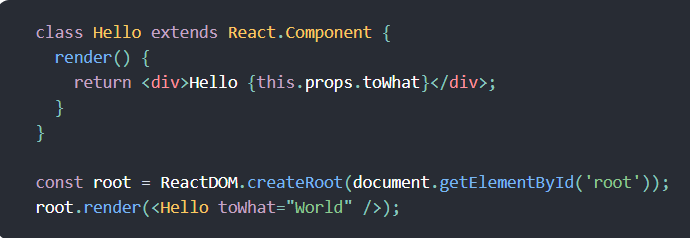
1. Is JSX mandatory for React?

JSX is not a requirement for using React. Using react without JSX is convenient when you don’t want to go through setting up compilation in your build environment.

Each JSX element is just a shortcut to React.createElement(component, props, …children) to make it look cleaner and easier to read.

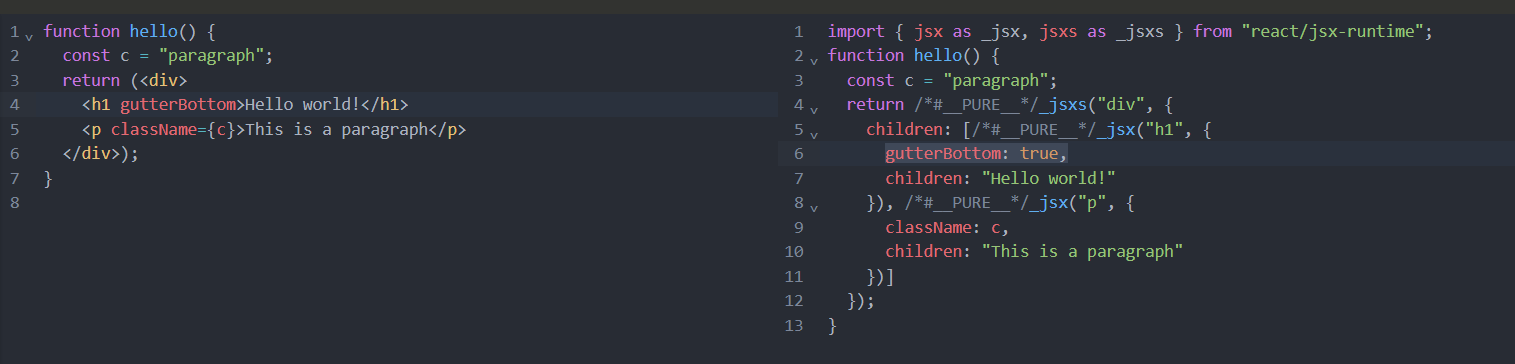
To give an example, the below code is written in JSX:



Which can be compiled to this code that does not use JSX:



Below is the screenshot of how babel converts jsx to normal javascript behind the scenes.



1. Is ES6 mandatory for Reactjs?

ES6 is not mandatory for React, but it is highly recommended. React can work in plain javascript, but using ES6 makes coding faster, easier, cleaner and more modern. Many ES6 features are used in React such as Arrow Functions, Classes, Destructing, Template literals, let and const, Promises, Modules, Spread Operator, etc. It simplifies coding with React.

4. How can I write comments in JSX?

To put comments inside JSX, you use the syntax {/\* \*/} to wrap around the comment text.

5. What is <React.Fragment></React.Fragmet> and <></>?

<Fragment>, often used via <> … </> syntax, lets group elements without a wrapper node. Wrap elements in <Fragment> to group them together in situations where you need a single element. Grouping elements in Fragment has no effect on the resulting DOM; it is the same as if the elements were not grouped. The empty JSX tag <></> is shorthand for <Fragment></Fragment> in most cases.

Some Caveats:

If you want to pass key to a Fragment, you can’t use the <> … </> syntax. You have to explicitly import Fragment from ‘react’ and render <Fragment key={yourKey}>… </Fragment>.

6. What is Virtual DOM?

The virtual DOM is a llightweight copy of the Real DOM. It allows React to manage changes more efficiently by minimizing the direct manipulation required on the real DOM. This greatly improves the performance of web apps. The virtual DOM is an in-memory representation of the real DOM elements. “In-memory” means it’s not in your screen but rather it’s like a copy of the actual DOM that React keeps in it’s workspace (memory). React uses this to copy and quickly figure out changes and updates the real page only when necessary.

Step by Step Process of the Virtual DOM working:

1. Step 1: Initial Rendering: when the app starts, the entire UI is represented as Virtual ODM. React elements created and rendered into the virtual structure.

2. Step 2: State and Props Changes: as the states and props change in the app, React re-renders the affected components in the virtual DOM. These changes do not immediately impact the real DOM.

3. Step 3 – Comparison Using diff algorithm: React then uses a diffing algorithm to compare the current version of the Virtual DOM with the previous version. This process identifies the differences (or “diffs”) between the two versions.

4. Step 4 – Reconciliation Process: based on the differences identified, React determines the most efficient way to update the real DOM. Only the parts of the real DOM that need to be updated are changes, rather than re-rendering the entire UI. This selective updating is quick and performant.

5. Step 5 – Update the Real DOM: finally, React applies the necessary to the real DOM. This might involve adding, removing, or updating elements based on the differences detected in step 3.

What are the benefits of using Virtual DOM?

It has simplified development. Virtual DOM let’s a developer writes their code in a more declarative style where you are simply describing the UI and React takes care of it. This is all because of React’s declarative syntax and it’s component based architecture.

Declarative syntax is a way of writing code where you focus on what you want to happen rather than how to do it step by step. It's more about describing the result and React (or the underlying system) figures out how to achieve it.

In React, declarative syntax is evident when you define how your UI should look based on the current state or props. Here’s how it looks:

Example of Declarative Syntax:

function Greeting({ name }) {

return <h1>Hello, {name}!</h1>;

}

What it does: This code simply describes that the UI should show "Hello, [name]!" where name is passed as a prop. You don't need to manually write instructions to create an <h1> element, set its text, or update it when name changes. React takes care of those details.

How It Differs from Imperative Syntax

In imperative syntax, you write detailed steps to make things happen. For example:

Imperative Way:

const h1 = document.createElement('h1');

h1.textContent = 'Hello, John!';

document.body.appendChild(h1);

Here, you explicitly tell the browser to:

Create an <h1> element.

Set its text to "Hello, John!"

Attach it to the page.

Declarative Benefits in React

React's declarative syntax allows you to focus on the "what":

"I want an <h1> that says Hello, {name}." Instead of the "how":

"Create an element, set its text, attach it, and update it later if needed."

This makes your code:

Easier to write and read.

Less error-prone.

Easier to maintain, especially as your UI becomes more complex.

It also Improves performance.

It has enhanced User Experience.

It is cross-platform development.

7. What is Reconciliation in React?

Reconciliation is a process that React follows which updates the real DOM. It checks for changes in the Virtual DOM and the real DOM and changes only the stuff that has been modified. This process is efficient, faster, and keeps the UI in sync with the current state or props of the application.

Virtual DOM is a lightweight copy of the actual DOM. Instead of directly changing the real DOM which is performance heavy it uses the Virtual DOM for performance optimization.

When a component’s state or props is changed it creates a new Virtual DOM tree which represents the updated UI.

React compares the newly created Virtual DOM with the older Virtual DOM. This comparison is called the “diffing” algorithm.

Diffing means difference between the Older Virtual DOM and the newly created Virtual DOM (e.g. which nodes were added, or were modified or removed, etc.)

Once the differences are identified, react updates only the parts of the Real DOM that has been changed instead of re-rendering the entire UI. This makes React fast and efficient as updating the real DOM is computationally expensive.

Some key aspects of reconciliation are:

1. Elements are of the same type: If two elements are of the same type (e.g. <div>), React reuses the existing DOM node and only changes the attributes or children if they are changed.
2. Elements of Different Type: If the type of the element changes (e.g. from <div> to <span>), React destroys the old DOM node and creates a new one.
3. Lists with Type: For lists, React uses keys to track individual items. Keys help React identify which items have changed, been added, or removed. Without keys, React re-renders the entire list.

**How react Handles Component Updates**:

State Changes: When you call setState() in a class component or use the useState hook in a functional component, React Triggers reconciliation for that specific component and its children.

Props Changes: When the props of a component change, React reconciles that component and its children.

ShouldComponentUpdate & Memoization: You can control how React reconciles components by using lifecycle methos like shouldComponentUpdate() (for class components) or React.memo() (for functional components). This helps prevent unnecessary updates.

The main reason reconciliation matters because it improves the performance of the app by reducing unnecessary DOM updates and it ensures that the UI stays consistent with the underlying application state or props.

A screenshot of a computer program

Description automatically generated

1. Initially, React renders the div, h1 and button in the real DOM.
2. When you click the button, setCount updates the state.
3. React creates a new Virtual Dom to represent the updated UI (count: 1 instead of Count: 0).
4. React compares the old Virtual DOM with the new one:
   1. The h1 content changes, so React updates only the h1 text in the real DOM.
   2. The button and div remain unchanged, so React reuses those DOM nodes.

In simple words, reconciliation is react’s way of figuring out what’s different in the UI and only changing what is necessary.

8. What is React Fiber?

React Fiber is the re-implementation of React’s core algorithm which is the reconciliation algorithm. It was introduced in React 16. The main goal of this update is for the UI to be more efficient, smooth and faster. Fiber is the “engine” under the hood of React that helps manage how your app updates and renders components.

Before Fiber, React used a stack-based algorithm, which was synchronous. This means that once React started updating a UI it couldn’t stop until it was done. For complex apps, this could cause UI freezes or slow performance because React couldn’t handle other important tasks (like user input) until it finished.

Fiber solved this by making React’s rendering process asynchronous, prioritized and interruptible. This makes apps feel smoother and more responsive, even when working with complex UIs or heavy computations.

React Fiber breaks rendering into smaller chunks, so React doesn’t have to do everything at once.

If something more important comes up (like a button click), React can pause what’s its doing and handle that first.

Updates are given priorities:

Such as High-Priority updates are user-input, animations.

Low-priority updates are background tasks, non-visible components.

React Fiber improves the process of comparing the current UI tree to the new UI tree (called diffing) and updating only the parts of the UI that have changed.

Fiber splits rendering work into chunks and spreads them over multiple frames. This ensures React doesn’t block the main thread for too long, allowing smoother animations and faster interactions.

Fiber treats each component and element in your app as a "work unit."

Instead of processing the entire tree of components in one go, it splits them into smaller units that can be paused and resumed later.

Each update (like a state or prop change) is assigned a priority.

Fiber uses these priorities to decide the order of updates.

For example, if you click a button (high priority) while an animation is running (lower priority), React will handle the button click first.

There are Two phases of rendering:

Render Phase:

Fiber figures out what needs to change in the DOM but doesn’t make changes yet.

This phase is interrupted if higher-priority work comes up.

**Commit Phase**:

Fiber applies the calculated changes to the DOM in one go.

This phase is synchronous and cannot be interrupted.

**Technical Details (Simplified)**

* **Fiber Nodes**:
  + Each component or element in the React tree gets its own **Fiber Node**, which is like a virtual representation of that component.
  + These nodes contain metadata like the component type, props, and state.
* **Linked List**:
  + React Fiber represents the UI tree as a linked list instead of a traditional tree. This structure makes it easier to pause and resume work.
* **Scheduler**:
  + Fiber uses a **scheduler** to manage tasks and assign priorities.
  + It ensures React works efficiently without overloading the browser.

**Benefits of React Fiber**

1. **Smooth User Experiences**
   * Apps feel more responsive because Fiber handles updates without blocking the UI.
2. **Better Animations and Transitions**
   * Time slicing ensures animations run smoothly, even when there are heavy updates in the background.
3. **Improved Performance for Complex Apps**
   * Fiber's ability to pause and resume work means that large applications don't freeze when rendering large UI trees.
4. **Enhanced Developer Experience**
   * Features like error boundaries and suspense (introduced after Fiber) are possible because of the new architecture.

**Examples of Fiber in Action**

1. **User Input During Animation**:
   * Imagine an animation is running, and you click a button.
   * With Fiber, React can pause the animation, handle the button click, and then resume the animation.
2. **Rendering Large Lists**:
   * If you have a list with 10,000 items, Fiber will process it in chunks instead of rendering it all at once, preventing the browser from freezing.

**How Fiber Changed React**

1. **Suspense and Concurrent Mode**
   * Fiber laid the groundwork for advanced features like Suspense and Concurrent Mode.
   * These features improve how React handles asynchronous tasks and data fetching.
2. **Backward Compatibility**
   * Even though Fiber changed React’s internal algorithm, it didn’t break existing apps. Developers didn’t need to rewrite their code to benefit from Fiber.

9. Why do we need keys in React? When do we need keys in React?

Answer: A key is a unique identifier for each item in a list of elements. React uses keys to keep track of which items have changed, added or removed when updating the UI. The best way to use a key is to use a string that uniquely identifies a list of items among its siblings. Most often it’s advisable to use ID’s from the data as keys.

So, for example:

<li key={todo.id}>{todo.text}</li>

When we don’t have a stable ID to rely on to render items, we might just use the index as a key as a last resort.

A screenshot of a computer code

Description automatically generated

It is not recommended to use indexes for keys if the order of the items change. This negatively impacts performance and may cause issues with component state.

Sometimes generating new ids maybe redundant and should be avoided such as the case of a translation or license terms where the items are fixed and not changed.

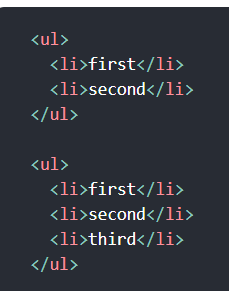
Three conditions when we can use index as keys:

1. The list and items are static-they are not computed and do not change.
2. The items in the list have no ids.
3. The list is never reordered

When all of them are met, it is safe to use index as a key.

By default, when recursing on the children of a DOM node, React just iterates over both lists of children at the same time and generates a mutation whenever there’s a difference.

For example, when adding an element at the end of the children, converting between the two trees works well:



React will match the two <li> first </li> trees, match the two <li> second </li> trees, and then insert the <li> third </li> tree.

If implemented naively, inserting an element at the beginning has worse performance. For example, converting between these two trees works poorly:

A screen shot of a computer program

Description automatically generated

So, here’s what happens:

* React compares <li>Duke</li> in the old tree with <li>Connecticut</li> in the new tree. They are **different**, so React thinks it needs to replace <li>Duke</li> with <li>Connecticut</li>.
* React then compares <li>Villanova</li> in the old tree with <li>Duke</li> in the new tree. They are **different**, so React thinks it needs to replace <li>Villanova</li> with <li>Duke</li>.
* Finally, React **adds <li>Villanova</li>** at the end.

Instead of realizing that <li>Duke</li> and <li>Villanova</li> could have stayed where they are, React **recreates everything**, which is slower and less efficient.

Keys: to solve this issue, React supports a key attribute. When children have keys, React uses the key to match children in the original tree with children in the subsequent tree. For example, adding a key to our inefficient example above can make the tree conversion efficient:

A screen shot of a computer program

Description automatically generated

10. Can we use index as keys in React?

Answer: Yes we can use index as keys in React only but it is not recommended as it might mess up the ordering of the items in a list. It is usually safe to use index as key based on 3 rules:

1. The list and items are static-they are not computed and do not change.
2. The items in the list have no ids.
3. The list is never reordered

11. What are props in React? Ways to use props?

Answer: Props (short for properties) are a way to apss data from one component to another in React, typically from a parent component to a child component. Props are read-only, meaning they cannot be modified by the child component receiving them. They allow us to make components dynamic and reusable by customizing their behavior or appearance based on the data passed to them.

Ways to use props:

1. Passing props to a child component:

A screen shot of a computer code

Description automatically generated

Here, the name prop is passed from the App component to the Greeting component.

1. Destructing props: Instead of accessing props using props.name, you can destructure them directly:

A screen shot of a computer code

Description automatically generated

1. Default props: You can defined default values for props if they are not provided:

A black screen with white text

Description automatically generated

12. What is Config Driven UI?

Answer: Config driven UI is a design pattern where the structure and behavior of the user interface are defined using configuration files rather than hard coded in the application. These configurations files are typically in formats like JSON or YAML. By separating the UI logic from the code, developers can easily modify the UI without changing the underlying codebase.

In config drive UI, we can update the configuration files, and the UI will automatically adjust based on the new settings. This approach simplifies the development process and makes the UI more adaptable to different requirements.

Benefits of Config Drive UI:

1. Flexibility and Scalability:
   1. The main benefit of config drive UI is flexibility. Since the UI is driven by the configuration file, it’s easier to make changes without changing the underlying codebase. This makes the UI scalable, offering developers the opportunity to add new features or modify existing ones with minimal effort.
2. Easier Maintenance and Updates:
   1. Config driven UI simplifies the maintenance process. Developers can make the change directly in the configuration file and don’t have to look up the code that needs to be changed. This helps reduce bugs and speeds up the update process.
   2. For example: Changing the Theme of the app:
      1. With config drive UI, the theme settings can be changed in the configuration file itself. Below is a screenshot of a configuration file with a theme that can be changed.

A screen shot of a computer

Description automatically generated

* + 1. By updating this configuration file, the entire application can switch to the new theme without any code changes.

Configuration files usually are files that contain information about components, their properties, and how they should be arranged on the screen.

Let’s take an example of changing a form in a web page using configuration file.

A screen shot of a computer code

Description automatically generated

This JSON configuration file can then be used to render the form dynamically in the UI.

Using this configuration file we can render components:

For example:

A computer screen with many white and green text

Description automatically generated

We use the Component in App.js

A computer screen shot of text

Description automatically generated

This setup fetches the configuration file, reads the form settings, and dynamically renders the form based on the configuration.

What we achieved by using Config-Drive UI in React is:

* Ease of Updates
* Dynamic UI
* Reduced code Duplication.