React.js – Complete Beginner-Friendly Guide (Step by Step)

---

1. What is React and Why Was It Introduced?

React is a **JavaScript library** used for building **user interfaces**, especially **single-page applications**. It was created by Facebook to solve problems related to managing complex and dynamic user interfaces.

Before React, developers used traditional multi-page websites where every user action caused a full page reload. This made applications slow and user experience poor.

React was introduced to: - Build fast and interactive UIs - Update only the required parts of the screen - Manage UI state efficiently

React uses a component-based architecture, which makes applications scalable and maintainable.

👉**Interview Line:**

> “React is a JavaScript library for building reusable and interactive user interfaces using components.”

---

2. What is a Component and Why Do We Need It?

In React, a **component** is the basic building block of the application. A component is simply a **JavaScript function** that returns UI (JSX). Instead of writing one large file with all HTML and logic mixed together, React encourages us to divide the UI into small, independent, reusable pieces called components.

Think of a component as a machine. It takes input (props), processes logic, and produces output (UI). Each component is responsible for **one specific part of the screen**.

Before components, developers reused code by copying and pasting HTML and JavaScript. This caused duplication, bugs, and maintenance issues. Components solve this problem by allowing reuse and separation of concerns.

If components did not exist: - Code would be very large and confusing - Reusability would be difficult - Updating UI would be risky and error-prone

Example Component

```
function Header() {
  return (
    <header>
```

```
      <h1>Welcome to My Website</h1>
    </header>
  );
}

export default Header;
```

👉**Interview Line:**

> “A component in React is a reusable JavaScript function that returns JSX and represents a part of the user interface.”

---

3. What is JSX?

JSX is a syntax extension that looks like HTML but runs inside JavaScript.

Example:

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

Browsers do not understand JSX directly. That is why we need **build tools**.

---

4. Why Do We Need Build Tools in React?

Build tools help us:

1. Convert JSX into JavaScript
2. Convert modern JavaScript into browser-compatible code
3. Bundle multiple files into one
4. Optimize performance
5. Provide fast development server

Without build tools, React development is not practical.

---

5. Creating React App Using CRA (Create React App)

Steps

```
npx create-react-app my-app
cd my-app
npm start
```

CRA: - Uses Webpack internally - Easy for beginners - Less control over configuration

---

6. Creating React App Using Vite (Recommended)

Steps

```
npm create vite@latest my-vite-app --template react
cd my-vite-app
npm install
npm run dev
```

Why Vite?

- Very fast
- Minimal configuration
- Modern development experience

---

7. Creating React App Using Webpack (Manual Setup)

Webpack gives full control but requires configuration.

Basic Steps

1. Initialize project

   ```
   npm init -y
   ```

2. Install dependencies

   ```
   npm install react react-dom
   npm install -D webpack webpack-cli webpack-dev-server
   npm install -D babel-loader @babel/core @babel/preset-env @babel/preset-react
   ```

3. Create files

4. index.html
5. index.js
6. App.js
7. webpack.config.js

Webpack is best for advanced users.

---

8. Project Structure (Vite Example)

```
src/
 ├── main.jsx
 ├── App.jsx
 ├── components/
 │    └── Header.jsx
```

---

9. What is App.js and Why Do We Need It?

App.js (or App.jsx) is the **root component** of the application.

- All other components are added inside App
- It represents the main UI structure

Example:

```
import Header from './components/Header';

function App() {
  return (
    <div>
      <Header />
      <h2>Main Content</h2>
    </div>
  );
}

export default App;
```

---

10. Root Element in React

The **root element** is the single HTML element where the entire React application is injected. Usually, this element is a `<div>` with an id of `root` inside `index.html`.

React does not control the entire HTML page. Instead, it controls only what is inside the root element.

Example

```
<div id="root"></div>
```

```
ReactDOM.createRoot(document.getElementById('root')).render(
  <App />
);
```

If there is no root element: - React cannot render the application - The app will not appear in the browser

👉**Interview Line:**

> “The root element is the container where React renders the entire application UI.”

---

11. Creating and Using Components

Create Component

```
function Card() {
  return <p>This is a card</p>;
}

export default Card;
```

Import and Use

```
import Card from './Card';

<Card />
```

---

12. What are Props and Why We Need Them?

Props (short for **properties**) are used in React to **pass data from a parent component to a child component**. Props allow components to be dynamic instead of hard-coded.

Without props, components would always display the same data and would not be reusable. Props make components flexible and configurable.

Props follow a **one-way data flow**, meaning data can only flow from parent to child. A child component cannot modify the props it receives.

Example Using Props

```
function User({ name, age }) {
  return (
    <p>
```

```
      {name} is {age} years old
    </p>
  );
}
```

```
<User name="Alice" age={25} />
<User name="Bob" age={30} />
```

Here, the same component is reused with different data.

👉**Interview Line:**

> “Props are used to pass data from parent to child components, making components reusable and dynamic.”

---

13. Iterating Arrays in React (map)

```
const fruits = ['Apple', 'Banana', 'Orange'];

function FruitList() {
  return (
    <ul>
      {fruits.map((fruit) => (
        <li key={fruit}>{fruit}</li>
      ))}
    </ul>
  );
}
```

---

14. What is Key and Why Is It Important?

In React, **key** is a special attribute that is used when we render a list of elements using the `map()` function. The key helps React **identify which items have changed, been added, or been removed** from the list.

When React updates the UI, it does not re-render the entire page. Instead, it compares the previous UI with the new UI using a process called **reconciliation**. During this comparison, React needs a reliable way to uniquely identify each element in a list. That is exactly what the `key` provides.

If we do not use a key, or if we use an incorrect key, React cannot correctly track list items. As a result: - React may re-render unnecessary elements - Input fields may lose focus - Performance will be reduced - UI bugs may appear

Example WITHOUT key (Wrong)

```
function UserList() {
  const users = ['Alice', 'Bob', 'Charlie'];

  return (
    <ul>
      {users.map((user) => (
        <li>{user}</li>
      ))}
    </ul>
  );
}
```

In this case, React does not know which `<li>` belongs to which user when the list changes.

Example WITH key (Correct)

```
function UserList() {
  const users = ['Alice', 'Bob', 'Charlie'];

  return (
    <ul>
      {users.map((user) => (
        <li key={user}>{user}</li>
      ))}
    </ul>
  );
}
```

Here, each list item has a **unique identity**. React can now efficiently update only the changed items.

👉Important Interview Line:

> “Key helps React uniquely identify elements in a list so that it can update the DOM efficiently during re-rendering.”

---

jsx key={index}

~~~
Good practice:
```jsx
key={id}
~~~

---

15. What are Fragments and Why We Need Them?

In React, a component must return **a single parent element**. If we try to return multiple sibling elements directly, React will throw an error. Traditionally, developers used an extra `<div>` to wrap elements, but this created unnecessary nodes in the DOM.

React Fragments solve this problem by allowing us to group multiple elements **without adding extra nodes to the DOM**.

Example WITHOUT Fragment

```
function Profile() {
  return (
    <div>
      <h1>User Profile</h1>
      <p>This is user information</p>
    </div>
  );
}
```

This works, but the extra `<div>` appears in the DOM. If many components use extra divs, the DOM becomes deeply nested, which affects performance and CSS layout.

Example WITH Fragment

```
function Profile() {
  return (
    <>
      <h1>User Profile</h1>
      <p>This is user information</p>
    </>
  );
}
```

With Fragment: - No extra `<div>` is added - DOM remains clean and shallow - Better performance - Easier CSS styling

👉**Important Interview Line:**

> “Fragments allow us to group multiple elements without adding extra nodes to the DOM, keeping the DOM clean and improving performance.”

---

jsx import React from 'react';

function Example() { return (<> <h1>Title</h1> <p>Description</p> </>); }

~~~~
---

## 19. Practice Guide – Step by Step (VERY IMPORTANT)

This section is added specially for **hands-on practice**. Follow these steps
slowly. Do not skip any step. Type the code manually.

---

### Step 1: Create a New React Application (Using Vite)

Open terminal and run:

```bash
npm create vite@latest react-practice --template react
cd react-practice
npm install
npm run dev
~~~~

Now open the project in VS Code.

You will see this structure:

```
src/
 ├── main.jsx
 ├── App.jsx
 └── index.css
```

At this stage, your React app is running successfully.

Step 2: Understand main.jsx (Entry Point)

`main.jsx` is the entry point of the React application. This is where React connects to the HTML file.

```
import ReactDOM from 'react-dom/client';
import App from './App';

ReactDOM.createRoot(document.getElementById('root')).render(
  <App />
);
```

Here: - `root` is the root HTML element - `<App />` is injected into the DOM

You normally **do not write UI here**.

Step 3: Understand App.jsx (Root Component)

`App.jsx` is the **root component** of your application.

```
function App() {
  return (
    <div>
      <h1>Hello React</h1>
    </div>
  );
}

export default App;
```

All other components will be added inside `App.jsx`.

Step 4: Create Your First Component

Create a folder:

```
src/components
```

Create a file:

```
Header.jsx
```

```
function Header() {
  return (
    <header>
      <h2>This is Header Component</h2>
    </header>
  );
}

export default Header;
```

Step 5: Import and Inject Component into App.jsx

Open `App.jsx` and modify:

```
import Header from './components/Header';

function App() {
  return (
    <div>
      <Header />
      <h1>Main Application</h1>
    </div>
  );
}

export default App;
```

Now you have successfully: - Created a component - Exported it - Imported it - Injected it into App.js

---

Step 6: Create Multiple Components

Create two more components:

Footer.jsx

```
function Footer() {
  return <p>Footer Section</p>;
}

export default Footer;
```

Card.jsx

```
function Card() {
  return <div>This is a Card</div>;
}

export default Card;
```

Step 7: Use Multiple Components in App.jsx

```
import Header from './components/Header';
import Footer from './components/Footer';
import Card from './components/Card';

function App() {
  return (
    <div>
      <Header />
      <Card />
      <Card />
      <Card />
      <Footer />
    </div>
  );
}

export default App;
```

This shows **component reusability**.

Step 8: Practice with Props

Update `Card.jsx`:

```
function Card({ title }) {
  return <h3>{title}</h3>;
}

export default Card;
```

Update `App.jsx`:

```
<Card title="React Basics" />
<Card title="Props Practice" />
<Card title="Components" />
```

Step 9: Practice Task (DO THIS)

1. Create `Sidebar.jsx`
2. Pass menu items as props
3. Render list using `map()`
4. Use `key`
5. Wrap elements using Fragment

Step 10: Daily Practice Rule

- Read one section
- Type code manually
- Break code intentionally
- Fix errors
- Explain in your own words

---

20. Final Advice

This document is now both: - **Concept-friendly** - **Practice-friendly**

If you practice everything here: - You will understand React fundamentals deeply - You will speak confidently in interviews - You will not need to re-read basics again

Keep practicing. You are on the right path.