Chapter 3 – Laying the foundation

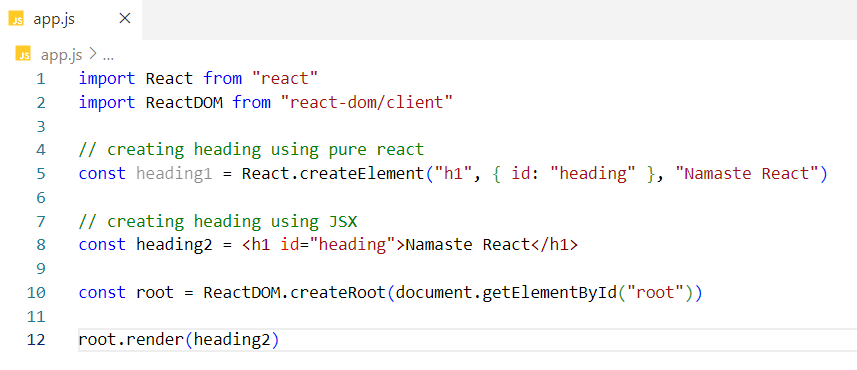
In the previous chapter, we used the command npx parcel index.html to start our app, and npx parcel build index.html to build it. Now, let's configure these commands in the package.json file so we can run them using npm.



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 using JSX



heading1 and heading2 are JavaScript objects when we log them 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 (JavaScript XML)** is a syntax extension for JavaScript that allows developers to write HTML-like code directly within their JavaScript files. It simplifies the process of creating React elements and components, making the code more readable and intuitive.

Common Misconceptions about JSX

JSX is part of React - This is a misconception. JSX is not a mandatory part of React; developers can write React code without using JSX. While JSX simplifies the process of creating React applications, it is not required for development.

JSX is HTML inside JavaScript - This is also incorrect. JSX resembles HTML, but it is not HTML. Rather, it is an XML-like syntax that allows for easier creation of React elements and components.

### 🔁 Difference between HTML and JSX

**Expressions and Functions:** JSX allows you to include JavaScript expressions and functions within its syntax using curly braces {}. HTML, on the other hand, only permits static content.

**Transpilation:** JSX must be transpiled into regular JavaScript using tools like Babel. HTML does not require this process.

**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 Points on JSX

**Understanding JSX:** JSX is not plain JavaScript. While JavaScript is understood by the browser’s engine, JSX needs to be converted into JavaScript before the browser can interpret it.

**How JSX is Rendered in the Browser:** Browsers cannot understand JSX directly. Tools like **Parcel** act as bundlers that help convert (transpile) JSX into browser-readable JavaScript using a transpiler.

**Role of Parcel and Babel:** Parcel does not perform transpilation itself—it delegates this responsibility to **Babel**, which is the actual transpiler. Parcel simply manages the process.

**Dependency**: Parcel includes Babel as a **transitive dependency**, meaning it depends on Babel indirectly to handle JSX transpilation behind the scenes.

What is Babel?

**Babel** is a JavaScript toolchain used to **transpile** modern JavaScript code—including **JSX**—into code that browsers can understand.

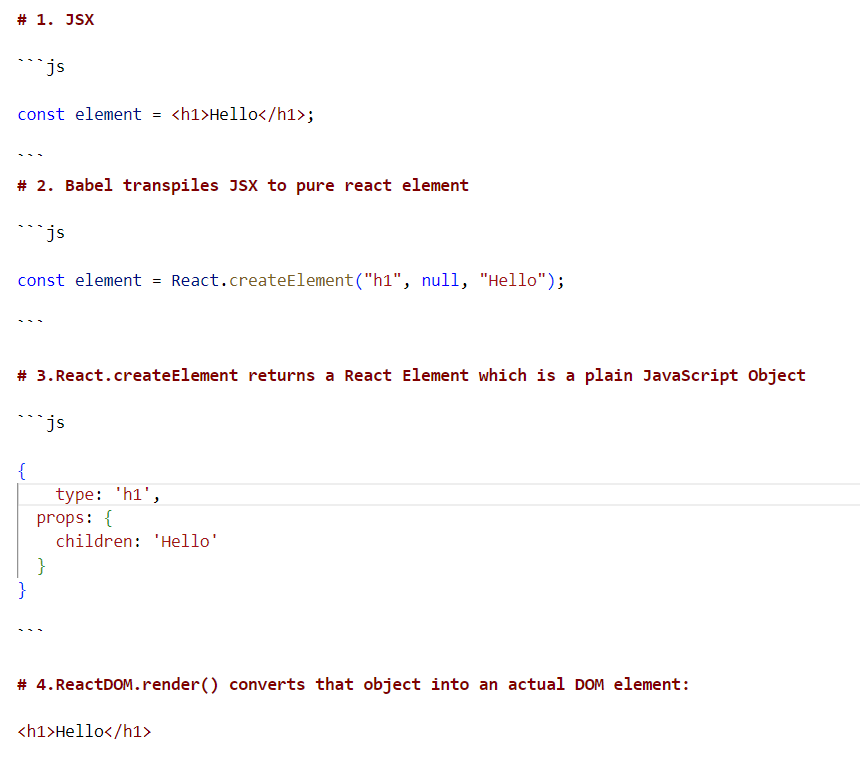
At its core, react code is simply JavaScript, and it can be loaded using CDN links.

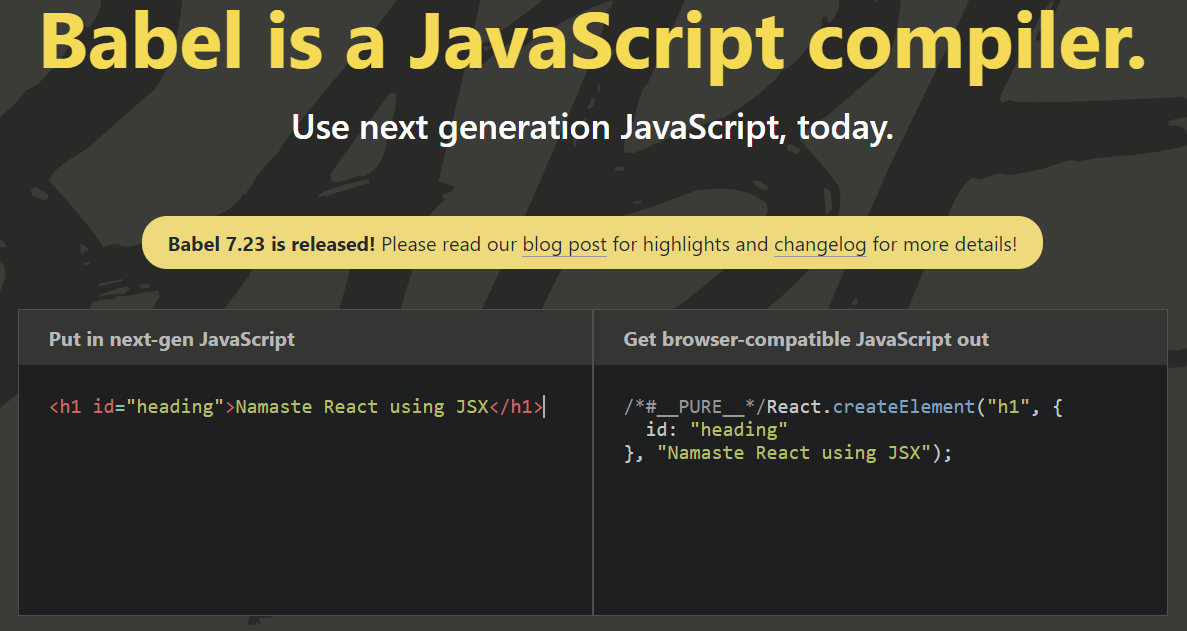
### 🧠 Why do we need Babel?

* Browsers do **not** understand JSX.
* JSX is **syntactic sugar** for React.createElement() calls.
* Babel **converts** (transpiles) JSX → JavaScript → HTML (via ReactDOM).

### 🔄 JSX to HTML: The Flow

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





### ⚙️ Tools Involved

* **Babel** → Transpiles JSX to JavaScript.
* **React.createElement** → Converts into a React element.
* **ReactDOM.render** → Renders it to the DOM.
* **Parcel/Webpack** → Uses Babel under the hood to automate this process.

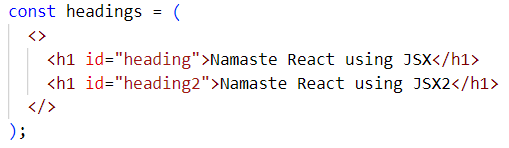
💡 Note: Parcel does not do the transpiling itself. It delegates the job to Babel (via a transitive dependency).

### ✅ 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

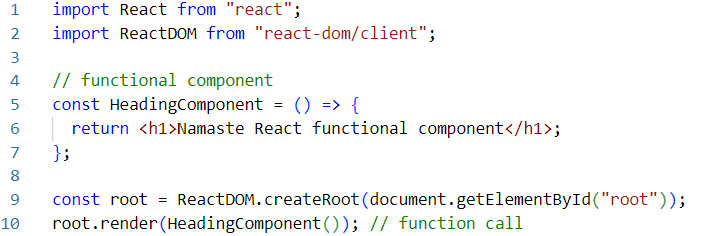
In React, there are two main types of components:

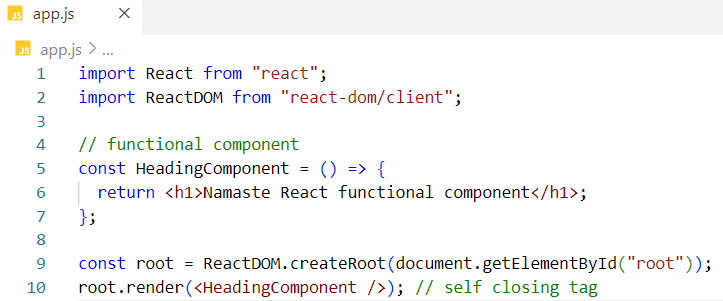
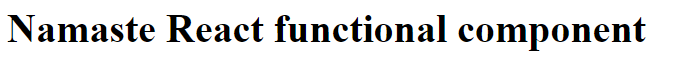
**Class-based Components**: The traditional way of writing components in React.

**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.

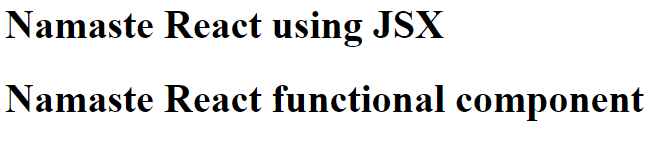




Component Composition (Using a Component Inside Another Component)

**Component Composition** is the technique of building complex UIs by combining smaller, reusable React components.

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

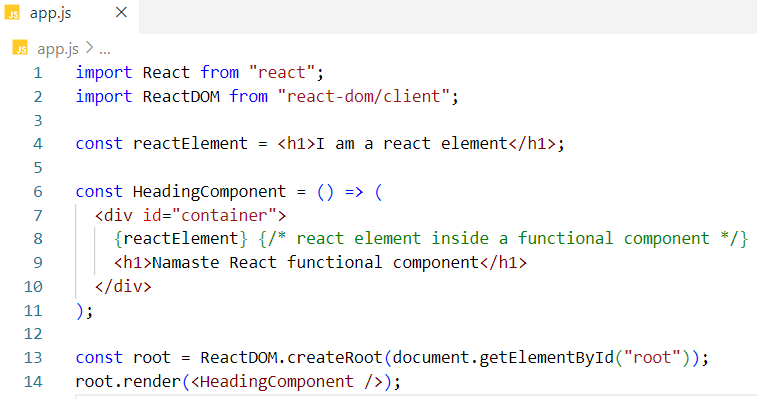
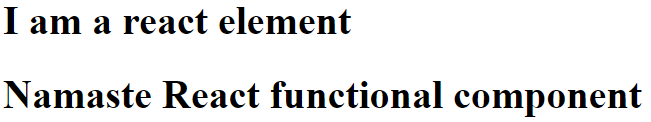


When including JavaScript expressions inside JSX, we wrap them in curly braces. For example: {1 + 2} or {console.log("Hello")}.



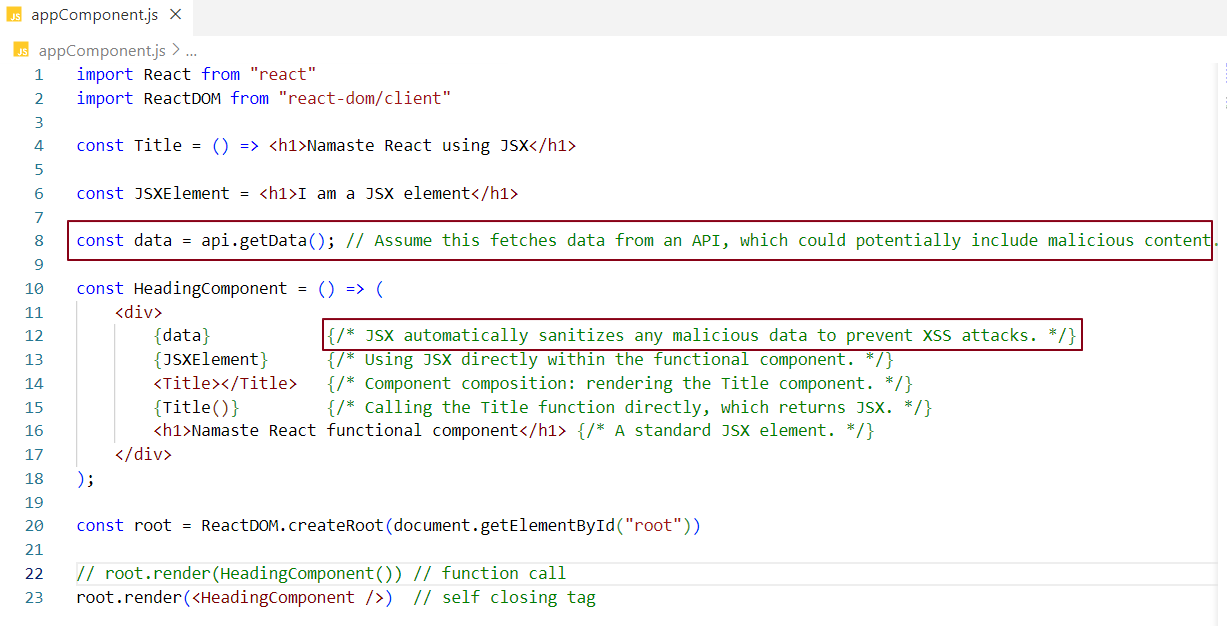
The result will be the same.

Using JSX Within a Functional Component



**JSX and XSS**

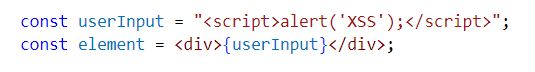
**JSX helps protect against XSS (Cross-Site Scripting) attacks by automatically escaping any data before it is added to the component.** This ensures that dangerous content, such as script tags or malicious code, is neutralized and cannot be executed, making the application safer.



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.