# React 19

# New Features 🚀

Rajat Saraswat
www.linkedin.com/in/rajatsaraswat9/

# React Compiler

## Forget about Memo

- The most exciting feature addition to React 19 is React Compiler. You may be familiar today with memoization APIs such as useMemo, useCallback, and React.memo.
- While powerful, it’s easy to forget to apply memoization or apply them incorrectly.

## Automatically Memoization of your Code.

- React Compiler automatically applies memoization to React components and hooks, thus ensuring better performance without manual intervention.

- It identifies parts of your code that can benefit from memoization, reducing manual overhead and potential errors.

The initial release of React Compiler is primarily focused on improving update performance (re-rendering existing components), so it focuses on these two use cases:

1. Skipping cascading re-rendering of components.
2. Skipping expensive calculations from outside of React.

# Actions

- Actions simplify handling asynchronous operations in React components, such as API requests or form submissions.

- They manage pending states, errors, optimistic updates, and sequential requests automatically,  making the developer's job easier.

**Key Features of Actions**

1. Pending State: Actions provide a pending state automatically during the async request, keeping the UI responsive.

2. Optimistic Updates: Using useOptimistic, you can show immediate feedback to users while the request is being processed.

3. Error Handling: Actions handle errors, allowing you to display error messages and revert optimistic updates if necessary.

4. Form Handling: <form> elements can now use functions for action and formAction props, which manage form submissions automatically and reset forms upon success.

**Rajat Saraswat**
www.linkedin.com/in/rajatsaraswat9/

# Example: Before Actions

```
function UpdateName({}) {
  const [name, setName] = useState("");
  const [error, setError] = useState(null);
  const [isPending, setIsPending] = useState(false);

  const handleSubmit = async () => {
    setIsPending(true);
    const error = await updateName(name);
    setIsPending(false);
    if (error) {
      setError(error);
      return;
    }
    redirect("/path");
  };

  return (
    <div>
      <input value={name} onChange={(event) => setName(event.target.value)} />
      <button onClick={handleSubmit} disabled={isPending}>
        Update
      </button>
      {error && <p>{error}</p>}
    </div>
  );
}
```

# Example: Using Actions

```
function UpdateName({}) {
  const [name, setName] = useState("");
  const [error, setError] = useState(null);
  const [isPending, startTransition] = useTransition();

  const handleSubmit = () => {
    startTransition(async () => {
      const error = await updateName(name);
      if (error) {
        setError(error);
        return;
      }
      redirect("/path");
    })
  };

  return (
    <div>
      <input value={name} onChange={(event) => setName(event.target.value)} />
      <button onClick={handleSubmit} disabled={isPending}>
        Update
      </button>
      {error && <p>{error}</p>}
    </div>
  );
}
```

**Rajat Saraswat**
www.linkedin.com/in/rajatsaraswat9/

# React Server Components

- RSC is a new architecture designed to run components on the server rather than the client.

- This reduces the size of the JavaScript bundle sent to the client side, which can result in faster initial page loads and optimize the app's performance.

**Key Features:**

- Async/Await for Data Fetching:
  Server Components use async/await for fetching data, making it simpler to manage asynchronous operations.

- Seamless Client-Server Communication:
  Allows Server Components to perform server-side operations and pass results to Client Components, reducing the need for extensive APIs.

- Server Actions: "use server" directive is used for Server Actions.
  Execute server-side logic from client-side components through Server Actions. This feature is designed to handle operations like database updates directly from client-side interactions.

Rajat Saraswat
www.linkedin.com/in/rajatsaraswat9/

# React 19
# New Hooks 🪝

- React 19 introduces several new hooks aimed at improving the handling of state, forms, and optimistic updates.
- These hooks simplify common patterns in React development and enhance performance and user experience.

Let’s discuss the three new hooks available in the new version :

- useActionState : Simplifies handling actions, especially those involving asynchronous operations.
- useFormStatus : Provides access to the status of a parent <form>, such as whether a form is pending.
- useOptimistic : Manages optimistic updates, rendering an optimistic state while an async request is in progress.

Rajat Saraswat
www.linkedin.com/in/rajatsaraswat9/

# React 19

## 🪝 New Hook : useActionState

- useActionState accepts an action function and an initial state.
- Returns an array with an error, a submitAction function, and a pending state.

```
import { useActionState } from 'react';

const [error, submitAction, isPending] = useActionState(
  async (previousState, newName) => {
    const error = await updateName(newName);
    if (error) {
      return error;
    }
    return null;
  },
  null,
);

function ChangeNameForm() {
  return (
    <form action={submitAction}>
      <input type="text" name="name" />
      <button type="submit" disabled={isPending}>Update</button>
      {error && <p>{error}</p>}
    </form>
  );
}
```

# React 19

# 🪝 New Hook : useFormStatus

- useFormStatus provides access to the status of a parent **<form>**, such as whether a form is pending.
- Useful in design systems where components need form status without prop drilling.

```jsx
import { useFormStatus } from 'react-dom';

function SubmitButton() {
  const { pending } = useFormStatus();
  return <button type="submit" disabled={pending}>Submit</button>;
}

function MyForm() {
  return (
    <form>
      <input type="text" name="name" />
      <SubmitButton />
    </form>
  );
}
```

# React 19

# 🪝 New Hook : useOptimistic

- useOptimistic takes an initial value and returns the optimistic state and a setter function.

- Enhances user experience by showing immediate feedback

```
import { useOptimistic } from 'react';

function ChangeName({ currentName, onUpdateName }) {
  const [optimisticName, setOptimisticName] = useOptimistic(currentName);

  const submitAction = async (formData) => {
    const newName = formData.get("name");
    setOptimisticName(newName);
    const updatedName = await updateName(newName);
    onUpdateName(updatedName);
  };

  return (
    <form action={submitAction}>
      <p>Your name is: {optimisticName}</p>
      <p>
        <label>Change Name:</label>
        <input type="text" name="name" disabled={currentName !== optimisticName} />
      </p>
    </form>
  );
}
```

# Asset Loading 

- React 19  automatically handles background loading for you particularly in handling images and resources like fonts, scripts, and stylesheets.

**Key Features :**

1. Background Asset Loading
    - React 19 allows images and other files to load in the background as users interact with the current page.
    - This reduces waiting times and ensures that users can interact with the content without interruptions, leading to faster page load times and a smoother user experience.

2. Lifecycle Suspense for Asset Loading
    - React 19 introduces lifecycle Suspense for loading assets like scripts, stylesheets, and fonts.
    - Suspense ensures that React determines when the content is ready to be displayed, eliminating the flash of unstyled content (FOUC) and enhancing the overall visual stability of the application.

3. New Resource Loading APIs : “preload” and “preinit”
    - These APIs provide greater control over when a resource should load and initialize.
    - Developers can fine-tune the loading strategy of their applications, optimizing performance by prioritizing critical resources.

# Document Metadata 

- React 19 introduces built-in support for rendering document metadata, such as <title>, <meta>, and  <link> tags directly within your component tree.

- This feature simplifies the management of document metadata and ensures consistent behavior across different environments, including client-side code, server-side rendering (SSR), and React Server Components (RSC).

## Built-in Metadata Support

  - Allows developers to manage document metadata within the React component tree.
  - This Ensures consistency and simplifies the process of updating document metadata, removing the need for external libraries like React Helmet.

```
function BlogPost({post}) {
  return (
    <article>
      <h1>{post.title}</h1>
      <title>{post.title}</title>
      <meta name="author" content="Josh" />
      <link rel="author" href="https://twitter.com/joshcstory/" />
      <meta name="keywords" content={post.keywords} />
      <p>
        Eee equals em-see-squared...
      </p>
    </article>
  );
}
```

# Support for async scripts 📄

- React 19 also improves the handling of async scripts, allowing them to be rendered anywhere in the component tree and ensuring that they load and execute efficiently.

- Async scripts can be included directly within components, avoiding the need to manage script relocation and deduplication.

Example :

```jsx
function MyComponent() {
  return (
    <div>
      <script async src="script.js"></script>
      Hello World
    </div>
  );
}

function App() {
  return (
    <html>
      <body>
        <MyComponent />
        {/* ... */}
        <MyComponent /> {/* No duplicate script in the DOM */}
      </body>
    </html>
  );
}
```

# Support for stylesheets 📄

- React 19 simplifies the management and composition of styles within React components.
- It waits for newly rendered stylesheets to load before committing the render,ensuring that styles are loaded efficiently and applied correctly.

Benefits :

- Local Reasoning: Stylesheets can be placed alongside the components that depend on them.
- Optimized Loading: Only the necessary stylesheets are loaded, improving performance.

```jsx
function ComponentOne() {
  return (
    <Suspense fallback="loading...">
      <link rel="stylesheet" href="foo.css" precedence="default" />
      <link rel="stylesheet" href="bar.css" precedence="high" />
      <article className="foo-class bar-class">
        {/* ... */}
      </article>
    </Suspense>
  );
}

function ComponentTwo() {
  return (
    <div>
      <p>{/* ... */}</p>
      <link rel="stylesheet" href="baz.css" precedence="default" />  {/* Inserted
    </div>
  );
}
```