Chapter 3 – Laying the foundation

As we have seen in previous chapter, we used a command npx parcel index.html to ignite our app and to build our app we used npx parcel build index.html. Let's execute these commands using npm. To do that we configure the scripts in package.json file.

package.json

```
{
  "name": "chapter---2---igniting-our-app",
  "version": "1.0.0",
  "description": "Learning Namaste React from Akshay Saini",
  "scripts": {
    "start":"parcel index.html",
    "build":"parcel build index.html",
    "test": "jest"
  },
  "author": "Amar",
  "license": "ISC",
  "devDependencies": {
    "parcel": "^2.10.3",
    "process": "^0.11.10"
  },
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0"
  }
}
```

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

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

While representing deep nested elements in the DOM, heavy use of React's createElement API made the code unreadable. To overcome this issue JSX is introduced by Facebook developers. JSX makes the code clean and readable.

Below is example where a react element is created using pure react code and the same element is created using JSX.

```
// creating heading using pure react
const heading1 = React.createElement("h1", { id: "heading" }, "Namaste React");

// creating heading using JSX
const heading2 = <h1 id="heading">Namaste React using JSX</h1>
```

In JSX syntax, it looks like we have written html inside JavaScript. No, this is not html. This is JSX which looks like html. There is a difference between html syntax and html like syntax.

What is JSX?

JSX is a JavaScript syntax which is easier to create react element.

Common misconceptions of JSX

* JSX is part of react – No, JSX is not a part of react, we can write react code without using JSX. We can develop react applications without using JSX. JSX makes developers life easy.

* JSX is html inside JavaScript. No. JSX is html / XML like syntax.

Difference between html and JSX

* JSX allows us to include expressions and functions within the syntax, while HTML only allows static text in its syntax.

* JSX is transpiled to JavaScript while HTML is not.

* We can use class as an attribute in html. We can't use class as attribute in JSX.in JSX we have className.

As we move ahead, we will see more dissimilarities.

More points on JSX

JavaScript is a code that JS engine understands. Browser does not understand JSX. Bottom line is JSX is not pure JavaScript. So,

How JSX content is displayed in the browser if JS engine does not understand it?

Answer is bundler. Parcel is doing the Job behind the scene. Even before the application code goes into JS engine, it is transpiled or converted into JS code. Then the JS code is taken by JS engine which browser understands.

Parcel does not transpile the code itself. It's like a manager who orders his employee to do the job and that employee is babel.  . Babel is the transpiler.

Parcel package has transitive depenancy (discussed in previous chapter) on babel package.

What is babel?

It's a package whose job is to transpile JSX code into React code which is understandable by browser. React code in the end of the day is JavaScript code. (discussed before)

Below is an example of JSX getting converted into pure react code with the help of babel.

JSX => React.createElement => React element => JavaScript object => html element (render)

If it's a single JSX expression, wrapping JSX inside parenthesis is optional, but if we expect multiple JSX elements, then we wrap them under a div tag or a react fragment enclosed by parenthesis.

```
const headings = (
  <>
    <h1 id="heading">Namaste React using JSX</h1>
    <h1 id="heading2">Namaste React using JSX2</h1>
  </>
);
```

We will discuss about the react fragment later.

React Components –

There are two types of components in react

1.Class based component (old way of writing code) - We will learn later.

2. Functional Component (New way of writing code)

React Functional Component –

React functional component is just a normal JavaScript function which returns some JSX or react element. Naming convention used in functional component is pascal case. We render functional component either by a function call or by function name with a self-closing tag <function Name/>.

```
1   import React from "react";
2   import ReactDOM from "react-dom/client";
3
4   // functional component
5   const HeadingComponent = () => {
6     return <h1>Namaste React functional component</h1>;
7   };
8
9   const root = ReactDOM.createRoot(document.getElementById("root"));
10  root.render(HeadingComponent()); // function call
```

app.js

app.js > ...

```
1  import React from "react";
2  import ReactDOM from "react-dom/client";
3
4  // functional component
5  const HeadingComponent = () => {
6    return <h1>Namaste React functional component</h1>;
7  };
8
9  const root = ReactDOM.createRoot(document.getElementById("root"));
10 root.render(<HeadingComponent />); // self closing tag
```

Namaste React functional component

Component inside another component – (Component composition)

In below example Title functional component is called inside HeadingComponent functional component.

app.js

app.js > ...

```
1  import React from "react";
2  import ReactDOM from "react-dom/client";
3
4  const Title = () => <h1 className="head">Namaste React using JSX</h1>;
5
6  const HeadingComponent = () => (
7    <div id="container">
8      <Title />
9      <h1>Namaste React functional component</h1>
10   </div>
11 );
12
13 const root = ReactDOM.createRoot(document.getElementById("root"));
14 root.render(<HeadingComponent />);
```

Namaste React using JSX

Namaste React functional component

When we write JavaScript expression inside JSX we enclose them within curly braces. examples – {1+2}, {console.log(“Hello”)}

app.js

app.js > ...

```
1  import React from "react";
2  import ReactDOM from "react-dom/client";
3
4  const Title = () => <h1 className="head">Namaste React using JSX</h1>;
5
6  const HeadingComponent = () => (
7    <div id="container">
8      {Title()} {/* javaScript function call */}
9      <h1>Namaste React functional component</h1>
10   </div>
11 );
12
13 const root = ReactDOM.createRoot(document.getElementById("root"));
14 root.render(<HeadingComponent />);
```

We will get the same result.

React element inside a functional component -

app.js

```
app.js > ...
1   import React from "react";
2   import ReactDOM from "react-dom/client";
3
4   const reactElement = <h1>I am a react element</h1>;
5
6   const HeadingComponent = () => (
7     <div id="container">
8       {reactElement} {/* react element inside a functional component */}
9       <h1>Namaste React functional component</h1>
10    </div>
11  );
12
13  const root = ReactDOM.createRoot(document.getElementById("root"));
14  root.render(<HeadingComponent />);
```

I am a react element

Namaste React functional component

Notes –

* JSX takes care of XSS attack. It does sanity check before the data is injected into the component.

* Inside JSX, we can call a component like

    <componentName></componentName>

    {componentName ()}

    < componentName />

* npx = npm nun