Why React?

Declarative, Component-Driven, Stateful UI

1. Introduction – The Problem with Traditional JS/jQuery

In traditional JavaScript or jQuery-based development, creating dynamic UIs involves manually updating the DOM and tracking UI state. This becomes complex and error-prone as applications grow.

Example (jQuery-style toggle):

```
<button id="toggle">Toggle</button>
<div id="box" style="display:none;">Hello!</div>

<script>
  $('#toggle').click(function () {
    $('#box').toggle();
  });
</script>
```

This approach mixes logic and UI, making code harder to maintain.

2. What is React?

React is a JavaScript library developed by Facebook for building user interfaces. It is focused on:

- **Declarative Programming**
- **Component-Driven Design**
- **Stateful UI**

3. React is Declarative

Instead of imperatively changing the UI, React lets you declare what the UI should look like based on the current state.

Example (Declarative Toggle in React):

```
function Greeting() {
  const [isVisible, setIsVisible] = React.useState(false);

  return (
    <div>
      <button onClick={() => setIsVisible(!isVisible)}>Toggle</button>
      {isVisible && <div>Hello!</div>}
    </div>
  );
}
```

React re-renders the UI automatically when the state changes.

4. React is Component-Driven

React encourages building the UI as a tree of reusable components.

Example (Reusable Button Component):

```
function Button({ onClick, label }) {
  return <button onClick={onClick}>{label}</button>;
}

function App() {
  const [count, setCount] = React.useState(0);

  return (
    <div>
      <Button label="Click me!" onClick={() => setCount(count + 1)} />
      <p>You clicked {count} times.</p>
    </div>
  );
}
```

5. React is Stateful

React uses hooks like `useState` to manage local component state and automatically update the UI.

Example (Counter):

```
function Counter() {
  const [count, setCount] = React.useState(0);

  return (
    <div>
```

```
      <p>Current count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increase</button>
    </div>
  );
}
```

6. Putting It Together – A Simple Todo App

Full Example:

```
function TodoApp() {
  const [todos, setTodos] = React.useState([]);
  const [input, setInput] = React.useState("");

  const addTodo = () => {
    if (input.trim() === "") return;
    setTodos([...todos, input]);
    setInput("");
  };

  return (
    <div>
      <h2>Todo List</h2>
      <input value={input} onChange={(e) => setInput(e.target.value)} />
      <button onClick={addTodo}>Add</button>

      <ul>
        {todos.map((todo, index) => (
          <li key={index}>{todo}</li>
        ))}
      </ul>
    </div>
  );
}
```

7. Understanding the Spread Operator (`...todos`)

In this line:

```
setTodos([...todos, input]);
```

`...todos` spreads the contents of the `todos` array into a new array, allowing React to track state immutably.

Why not use push()?

```
// Mutates state directly:
todos.push(input);

// Creates a new array:
setTodos([...todos, input]);
```

Table: Spread Operator Explained

Syntax	Meaning	Result
`...todos`	Spread existing items	`'Buy milk', 'Walk dog'`
`[...todos]`	Copy of existing array	`['Buy milk', 'Walk dog']`
`[...todos, x]`	Append new item	`['Buy milk', 'Walk dog', x]`

8. Recap – Why React?

Feature	What It Means	Benefit
Declarative UI	Describe UI as a function of state	Easy to reason about
Component-Driven	UI built from modular components	Reusable and maintainable
Stateful UI	Built-in hooks like `useState`	Reactive UI updates

9. Final Notes

React is a UI-focused library (not a full framework). You can enhance it with:

- **React Router** – for client-side navigation
- **Redux** or **Zustand** – for global state
- **Next.js** – for full-stack apps and SSR

React encourages clean, predictable, and scalable front-end architecture.