##

React JS By Prasad

##

File Structure:

Once you created react app, you will get lot of files, let’s understand one by one.

Now, let's explain the purpose of each key file and folder:

node_modules/: This folder contains all the dependencies your app requires. It's usually managed by npm (Node Package Manager) or Yarn.

public/: This directory contains static assets that are directly copied to the build output. The index.html file here serves as the main entry point for your app.

index.html: The HTML template that acts as the root container for your React app. This is where your app's content will be injected by React.

src/: This is where the majority of your source code resides.

index.js: The entry point of your application. It typically renders the main React component and attaches it to the DOM.

App.js: The main component of your application. This is where you structure the overall layout of your app and include other components.

package.json: The configuration file that holds metadata about your project, including dependencies, scripts, and project information.

package-lock.json: This file is automatically generated and used to lock down the specific versions of dependencies that were installed, ensuring consistency across different environments.

README.md: A documentation file that provides information about your project, its setup, and usage.

This structure is suitable for a basic React app. As your app grows, you might introduce additional folders for features, routes, state management, and more, depending on the complexity of your project and your preferred project organization.

JSX:

JSX, which stands for JavaScript XML, is a syntax extension for JavaScript often used with libraries like React for building user interfaces. It allows developers to write HTML-like code directly within JavaScript code, making it easier to create and manipulate the structure of user interfaces.

Here's a comprehensive explanation of JSX:

Embedding HTML in JavaScript: JSX enables you to write HTML elements and components directly in your JavaScript code. This is particularly useful when building user interfaces, as it allows you to describe the structure of your UI using familiar HTML-like syntax.

```
const element = <h1>Hello, JSX!</h1>;
```

Components: In JSX, you can define your own custom components. Components are like building blocks for your UI. They encapsulate functionality and can be reused throughout your application.

```
function Greeting(props) {
  return <h1>Hello, {props.name}!</h1>;
}
```

```
const element = <Greeting name="John" />;
```

Expressions: You can embed JavaScript expressions within curly braces {} in JSX. This allows you to dynamically generate content based on variables or calculations.

```
const name = "Alice";

const element = <p>Hello, {name}!</p>;
```

Attributes and Props: JSX elements can have attributes just like HTML elements. These attributes are passed as props to the corresponding React components.

```
const element = <input type="text" placeholder="Enter your name" />;
```

Conditional Rendering: JSX allows you to conditionally render elements based on certain conditions using JavaScript's if statements or ternary operators.

```
const isLoggedIn = true;

const element = isLoggedIn ? <p>Welcome back!</p> : <p>Please log in.</p>;
```

Mapping Arrays to JSX: You can map over arrays of data and render JSX elements dynamically.

```
const numbers = [1, 2, 3, 4, 5];

const listItems = numbers.map(number => <li key={number}>{number}</li>);

const element = <ul>{listItems}</ul>;
```

Styling: While inline styles can be applied to JSX elements, it's more common to use CSS classes and classnames to style components.

```
const element = <div className="my-container">Styled using a CSS class</div>;
```

Nesting and Composition: JSX elements can be nested just like HTML elements, allowing you to create complex UI structures through composition.

```
const element = (
  <div>
    <h1>Title</h1>
    <p>Content goes here.</p>
  </div>
);
```

Event Handling: JSX allows you to attach event handlers to elements in a way similar to HTML.

```
function handleClick() {
  console.log('Button clicked!');
}

const element = <button onClick={handleClick}>Click me</button>;
```

Fragment: When you want to return multiple JSX elements from a component without introducing an extra wrapping element, you can use the React.Fragment or the shorthand <> ... </> syntax.

```
const element = (
  <>
```

```
    <p>Paragraph 1</p>
    <p>Paragraph 2</p>
  </>
);
```

JSX is not understood by browsers directly. It needs to be transpiled (converted) into regular JavaScript code using tools like Babel before it can be executed in a browser environment. This transpilation step is necessary to ensure compatibility with older browsers and to convert JSX syntax into JavaScript function calls that create and manipulate UI elements.