

New Features in **React** 19





WHAT'S NEW IN REACT 19?



1 REACT COMPILER

What's New?

- React automatically optimizes re-renders without manual intervention.
- No more need for:
- `useMemo()`
- `useCallback()`
- `React.memo`

Why React Compiler?

- Manual Optimization was tedious for developers.
- React now handles state changes and re-rendering logic automatically.

Benefits

- ✓ React decides what to re-render and when.
- ✓ Reduces developer effort.
- ✓ Cleaner, simpler codebase.
- ✓ Currently powering Instagram in production.

2 SERVER COMPONENTS

What's New?

- **Components that run on the server, not the client.**
- **Previously pioneered by Next.js (default in v13).**
- **In React 19, server components are integrated natively.**

Benefits

- ✓ **SEO Friendly: Enhances search engine optimization.**
- ✓ **Performance Boost: Faster initial page loads.**
- ✓ **Server-Side Execution: Efficiently handles tasks like API calls.**

How to Use Server Components?

1- Client-Side (Default):

All React components run on the client side.

2 – Server-Side:

Add 'use server' as the first line of your component:

```
'use server';

export default async function requestUsername(formData) {
  const username = formData.get('username');
  return canRequest(username) ? 'successful' : 'failed';
}
```

Key Notes

- Server Components do not run on the client.
- Combined with Actions, they streamline tasks like form handling and API requests.
- Next.js users can already leverage server components; React 19 brings this directly to React.

3 ACTIONS

- BEFORE REACT 19: MANUAL HANDLING OF ASYNC STATES.

```
function handleSubmit() {  
  setLoading(true);  
  try {  
    const result = await saveData();  
    setData(result);  
  } catch (err) {  
    setError(err);  
  } finally {  
    setLoading(false);  
  }  
}
```

- AFTER REACT 19: USING USEACTIONSTATE FOR CLEAN ASYNC HANDLING.

```
const [state, formAction] = useActionState(async (prevState, formData) => {  
  const result = await saveData(formData);  
  return result;  
}, null);
```



**AUTOMATICALLY HANDLES LOADING, ERRORS,
AND STATE UPDATES.**

4 NEW HOOKS

USEACTIONSTATE: FORM HANDLING

Automatically manages submission state and form validation.

- BEFORE REACT 19:

You manually tracked form submission.

```
const [isSubmitting, setIsSubmitting] = useState(false);

async function handleSubmit() {
  setIsSubmitting(true);
  try {
    await submitForm();
  } finally {
    setIsSubmitting(false);
  }
}
```

- AFTER REACT 19:

useActionState simplifies form submission.

```
const [state, formAction] = useActionState(async (prevState, formData) => {
  return await submitForm(formData);
});
```


USEFORMSTATUS: TRACK FORM STATUS

Access form states like pending, success, or error.

```
const status = useFormStatus();

<button type="submit" disabled={status.pending}>
  {status.pending ? "Submitting..." : "Submit"}
</button>
```

USEOPTIMISTIC: OPTIMISTIC UI UPDATES

Simplifies optimistic updates for better UX.

- **BEFORE REACT 19:**

You updated UI optimistically, then handled reverts.

```
function handleClick() {
  setItems([...items, newItem]);
  saveItem(newItem).catch(()) => setItems(items); // Revert on error
}
```


- **AFTER REACT 19:**

useOptimistic makes it declarative.



```
const [optimisticItems, addOptimisticItem] = useOptimistic(items);

function handleClick() {
  addOptimisticItem(newItem);
}
```




5 THE USE() HOOK

What is use()?

The use() hook simplifies async data fetching and context consumption by handling promises directly.

- BEFORE REACT 19: USING USEEFFECT

Manual handling of async states.




```
const [data, setData] = useState(null);

useEffect(() => {
  fetchData().then((result) => setData(result));
}, []);
```

- AFTER REACT 19: THE USE() HOOK

Directly fetch async data and resolve promises.



```
const data = use(fetchData());
```

Advantages:

- No need for `useEffect` or `useState`.
- Cleaner and more declarative code.
- Handles promises seamlessly.



6 SERVER COMPONENTS

What are Server Components?

Server Components render on the server and send lightweight results to the client, improving performance.

- **BEFORE REACT 19: TRADITIONAL CLIENT-SIDE RENDERING**

You had to use `useEffect` for data fetching and SSR required complex tools.

```
useEffect(() => {  
  fetchData().then(setData);  
}, []);
```

- **AFTER REACT 19: SERVER-SIDE RENDERING**

React Server Components integrate seamlessly.

```
export default async function Page() {  
  const data = await fetchData();  
  return <div>{data.title}</div>;  
}
```

Advantages:

- Faster page loads (no heavy client-side JS).
- Improved SEO and reduced bundle size.



7 ENHANCED ASSET LOADING

React 19 improves how assets like images and scripts load in the background.

BEFORE REACT 19:

You needed manual configurations.



```
<script src="file.js" defer></script>
```

AFTER REACT 19: NATIVE LAZY LOADING

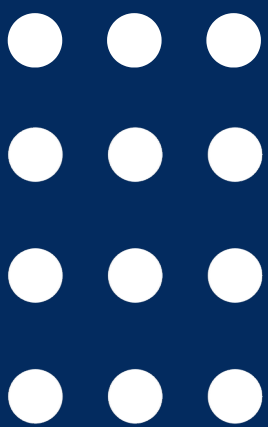
Assets now load efficiently without intervention.



```
  
<script async src="file.js"></script>
```

Advantages:

- Faster load times.
- Better user experience with reduced delays.

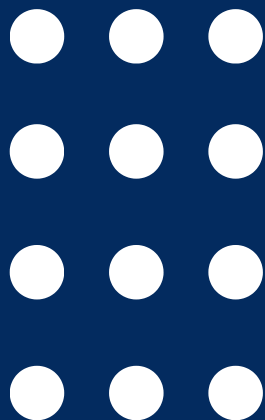


CONCLUSION

React 19 brings:

- Simplified async state management with Actions.
- New hooks like `useActionState`, `useFormStatus`, and `useOptimistic`.
- `use()` hook for easy async data handling.
- Seamless Server Components for better performance.
- Cleaner ref management and enhanced asset loading.





THANK YOU !

