Fetching data with Next.js

In Next.js 15,
Fetching Data with Server Components. Also provide following :
1. Loading state
2. Error state

In **Next.js 15**, you can fetch data using **Server Components** by default (as most components in the app directory are Server Components unless marked otherwise). Here's how you can fetch data, along with handling **loading** and **error** states using the `<Suspense>` and `<ErrorBoundary>` pattern.

✅ Summary

You will:

1. Create a **Server Component** that fetches data.
2. Wrap it with `<Suspense>` for a loading state.
3. Wrap it with a **custom Error Boundary** to handle errors.

🔧 1. Server Component Example with Fetching Data

`/app/page.tsx` – Main Page

tsx

```tsx
import React, { Suspense } from 'react';
import Users from './components/Users';
import Loading from './loading';
import ErrorBoundary from './components/ErrorBoundary';

export default function HomePage() {
  return (
    <main>
      <h1>Users</h1>

      <ErrorBoundary>
        <Suspense fallback={<Loading />}>
```

```
        <Users />
      </Suspense>
    </ErrorBoundary>
  </main>
 );
}
```

🔹 2. Server Component: `/app/components/Users.tsx`

```tsx
// Server Component (by default in /app)
const getUsers = async () => {
 const res = await fetch('https://jsonplaceholder.typicode.com/users');

 if (!res.ok) {
  throw new Error('Failed to fetch users');
 }

 return res.json();
};

export default async function Users() {
 const users = await getUsers();

 return (
  <ul>
   {users.map((user: any) => (
    <li key={user.id}>{user.name}</li>
   ))}
  </ul>
 );
}
```

🌀 3. Loading State: `/app/loading.tsx`

Next.js automatically shows this during Suspense fallback.

```tsx
```

```tsx
export default function Loading() {
  return <p>Loading users...</p>;
}
```

❌ 4. Error State: `/app/components/ErrorBoundary.tsx`

You need to create a client component that can catch errors.

```tsx
'use client';

import React from 'react';

type Props = {
  children: React.ReactNode;
};

type State = {
  hasError: boolean;
  error: Error | null;
};

export default class ErrorBoundary extends React.Component<Props, State> {
  constructor(props: Props) {
    super(props);
    this.state = { hasError: false, error: null };
  }

  static getDerivedStateFromError(error: Error): State {
    return { hasError: true, error };
  }

  render() {
    if (this.state.hasError) {
      return <p>Something went wrong: {this.state.error?.message}</p>;
    }

    return this.props.children;
```

```
  }
}
```

✅ Final Result:

When the data is:

- **Loading** → it shows the `<Loading />` component.
- **Fetched successfully** → it displays the user list.
- **Error occurs** → it displays the error message inside the `ErrorBoundary`.

Let me know if you'd like to adapt this to **client-side fetching** or handle **retry** logic as well.