



# 조회수 중복 증가 버그 해결 과정

React Strict Mode 환경에서의 안정적인 조회수 관리 시스템 구축



## 목차

1. 문제 개요
2. 문제 분석 과정
3. 최종 해결책
4. 핵심 해결 원리
5. 성능 및 안정성 개선
6. 테스트 시나리오
7. 디버깅 로그
8. 학습된 교훈
9. 향후 개선 방향



## 문제 개요

### 초기 문제

- 게시글 상세 페이지 진입 시 조회수가 **2씩 증가** 하는 현상 발생
- React Strict Mode에서 **useEffect**가 두 번 실행 되어 중복 호출 발생
- 서버에서는 정상적으로 1씩 증가하지만 UI에서는 2씩 증가하는 것으로 표시

## 기술적 배경

- 프레임워크: Next.js 14 (App Router)
- 개발 모드: React Strict Mode 활성화
- 상태 관리: React Hooks (useState, useEffect, useCallback)
- 백엔드: Supabase (PostgreSQL)

## 문제 분석 과정

1

### 1단계: 초기 진단

```
useEffect(() => { fetchComments() incrementViewCount()
  조회수 증가 }, [post.id])
```

Code

#### 문제점:

- React Strict Mode에서 useEffect가 두 번 실행
- 중복 방지 로직이 없어서 API 호출이 두 번 발생

2

### 2단계: 중복 방지 로직 추가 (실패)

```
// 첫 번째 시도: useState 기반 중복 방지 const [hasIncremented,
setHasIncremented] = useState(false) const
incrementViewCount = useCallback(async () => { if
(hasIncremented) return setHasIncremented(true) // API 호
출... }, [hasIncremented])
```

Code

**문제점:**

- React Strict Mode에서 상태 업데이트가 비동기적으로 처리되어 중복 방지 실패
- useEffect 의존성 배열에 hasIncremented가 포함되어 무한 루프 발생 가능

3

**3단계: 강화된 중복 방지 로직 (실패)**

```
// 두 번째 시도: 다중 상태 + 지연 시간 const [hasIncremented,
setHasIncremented] = useState(false) const [isProcessing,
setIsProcessing] = useState(false) const incrementViewCount
= useCallback(async () => { if (hasIncremented ||
isProcessing) return setHasIncremented(true)
setIsProcessing(true) await new Promise(resolve =>
setTimeout(resolve, 100)) // 이중 체크 로직... },
[hasIncremented, isProcessing])
```

Code

**문제점:**

- 과도한 복잡성으로 인한 예측 불가능한 동작
- 지연 시간이 사용자 경험에 부정적 영향
- 여전히 React Strict Mode의 중복 실행을 완전히 차단하지 못함

**✓ 최종 해결책****4단계: useRef 기반 중복 방지 (성공)**

Code

```
// 최종 해결책: useRef 기반 중복 방지
const hasIncrementedRef =
useRef(false)
const hasInitializedRef = useRef(false)
const
incrementViewCount = useCallback(async () => { // useRef 기반 중복
방지 체크 (React Strict Mode 완전 차단)
if
(hasIncrementedRef.current) { console.log('🚫 이미 조회수 증가됨
(useRef), 중복 호출 방지') return } // 즉시 ref 설정하여 중복 호출 방지
hasIncrementedRef.current = true
setHasIncremented(true)
try {
const response = await
fetch(`/api/board/posts/${post.id}/view`, { method: 'POST',
headers: { 'Content-Type': 'application/json' } })
const result
= await response.json()
if (response.ok && result.success) { //
서버에서 받은 정확한 조회수로 UI 업데이트
const newViewCount =
result.viewCount || (viewCount + 1)
setViewCount(newViewCount)
} } catch (error) { // 오류 시 ref와 상태 리셋
hasIncrementedRef.current = false
setHasIncremented(false)
} },
[post.id, viewIncrementKey]) // 컴포넌트 마운트 시 한 번만 실행
useEffect(() => { if (!hasInitializedRef.current) {
hasInitializedRef.current = true
fetchComments()
incrementViewCount()
} }, [post.id, fetchComments,
incrementViewCount])
```

#### ✅ 성공 요인:

- useRef는 리렌더링 간에 값을 유지하여 React Strict Mode의 중복 실행에서도 안정적
- 동기적 업데이트로 즉시 중복 방지 효과
- 의존성 배열에 영향을 주지 않아 무한 루프 방지



## 핵심 해결 원리

1

useRef의 특성 활용

2

서버 기반 상태 관리

# 3

## 상태 리셋 전략

### 1. useRef의 특성 활용

- 리렌더링 간 값 유지: React Strict Mode의 중복 실행에서도 안정적
- 동기적 업데이트: `ref.current = true`는 즉시 반영되어 중복 방지 효과적
- 의존성 배열 불필요: ref 값 변경이 리렌더링을 트리거하지 않음

### 2. 서버 기반 상태 관리

- 클라이언트 계산 제거: `setViewCount(prev => prev + 1)` 대신 서버 값 사용
- 단일 진실의 원천: 서버의 조회수가 유일한 정확한 값
- 동기화 보장: API 응답의 정확한 값을 UI에 반영

### 3. 상태 리셋 전략

- 게시글 변경 시: 모든 ref와 상태를 리셋하여 새로운 게시글에서 정상 작동
- 초기값 분리: `post.viewCount` 변경 시에만 초기값 설정
- 의존성 최소화: 불필요한 의존성을 제거하여 예측 가능한 동작

## 성능 및 안정성 개선

#### Before (문제 상황)

- ✗ 조회수 2씩 증가
- ✗ React Strict Mode 중복 실행

#### After (해결 후)

- ✓ 조회수 1씩 정확히 증가
- ✓ React Strict Mode 완전 차단

❌ 예측 불가능한 상태 업데이트

❌ 서버와 UI 불일치

✅ 안정적인 상태 관리

✅ 서버와 UI 완벽 동기화

## 테스트 시나리오



### 1. 기본 기능 테스트

게시글 클릭 → 조회수 1 증가 확인

목록으로 돌아가기 → 같은 게시글 재클릭 → 조회수 1 증가 확인



### 2. React Strict Mode 테스트

개발 모드에서 Strict Mode 활성화 상태에서 테스트  
useEffect가 두 번 실행되어도 조회수는 1씩만 증가






### 3. 에러 처리 테스트

네트워크 오류 시 → ref 리셋 확인

API 오류 시 → 재시도 가능 확인

## 디버깅 로그

### 성공적인 실행 로그

 PostDetail 컴포넌트 렌더링: [게시글ID]  초기 viewCount 설정: [게시글ID]  
회수]  post.id 변경 시 상태 및 ref 리셋: [게시글ID] hasIncremented:

Code

```
false hasIncrementedRef: false 🚩 PostDetail 초기화 실행 (useRef):
[게시글ID] hasIncrementedRef: false 🚩 incrementViewCount 호출: [게
시글ID] hasIncrementedRef: false 키: view_[게시글ID]_[타임스탬프]_[랜
덤] 🚩 조회수 증가 ref 설정됨 (동기) 🚩 조회수 증가 API 호출: [게시글ID] 키:
view_[게시글ID]_[타임스탬프]_[랜덤] 🚩 조회수 증가 API 응답: {success:
true, viewCount: [새로운조회수], ...} 🚩 조회수 증가 성공, UI 업데이트:
이전: [이전조회수] 새로운: [새로운조회수]
```

## 중복 방지 로그

```
🚩 incrementViewCount 호출: [게시글ID] hasIncrementedRef: true ...
view_[게시글ID]_[타임스탬프]_[랜덤] 🚩 이미 조회수 증가됨 (useRef), 중복 호
출 방지
```

Code

## 학습된 교훈

### 1. React Strict Mode 대응

- **useRef 활용:** 상태 기반 중복 방지보다 ref 기반이 더 안정적
- **동기적 업데이트:** ref 값 변경은 즉시 반영되어 중복 방지 효과적
- **의존성 최소화:** 불필요한 의존성을 제거하여 예측 가능한 동작

### 2. 상태 관리 원칙

- **단일 진실의 원천:** 서버 데이터를 기준으로 클라이언트 상태 관리
- **서버 기반 계산:** 클라이언트 계산보다 서버 계산이 더 안정적
- **명확한 리셋 전략:** 상태 변경 시점에 명확한 리셋 로직 필요

### 3. 디버깅 전략

- **상세한 로깅:** 각 단계별 상태 변화를 추적할 수 있는 로그
- **단계별 검증:** 각 해결책을 단계별로 적용하고 검증

- 실제 환경 테스트: 개발 환경과 프로덕션 환경 모두에서 테스트



## 향후 개선 방향

1

### 코드 최적화

- 커스텀 훅으로 조회수 증가 로직 분리
- 타입 안정성 강화
- 에러 바운더리 추가

2

### 성능 개선

- 조회수 증가 API 캐싱
- 배치 업데이트 고려
- 실시간 동기화 최적화

3

### 사용자 경험

- 조회수 증가 애니메이션
- 로딩 상태 표시
- 오프라인 지원

## 문서 정보

문서 작성일: 2024년 12월 19일

작성자: AI Assistant

프로젝트: 교회 청년부 커뮤니티 MVP

버전: 1.0.0