Fetch error troubleshooting

Runtime TypeError
Server

fetch failed

src\app\users-server\page.tsx (15:22) @ usersServer

```
  13 |
  14 | export default async function usersServer() {
> 15 |     const response = await
fetch("https://jsonplaceholder.typicode.com/users")
     |                   ^
  16 |     const users : User[] = await response.json();
  17 |
  18 |     console.log(users);
```

Call Stack
6

Show 5 ignore-listed frame(s)
usersServer
src\app\users-server\page.tsx (15:22)

,

```
type User = {
    name: string,
    username: string,
    email: string,
    phone: string,
    address: {
        city: string,
    },
    company: {
        name: string,
    },
}
```

```
export default async function usersServer() {
   const response = await
fetch("https://jsonplaceholder.typicode.com/users")
   const users : User[] = await response.json();

   console.log(users);
   return <div>Server components</div>
}
```

You're encountering a `TypeError: fetch failed` in your Next.js `page.tsx` file when trying to use the Fetch API in a **server component**.

🔍 Cause of the Error

In server components (like your `usersServer` function), the `fetch` call is failing. This usually happens for one of the following reasons:

✅ Fixes to Try

✅ 1. Check Node.js version

Make sure your environment is running **Node.js 18 or higher**. Native `fetch` support was added in Node 18.

- You can check your version with:

  ```bash
  node -v
  ```

- If you're using an older version (like 16), either upgrade Node or use a library like `node-fetch`.

✅ 2. Handle fetch errors gracefully

Wrap your fetch in a try-catch block to prevent the app from crashing and log detailed errors:

```ts
```

```
export default async function usersServer() {
  try {
    const response = await fetch("https://jsonplaceholder.typicode.com/users");

    if (!response.ok) {
      throw new Error(`Fetch failed with status: ${response.status}`);
    }

    const users: User[] = await response.json();
    console.log(users);
  } catch (error) {
    console.error("Error fetching users:", error);
  }

  return <div>Server components</div>;
}
```

✅ 3. Check network environment

- If you're running this locally in an offline or firewalled environment, the fetch call to `https://jsonplaceholder.typicode.com/users` will fail.
- Try accessing the URL directly in a browser or with `curl` to confirm it's reachable.

✅ 4. Use Next.js recommended `fetch` pattern (optional)

For server-side data fetching, make sure you're using server components (which you are), or fetch inside `getServerSideProps` / `getStaticProps` if using pages.

✅ Final Thoughts

If you're using a correct Node version and still getting `fetch failed`, it's likely due to:

- A bad network / VPN / proxy setting
- Running inside a container with no internet access

Let me know if you'd like help checking those.

```
PS C:\Users\shubh> node -v
v22.16.0
PS C:\Users\shubh>
```

Great — you're running **Node.js v22.16.0**, which **fully supports the native `fetch` API**, so the version is not the issue. ✅

Since you're still getting a `TypeError: fetch failed`, the most likely causes now are:

🔍 Most Probable Causes

1. No internet access or DNS resolution issue
Your server component is trying to fetch `https://jsonplaceholder.typicode.com/users` — if that URL can't be reached due to lack of internet or DNS issues, fetch will fail.

👉 **Try this:**
Run this in your terminal to verify connectivity:

```bash
curl https://jsonplaceholder.typicode.com/users
```

Or just open that URL in your browser.

---

2. Firewall, Proxy, or VPN interference

- If you're behind a corporate network, VPN, or using a proxy, it might block external HTTP requests.
- Some environments (like strict enterprise settings) block requests made from Node.js apps.

👉 **Try temporarily disabling** any VPN or proxy and rerun your app.

---

3. SSL certificate issue
Sometimes, Node.js has trouble validating SSL certificates in some environments.

👉 **Quick fix (for testing only):**

Try setting this **just for testing** to bypass SSL verification:

```ts
process.env['NODE_TLS_REJECT_UNAUTHORIZED'] = '0';
```

Add it **before** the fetch call:

```ts
process.env['NODE_TLS_REJECT_UNAUTHORIZED'] = '0';

const response = await fetch("https://jsonplaceholder.typicode.com/users");
```

⚠ **Don't use this in production** — it disables SSL checks and is insecure.

---

4. Check if `fetch` is polyfilled or monkey-patched

Make sure you're not overriding `fetch` accidentally in your code or config.

🛠 Updated Safe Version with Debugging

Here's a safe version with error logging:

```ts
type User = {
  name: string,
  username: string,
  email: string,
  phone: string,
  address: {
    city: string,
  },
  company: {
    name: string,
  },
};

export default async function usersServer() {
```

```
  try {
    const response = await fetch("https://jsonplaceholder.typicode.com/users");

    if (!response.ok) {
      throw new Error(`Failed to fetch users: ${response.statusText}`);
    }

    const users: User[] = await response.json();
    console.log(users);
  } catch (error) {
    console.error("Fetch error:", error);
  }

  return <div>Server components</div>;
}
```

Would you like to test the `curl` command or disable SSL just to confirm it's a network/SSL issue? I can walk you through that.

```
type User = {
  name: string,
  username: string,
  email: string,
  phone: string,
  address: {
    city: string,
  },
  company: {
    name: string,
  },
};

export default async function usersServer() {
  try {
    const response = await
fetch("https://jsonplaceholder.typicode.com/users");

    if (!response.ok) {
```

```
      throw new Error(Failed to fetch users: ${response.statusText});
    }

    const users: User[] = await response.json();
  } catch (error) {
    console.error("Fetch error:", error);
  }

  return (
    <div className="m-15 p-3 bg-black text-center">
 <ul className="flex flex-wrap gap-4 m-auto">
  {users.map((user) => (
   <li
    key={user.id}
    className="w-64 bg-white shadow-md rounded-lg p-4 text-gray-700"
   >
    <div className="font-bold text-gray-900">{user.name}</div>
    <div className="text-sm mt-2 space-y-1">
     <div>Username: {user.username}</div>
     <div>Email: {user.email}</div>
     <div>Phone: {user.phone}</div>
     <div>City: {user.address.city}</div>
     <div>Company: {user.company.name}</div>
    </div>
   </li>
  ))}
 </ul>
</div>
  );
}
```

You're very close to what you want, but there's a **key issue**:

❌ Problem

You're trying to use `users.map(...)` **outside** the `try` block — but `users` is declared **inside** the `try`, so it doesn't exist in the outer scope.

If the fetch fails, `users` will be undefined, and even if it works, you'll get a TypeScript error like:

```lua
Cannot find name 'users'
```

✅ Solution

You need to declare `users` **outside** the `try` block so it can be accessed in the return JSX.

Here's the **fixed and complete version**:

✅ Corrected Code

```tsx
type User = {
  id: number;
  name: string;
  username: string;
  email: string;
  phone: string;
  address: {
    city: string;
  };
  company: {
    name: string;
  };
};

export default async function usersServer() {
  let users: User[] = [];

  try {
    const response = await fetch("https://jsonplaceholder.typicode.com/users");

    if (!response.ok) {
```

```
      throw new Error(`Failed to fetch users: ${response.statusText}`);
    }

    users = await response.json();
  } catch (error) {
    console.error("Fetch error:", error);
  }

  return (
    <div className="m-15 p-3 bg-black text-center">
      <ul className="flex flex-wrap gap-4 m-auto">
        {users.map((user) => (
          <li
            key={user.id}
            className="w-64 bg-white shadow-md rounded-lg p-4 text-gray-700"
          >
            <div className="font-bold text-gray-900">{user.name}</div>
            <div className="text-sm mt-2 space-y-1">
              <div>Username: {user.username}</div>
              <div>Email: {user.email}</div>
              <div>Phone: {user.phone}</div>
              <div>City: {user.address.city}</div>
              <div>Company: {user.company.name}</div>
            </div>
          </li>
        ))}
      </ul>
    </div>
  );
}
```

✅ Changes Made:

1. Declared `users` **outside** the try block.
2. Added `id: number` to `User` type (it's required for `key={user.id}`).
3. Code styling improvements for readability.

Let me know if you also want to handle the error case visually (e.g. display a message when `users` is empty).

Explain Request Memoization

🧠 What Is Request Memoization?

Request memoization refers to **caching the result of a data-fetching operation** (like `fetch`) so that **the same request isn't repeated multiple times during a render cycle** — especially on the server.

This is **especially important in frameworks like Next.js** (particularly in server components), where a component might be **rendered multiple times**, and redundant fetches can hurt performance or cause bugs.

🔁 Without Memoization:

If you do this:

```ts
const users1 = await fetch('https://api.com/users');
const users2 = await fetch('https://api.com/users');
```

Both `fetch` calls **will hit the network**, even though they're the same. This is **wasteful**.

✅ With Request Memoization:

With memoization:

```ts
const users1 = await fetch('https://api.com/users'); // Fetches from network
const users2 = await fetch('https://api.com/users'); // Returned from cache
```

The second call **reuses** the result of the first — saving bandwidth, time, and CPU.

📦 In Next.js — It Happens Automatically

Next.js **automatically memoizes fetch calls** in **server components** by default (when using the native `fetch` API). So:

```ts
await fetch('https://api.com/data');
await fetch('https://api.com/data');
```

Only one network call is made, **as long as the URL and options are identical**.

This only works:

- In server components (or `getServerSideProps`)
- When `fetch` is used natively (not via Axios or custom wrapper)
- When fetch options (headers, body, etc.) are the same

🔧 You Can Control It

You can **customize memoization behavior** using the `next` option in `fetch`:

```ts
await fetch('https://api.com/data', {
  next: {
    revalidate: 60, // cache and revalidate after 60 seconds
  }
});
```

Or disable it:

```ts
await fetch('https://api.com/data', {
  cache: 'no-store' // disables memoization
});
```

🧠 Why It Matters

Memoization avoids:

- **Duplicate API calls**

- **Longer load times**
- **Extra server load**
- **Unnecessary database queries (in backend APIs)**

It's especially useful in:

- SSR (server-side rendering)
- Server components
- Incremental Static Regeneration (ISR)

🧪 Bonus: Custom Memoization

If you're not using `fetch`, or using Axios or custom logic, you can implement your own memoization with something like this:

```ts
const cache = new Map();

async function memoizedFetch(url: string) {
  if (cache.has(url)) return cache.get(url);
  const res = await fetch(url);
  const data = await res.json();
  cache.set(url, data);
  return data;
}
```

Summary

Feature	With Memoization
Network calls	Avoid duplicates
Performance	Faster rendering
Memory/cache usage	More efficient
Default in Next.js?	✅ Yes (for fetch in server components)

Let me know if you want a live code example or comparison demo!