Chapter 3 – Laying the foundation

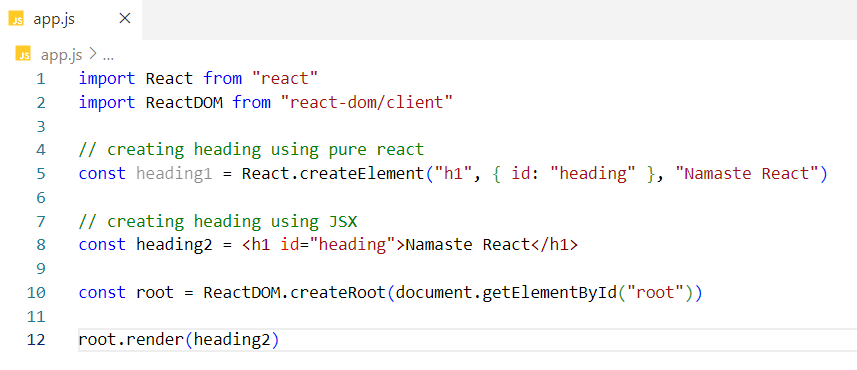
In the previous chapter, we used the command npx parcel index.html to start our app, and we used npx parcel build index.html to build it. Now, let’s execute these commands using npm by configuring the scripts in the package.json file.



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

When representing deeply nested elements in the DOM, extensive use of React's createElement API can result in code that is hard to read. To address this issue, Facebook developers introduced JSX, which enhances code readability and clarity.

Below is an example of a React element created using pure React code, along with the same element created using JSX.



heading1 and heading2 are JavaScript objects when we log them to the console. In JSX syntax, it may appear that we are writing HTML within JavaScript, but it is important to note that this is not HTML; it is JSX, which resembles HTML. There is a distinction between HTML syntax and HTML-like syntax.

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 us to include JavaScript expressions and functions within its syntax using curly braces {}, whereas HTML only permits static text.

**Transpilation:** JSX is transpiled into JavaScript by tools like Babel, while HTML is not subjected to the same process.

**Attribute Naming:** In HTML, we can use class as an attribute to define CSS classes. However, in JSX, we cannot use class due to conflicts with the JavaScript class keyword; instead, we use className.

More Points on JSX

**\*** Understanding JSX**:** JavaScript is the language that the JavaScript engine understands, while browsers do not directly understand JSX. Therefore, it is important to note that JSX is not pure JavaScript.

**\*** How JSX is Rendered in the Browser: So, how is JSX content displayed in the browser if the JavaScript engine does not understand it? The answer lies in the use of a bundler. Tools like Parcel handle this process behind the scenes. Before the application code reaches the JavaScript engine, it is transpiled or converted into standard JavaScript code. This transpiled code is then processed by the JavaScript engine, which the browser can understand.

**\*** Role of Parcel and Babel: It’s worth mentioning that Parcel itself does not perform the transpilation. Instead, it acts like a manager that delegates the task to its employee—Babel, the transpiler.

**\*** Dependency: The Parcel package has a transitive dependency on the Babel package, allowing it to leverage Babel's capabilities for transpiling JSX into JavaScript.

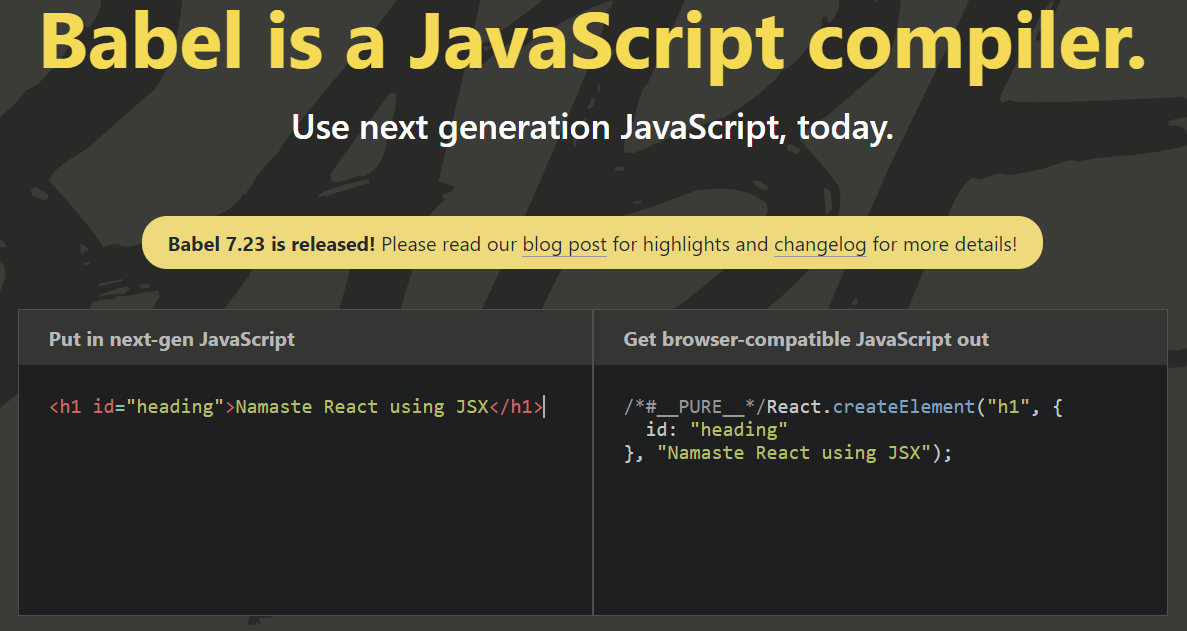
What is Babel?

Babel is a JavaScript toolchain primarily used to transpile (convert) JSX code into standard JavaScript code that can be understood by browsers. At its core, React code is ultimately JavaScript code, which can be loaded via CDN links.

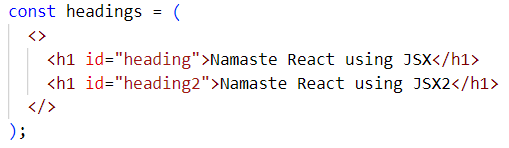
Here's a simplified flow of how JSX is converted into React code -

1.JSX → 2. React.createElement → 3. React Element → 4. JavaScript Object → 5. HTML Element (rendered)

In this process, Babel transforms JSX syntax into calls to React.createElement, which then creates React elements as JavaScript objects that the browser can render as HTML elements.



If it's a single JSX expression, wrapping the JSX in parentheses is optional. However, when expecting multiple JSX elements, we should wrap them in a <div> tag or a React fragment, and enclose them in parentheses.



We will discuss React fragments later.

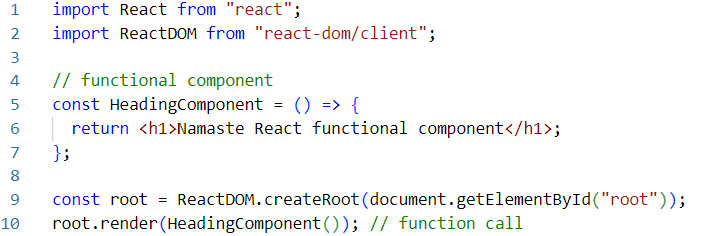
React Components –

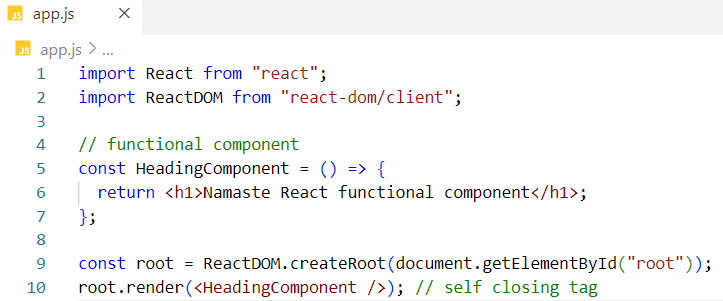
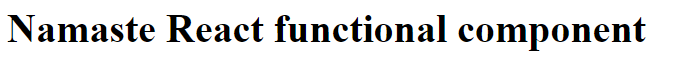
In React, there are two types of components:

1. Class-based Components: These represent the traditional way of writing components in React. We will cover these in more detail later.
2. Functional Components: This is the modern approach to writing components in React.

React Functional Component -

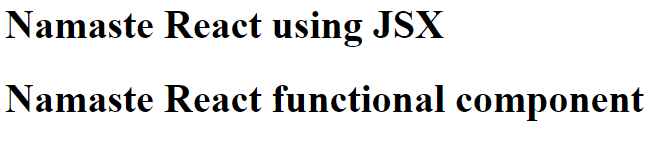
A React functional component is simply a standard JavaScript function that returns JSX or a React element. The naming convention for functional components follows PascalCase. We can render a functional component either by calling the function directly or by using its name with a self-closing tag, like this: <FunctionName />.





Component inside another component – (Component composition)

In the example below, the Title functional component is called within the HeadingComponent functional component.

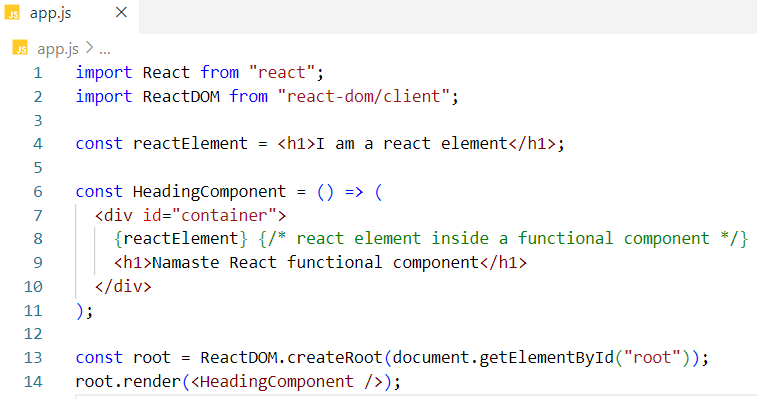
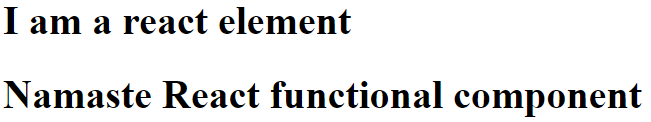


When we include JavaScript expressions inside JSX, we enclose them in curly braces. For example: {1 + 2} or {console.log("Hello")}.



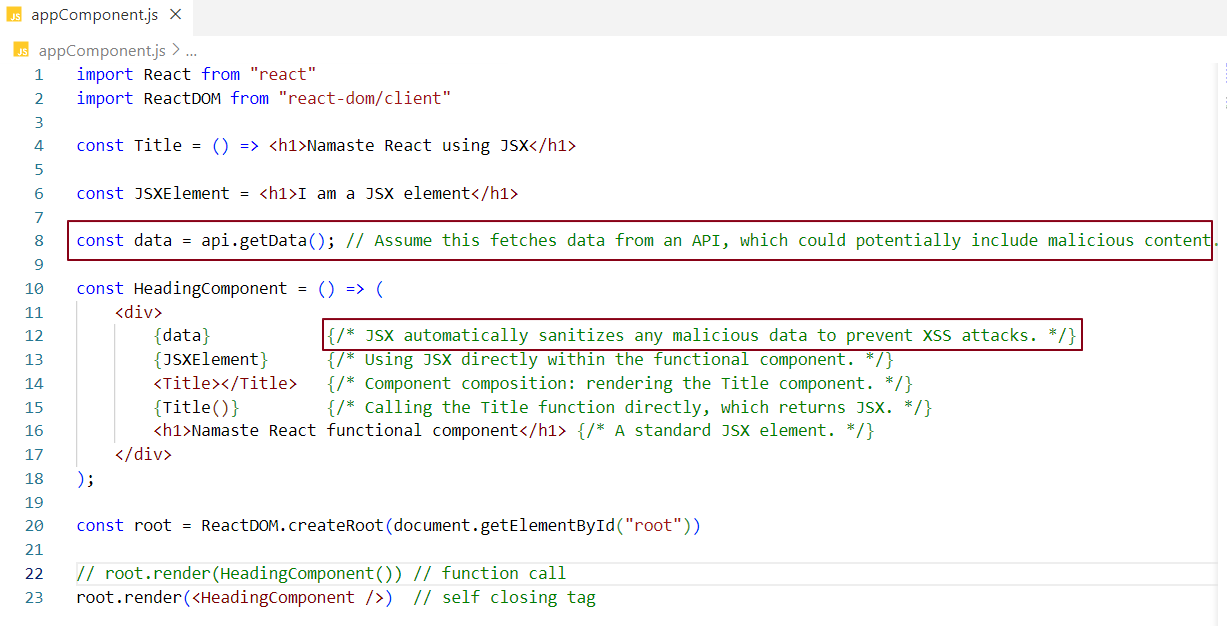
We will achieve the same result.

Using JSX inside a functional component -



**JSX and XSS**

JSX helps protect against XSS (Cross-Site Scripting) attacks by checking the data before it gets added to the component. This means it automatically removes any dangerous code, making your 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 any potentially harmful characters. This means that characters like <, >, and & are converted to their respective HTML entities (&lt;, &gt;, and &amp;). For example:

const userInput = "<script>alert('XSS');</script>";

const element = <div>{userInput}</div>;

This will render as <div>&lt; script&gt; alert('XSS'); &lt;/script&gt;</div>

So instead of executing the script, it displays the text literally.

Note - We can also use arrow functions to create function components.