



Menu

On this page

디버깅하기

준비

React Native Debugger를 사용해서 JavaScript 디버깅을 하려면 Chrome 브라우저가 필요해요. 다음 링크로 브라우저를 다운로드하세요.

- [Chrome 웹브라우저 다운로드 링크](#)

Metro 개발 서버 실행

Metro 개발 서버는 React Native 애플리케이션을 개발할 때 코드 번들링, 핫 리로딩, 자원 제공 등의 역할을 해주는 번들러예요. 쉽게 말해서, 앱이 실행될 수 있도록 JavaScript 코드를 준비하고 관리하는 도구라고 생각하면 돼요.

다음 명령어로 Metro 개발 서버를 실행해서 디버깅을 시작해 볼게요.

bash

```
yarn dev  
# 또는  
yarn bedrock dev
```

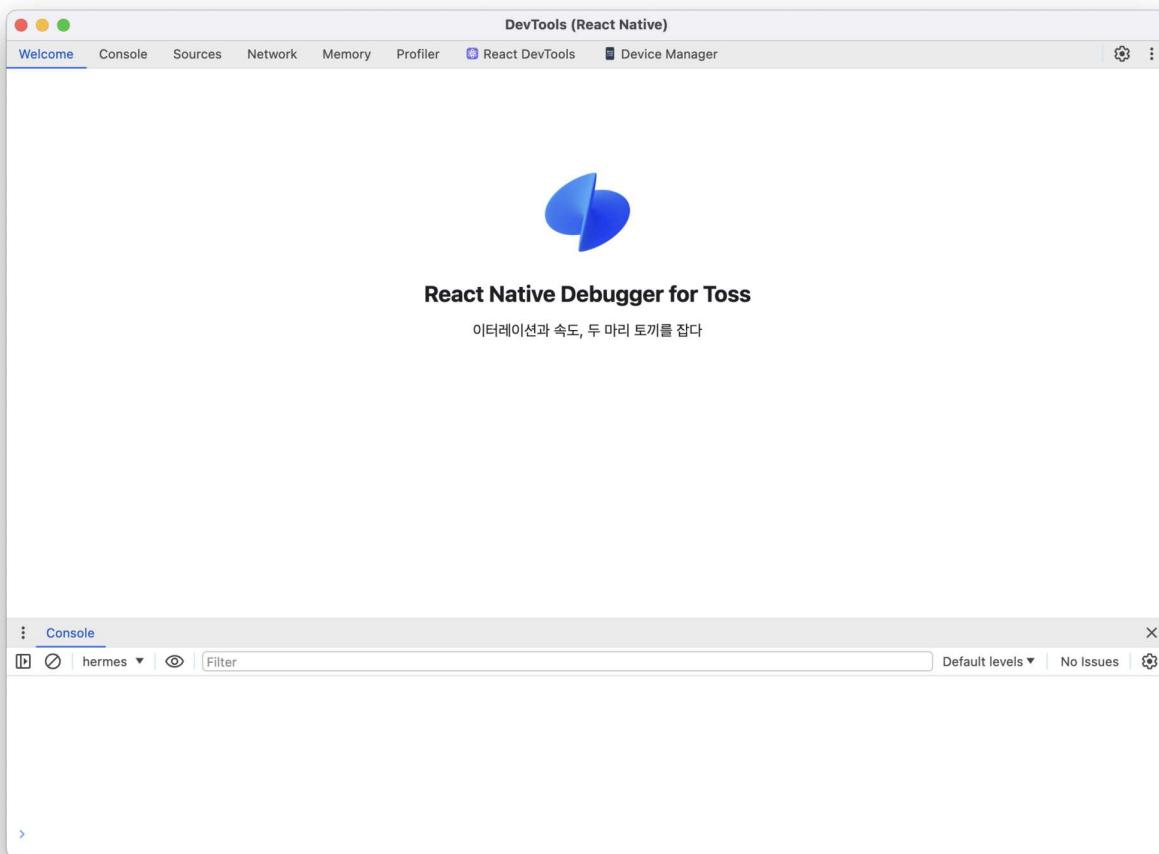


개발 서버가 실행됐다면, 터미널에서 `j` 키를 눌러 React Native Debugger를 열 수 있어요. 다만, 아래와 같이 Metro와 테스트 기기 간의 연결이 이루어진 상태에서만 React Native Debugger가 열려요.



탭 소개

Metro 개발 서버가 제공하는 디버깅 도구를 사용해서 편리하게 애플리케이션의 상태를 분석하고, 문제를 찾아 해결할 수 있어요. 각 탭에서는 모니터링하거나 조작할 수 있는 항목을 알아봐요.



- **Console:** [Console API](#)를 통해 기록한 로그를 확인할 수 있어요. 또한, REPL(Read–Eval–Print Loop) 환경을 지원하기 때문에 코드를 입력하면 결과를 콘솔에서 확인할 수 있어요.
- **Source:** 현재 실행 중인 코드를 보고 중단점을 추가할 수 있어요.
- **Network:** 개발 서버에 연결된 서비스의 네트워크 활동을 확인할 수 있어요.
- **Memory:** Memory Snapshot을 기록해서 Hermes 엔진의 메모리 사용 상황을 프로파일링할 수 있어요.
- **Profiler:** 코드 실행 성능을 측정할 수 있는 도구예요.

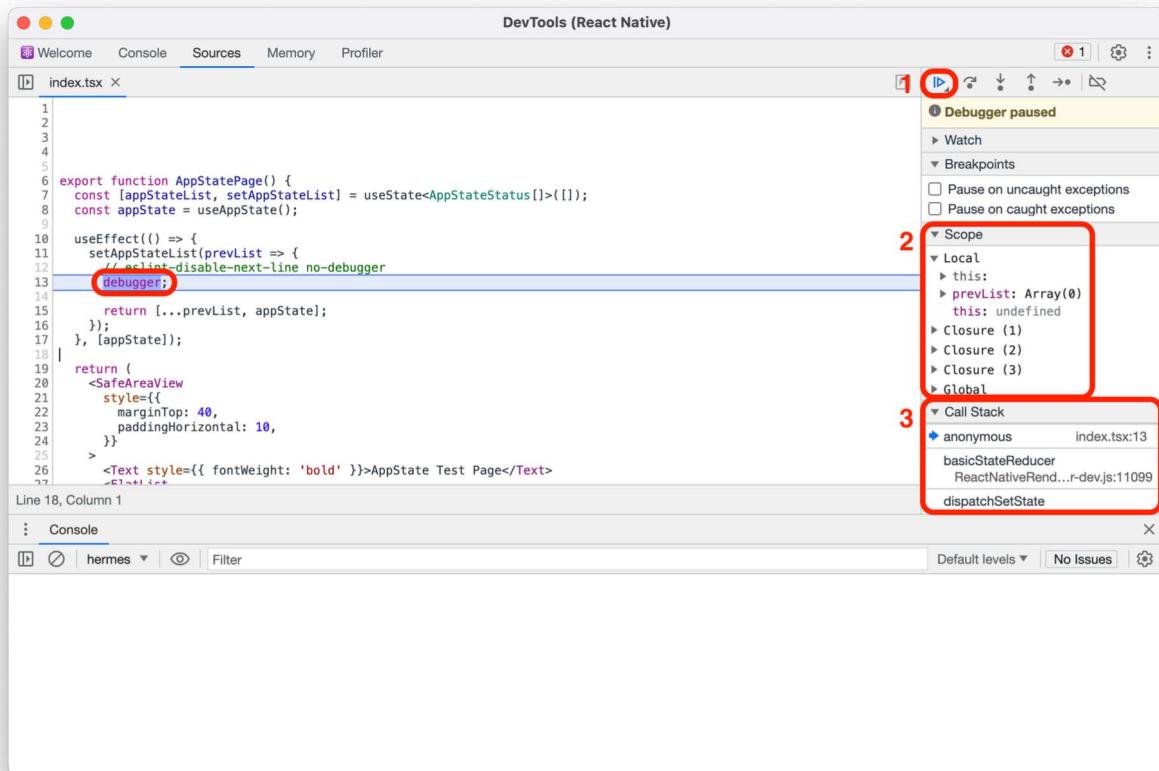
기본 사용법

이제 Metro 개발 서버를 사용해서 React Native 애플리케이션에서 다양한 디버깅 기능을 활용하는 방법을 알아볼게요.

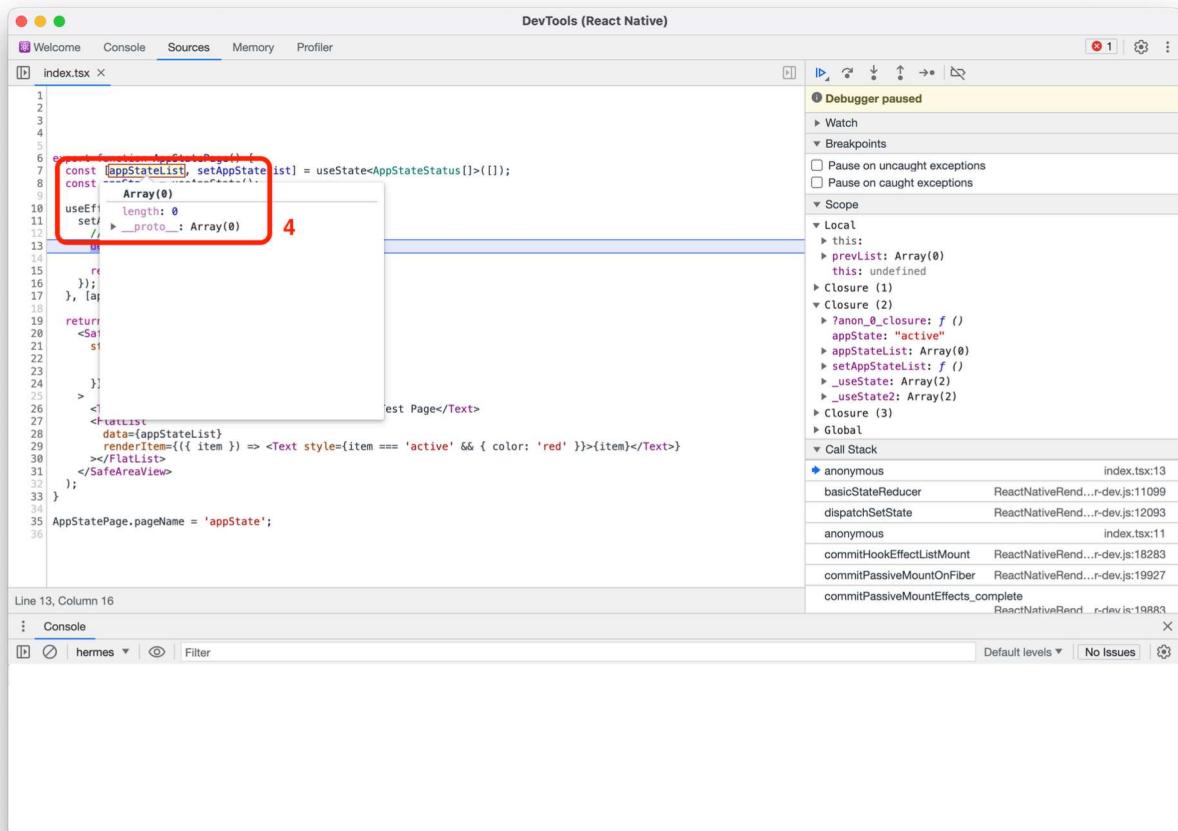
디버깅 공통

React Native 디버거를 사용하면 코드에서 발생하는 문제를 쉽게 찾고 해결할 수 있어요. 특히 유용한 기능은 Breakpoints예요. 이 기능을 사용하면 코드가 중단된 시점의 상태를 확인할 수 있어요. 가장 기본적인 기능은 다음과 같아요.

1. **Resume**: 일시 정지된 코드를 다시 실행해요.
2. **Scope 정보**: 현재 Scope에 접근 할 수 있는 변수와 그 값을 확인할 수 있어요.
3. **Call Stack**: 코드가 실행된 순서와 호출된 함수 목록, Call Stack을 확인할 수 있어요.



일시정지된 시점에서 변수의 값을 바로 확인할 수도 있어요. 다음과 같이 변수 위로 마우스 커서를 올리면 돼요.



debugger 키워드로 디버깅하기

또 다른 유용한 기능은 `debugger` 키워드예요. 이 키워드를 소스 코드에 추가하면 코드가 해당 지점에 도달했을 때 자동으로 중단되어 현재 상태를 살펴볼 수 있어요.

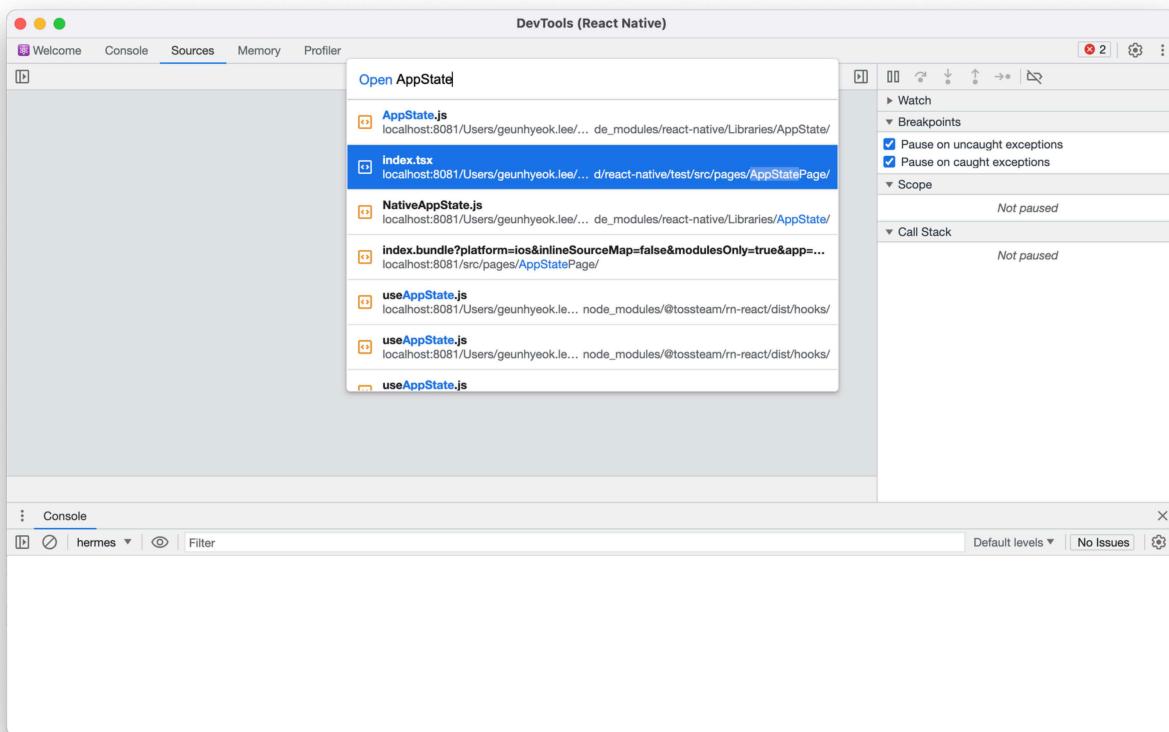
```
export function AppStatePage() { 
  const [appStateList, setAppStateList] = useState<AppStateStatus[]>([]); 
  const appState = useAppState(); 

  Complexity is 3 Everything is cool! 
  useEffect(() => { 
    setAppStateList(prevList => { 
      // eslint-disable-next-line no-debugger 
      debugger; 

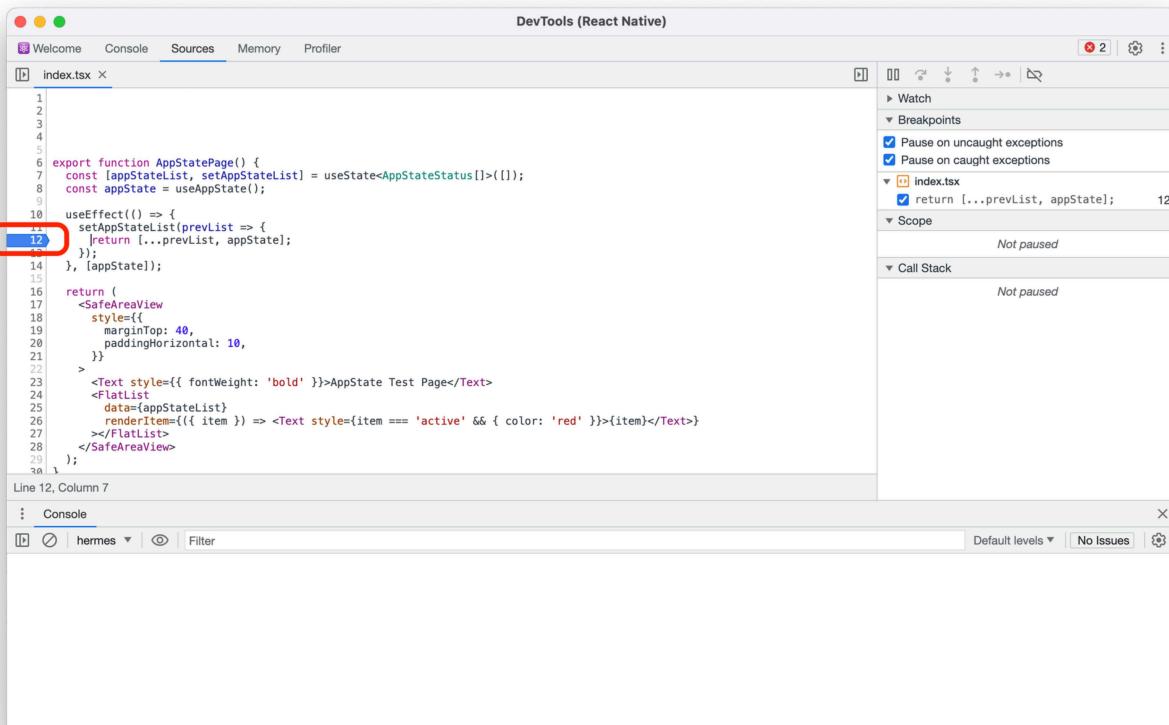
      return [...prevList, appState]; 
    }); 
  }, [appState]);
```

중단점으로 디버깅하기

중단점을 설정해서 특정 위치에서 코드 실행을 멈추는 기능도 사용할 수 있어요. 중단점을 추가하려면 커맨드(Command) + P 키를 눌러 파일 검색 창을 띄우고, 확인하고 싶은 파일명을 입력하세요.



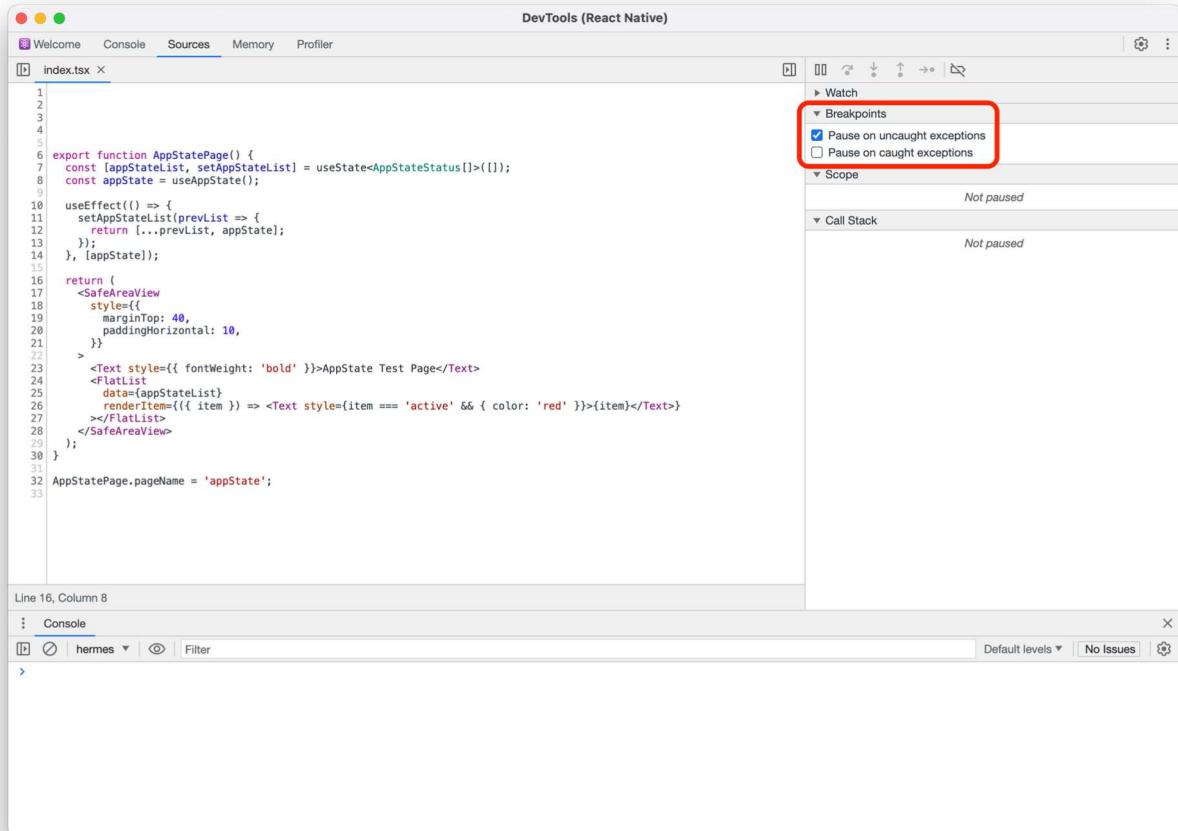
파일을 열었다면, 중단점을 설정하고 싶은 위치를 클릭해서 추가할 수 있어요. 중단점에 도달했을 때 코드 실행이 멈추면 현재 상태를 확인할 수 있어요.



예외 상황 디버깅하기

코드에서 예외(에러)가 발생했을 때 자동으로 그 지점에서 일시 정지되도록 설정할 수 있어요. 이 기능을 사용하면 미리 예상하지 못한 에러와 그 원인을 쉽게 파악할 수 있죠.

이 기능은 Metro 디버거의 **Source** 탭에서 설정할 수 있어요. 우측 상단의 Breakpoints 섹션에서 "Pause on uncaught exceptions" 을 활성화하면 예기치 못한 예외 발생 시 코드가 자동으로 중단돼요. "Pause on caught exceptions" 을 활성화하면 모든 예외(핸들링 여부와 관계없이)에서 일시 정지 할 수 있어요.



The screenshot shows the React Native DevTools interface. The Sources tab is active, displaying the code for `index.tsx`. The code contains a `useEffect` hook that throws an error. The Breakpoints panel on the right shows a breakpoint at line 11, which is currently active (indicated by a red dot). The Call Stack panel shows the execution path from the user's code down to the React Native rendering stack.

```

1
2
3
4
5
6 export function AppStatePage() {
7   const [appStateList, setAppStateList] = useState<AppStateStatus>([]);
8   const appState = useAppState();
9
10  useEffect(() => {
11    setAppStateList(prevList => {
12      throw new Error('Some error!');
13      return [...prevList, appState];
14    });
15  }, [appState]);
16
17  return (
18    <SafeAreaView
19      style={{
20        marginTop: 40,
21        paddingHorizontal: 10,
22      }}
23    >
24      <Text style={{ fontWeight: 'bold' }}>AppState Test Page</Text>
25      <FlatList
26        data={appStateList}
27        renderItem={({ item }) => <Text style={item === 'active' && { color: 'red' }}>{item}</Text>}
28      >
29    </FlatList>
30  </SafeAreaView>
31);
32

```

유의하세요

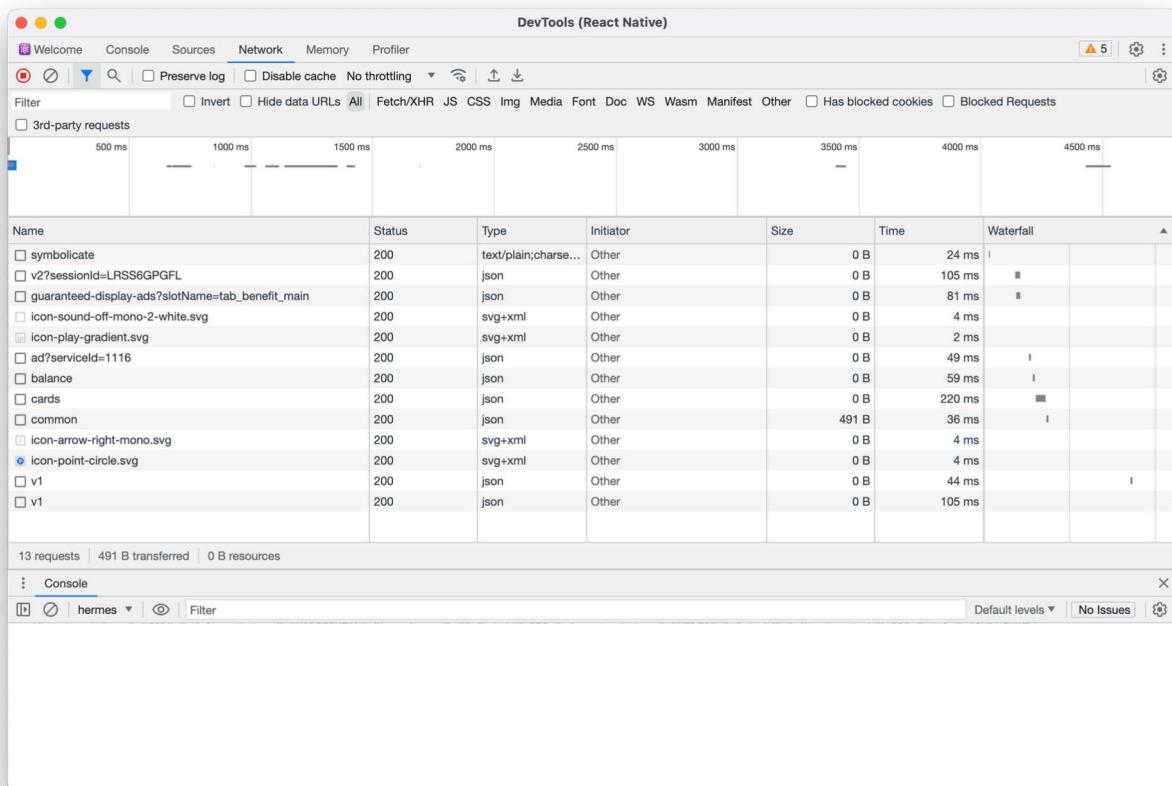
서비스 코드에서 심각한 예외가 발생해 서비스가 완전히 중단된 후에 예외 Breakpoints가 제대로 동작하지 않는 버그가 있어요.

임시 해결 방법은 개발 서버와 React Native Debugger를 재시작하는 거예요.

네트워크 활동 검사

React Native 애플리케이션의 네트워크 활동을 기록하고 확인할 수 있는 네트워크 인스펙터 기능을 사용해 볼게요.

네트워크 인스펙터를 사용하면 요청과 응답 데이터를 포함한 네트워크 활동을 다음과 같이 확인할 수 있어요. 이 도구를 사용하면 각 요청의 헤더와 응답 데이터를 상세히 분석할 수 있어요.



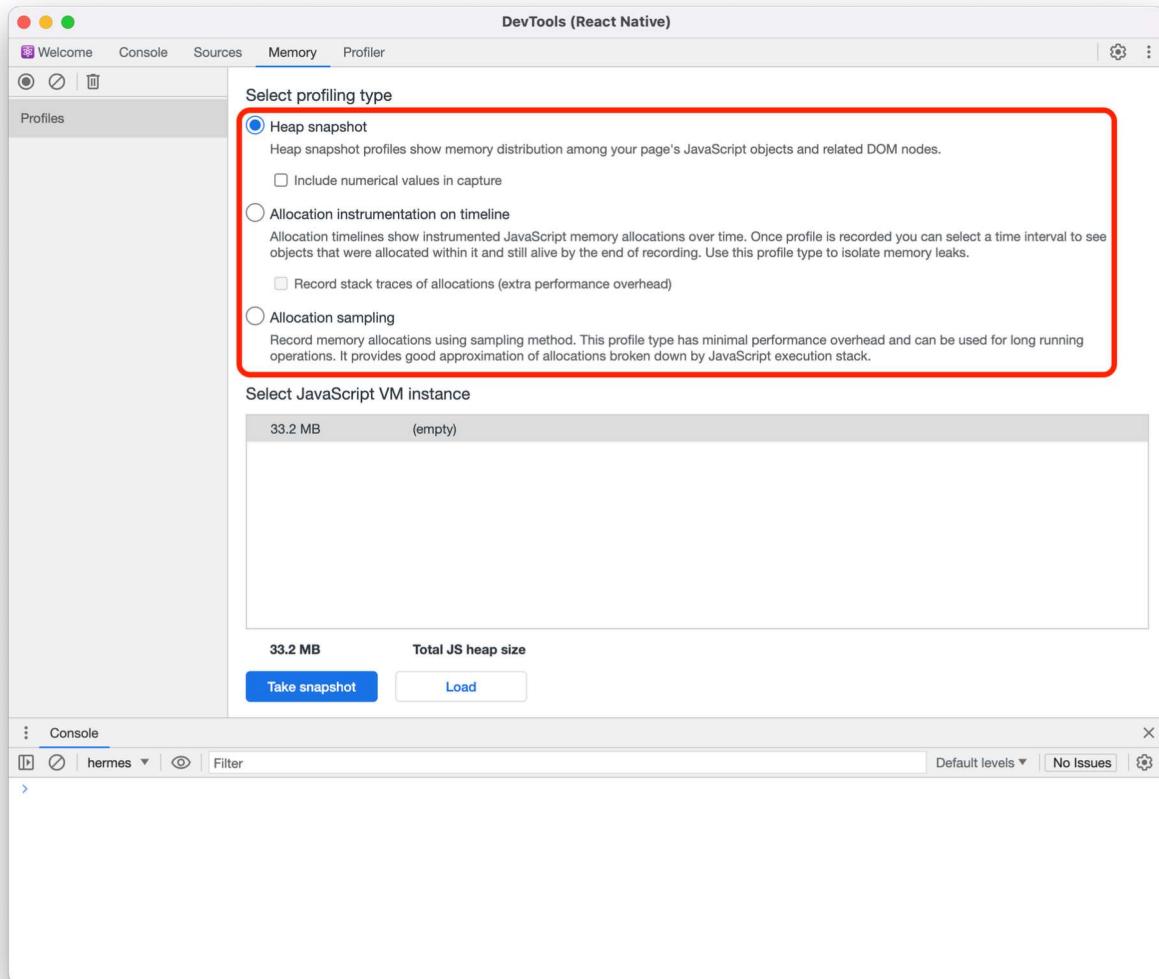
프로파일링

프로파일링 도구를 사용하면 React Native 애플리케이션의 메모리 사용량과 코드 실행 성능을 상세히 분석할 수 있어요. 이를 통해 성능 최적화와 메모리 누수 문제를 발견하고 해결할 수 있죠.

메모리

메모리 프로파일링 기능은 메모리 사용량을 분석하고, 메모리 누수를 탐지하는 데 유용해요. 이 도구를 사용해 앱이 메모리를 어떻게 관리하고 있는지 확인할 수 있어요.

1. 먼저, 프로파일링 유형을 선택하세요.
2. "Take snapshot" 버튼을 눌러 스냅샷을 기록할 수 있어요.
3. 기록된 스냅샷을 보면 메모리 사용 상태를 분석하고 필요한 정보를 찾아요.



Profiles

HEAP SNAPSHTOS

Snapshot 1 **Save**

Summary **Class filter** **All objects**

		Distance	Shallow Size	Retained Size
Constructor		3	80 208	0 %
Object		12	280	0 %
actualDuration: 0.3737945556640625		11	2 600	0 %
actualStartTime: 1364581419.894166		11	320	0 %
alternate: Object		13	44 520	0 %
child: Object		5	263 960	0 %
childLanes: 0		19	120	0 %
deletions: null		4	2 720	0 %
dependencies: null		—	47 360	0 %
elementType: Object		17	40	0 %
flags: 1		12	48	0 %
index: 0		18	40	0 %
key: null		18	40	0 %
lanes: 0		18	40	0 %
memoizedProps: Object		16	40	0 %
memoizedState: null		18	40	0 %
memoizeIndex: 0		18	40	0 %
child :: FiberNode @1355123	ReactNativeRenderer-dev.js:23023	16	40	0 %
_debugOwner :: FiberNode @1355031	ReactNativeRenderer-dev.js:23023	18	40	0 %
return :: FiberNode @1355031	ReactNativeRenderer-dev.js:23023	18	40	0 %
memoizedProps :: JSObject<accessible, accessibilityLabel, accessibilityHint,		18	40	0 %

Retainers

	Distance	Shallow Size	Retained Size
Object			
0 in ArrayStorageSmall[] @1355145	17	112	0 %
propStorage in FiberNode @1355107	16	40	0 %
0 in ArrayStorageSmall[] @1355125	16	112	0 %
propStorage in FiberNode @1353861	15	40	0 %
0 in ArrayStorageSmall[] @1355053	17	112	0 %
propStorage in FiberNode @1353963	16	40	0 %
0 in ArrayStorageSmall[] @1353977	17	112	0 %

Console

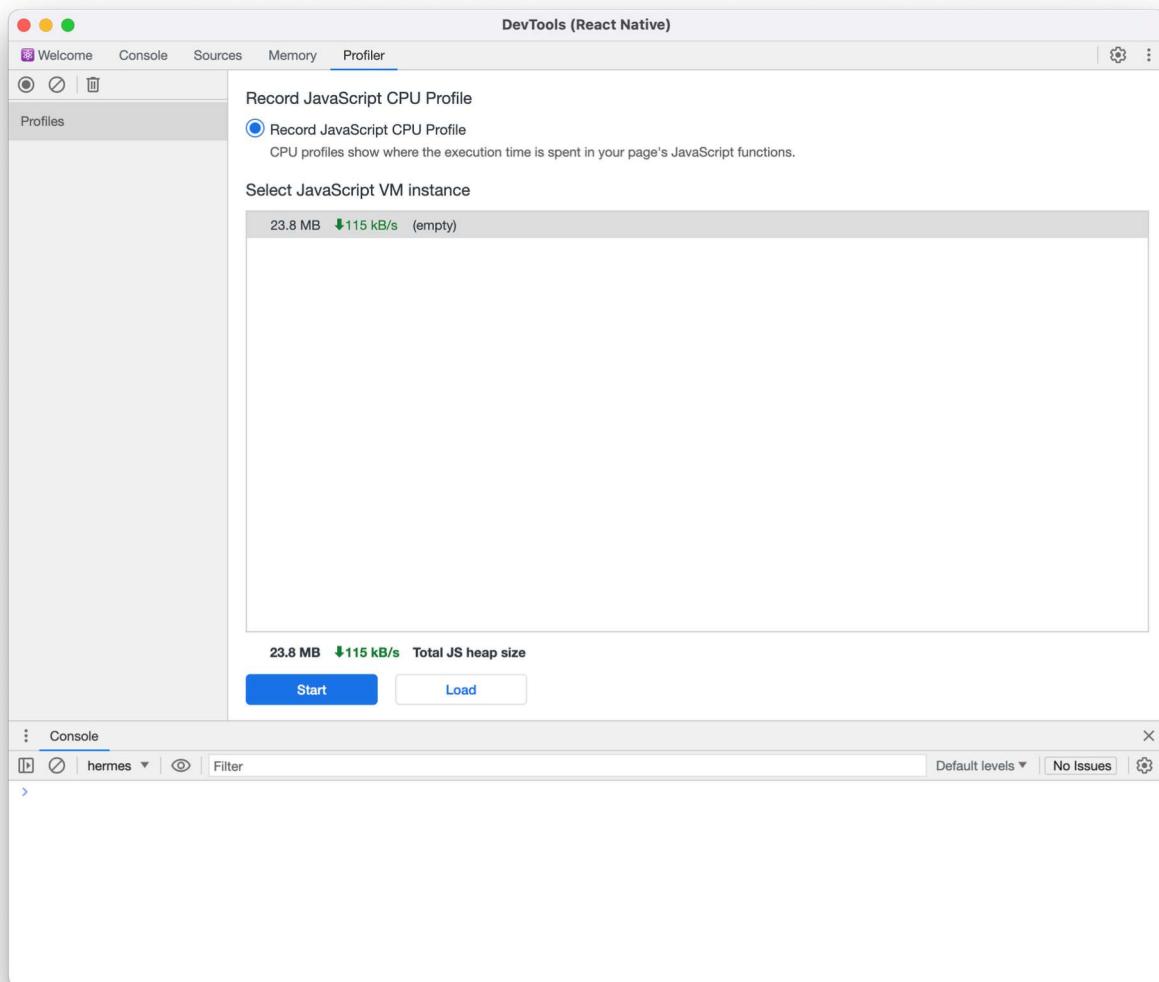
hermes Filter Default levels No Issues

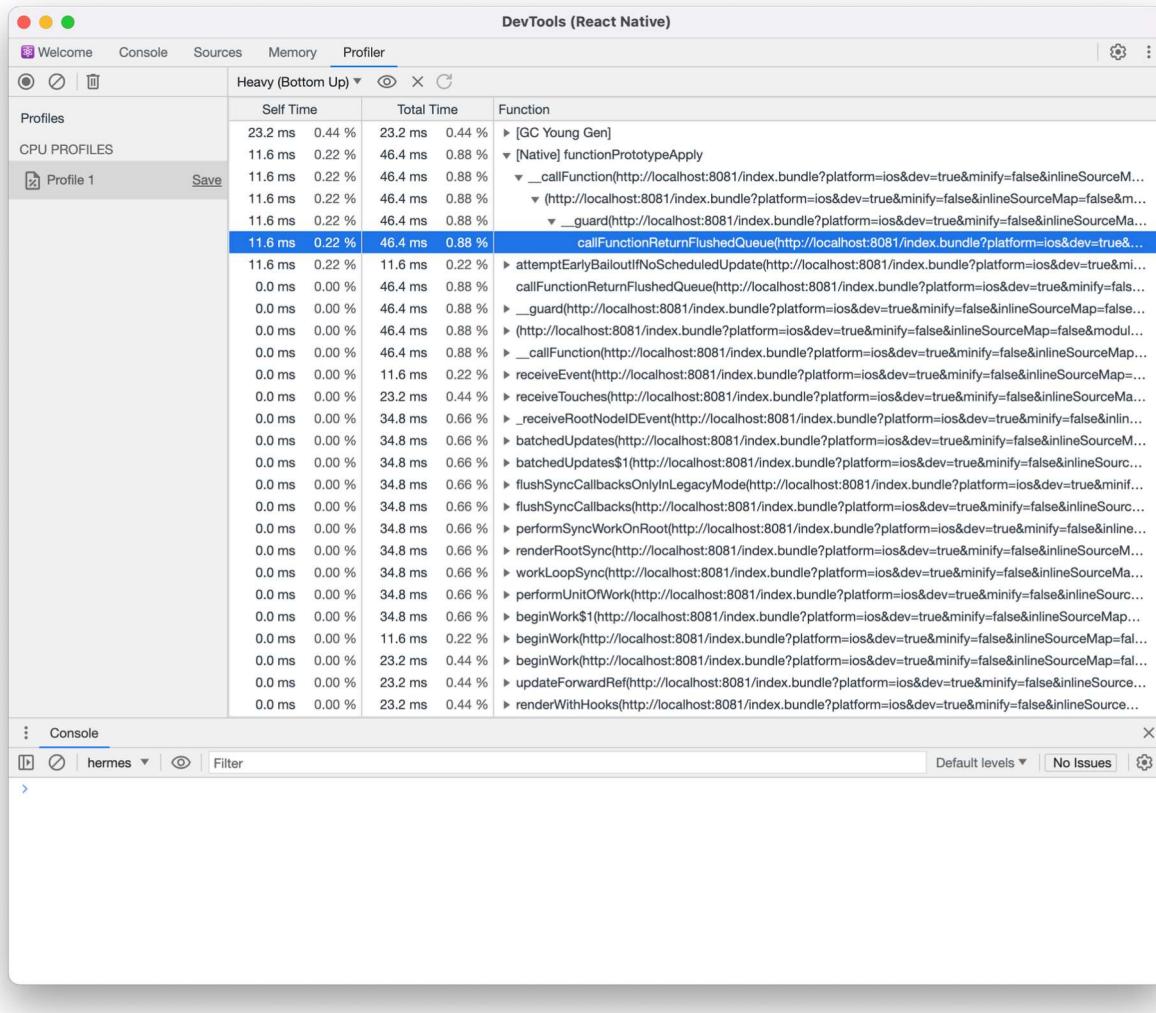
```
>
```

성능 측정

성능 측정 도구를 사용하면 코드의 실행 성능을 분석할 수 있어요. 이 기능은 성능 최적화가 필요한 부분을 식별하고 개선하는 데 유용해요.

다음은 성능 측정 도구의 사용 예시에요. 이 도구를 사용해 기록을 시작하고, 실행된 코드의 성능을 평가할 수 있어요.





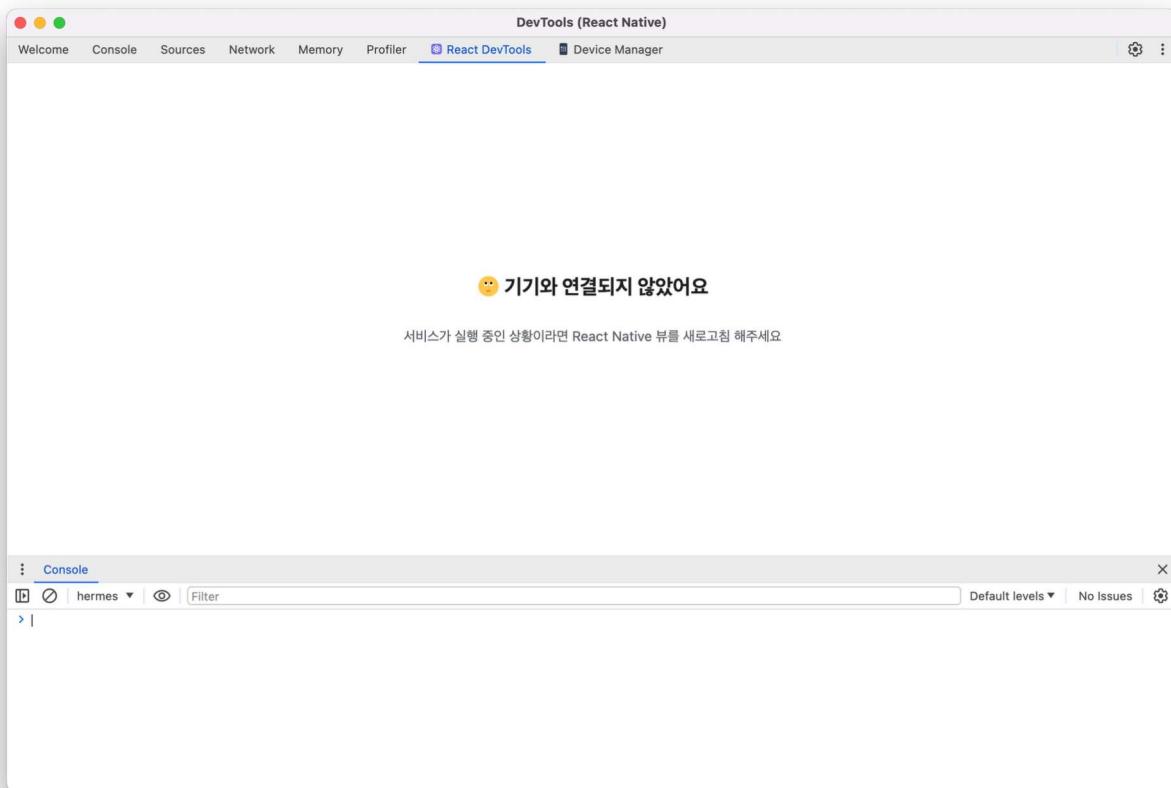
React DevTools로 디버깅하기

React DevTools는 React Native 애플리케이션의 컴포넌트 구조를 시각적으로 탐색하고 디버깅할 수 있는 도구예요. 디버깅 도구의 React DevTools에서 바로 사용할 수 있어요.

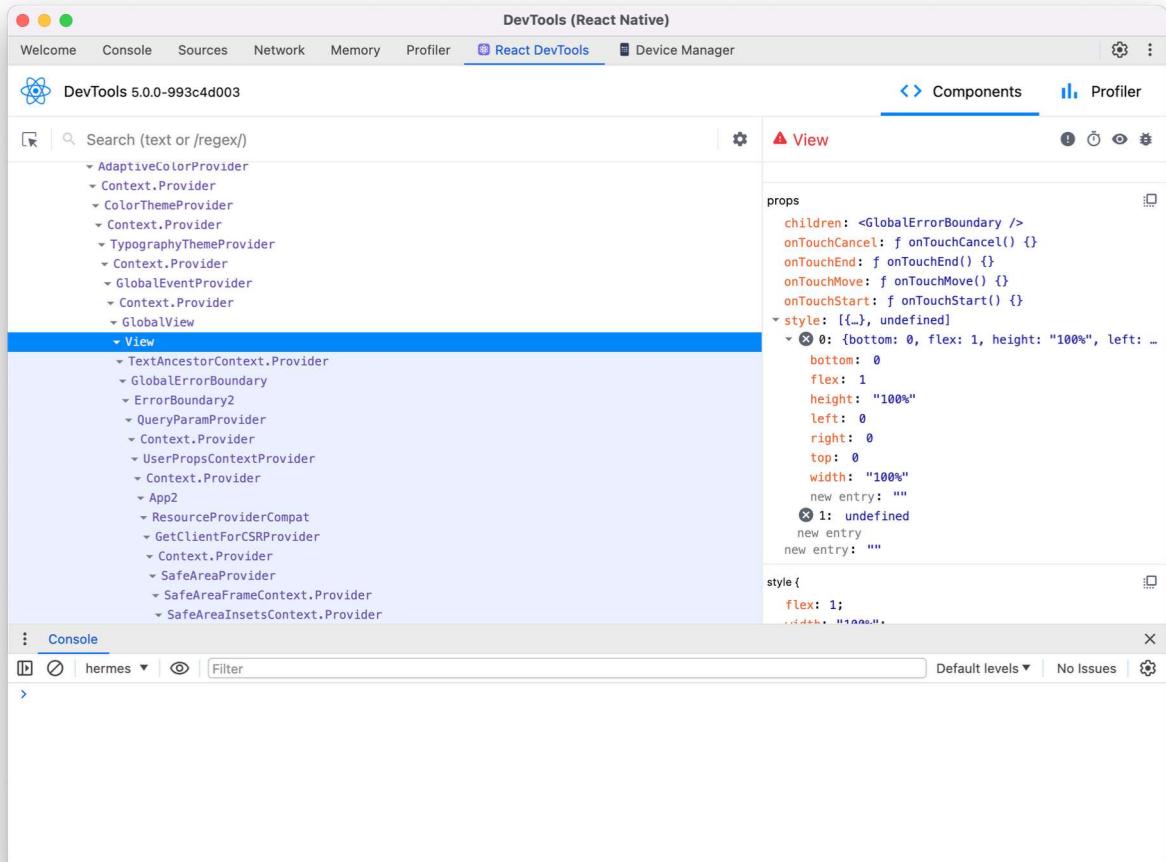
기기와 연결하기

React DevTools를 사용하려면 먼저 개발 중인 애플리케이션의 연결이 필요해요.

서비스가 실행되기 전이라면 개발 모드에서 React Native 서비스를 실행해 주세요. 서비스가 이미 실행 중이라면, 개발 모드 RN 뷰를 새로고침해야 해요. **R** 키를 눌러서 새로고침해주세요.



다음과 같이 React DevTools 화면이 나타나면 연결이 완료된 거예요.



안드로이드 기기를 사용한다면, adb reverse tcp:8097 tcp:8097 명령어를 입력해서 포트를 열어야

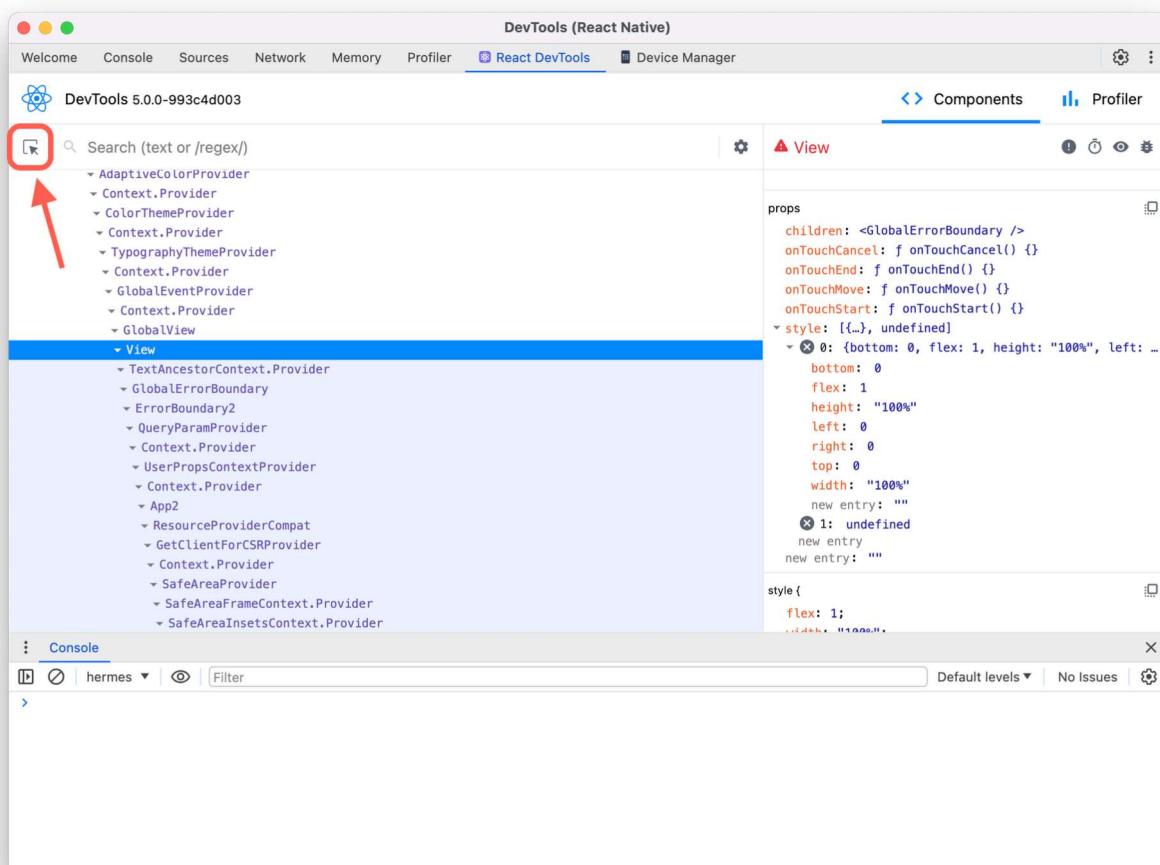
React DevTools가 정상적으로 동작해요. 만약 문제가 있다면 담당자에게 문의해주세요.

사용 팁

React DevTools에 있는 여러 유용한 기능을 효과적으로 사용하는 몇 가지 팁을 알려드릴게요.

요소 인스펙팅

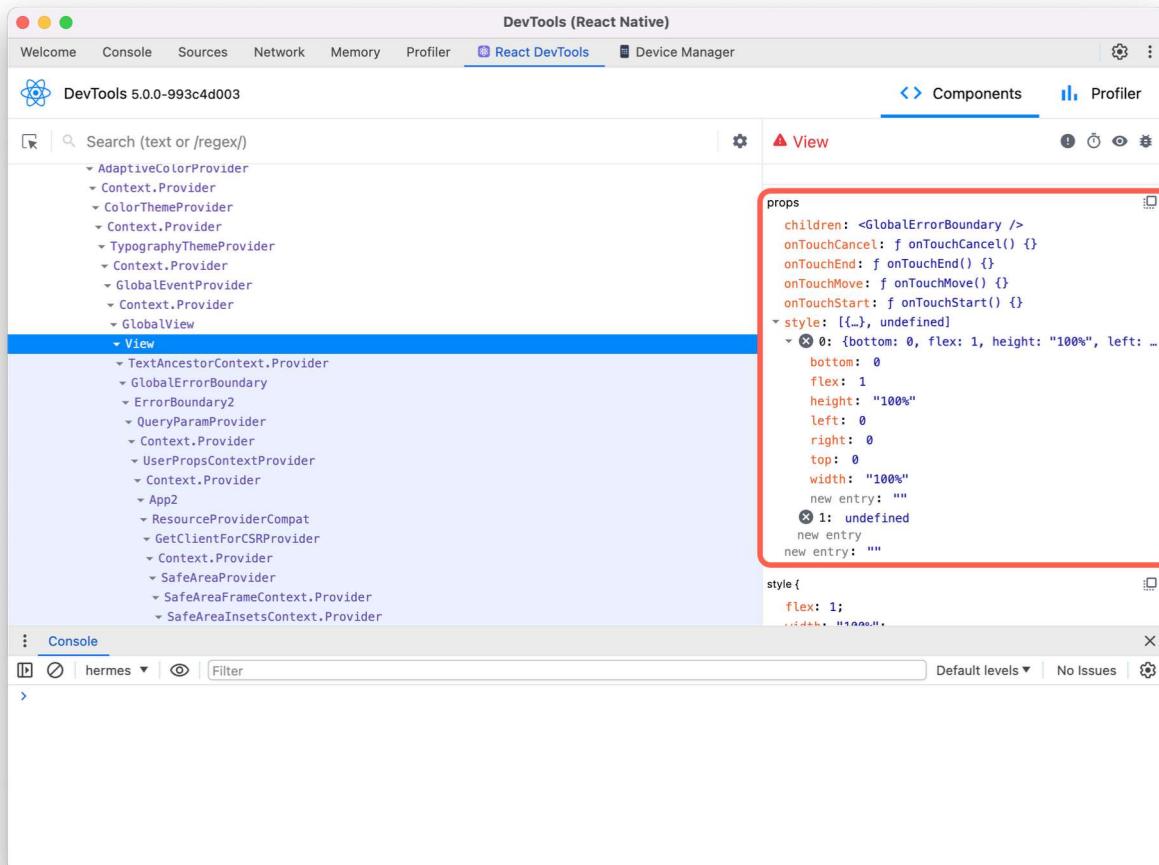
특정 요소를 쉽게 찾고 확인할 수 있는 기능이에요. 요소 선택 버튼을 누른 뒤, 테스트 중인 기기에서 확인할 요소를 터치하면 React DevTools에서 해당 요소로 바로 이동해요.



0:00 / 0:13

Prop 변경하기

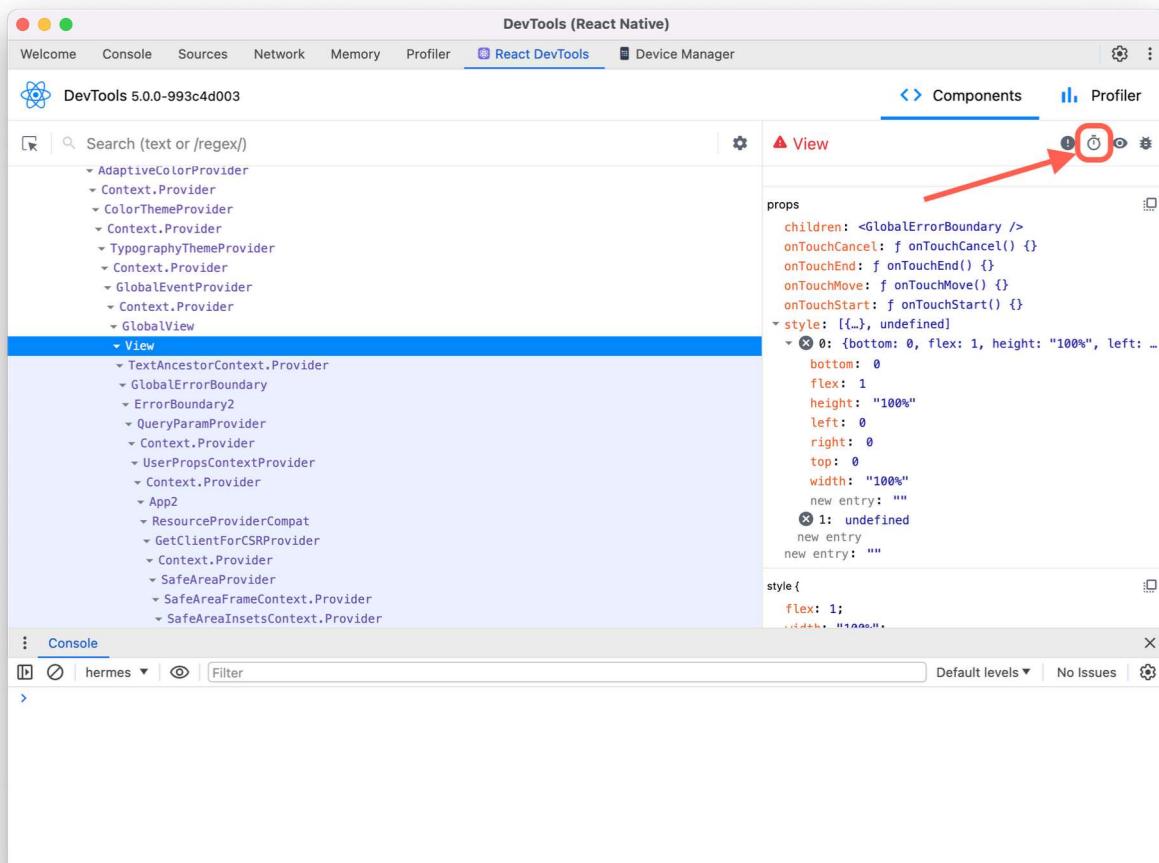
선택한 컴포넌트의 Prop(속성)을 확인하고, 실시간으로 변경할 수 있어요. 원하는 Prop을 더블 클릭한 뒤 값을 입력하면 바로 반영됩니다.



0:00 / 0:21

Suspense 테스트하기

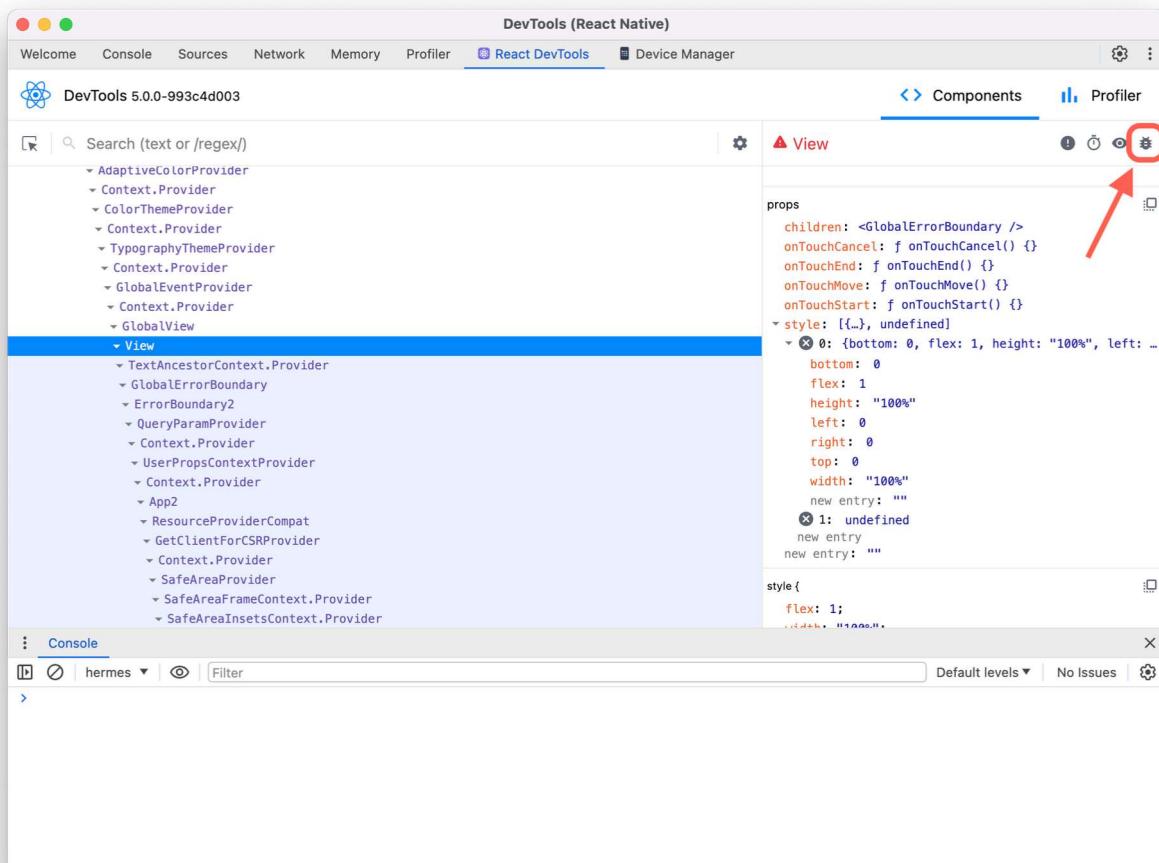
Suspense 요소의 대기 상태를 시뮬레이션할 수 있는 기능이에요. 타이머 아이콘을 눌러 Suspense 대기 상태를 테스트해볼 수 있어요.



0:00 / 0:09

상세 정보 로깅하기

특정 요소의 세부 정보를 콘솔에서 확인할 수 있어요. 디버그 버튼을 누르면 선택한 요소의 세부 정보가 콘솔에 기록됩니다.



0:00 / 0:27



트러블슈팅

- ▶ 연결 가능한 기기가 없다고 떠요
- ▶ REPL가 동작하지 않아요
- ▶ 네트워크 인스펙터가 동작하지 않아요

기타 다른 문제가 발생했다면, 다음을 시도해 보세요.

1. 앱을 완전히 종료해요.
2. 개발 서버를 중단하고 네트워크 인스펙터를 닫아요.
3. 앱을 다시 시작하고 `dev` 스크립트를 실행해 개발 서버를 재실행해요.

이 절차로도 문제가 해결되지 않으면, 담당자에게 제보해 주세요.

Previous page
[개발 서버 연결하기](#)

Next page
[WebView로 시작하기](#)

