16. 페이지 간 이동하기

지난 챕터까지

대시보드 레이아웃과 페이지를 만들었습니다.
이제 사용자가 **대시보드 경로들 간 이동**할 수 있도록 링크를 추가할 차례입니다.

이 챕터에서 배울 것

- `next/link` 컴포넌트 사용 방법
- `usePathname()` 훅으로 현재 활성 링크 표시
- Next.js 내비게이션 작동 방식

왜 내비게이션을 최적화해야 할까?

전통적인 방법으로는 `<a>` 태그를 사용해 페이지를 이동합니다.
현재 사이드바에 있는 링크들도 `<a>` 태그를 사용 중입니다.

직접 홈, 인보이스, 커스터머 페이지를 이동해보세요.

봤나요?

- 매번 **전체 페이지 리로드**가 발생합니다.

`<Link>` 컴포넌트

Next.js에서는 **`<Link />` 컴포넌트**를 사용해서 페이지 간 이동을 합니다.
`<Link>`를 사용하면 **클라이언트 사이드 내비게이션**을 수행할 수 있습니다.

사용 방법:

1. `/app/ui/dashboard/nav-links.tsx` 파일을 엽니다.
2. `next/link` 모듈에서 `Link`를 import 합니다.
3. 기존 `<a>` 태그를 `<Link>`로 교체합니다.

/app/ui/dashboard/nav-links.tsx

```
import {
  UserGroupIcon,
  HomeIcon,
  DocumentDuplicateIcon,
} from '@heroicons/react/24/outline';
import Link from 'next/link';

// ...

export default function NavLinks() {
  return (
    <>
      {links.map((link) => {
        const LinkIcon = link.icon;
        return (
          <Link
            key={link.name}
            href={link.href}
            className="flex h-[48px] grow items-center justify-center gap-
2 rounded-md bg-gray-50 p-3 text-sm font-medium hover:bg-sky-100
hover:text-blue-600 md:flex-none md:justify-start md:p-2 md:px-3"
          >
            <LinkIcon className="w-6" />
            <p className="hidden md:block">{link.name}</p>
          </Link>
        );
      })}
    </>
  );
}
```

변경 후 결과:

- 전체 페이지 리프레시 없이 페이지가 전환됩니다.
- 앱이 **더 빠르고 부드럽게** 느껴집니다.
- 서버 렌더링을 하더라도 클라이언트 전환이 가능합니다.

자동 코드 분할(Automatic code-splitting)과 프리페치(prefetching)

Next.js는 경로(route) 단위로 코드를 자동 분할합니다.

전통적인 React SPA는 처음에 모든 코드를 다운로드하지만,
Next.js는 필요한 경로의 코드만 로딩합니다.

장점:

- 페이지 하나가 오류를 내도 앱 전체가 멈추지 않음
- 브라우저가 해석해야 할 코드량이 줄어듦 → 로딩 속도 개선

또한, 프로덕션 환경에서는 `<Link>`가 뷰포트에 들어오는 순간
해당 경로의 코드를 **백그라운드에서 미리 프리페치**합니다.

→ 사용자가 클릭하기 전에 이미 코드가 로드 완료!

패턴: 현재 활성 링크 표시하기

일반적으로, 현재 사용자가 보고 있는 페이지를
사이드바 등에서 **하이라이트**하는 UI를 만듭니다.

이를 위해 **현재 URL 경로**를 가져와야 합니다.

Next.js는 `usePathname()`라는 훅을 제공합니다.

Client Component로 변환하기

`usePathname()`은 훅이기 때문에,
`nav-links.tsx`를 **클라이언트 컴포넌트**로 바꿔야 합니다.

수정:

1. 파일 맨 위에 `'use client';` 추가
2. `next/navigation`에서 `usePathname()` import
3. 컴포넌트 내부에서 `pathname` 변수에 현재 경로 저장

/app/ui/dashboard/nav-links.tsx

```
'use client';

import {
  UserGroupIcon,
  HomeIcon,
  DocumentDuplicateIcon,
} from '@heroicons/react/24/outline';
import Link from 'next/link';
```

```
import { usePathname } from 'next/navigation';

// ...

export default function NavLinks() {
  const pathname = usePathname();
  // ...
}
```

활성 링크 스타일링하기

`clsx` 라이브러리를 이용해,
현재 경로(`pathname`)가 `link.href` 와 같으면
파란색 텍스트 + 연한 파란색 배경을 적용합니다.

최종 코드:

/app/ui/dashboard/nav-links.tsx

```
'use client';

import {
  UserGroupIcon,
  HomeIcon,
  DocumentDuplicateIcon,
} from '@heroicons/react/24/outline';
import Link from 'next/link';
import { usePathname } from 'next/navigation';
import clsx from 'clsx';

// ...

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

  return (
    <>
      {links.map((link) => {
        const LinkIcon = link.icon;
        return (
          <Link
            key={link.name}
            href={link.href}
            className={clsx(
              'flex h-[48px] grow items-center justify-center gap-2
rounded-md bg-gray-50 p-3 text-sm font-medium hover:bg-sky-100 hover:text-
```

```
blue-600 md:flex-none md:justify-start md:p-2 md:px-3',
              {
                'bg-sky-100 text-blue-600': pathname === link.href,
              },
            )}
          >
            <LinkIcon className="w-6" />
            <p className="hidden md:block">{link.name}</p>
          </Link>
        );
      })}
    </>
  );
}
```

결과:

- 현재 페이지의 링크는 배경이 연한 파란색, 텍스트는 파란색으로 하이라이트 됩니다.
- 다른 링크들은 기본 스타일을 유지합니다.