Active link styling

how to apply 'isActive' css styling on these link :

```
"use client";
import Link from "next/link";
import { usePathname } from "next/navigation";

export default function Navbar() {
  const pathname = usePathname();

  return (
    <nav className="w-full bg-gray-900 shadow-md p-4 flex justify-between items-center">
      <div className="text-xl font-bold">
        <Link href="/">Practice-App_1</Link>
      </div>

      <div className="space-x-4">
        <Link href="/about" replace>About</Link>
        <Link href="/products">Products</Link>
        <Link href="/docs">Documentation</Link>
        <Link href="/counter">Counter</Link>
      </div>

      <div className="space-x-4">
        <Link href="/login">Sign-In</Link>
        <Link href="/register">Sign-Up</Link>
        <Link href="/profile">Profile</Link>
      </div>
    </nav>
  );
}
```

To apply an `isActive` CSS class to links based on the current route (`pathname`), you can conditionally add a class to each `<Link>` component using `pathname` from `usePathname()`.

Here’s how you can update your `Navbar` component to apply an `isActive` class to the active link:

✅ 1. Define a Utility Function (Optional but Clean)

To avoid repeating code, you can define a helper function to determine if a link is active.

✅ 2. Add Conditional Class Names

Use `className` with a conditional expression to apply styles.

✅ Updated `Navbar` Component Example

```tsx
"use client";
import Link from "next/link";
import { usePathname } from "next/navigation";
import clsx from "clsx"; // optional, for cleaner className handling

export default function Navbar() {
  const pathname = usePathname();

  const isActive = (href: string) => pathname === href;

  const linkClass = (href: string) =>
    clsx(
      "text-white px-3 py-2 rounded-md hover:bg-gray-700 transition",
      {
        "bg-blue-600 font-semibold": isActive(href),
      }
    );

  return (
    <nav className="w-full bg-gray-900 shadow-md p-4 flex justify-between items-center">
      <div className="text-xl font-bold text-white">
        <Link href="/" className={linkClass("/")}>
          Practice-App_1
        </Link>
      </div>

      <div className="space-x-4">
        <Link href="/about" className={linkClass("/about")}>
          About
        </Link>
        <Link href="/products" className={linkClass("/products")}>
          Products
        </Link>
        <Link href="/docs" className={linkClass("/docs")}>
          Documentation
        </Link>
        <Link href="/counter" className={linkClass("/counter")}>
          Counter
        </Link>
      </div>

      <div className="space-x-4">
        <Link href="/login" className={linkClass("/login")}>
          Sign-In
        </Link>
        <Link href="/register" className={linkClass("/register")}>
          Sign-Up
        </Link>
        <Link href="/profile" className={linkClass("/profile")}>
          Profile
        </Link>
      </div>
    </nav>
  );
}
```

✅ Tailwind Styles Explained

- Default: `text-white px-3 py-2 rounded-md hover:bg-gray-700 transition`
- Active: `bg-blue-600 font-semibold` (highlight active link)

🧩 Optional: Use `startsWith()` for Parent Route Matching

If you want parent routes to match child routes (e.g., `/products/1` should activate `/products`), modify `isActive`:

```ts
```

```
const isActive = (href: string) => pathname.startsWith(href);
```

Just be careful with `/` and `/about` — you might need edge-case handling.

Let me know if you want this styled differently or made into a component!

is there any other way

Yes, there are **several other ways** to apply active link styling in a Next.js app (especially with App Router). Here are **3 alternative approaches**, each with trade-offs:

✅ Option 1: Use `usePathname()` + Inline ClassName (Without helper or `clsx`)

If you don't want to use `clsx` or create helper functions:

```tsx
<Link
  href="/about"
  className={`px-3 py-2 text-white rounded-md hover:bg-gray-700 transition ${
    pathname === "/about" ? "bg-blue-600 font-semibold" : ""
  }`}
>
  About
</Link>
```

Pros: Simple, no extra imports

Cons: Repetitive if used in many places

✅ Option 2: Create a Custom `ActiveLink` Component

Encapsulate the active logic in a reusable component.

1. Create `components/ActiveLink.tsx`

```tsx
'use client';
import Link from 'next/link';
import { usePathname } from 'next/navigation';
import { ReactNode } from 'react';

interface Props {
  href: string;
  children: ReactNode;
  exact?: boolean;
}

export default function ActiveLink({ href, children, exact = true }: Props) {
  const pathname = usePathname();
  const isActive = exact ? pathname === href : pathname.startsWith(href);

  return (
    <Link
      href={href}
      className={`px-3 py-2 text-white rounded-md hover:bg-gray-700 transition ${
        isActive ? 'bg-blue-600 font-semibold' : ''
      }`}
    >
      {children}
    </Link>
  );
}
```

2. Use It in Your `Navbar`

```tsx
import ActiveLink from './ActiveLink';

<ActiveLink href="/about">About</ActiveLink>
<ActiveLink href="/products">Products</ActiveLink>
```

Pros: DRY, scalable, reusable

Cons: Slight abstraction, but easy to maintain

✅ Option 3: Using Tailwind's `group` + `aria-current` for Semantics

Next.js `Link` supports `aria-current="page"` which you can use for accessibility **and styling**:

```tsx
<Link
  href="/about"
  aria-current={pathname === "/about" ? "page" : undefined}
  className="group px-3 py-2 text-white rounded-md hover:bg-gray-700 transition
          aria-[current=page]:bg-blue-600 aria-[current=page]:font-semibold"
>
  About
</Link>
```

Pros: Semantic HTML, accessible, uses Tailwind utilities

Cons: Less known, might confuse new devs

🧠 Summary: Which Should You Use?

Method	Use If...
Inline ClassName	You want simplicity for a small number of links
`clsx` + helper function	You want clean conditional logic without extra components
`ActiveLink` component	You want reusability and clean JSX
`aria-current` with Tailwind	You care about accessibility and want a semantic approach

---

Let me know your preferences (e.g., no extra packages, minimal JSX, or semantic HTML), and I'll tailor it further for you.