

API 참고서 > 컴포넌트 >

# <ViewTransition>

## Canary

The `<ViewTransition />` API is currently only available in React's Canary and Experimental channels.

[Learn more about React's release channels here.](#)

<ViewTransition> 을 사용하면 Transition 내부에서 업데이트되는 엘리먼트에 애니메이션을 적용할 수 있습니다.

```
import {ViewTransition} from 'react';

<ViewTransition>
  <div>...</div>
</ViewTransition>
```

- [레퍼런스](#)
  - [<ViewTransition>](#)
  - [View Transition 클래스](#)
  - [View Transition 스타일링](#)
- [사용법](#)
  - [enter/exit에서 엘리먼트 애니메이션 적용하기](#)
  - [공유 엘리먼트 애니메이션 적용하기](#)
  - [목록에서 항목 순서 변경 애니메이션 적용하기](#)

- Suspense 콘텐츠에서 애니메이션 적용하기
  - 애니메이션 제외하기
  - 애니메이션 커스터마이징
  - 타입으로 애니메이션 커스터마이징하기
  - View Transition 지원 라우터 구축하기
- 문제 해결
    - <ViewTransition> 이 활성화되지 않습니다
    - ”동일한 이름으로 마운트된 <ViewTransition name=%s> 컴포넌트가 두 개 있습니다.”라는 오류가 발생합니다

---

## 레퍼런스

### <ViewTransition>

엘리먼트를 <ViewTransition> 으로 감싸면 Transition 내부에서 업데이트할 때 애니메이션을 적용할 수 있습니다. React는 다음 휴리스틱을 사용하여 View Transition이 애니메이션에 활성화되는지 판단합니다.

- enter : 해당 Transition에서 ViewTransition 자체가 삽입되면 활성화됩니다.
- exit : 해당 Transition에서 ViewTransition 자체가 삭제되면 활성화됩니다.
- update : ViewTransition 내부에서 React가 수행하는 DOM 변경(예: 프로퍼티 변경)이 있거나 인접한 형제 엘리먼트의 영향으로 ViewTransition 경계 자체의 크기나 위치가 변경되는 경우 활성화됩니다. 중첩된 ViewTransition 이 있으면 변경이 부모가 아닌 해당 항목에 적용됩니다.
- share : 이름이 지정된 ViewTransition 이 삭제된 서브트리 내부에 있고 같은 이름을 가진 다른 이름 있는 ViewTransition 이 같은 Transition에서 삽입된 서브트리의 일부인 경우 공유 엘리먼트 Transition을 형성하며, 삭제된 것에서 삽입된 것으로 애니메이션됩니다.

기본적으로 <ViewTransition> 은 부드러운 크로스 페이드(브라우저 기본 View Transition)로 애니메이션됩니다. <ViewTransition> 컴포넌트에 [View Transition 클래스](#)를 제공하여 애니메이션을 커스터마이징할 수 있습니다. 각 트리거 유형에 대해 애니메이션을 커스터마이징할 수 있습니다([View Transition 스타일링](#) 참고).

## □ 자세히 살펴보기

### <ViewTransition> 은 어떻게 작동하나요?

자세히 보기

## Props

기본적으로 <ViewTransition> 은 부드러운 크로스 페이드로 애니메이션됩니다. 이러한 프로퍼티로 애니메이션을 커스터마이즈하거나 공유 엘리먼트 Transition을 지정할 수 있습니다.

- **optional** `enter`: 문자열 또는 객체. “enter”가 활성화될 때 적용할 [View Transition 클래스](#)입니다.
- **optional** `exit`: 문자열 또는 객체. “exit”이 활성화될 때 적용할 [View Transition 클래스](#)입니다.
- **optional** `update`: 문자열 또는 객체. “update”가 활성화될 때 적용할 [View Transition 클래스](#)입니다.
- **optional** `share`: 문자열 또는 객체. 공유 엘리먼트가 활성화될 때 적용할 [View Transition 클래스](#)입니다.
- **optional** `default`: 문자열 또는 객체. 다른 일치하는 활성화 프로퍼티가 없을 때 사용되는 [View Transition 클래스](#)입니다.
- **optional** `name`: 문자열 또는 객체. 공유 엘리먼트 transition에 사용되는 View Transition의 이름입니다. 제공되지 않으면 React는 예상치 못한 애니메이션을 방지하기 위해 각 View Transition에 대해 고유한 이름을 사용합니다.

## 콜백

이 콜백을 사용하면 [animate API](#)를 사용하여 애니메이션을 명령적으로 조정할 수 있습니다.

- **optional** `onEnter`: 함수. React는 “enter” 애니메이션 후에 `onEnter` 를 호출합니다.
- **optional** `onExit`: 함수. React는 “exit” 애니메이션 후에 `onExit` 를 호출합니다.
- **optional** `onShare`: 함수. React는 “share” 애니메이션 후에 `onShare` 를 호출합니다.
- **optional** `onUpdate`: 함수. React는 “update” 애니메이션 후에 `onUpdate` 를 호출합니다.

각 콜백은 다음을 인수로 받습니다.

- element : 애니메이션된 DOM 엘리먼트입니다.
- types : 애니메이션에 포함된 Transition 타입입니다.

## View Transition 클래스

View Transition 클래스는 ViewTransition이 활성화될 때 Transition 중에 React가 적용하는 CSS 클래스 이름입니다. 문자열 또는 객체일 수 있습니다.

- string : 활성화될 때 자식 엘리먼트에 추가되는 class입니다. 'none'이 제공되면 클래스가 추가되지 않습니다.
- object : 자식 엘리먼트에 추가되는 클래스는 addTransitionType으로 추가된 View Transition 타입과 일치하는 키입니다. 객체는 일치하는 타입이 없을 때 사용할 default도 지정할 수 있습니다.

값 'none'은 특정 트리거에 대해 View Transition이 활성화되지 않도록 하는 데 사용할 수 있습니다.

## View Transition 스타일링

### ▣ 중요합니다!

웹에서 View Transition의 많은 초기 예시에서 `view-transition-name`을 사용한 다음 `::view-transition-... (my-name)` 선택자를 사용하여 스타일을 지정하는 것을 볼 수 있습니다. 그러나 이러한 방식으로 스타일링하는 것을 권장하지 않습니다. 대신, 일반적으로 View Transition 클래스를 사용하는 것을 권장합니다.

<ViewTransition>의 애니메이션을 커스터マイ즈하려면 활성화 프로퍼티 중 하나에 View Transition 클래스를 제공할 수 있습니다. View Transition 클래스는 ViewTransition이 활성화될 때 React가 자식 엘리먼트에 적용하는 CSS 클래스 이름입니다.

예를 들어 “enter” 애니메이션을 커스터マイ즈하려면 enter 프로퍼티에 클래스 이름을 제공합니다.

```
<ViewTransition enter="slide-in">
```

<ViewTransition>이 “enter” 애니메이션을 활성화하면 React는 클래스 이름 slide-in 을 추가합니다. 그런 다음 [View Transition 가상 선택자](#)를 사용하여 이 클래스를 참조하여 재사용 가능한 애니메이션을 구축할 수 있습니다.

```
::view-transition-group(.slide-in) {  
}  
::view-transition-old(.slide-in) {  
}  
::view-transition-new(.slide-in) {  
}
```

향후 CSS 라이브러리에서 View Transition 클래스를 사용한 내장 애니메이션을 추가하여 사용하기 쉽게 만들 수 있습니다.

## 주의 사항

- 기본적으로 setState 는 즉시 업데이트되며 <ViewTransition> 을 활성화하지 않습니다. [Transition](#)으로 감싼 업데이트만 활성화됩니다. <Suspense> ](/reference/react/Suspense) 를 사용하여 Transition을 선택적으로 적용하고 [콘텐츠를 한 번에 표시](#)할 수도 있습니다.
- <ViewTransition> 은 이동, 크기 조정 및 크로스-페이드가 가능한 이미지를 생성합니다. React Native나 Motion에서 볼 수 있는 레이아웃 애니메이션과 달리, 내부의 모든 개별 요소가 위치를 애니메이션하는 것이 아닙니다. 이는 모든 개별 요소를 애니메이션하는 것에 비해 더 나은 성능과 더 연속적이고 부드러운 애니메이션을 제공할 수 있습니다. 하지만 독립적으로 움직여야 하는 요소들의 연속성을 잃을 수도 있습니다. 따라서 수동으로 더 많은 <ViewTransition> 경계를 추가해야 할 수 있습니다.
- 많은 사용자가 페이지의 애니메이션을 선호하지 않을 수 있습니다. React는 이 경우에 대해 자동으로 애니메이션을 비활성화하지 않습니다. 사용자 선호도에 따라 애니메이션을 비활성화하거나 줄이기 위해 @media (prefers-reduced-motion) 미디어 쿼리를 사용할 것을 권장 합니다. 향후 CSS 라이브러리에서 프리셋에 이 기능이 내장될 수 있습니다.
- 현재 <ViewTransition> 은 DOM에서만 작동합니다. React Native 및 기타 플랫폼에 대한 지원을 추가하기 위해 작업 중입니다.

# 사용법

## enter/exit에서 엘리먼트 애니메이션 적용하기

Enter/Exit Transition은 <ViewTransition>이 Transition에서 컴포넌트에 의해 추가되거나 제거될 때 발생합니다.

```
function Child() {
  return (
    <ViewTransition>
      <div>Hi</div>
    </ViewTransition>
  );
}

function Parent() {
  const [show, setShow] = useState();
  if (show) {
    return <Child />;
  }
  return null;
}
```

setShow가 호출되면 show가 true로 바뀌고 Child 컴포넌트가 렌더링됩니다. setShow가 startTransition 내부에서 호출되고 Child가 다른 DOM 노드보다 먼저 ViewTransition을 렌더링하면 enter 애니메이션이 발생합니다.

show가 다시 false로 바뀌면 exit 애니메이션이 발생합니다.

### App.js

↪ 새로고침 × Clear ✎ 포크

```
import {
  ViewTransition,
  useState,
  startTransition
} from 'react';
import {Video} from "./Video";
```

```
import videos from "./data"
```

```
function Item() {  
  return (  
    <ViewTransition>
```

▼ 자세히 보기

## ⚠ 주의하세요!

<ViewTransition>은 DOM 노드보다 앞에 배치되어야만 활성화됩니다. child가 다음과 같다면 애니메이션이 발생하지 않습니다.

```
function Component() {  
  return <ViewTransition>Hi</ViewTransition>;  
}
```

# 공유 엘리먼트 애니메이션 적용하기

일반적으로 `<ViewTransition>`에 이름을 할당하는 것보다 React가 자동으로 이름을 할당하도록 하는 것을 권장합니다. 이름을 할당하고 싶은 경우는 하나의 트리가 마운트 해제되고 다른 트리가 동시에 마운트될 때 완전히 다른 컴포넌트 간에 애니메이션을 적용하여 연속성을 보존하고자 할 때입니다.

```
<ViewTransition name={UNIQUE_NAME}>
  <Child />
</ViewTransition>
```

하나의 트리가 마운트 해제되고 다른 트리가 마운트될 때 마운트 해제되는 트리와 마운트되는 트리에서 동일한 이름이 존재하는 쌍이 있으면 둘 다에서 “share” 애니메이션이 발생합니다. 마운트 해제되는 쪽에서 마운트되는 쪽으로 애니메이션이 적용됩니다.

`exit/enter` 애니메이션과 달리 삭제되거나 새로 마운트된 트리의 깊숙한 곳에서도 적용될 수 있습니다. `<ViewTransition>` 이 `exit/enter`에도 해당한다면 “share” 애니메이션이 우선순위를 갖습니다.

Transition이 먼저 한쪽을 마운트 해제하고 새로운 이름이 마운트되기 전에 `<Suspense>` 풀백이 표시되는 경우 공유 엘리먼트 Transition은 발생하지 않습니다.

[App.js](#) [Video.js](#)

⌚ 새로고침 X Clear ☰ 포크

```
import {
  ViewTransition,
  useState,
  startTransition
} from "react";
import {Video, Thumbnail, FullscreenVideo} from "./Video";
import videos from "./data";

export default function Component() {
  const [fullscreen, setFullscreen] = useState(false);
  if (fullscreen) {
    return <FullscreenVideo
```

▼ 자세히 보기

## 중요합니다!

한 쌍의 마운트된 쪽이나 마운트 해제된 쪽 중 하나가 뷔포트 밖에 있으면 쌍이 형성되지 않습니다. 이는 무언가가 스크롤될 때 뷔포트 안팎으로 날아가는 것을 방지합니다. 대신 일반적인 `enter/exit`로 자체적으로 처리됩니다.

동일한 컴포넌트 인스턴스가 위치를 변경하는 경우에는 이런 일이 발생하지 않으며 “update”가 발생합니다. 한 위치가 뷔포트 밖에 있어도 애니메이션이 적용됩니다.

현재 한 가지 특이한 점이 있는데, 깊게 중첩된 마운트 해제된 `<ViewTransition>` 이 뷔포트 안에 있고, 마운트되는 쪽이 뷔포트 밖에 있는 경우, 해당 마운트 해제된 요소는 부모 애니메이션의 일부로 동작하는 대신, 깊게 중첩되어 있더라도 자체적인 “exit” 애니메이션으로 동작하게 됩니다.

## 주의하세요!

전체 앱에서 동시에 동일한 name으로 마운트된 것이 하나만 있어야 한다는 것이 중요합니다. 따라서 충돌을 피하기 위해 name에 고유한 네임스페이스를 사용하는 것이 중요합니다. 이를 확실히 하기 위해 가져올 수 있는 별도 모듈에 상수를 추가하는 것이 좋습니다.

```
export const MY_NAME = "my-globally-unique-name";
import {MY_NAME} from './shared-name';
...
<ViewTransition name={MY_NAME}>
```

## 목록에서 항목 순서 변경 애니메이션 적용하기

```
items.map(item => <Component key={item.id} item={item} />)
```

콘텐츠를 업데이트하지 않고 목록 순서를 변경할 때 DOM 노드 밖에 있으면 목록의 각 <ViewTransition>에서 “update” 애니메이션이 발생합니다. enter/exit 애니메이션과 유사합니다.

이는 이 <ViewTransition>에서 애니메이션이 발생한다는 의미입니다.

```
function Component() {
  return <ViewTransition><div>...</div></ViewTransition>;
}
```

### App.js

↺ 새로고침 × Clear ☒ 포크

```
import {
  ViewTransition,
  useState,
  startTransition
} from "react";
```

```
import {Video} from "./Video";
import videos from "./data";

export default function Component() {
  const [orderedVideos, setOrderedVideos] = useState(videos);
  const reorder = () => {
```

▼ 자세히 보기

하지만 다음은 각 개별 항목에 애니메이션을 적용하지 않습니다.

```
function Component() {
  return <div><ViewTransition>...</ViewTransition></div>;
}
```

대신 부모 <ViewTransition> 이 크로스 페이드됩니다. 부모 <ViewTransition> 이 없으면 별도의 애니메이션이 적용되지 않습니다.

## App.js

↪ 새로고침 × Clear ☐ 포크

```
import {
  ViewTransition,
```

```
useState,  
startTransition  
} from "react";  
  
import {Video} from "./Video";  
import videos from "./data";  
  
export default function Component() {  
  const [orderedVideos, setOrderedVideos] = useState(videos);  
  const reorder = () => {  
    const items = [...orderedVideos];  
    const index = Math.floor(Math.random() * items.length);  
    const item = items[index];  
    const newItems = [...items];  
    newItems.splice(index, 1);  
    newItems.push(item);  
    setOrderedVideos(newItems);  
  };  
}
```

### ▼ 자세히 보기

이는 컴포넌트가 자체적으로 순서 변경 애니메이션을 제어할 수 있도록 하고 싶을 때는 리스트 안에 래퍼 요소를 두지 않는 것이 좋다는 뜻입니다.

```
items.map(item => <div><Component key={item.id} item={item} /></div>)
```

위 규칙은 항목 중 하나가 크기 조정을 위해 업데이트되어 형제 항목들이 크기 조정되는 경우에도 적용되며, 이는 형제 <viewTransition> 도 애니메이션시키지만 직접적인 형제인 경우에만 해당합니다.

이것은 업데이트가 발생하여 레이아웃이 크게 변경될 때, 페이지에 있는 모든 `<ViewTransition>` 을 각각 개별적으로 애니메이션하지 않는다는 뜻입니다. 그렇게 하면 실제 변화와 관계없는 많은 산만한 애니메이션이 발생해 주의를 흐트러뜨리게 됩니다. 따라서 React는 개별 애니메이션을 언제 트리거할지에 대해 보다 보수적으로 동작합니다.

## ⚠ 주의하세요!

목록 순서를 변경할 때 아이덴티티를 보존하기 위해 키를 적절히 사용하는 것이 중요합니다. “name”이나 공유 엘리먼트 Transition을 사용하여 순서 변경을 애니메이션할 수 있을 것 같지만 한쪽이 뷔포트 밖에 있으면 발생하지 않습니다. 리스트를 재정렬하는 애니메이션을 만들 때는, 해당 항목이 화면에 보이지 않는 위치로 이동했음을 사용자에게 보여주는 것이 중요한 경우가 많습니다.

## Suspense 콘텐츠에서 애니메이션 적용하기

다른 Transition과 마찬가지로 React는 애니메이션을 실행하기 전에 데이터와 새로운 CSS(`<link rel="stylesheet" precedence="...">`)를 기다립니다. 이에 더해 ViewTransition은 새로운 폰트가 나중에 깜빡이는 것을 방지하기 위해 애니메이션을 시작하기 전에 새로운 폰트가 로드될 때까지 최대 500ms까지 기다립니다. 같은 이유로 ViewTransition으로 래핑된 이미지는 이미지가 로드될 때까지 기다립니다.

새로운 Suspense 경계 인스턴스 내부에 있으면 폴백이 먼저 표시됩니다. Suspense 경계가 완전히 로드된 후 `<ViewTransition>` 이 콘텐츠로 전환되는 애니메이션을 실행합니다.

현재 이 동작은 클라이언트 측 Transition에서만 발생합니다. 향후에는 초기 로드 중에 서버의 콘텐츠가 일시 중단될 때 스트리밍 SSR에 대한 Suspense 경계도 애니메이션할 예정입니다.

`<ViewTransition>` 을 배치하는 위치에 따라 Suspense 경계를 애니메이션하는 두 가지 방법이 있습니다.

Update:

```
<Suspense fallback={<A />}>
  <B />
</Suspense>
</ViewTransition>
```

이 시나리오에서 컨텐츠가 A에서 B로 바뀔 때 “update”로 처리되며 적절한 경우 해당 클래스를 적용합니다. A와 B 모두 동일한 view-transition-name을 갖게 되므로 기본적으로 크로스 페이드로 작동합니다.

## App.js

↪ 새로고침 ✕ Clear ⌂ 포크

```
import {
  ViewTransition,
  useState,
  startTransition,
  Suspense
} from 'react';
import {Video, VideoPlaceholder} from './Video';
import {useLazyVideoData} from "./data"

function LazyVideo() {
  const video = useLazyVideoData();
  return (
    <ViewTransition>
      <Suspense fallback={<VideoPlaceholder />}>
        <Video />
      </Suspense>
    </ViewTransition>
  );
}
```

▼ 자세히 보기

Enter/Exit:

```
<Suspense fallback={<ViewTransition><A /></ViewTransition>}>
  <ViewTransition><B /></ViewTransition>
</Suspense>
```

이 시나리오에서는 각각 고유한 `view-transition-name` 을 갖는 두 개의 별도 `ViewTransition` 인스턴스입니다. 이는 `<A>` 의 “exit”와 `<B>` 의 “enter”로 처리됩니다.

`<ViewTransition>` 경계를 배치하는 위치에 따라 다른 효과를 얻을 수 있습니다.

## 애니메이션 제외하기

때로는 전체 페이지와 같은 큰 기존 컴포넌트를 래핑하고 테마 변경과 같은 일부 업데이트를 애니메이션하고 싶지만 전체 페이지 내부의 모든 업데이트가 업데이트될 때 크로스 페이드에 포함되는 것을 원하지 않을 수 있습니다. 특히 점진적으로 더 많은 애니메이션을 추가하는 경우에 그렇습니다.

클래스 “`none`”을 사용하여 애니메이션을 제외할 수 있습니다. 자식을 “`none`”으로 래핑하면 부모가 여전히 발생하는 동안 자식에 대한 업데이트 애니메이션을 비활성화할 수 있습니다.

```
<ViewTransition>
  <div className={theme}>
    <ViewTransition update="none">
      {children}
    </ViewTransition>
  </div>
</ViewTransition>
```

이는 테마가 변경될 때만 애니메이션되며 자식만 업데이트될 때는 애니메이션되지 않습니다. 자식은 여전히 자체 `<ViewTransition>` 으로 다시 참여할 수 있지만 최소한 다시 수동으로 제어하는 방식이 됩니다.

# 애니메이션 커스터마이징

기본적으로 `<ViewTransition>` 은 브라우저의 기본 크로스 페이드를 포함합니다.

애니메이션을 커스터마이징하려면 `<ViewTransition>` 컴포넌트에 `props`를 제공하여 `<ViewTransition>` 이 활성화되는 방식에 따라 사용할 애니메이션을 지정할 수 있습니다.

예를 들어 기본 크로스 페이드 애니메이션을 느리게 할 수 있습니다.

```
<ViewTransition default="slow-fade">
  <Video />
</ViewTransition>
```

그리고 View Transition 클래스를 사용하여 CSS에서 slow-fade를 정의합니다.

```
::view-transition-old(.slow-fade) {
  animation-duration: 500ms;
}

::view-transition-new(.slow-fade) {
  animation-duration: 500ms;
}
```

## App.js

↺ 새로고침 X Clear ☒ 포크

```
import {
  ViewTransition,
  useState,
  startTransition
} from 'react';
import {Video} from "./Video";
import videos from "./data"

function Item() {
  return (
    <ViewTransition default="slow-fade">
      <Video video={videos[0]} />
    </ViewTransition>
  )
}

export default Item
```

▼ 자세히 보기

default 설정 외에도 enter, exit, update, share 애니메이션에 대한 구성을 제공할 수 있습니다.

## App.js

↪ 새로고침 ✖ Clear ⌂ 포크

```
import {  
  ViewTransition,  
  useState,  
  startTransition  
} from 'react';  
import {Video} from "./Video";  
import videos from "./data"  
  
function Item() {  
  return (  
    <ViewTransition enter="slide-in" exit="slide-out">  
      <Video video={videos[0]} />  
    </ViewTransition>  
  );  
}  
  
export default Item;
```

▼ 자세히 보기

## 타입으로 애니메이션 커스터마이징하기

특정 활성화 트리거에 대해 특정 Transition 타입이 활성화될 때 자식 엘리먼트에 클래스 이름을 추가하기 위해 `addTransitionType` API를 사용할 수 있습니다. 이를 통해 각 Transition 타입에 대한 애니메이션을 커스터마이징할 수 있습니다.

예를 들어 모든 앞으로 및 뒤로 네비게이션에 대한 애니메이션을 커스터마이징하려면,

```
<ViewTransition default={{
  'navigation-back': 'slide-right',
  'navigation-forward': 'slide-left',
}}>
<div>...</div>
</ViewTransition>

// 라우터에서:
startTransition(() => {
  addTransitionType('navigation-' + navigationType);
});
```

`ViewTransition`이 “navigation-back” 애니메이션을 활성화하면 React는 “slide-right” 클래스 이름을 추가합니다. `ViewTransition`이 “navigation-forward” 애니메이션을 활성화하면 React는 “slide-left” 클래스 이름을 추가합니다.

향후 라우터와 다른 라이브러리들이 표준 view-transition 타입과 스타일에 대한 지원을 추가할 수 있습니다.

## App.js

↪ 새로고침 X Clear ⌂ 포크

```
import {  
  ViewTransition,  
  addTransitionType,  
  useState,  
  startTransition,  
} from "react";  
  
import {Video} from "./Video";  
import videos from "./data"  
  
function Item() {  
  return (  
    <ViewTransition enter={
```

▼ 자세히 보기

## View Transition 지원 라우터 구축하기

스크롤 복원이 애니메이션 중에 정상적으로 동작하도록, React는 대기 중인 내비게이션이 완료될 때까지 기다립니다. 네비게이션이 React에서 차단되는 경우 useEffect는 교착 상태로 이어질

수 있으므로 라우터는 `useLayoutEffect`에서 차단을 해제해야 합니다.

”뒤로” 네비게이션 중처럼 레거시 `popstate` 이벤트에서 `startTransition`이 시작되면 스크롤과 품복원이 올바르게 작동하도록 동기적으로 완료되어야 합니다. 이는 View Transition 애니메이션 실행과 충돌합니다. 따라서 React는 `popstate`에서 애니메이션을 건너뜁니다. 따라서 뒤로 버튼에 대해서는 애니메이션이 실행되지 않습니다. Navigation API를 사용하도록 라우터를 업그레이드하여 이를 해결할 수 있습니다.

## 문제 해결

### <ViewTransition> 이 활성화되지 않습니다

<`ViewTransition`> only activates if it is placed before any DOM node:

```
function Component() {
  return (
    <div>
      <ViewTransition>Hi</ViewTransition>
    </div>
  );
}
```

해결하려면 <`ViewTransition`> 이 다른 DOM 노드보다 앞에 오도록 하세요.

```
function Component() {
  return (
    <ViewTransition>
      <div>Hi</div>
    </ViewTransition>
  );
}
```

”동일한 이름으로 마운트된 <`ViewTransition name=%s`> 컴포넌트가 두 개 있습니다.”라는 오류가 발생합니다

이 오류는 동일한 name 을 가진 두 개의 <ViewTransition> 컴포넌트가 동시에 마운트될 때 발생합니다.

```
function Item() {
  // ▶ 모든 항목이 동일한 "name"을 갖게 됩니다.
  return <ViewTransition name="item">...</ViewTransition>;
}

function ItemList({items}) {
  return (
    <>
    {items.map(item => <Item key={item.id}>/)}
    </>
  );
}
```

이는 View Transition에서 오류를 발생시킵니다. 개발 중에 React는 이 문제를 감지하여 표면화하고 두 개의 오류를 기록합니다.

#### Console

- ✖ There are two <ViewTransition name=%s> components with the same name mounted at the same time. This is not supported and will cause View Transitions to error. Try to use a more unique name e.g. by using a namespace prefix and adding the id of an item to the name.  
at Item  
at ItemList
- ✖ The existing <ViewTransition name=%s> duplicate has this stack trace.  
at Item  
at ItemList

해결하려면 name 이 고유하도록하거나 이름에 id 를 추가하여 전체 앱에서 동일한 이름을 가진 <ViewTransition> 이 한 번에 하나만 마운트되도록 하세요.

```
function Item({id}) {
  // ✓ 모든 항목이 고유한 "name"을 갖게 됩니다.
  return <ViewTransition name={`${item}-${id}`}>...</ViewTransition>;
}

function ItemList({items}) {
  return (
```

```
<>  
    {item.map(item => <Item key={item.id} item={item} />)}  
</>  
);  
}
```

이전

<Activity>

다음

API

 Meta Open Source

Copyright © Meta Platforms, Inc

uwu?

React 학습하기

빠르게 시작하기

설치하기

UI 표현하기

상호작용성 더하기

State 관리하기

탈출구

API 참고서

React APIs

React DOM APIs

커뮤니티

행동 강령

팀 소개

문서 기여자

감사의 말

더 보기

블로그

React Native

개인 정보 보호

약관

