

Tech Assembly 2 juillet 2025

# React 19 Server Components : qu'est-ce que ça change ?

✖ ▶ Uncaught Error: An unknown Component is an async [react-dom-client.development.js:5025](#)  
Client Component. Only Server Components can be async at the moment. This error is often  
caused by accidentally adding ``use client`` to a module that was originally written for  
the server.

```
at trackUsedThenable (react-dom-client.development.js:5025:21)  
at unwrapThenable (react-dom-client.development.js:7070:14)  
at reconcileChildFibersImpl (react-dom-client.development.js:8004:17)  
at react-dom-client.development.js:8057:33  
at reconcileChildren (react-dom-client.development.js:8621:13)  
at updateFunctionComponent (react-dom-client.development.js:8914:7)  
at beginWork (react-dom-client.development.js:10522:18)  
at runWithFiberInDEV (react-dom-client.development.js:1522:13)  
at performUnitOfWork (react-dom-client.development.js:15132:22)  
at workLoopSync (react-dom-client.development.js:14956:41)
```

⚠ ▶ An error occurred in the `<ServerComponent>` [react-dom-client.development.js:8283](#)  
component.

Consider adding an error boundary to your tree to customize error handling behavior.  
Visit <https://react.dev/link/error-boundaries> to learn more about error boundaries.



## Antoine Thompson

*React Lead Developer - Fasst.*

Basketball Player and Darts Junkie.



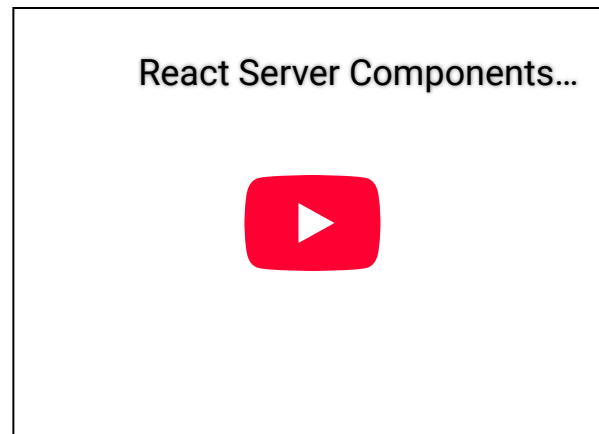
## Sébastien Auguste

*React Lead Developer - Fasst.*

Game Master and Occasional Brewer.






Vous avez déjà entendu parler des React Server Components.

Vous êtes déjà curieux à propos des React Server Components.



Retrouvez ces liens à la fin

# Les React Server Components expliqués à ma mère

-  Run exclusively on the server.
-  Allow database queries, API calls or whatsoever directly from the component.
-  Lighter bundle.
-  Incompatible with React's API : no state, no effects, no interactivity.
-  Render into a special data format : the payload.



# Historique des React Server Components

2020

## Présentation des RSC

Par la Core Team React

2021

## Expérimentation

Via Next.js comme terrain de test

2023

## Support Canary

RSC disponibles dans la version canary de React

2024

## React 19

Release officielle avec support complet des RSC



# Server Component

```
import LikeButton from '@app/ui/like-button'
import { getPost } from '@lib/data'

export default async function Page({ params }: { params: { id: string } }) {
  const post = await getPost(params.id)

  return (
    <div>
      <main>
        <h1>{post.title}</h1>
        { /* ... */ }
        <LikeButton likes={post.likes} />
      </main>
    </div>
  )
}
```

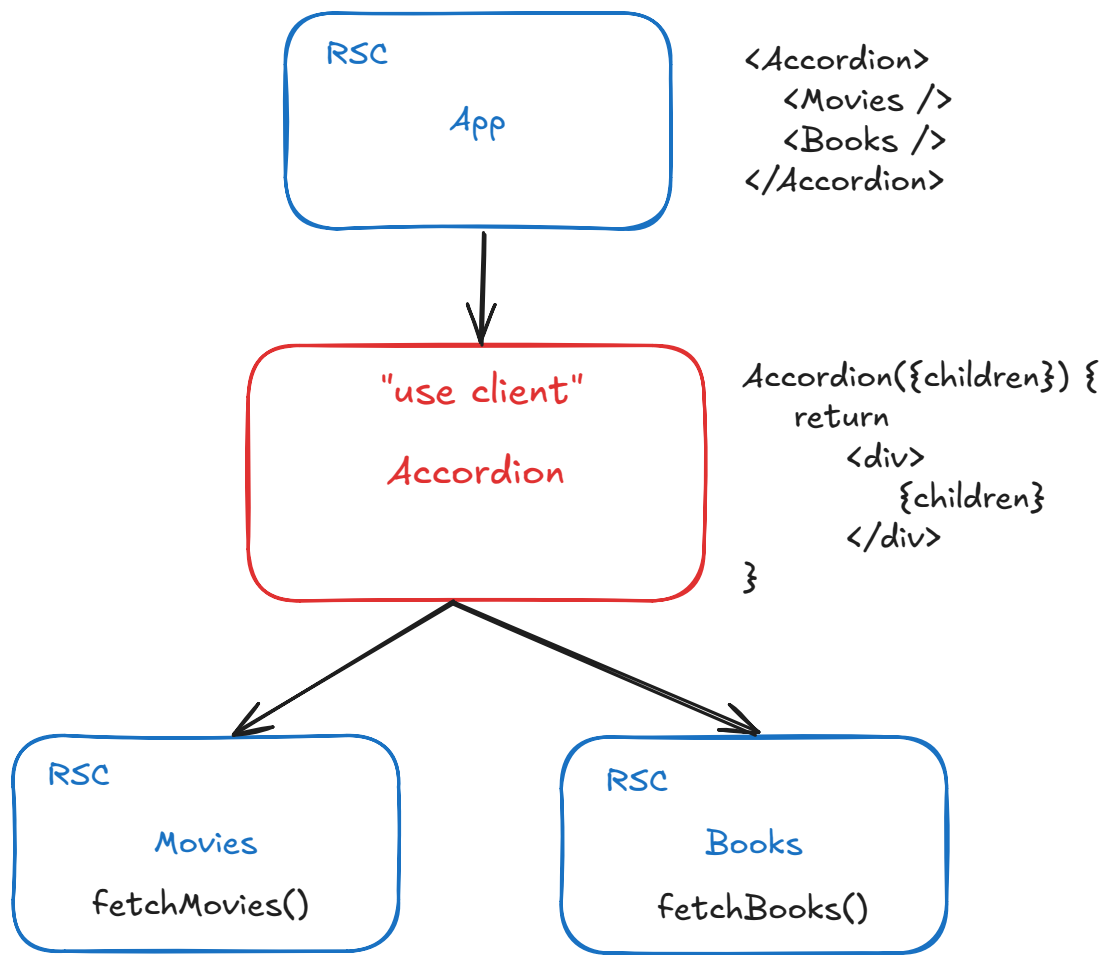




# Client Component

```
'use client'  
  
import { useState } from 'react'  
  
export default function LikeButton({ likes }: { likes: number }) {  
  // ...  
}
```





# Next.js

Support des RSC



Server functions



Composition



Ready for production



Binder : react-server-dom-turbopack en dev et react-server-dom-webpack en prod.





# Server Component

```
import db from 'some-db';
import { Gallery } from '../components/gallery';

export const Store = async () => {
  const products = await db.query('SELECT * FROM products');

  return <Gallery products={products} />;
};
```

# Client Component

```
'use client';

import { useState } from 'react';

export const Counter = () => {
  const [count, setCount] = useState(0);

  return (
    <div>Count: {count}</div>
    <button onClick={() => setCount((c) => c + 1)}>Increment</button>
  </div>
  );
};
```

# Composition

## WEAVING PATTERNS

Server components can import client components and doing so will create a server-client boundary. Client components cannot import server components, but they can accept server components as props such as `children`. For example, you may want to add global context providers this way.

# Waku

Support des RSC



Server functions



Composition



Ready for production



Binder : react-server-dom-webpack

Deployment : Vercel, Netlify, and Cloudflare, Partykit, Deno Deploy, AWS Lamba (all experimental).



# Roadmap

- Migrate to react-server-dom-vite binder.
- Migrate to Vite Environnement API.
- Waku Islands (?).
- Migrate from EsBuild to Rolldown.
- Jotai native integration.





# PARCEL

# Server Component

```
"use server-entry";

export async function Comments() {
  let comments = await db.getComments();

  return comments.map((comment) => (
    <article key={comment.id}>
      <p>Posted by: {comment.user}</p>
      {renderMarkdown(comment.body)}
    </article>
  ));
}
```

# Server

```
import express from "express";
import { renderRSC } from "@parcel/rsc/node";
import { Comments } from "../Comments";

const app = express();
app.get("/comments", (req, res) => {
  // Render the server component to an RSC payload.
  let stream = renderRSC(<Comments />);
  res.set("Content-Type", "text/x-component");
  stream.pipe(res);
});

app.listen(3000);
```

# App





```
import { Suspense } from 'react';
import { fetchRSC } from '@parcel/rsc/client';

export function App() {
  return (
    <
      <h1>Client rendered</h1>
      <Suspense fallback={<Loading comments ... </>>}>
        <Comments />
      </Suspense>
    </>
  );
}

let request = null;

function Comments() {
  request ??= fetchRSC('http://localhost:3000/comments');
  return request;
}
```

# Parcel

Support des RSC	
Server functions	
Composition	
Ready for production	
Binder : react-server-dom-parcel	



# Méthode 1 : La classique

```
export async function ServerComponent({ params }) {  
  let project = await loadProduct(params.projectId);  
  return (  
    <  
      <title>{project.name}</title>  
      <ProjectScreen project={project} />  
    </>  
  );  
}
```



# Méthode 2 : Les loaders

```
export async function loader({ params }) {
  let { contentBlocks, ...product } = await getProduct(params.productId);
  return {
    product,
    content: (
      <div>
        {contentBlocks.map((block) => {
          switch (block.type) {
            case "image":
              return <ImageBlock { ... block} />;
            case "gallery":
              return <GalleryBlock { ... block} />;
          }
        })}
      </div>
    ),
  };
}

export default function Article({ loaderData }) {
  return (
    <ProductLayout product={loaderData.product}>
      {loaderData.content}
    </ProductLayout>
  );
}
```

# React Router v7

Support des RSC



Server functions



Composition



Ready for production



Binder : react-server-dom-parcel



# Bighorn

the Next Development Epoch



# Cell

```
import { Link, routes } from '@redwoodjs/web'

export const QUERY = gql`
  query GetProducts {
    products {
      id
      name
    }
  }
`

export const Loading = () => <div>Loading products ... </div>
export const Failure = (error) => <div>Something went wrong: {error.message}</div>
export const Empty = () => <div>No products found</div>
export const Success = ({ products }) => (
  <ul>
    {products.map((product) => (
      <li key={product.id}>
        <Link to={routes.product({id: product.id})}>
          {product.name}
        </Link>
      </li>
    )}
  </ul>
)
```



# Server Component

```
import { Link, routes } from '@redwoodjs/web'

export const data = () => {
  return { products: db.products.findMany() }
}

export const Loading = () => <div>Loading products ... </div>
export const Failure = (error) => <div>Something went wrong: {error.message}</div>
export const Empty = () => <div>No products found</div>
export const Success = ({ products }) => (
  <ul>
    {products.map((product) => (
      <li key={product.id}>
        <Link to={routes.product({id: product.id})}>
          {product.name}
        </Link>
      </li>
    ))}
  </ul>
)
```



# Redwoodjs

Support des RSC



Server functions



Composition



Ready for production



Binder : react-server-dom-webpack



# Roadmap

- Server functions full support.
- Adapt integrated auth to RSC.
- Generators creation.



# And also

- Expo.
- TanStack.
- Vinxi.
- Twofold.
- Kotekan.

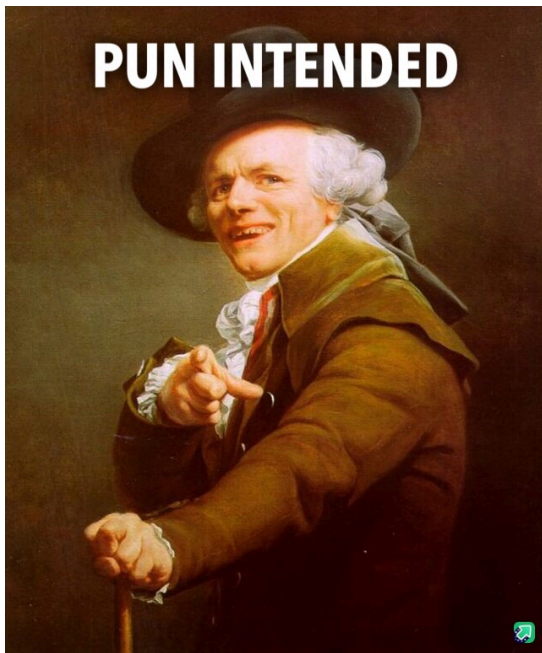


# Wrap-up

- Ready for production ? **Next.js**.
- Add to an existing project ? **Parcel**.
- Something minimalist, RSC-opinionated ? **Waku**.
- Something opinionated, data-oriented with scaffolding functionalities ? **Redwoodjs**.
- Something between all of that ? **React Router v7**.



# What's Next ?



- Vercel's influence.
- Developer EXperience ?
- Developer support ?



Slides & References



Open Feedback