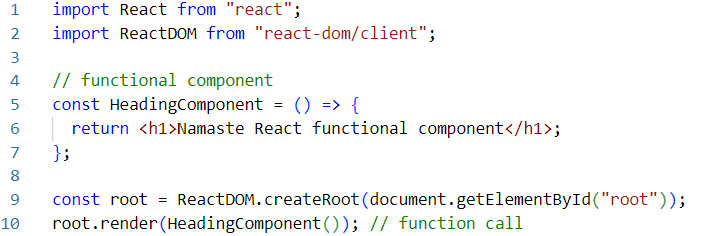
**Functional Components:** The modern and preferred way of writing components in React.

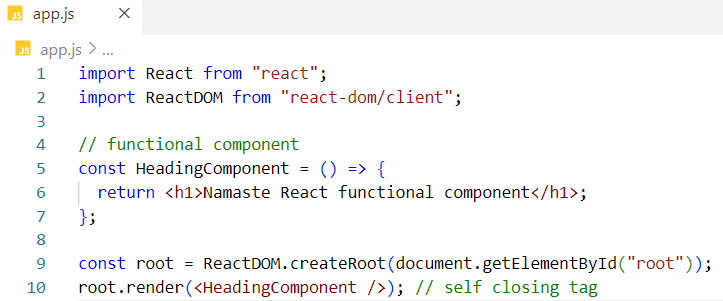
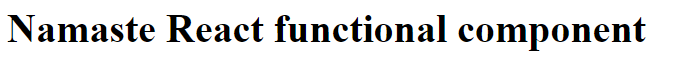
React Functional Component -

A React functional component is simply a standard JavaScript function that returns JSX or a React element.

The naming convention for functional components follows **PascalCase**.

We can render a functional component either by **invoking it as a JSX element** using a self-closing tag like <FunctionName />, or (less commonly) by calling the function directly.

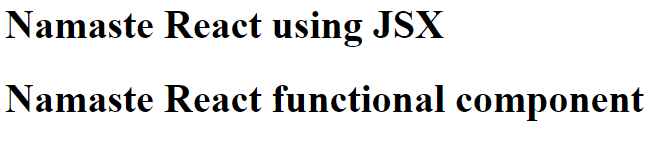




Component Composition in React-

Component composition means **building complex UIs by combining smaller, reusable components inside other components.**

In the example below, the Title functional component is used inside the HeadingComponent functional component.

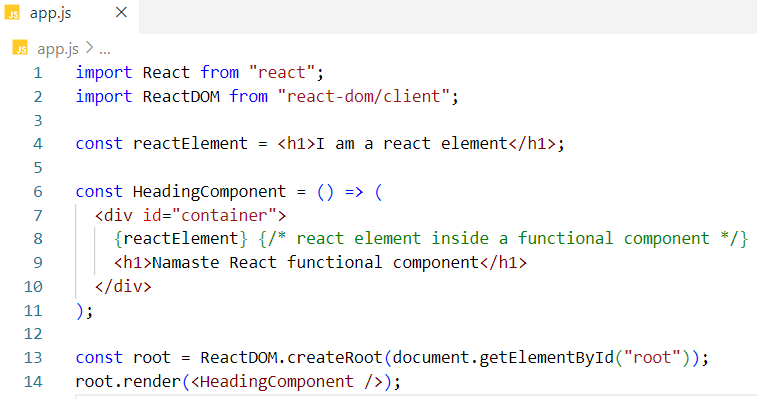
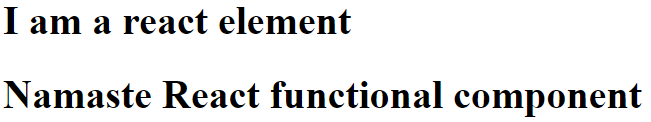


When including JavaScript expressions inside JSX, we wrap them in curly braces. For example: {1 + 2} or {console.log("Hello")}.



The result will be the same.

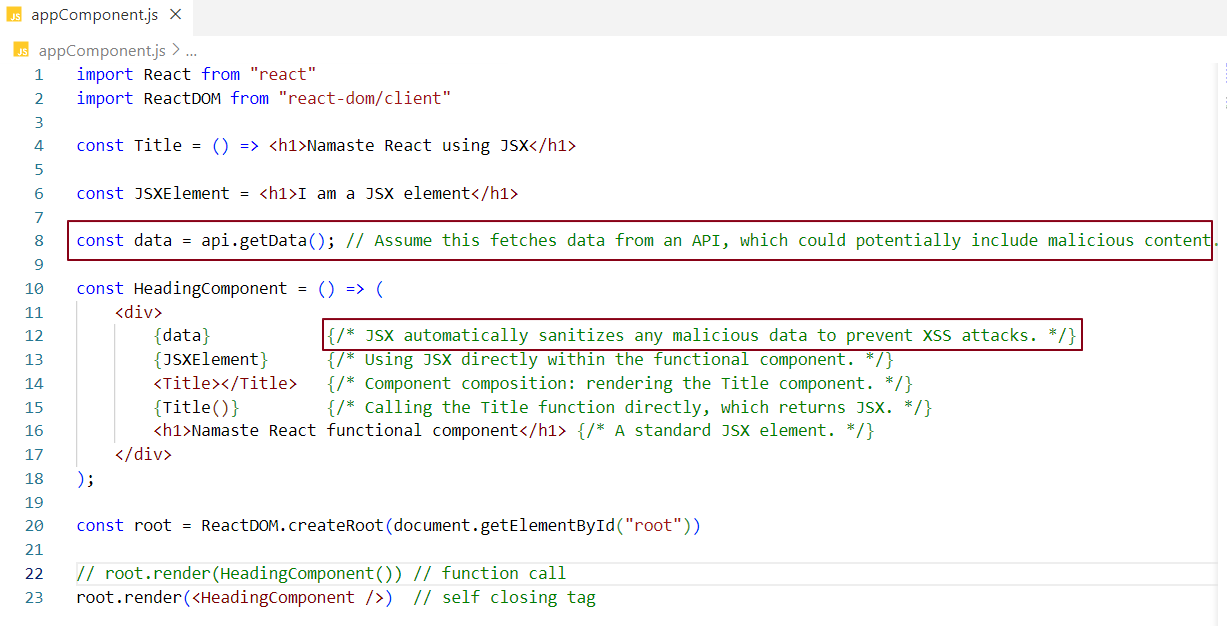
Using JSX Within a Functional Component -



**JSX & XSS -**

**JSX helps protect against XSS (Cross-Site Scripting) attacks by automatically escaping any data before it is added to the component.**

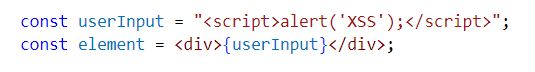
This ensures that dangerous content, such as script tags or malicious code, is neutralized and cannot be executed, making the application safer.



React sanitizes data to prevent XSS (Cross-Site Scripting) attacks through a combination of practices and built-in mechanisms. Here's how React handles sanitization:

**Automatic Escaping:** When you use curly braces {} in JSX to include dynamic data, react automatically escapes potentially harmful characters. This means that characters like <, >, and & are converted into their respective HTML entities (&lt; &gt; and &amp;).

For example:



React will render the above as:



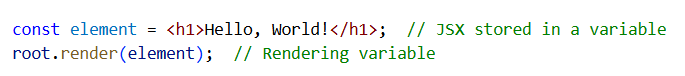
Instead of executing the script, react safely displays the text literally, preventing XSS attacks.

**Note:** We can also use arrow functions to create function components in React.

Can we render JSX directly inside root.render() in React?

Yes, we can render JSX directly in root.render() without storing it in a variable. React accepts JSX, component references, or React elements directly as arguments.

### *Example 1 (Using a variable) -*



### *Example 2 (Direct JSX) -*



Both will produce the same output on the browser: **Hello, World!**