React JS Tutorial for Beginners

Introduction

Are you tired of complex State management or confusing UI updates in web development? React simplifies building fast, interactive user interfaces using reusable Components. Many beginners struggle with the initial setup and understanding JSX and Hooks. This tutorial breaks down these concepts step-by-step, transforming complex tasks into clear, manageable steps. Start your journey to becoming a confident React developer today! To see precisely what you'll master, review our complete **React JS Course Syllabus**.

Why Students or Freshers Learn React JS?

Students or freshers should learn React JS because it offers significant career and technical advantages in modern web development:

- **High Job Demand & Earning Potential:** React developers are highly sought after by startups and tech giants, leading to excellent job opportunities and competitive salaries.
- **Component-Based Architecture:** It encourages building reusable UI components (like buttons or cards), which makes large applications easier to build, maintain, and scale.
- **Improved Performance (Virtual DOM):** React's Virtual DOM efficiently updates only the necessary parts of the webpage, ensuring fast loading times and a smooth user experience.
- **Cross-Platform Capability:** Skills easily transfer to mobile development using React Native, expanding your career scope.
- **Gentle Learning Curve:** While requiring basic JavaScript knowledge, React's concepts like JSX and Hooks are well-documented and designed for a smoother entry into complex front-end work.

Ready to ace your first interview? Download our curated list of **React JS Interview Questions and Answers for beginners**.

Step-by-Step React JS Tutorial for Beginners

This comprehensive tutorial will guide you through the fundamental steps of setting up a React project and building your first dynamic web application using modern functional components and Hooks.

Step 1: Installation and Setup

Before you start building with React, you need to set up your local development environment.

1.1 Install Node.js and npm

React projects are built using tools written in Node.js. You must have Node.js and its package manager, npm (Node Package Manager), installed on your computer.

Check Installation: Open your terminal or command prompt and run:

node -v

npm -v

You are all set if you see version numbers (ex., v18.x.x and 9.x.x). If not download, Node JS from its website.

1.2. Setup a React App

The easiest way to start a new single-page application (SPA) in React is by using a tool like Vite or the older Create React App (CRA). We will use Vite for a modern, faster setup.

Run the creation command: Open your terminal and run the following command to create a new project folder named my-react-app:

npm create vite@latest my-react-app — –template react

Navigate and Install: Change into that directory and install the requirements:

cd my-react-app

npm install

Start the Development Server: Run the command to start the application. It will typically open a browser window at *http://localhost:5173/* (or similar port).

npm run dev

1.3. Overview of File Structure

Open the my-react-app folder in your code editor (like VS Code). The key files you will be working with are in the src directory:

- **src/main.jsx:** The entry point where your main React component (App) is rendered into the root of the HTML document (index.html).
- **src/App.jsx:** The main component of your application. This is where you will write most of your code initially.
- **src/index.css / src/App.css:** Stylesheet for your aplicacion.

Step 2: Understanding Components & JSX

React is built around the idea of Components, which are isolated, reusable pieces of code that return HTML-like markup.

2.1. Functional Components

A functional component is a plain JavaScript function that returns JSX. Component names must always start with a capital letter.

- **Open src/App.jsx** and replace its contents with the following simple component:

```
// src/App.jsx

function App() {

  return (

    <div className="container">
```

```
      <h1>Hello, React World!</h1>

      <p>This is my first React component.</p>

    </div>

  );

}

export default App;
```

2.2. What is JSX?

JSX (JavaScript XML) is a syntax extension for JavaScript that allows you to write HTML-like structures directly within your JavaScript code. React uses a tool (like Babel or the one built into Vite) to convert this JSX into regular JavaScript function calls that build the UI.

JSX Rules:

- It looks like HTML, but uses camelCase for attributes (e.g., className instead of class).
- Components must return a single root element (or use a <> fragment).
- You embed JavaScript expressions inside JSX using curly braces {}.

Adding JavaScript (Expressions): In the src/App.jsx file add a JavaScript variable:

```
// src/App.jsx

function App() {

  const userName = “Beginner”;
```

```
  const currentYear = new Date().getFullYear(); // JavaScript expression

  return (

    <div className="container">

      {/* Embed JavaScript using curly braces */}

      <h1>Hello, {userName}!</h1>

      <p>Learning React in {currentYear}.</p>

    </div>

  );

}

export default App;
```

Step 3. Reusable Components and Props

A core strength of React is composition—building complex UIs by combining smaller, reusable components.

3.1. Writing a Reusable Component

Let's start by creating a new component for a simple header.

Create a file: Create a new file, src/components/Header.jsx. (You might need to create the components folder first).

Add the component code:

```
// src/components/Header.jsx
```

```
function Header() {
  return (
    <header>
      <h2>My Component-Based App</h2>
      <nav>
        <a href="#">Home</a> | <a href="#">About</a>
      </nav>
    </header>
  );
}
export default Header;
```

3.2. Nested Components

Now, import and use the Header component inside your main App component.

- **Update** *src/App.jsx*:

```
// src/App.jsx
import Header from './components/Header'; // 1. Import the component
function App() {
  const userName = "Beginner";
```

```
  return (
    <div className="app-container">
      {/* 2. Use the component as a custom HTML tag */}
      <Header />
      <main>
        <h1>Welcome, {userName}!</h1>
        <p>This is the main content area.</p>
      </main>
    </div>
  );
}

export default App;
```

3.3. Using Props (Properties)

Props are arguments passed to React components. They allow parent components to pass data down to child components, making them dynamic and reusable.

- **Define Props in the Child Component (Header.jsx):** We will modify Header to accept a title prop.

```
// src/components/Header.jsx
```

```
// Use object destructuring to get the ‘title’ prop

function Header({ title }) {

  return (

    <header>

      {/* Display the value of the prop */}

      <h2>{title}</h2>

      <nav>

        <a href=”#”>Home</a> | <a href=”#”>About</a>

      </nav>

    </header>

  );

}

export default Header;
```

- **Pass Props from the Parent Component (App.jsx):** Pass the title as an attribute when using the component.

```
// src/App.jsx

import Header from ‘./components/Header’;

function App() {
```

```
  const appTitle = “My First Dynamic App”; // Data to pass down

  return (

    <div className=”app-container”>

      {/* Pass ‘appTitle’ as the ‘title’ prop */}

      <Header title={appTitle} />

      <main>

        <h1>Welcome!</h1>

        <p>Content goes here.</p>

      </main>

    </div>

  );

}

export default App;
```

Step 4. Adding State and Hooks

To make your application interactive, you need a way for components to remember data and trigger re-renders when that data changes. This is handled by State.

4.1. The useState Hook

The useState Hook is a special function that lets you add state variables to functional components. Hooks were introduced in React 16.8 and are the modern way to manage component logic.

Syntax:

const [stateVariable, setStateFunction] = useState(initialValue);

- **stateVariable:** The present worth of the state.
- **setStateFunction:** A function that updates the state variable.
- **initialValue:** The value from which the state starts.

4.2. Implementing a Simple Counter Component

Let's create a component that displays a count and can be incremented with a button.

Create a file: Create *src/components/Counter.jsx.*

Add the component code:

```
// src/components/Counter.jsx

import { useState } from 'react'; // 1. Import useState

function Counter() {

  // 2. Declare a state variable named 'count' initialized to 0.

  //    'setCount' is the function used to update it.

  const [count, setCount] = useState(0);

  // 3. Define the event handler function
```

```
  const handleClick = () => {
    // 4. Use the setter function to update the state
    setCount(count + 1);
    // This triggers React to re-render the component with the new 'count' value.
  };
  return (
    <section>
      <h2>Interactive Counter</h2>
      {/* Display the current state value */}
      <p>Count: <strong>{count}</strong></p>
      {/* Attach the event handler to the button's onClick attribute */}
      <button onClick={handleClick}>
        Increment
      </button>
    </section>
  );
}
```

```
export default Counter;
```

4.3 Integrating the Counter

Update *src/App.jsx:*

```
// src/App.jsx

import Header from './components/Header';

import Counter from './components/Counter'; // Import the new component

function App() {

  const appTitle = "My First Dynamic App";

  return (

    <div className="app-container">

      <Header title={appTitle} />

      <main>

        <Counter /> {/* Add the Counter here */}

      </main>

    </div>

  );

}

export default App;
```

Step 5. Processing User Input – Forms

State is essential for handling form inputs, known as Controlled Components in React. This means the form data is handled by React state, making it the “single source of truth.”

5.1. Building an Input Component

Let’s create a component for capturing the user’s name.

Create a file: Create *src/components/InputForm.jsx*.

Add the block code:

```
// src/components/InputForm.jsx

import { useState } from ‘react’;

function InputForm() {

  // State to store the current input value

  const [name, setName] = useState(”);

  // Event handler for input change

  const handleChange = (event) => {

    // Get the current value from the input field

    setName(event.target.value);

  };

  // Event handler for form submission (prevents default browser refresh)
```

```
const handleSubmit = (event) => {
  event.preventDefault();
  alert(`Hello, ${name}! Your form has been submitted.`);
  setName(''); // Clear the input after submission
};
return (
  <section>
    <h2>Input Form Example</h2>
    {/* The form calls handleSubmit on submission */}
    <form onSubmit={handleSubmit}>
      <label htmlFor="name-input">Enter your Name:</label>
      <input
        id="name-input"
        type="text"
        value={name} // 1. Bind the input value to the 'name' state
        onChange={handleChange} // 2. Update state on every keystroke
        placeholder="Your Name"
```

```
        />
        <button type="submit">Submit</button>
      </form>
      {/* Display the current input state in real-time */}
      <p>Current Input: {name}</p>
    </section>
  );
}
export default InputForm;
```

5.2 Integrating the Input Form

Update *src/App.jsx:*

```
// src/App.jsx
import Header from './components/Header';
import Counter from './components/Counter';
import InputForm from './components/InputForm'; // Import the new component
function App() {
  const appTitle = "My First Dynamic App";
  return (
```

```
    <div className="app-container">
      <Header title={appTitle} />
      <main>
        <Counter />
        <hr/>
        <InputForm /> {/* Add the InputForm here */}
      </main>
    </div>
  );
}

export default App;
```

Step 6. Fetching Data with useEffect

In real-world applications, components often need to fetch data from an API after they render. This is a side effect and is managed using the useEffect Hook.

6.1. The useEffect Hook

The useEffect Hook lets you perform side effects (like data fetching, manual DOM changes, or setting up subscriptions) in functional components.

Syntax:

```
useEffect(() => {
```

```
  // Code to run (the side effect)

  return () => {

    // Optional cleanup function (runs before re-render or unmount)

  };

}, [dependencyArray]); // Control when the effect runs
```

Dependencies Array ([]):

- **Empty array ([]):** Runs the effect only once after the initial render (perfect for initial data fetching).
- **No array (omitted):** Runs the effect after every render.
- **Variables/Props ([prop1, state2]):** Runs the effect only when the values in the array change.

6.2. Fetching Dummy Data

Setup a state to store a list of users fetched from a public API using useEffect.

- **Create a file:** Create src/components/UserList.jsx.
- **Add the component code:**

```
// src/components/UserList.jsx

import { useState, useEffect } from 'react';

function UserList() {

  // State to hold the fetched data

  const [users, setUsers] = useState([]);
```

```
// State to handle loading status
const [isLoading, setIsLoading] = useState(true);
useEffect(() => {
  // 1. Define the asynchronous function to fetch data
  const fetchUsers = async () => {
    try {
      const response = await fetch('https://jsonplaceholder.typicode.com/users');
      const data = await response.json();
      setUsers(data); // 2. Update the state with the fetched data
    } catch (error) {
      console.error("Failed to fetch users:", error);
    } finally {
      setIsLoading(false); // 3. Set loading to false once done
    }
  };
  fetchUsers();
}, []); // Empty dependency array: runs only on the first render
```

```
// Conditional Rendering based on loading state
if (isLoading) {
  return <p>Loading users…</p>;
}
return (
  <section>
    <h2>User List (Data Fetching Example)</h2>
    <ul>
      {/* 4. Map over the 'users' array and render a list item for each */}
      {users.map(user => (
        // Always include a unique 'key' prop when rendering lists
        <li key={user.id}>
          <strong>{user.name}</strong> – {user.email}
        </li>
      ))}
    </ul>
  </section>
```

```
  ); }

export default UserList;
```

6.3 Final Integration

Update *src/App.jsx* for the final time:

```
// src/App.jsx

import Header from ‘./components/Header’;

import Counter from ‘./components/Counter’;

import InputForm from ‘./components/InputForm’;

import UserList from ‘./components/UserList’; // Import the new component

function App() {

  const appTitle = “My Complete Beginner App”;

  return (

    <div className=”app-container”>

      <Header title={appTitle} />

      <main>

        {/* Basic Interaction */}

        <Counter />

        <hr/>
```

```
      {/* Form Handling */}
      <InputForm />
      <hr/>
      {/* Data Fetching */}
      <UserList />
    </main>
  </div>
 ); }

export default App;
```

You have successfully set up a React development environment, built a component hierarchy, used JSX for markup, implemented interactivity with the useState Hook, and handled data fetching with the useEffect Hook. Ready to solidify your skills and tackle real-world coding problems? Take on our practical **React JS Coding Challenges and Solutions** to advance your development abilities.

Real Time Examples for React JS Tutorial for Learners

Here are some excellent real-time examples demonstrating key React JS concepts for learners:

Live Search/Filtering Component: Using useState, Event Handlers

- **Abstracted Concept:** State Management; handling user input events – onchange

- **Example:** A list of items (e.g., product names, country list) that updates instantly as the user types into an input field.
- **How it Works:**
    - The search bar's input value is stored in a state variable using useState.
    - The onChange handler updates this state on every keystroke.
    - The component then uses JavaScript's filter() method on the original list, based on the current state value, and re-renders only the matching items in real-time.
    - This is a foundational example of creating a fast, reactive UI.

Chat Application in Real Time-Using useEffect and WebSockets

- **Concept Demonstrated:** Side Effects (useEffect) and integrating with external services (WebSockets).
- **Example:** A basic chat interface where messages appear immediately without refreshing the browser.
- **How it Works:**
    - The useEffect Hook is used to establish and maintain a connection to a WebSocket server when the component mounts.
    - When a new message arrives from the server, the connection listener updates the component's message array state, triggering an instant re-render to display the new message.
    - This shows how React handles continuous, asynchronous data streams.

Shopping Cart Component (By Using State Lifting and Props)

- **Concept Demonstrated:** Component Communication (Props) and State Lifting (managing shared state in a common parent).
- **Example:** A complex page with separate components for ProductCard, CartButton, and CartSummary.
- **How it Works:**

- The cart data (the main state) lives in a common parent component.
- When a user clicks “Add to Cart” in the ProductCard (a child component), it calls a function passed down via props from the parent.
- This function updates the cart state in the parent, which then passes the updated cart data down to the CartSummary component (another child), ensuring all parts of the UI are perfectly synchronized.

Ready to build your own reactive applications? Explore our curated list of exciting **React JS Project Ideas** to start coding today!

FAQs About React JS Tutorial for Beginners

1. How to learn React JS for beginners?

First, master JavaScript fundamentals (ES6+). Then, learn core React concepts like Components, JSX, Props, and Hooks (useState, useEffect) by building small projects.

2. Is React JS easy to learn?

React is considered relatively easy once you have a solid foundation in JavaScript. Without strong JS knowledge, concepts like state and functional programming can be challenging.

3. Can I learn React in 5 days?

You can grasp the basics (Components, JSX, simple State) in 5 days, but becoming proficient enough for a job requires several weeks or months of practice and project building.

4. Which tool is used for React JS?

Vite or Create React App (CRA) are used for initial project setup. Node.js and npm/Yarn manage dependencies, and Babel transpiles JSX into browser-readable JavaScript.

5. Will AI replace React?

No. AI tools like Copilot will automate repetitive tasks and boilerplate code. Developers will focus more on complex architecture, business logic, and unique UX/UI design.

6. Can I get a job if I learn React?

Yes, absolutely. React is highly in-demand for building modern web and mobile apps (via React Native). Strong React skills lead to excellent job prospects globally.

7. Is React harder than Python?

For an absolute beginner, Python is generally easier due to its simple, readable syntax. React requires first mastering JavaScript, which gives it a steeper initial learning curve.

8. Is React high paying?

Yes, highly skilled React developers are in demand and command competitive salaries. Senior developers often earn top-tier compensation in the software industry.

9. Is Netflix using React?

Yes. Netflix originally used a custom React implementation (React-Gibbon) for its TV interface and still uses React for various parts of its platform and React Native for some mobile apps.

10. What is the salary of React developer in TCS?

The average **React Developer Salary at TCS in India** ranges from around ₹2.5 Lakhs to ₹9.7 Lakhs for those with 1 to 4 years of experience.

Conclusion

You've successfully completed the foundational steps of React JS, from setting up your project and understanding JSX and Components to implementing interactivity with State and Hooks. You now have the power to build dynamic, reusable UIs. Keep building small projects to solidify these concepts and master the art of component architecture. Ready to dive deeper and cover advanced topics like Redux, Routing, and Testing? Enroll in our comprehensive **React JS Master Course in Chennai** to accelerate your career.