실전으로 배우는

웹 성능 최적화 for React

Lecture #4

실습내용

- 이미지 지연(lazy) 로딩
- Layout Shift 피하기
- useSelector 렌더링 문제 해결
- Redux Reselect를 통한 렌더링 최적화
- 병목 함수에 memoization 적용
- 병목 함수 로직 개선하기

실습내용

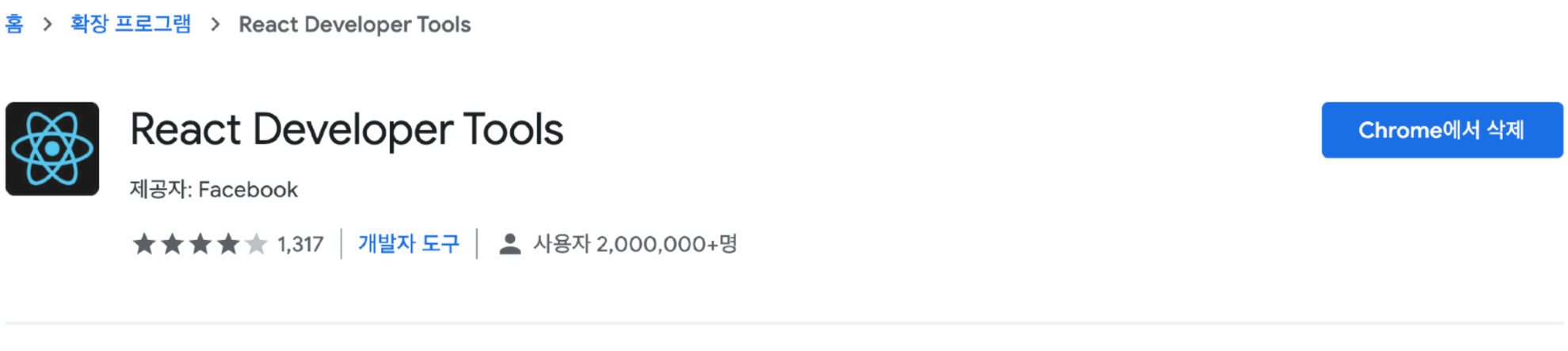
- 이미지 지연(lazy) 로딩 — 로딩 성능 최적화
- Layout Shift 피하기
- useSelector 렌더링 문제 해결
- Redux Reselect를 통한 렌더링 최적화
- 병목 함수에 memoization 적용
- 병목 함수 로직 개선하기

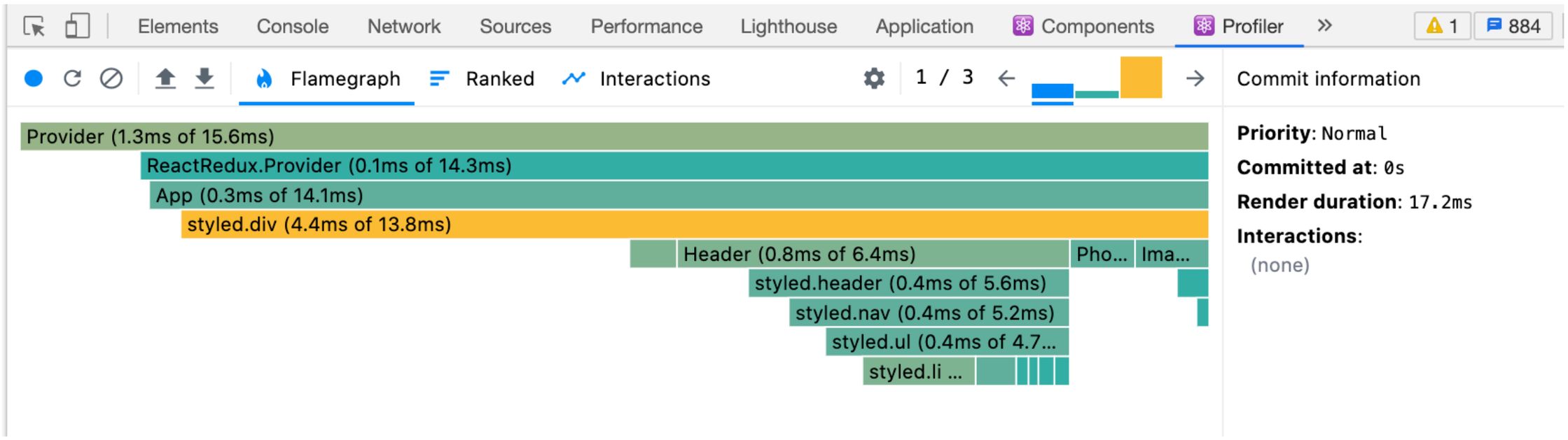
렌더링 성능 최적화

분석 툴

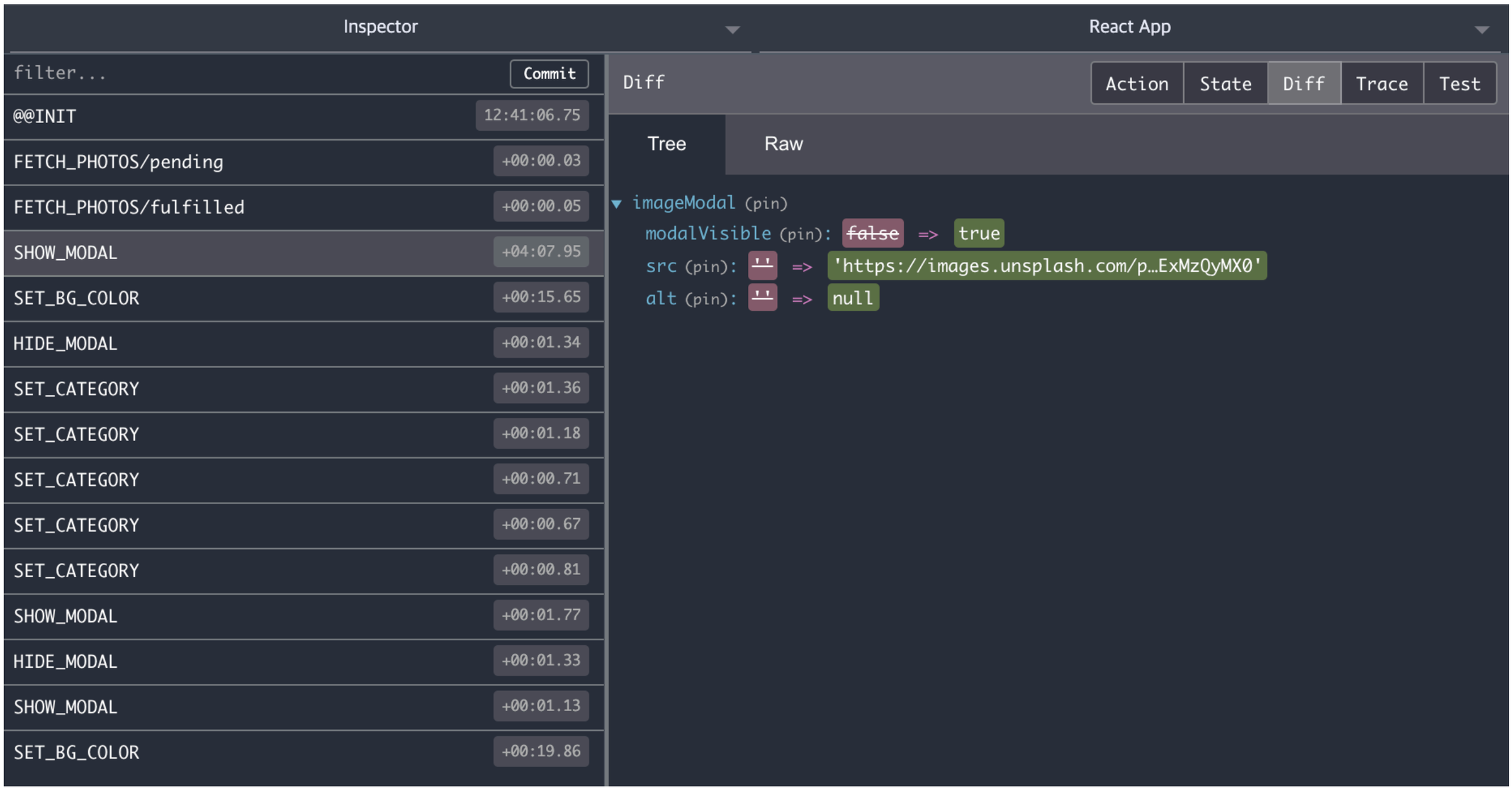
- 크롬 Network 탭
- 크롬 Performance 탭
- Lighthouse
- React Developer Tools (Profiler)
- Redux DevTools

React Developer Tools (Profiler)

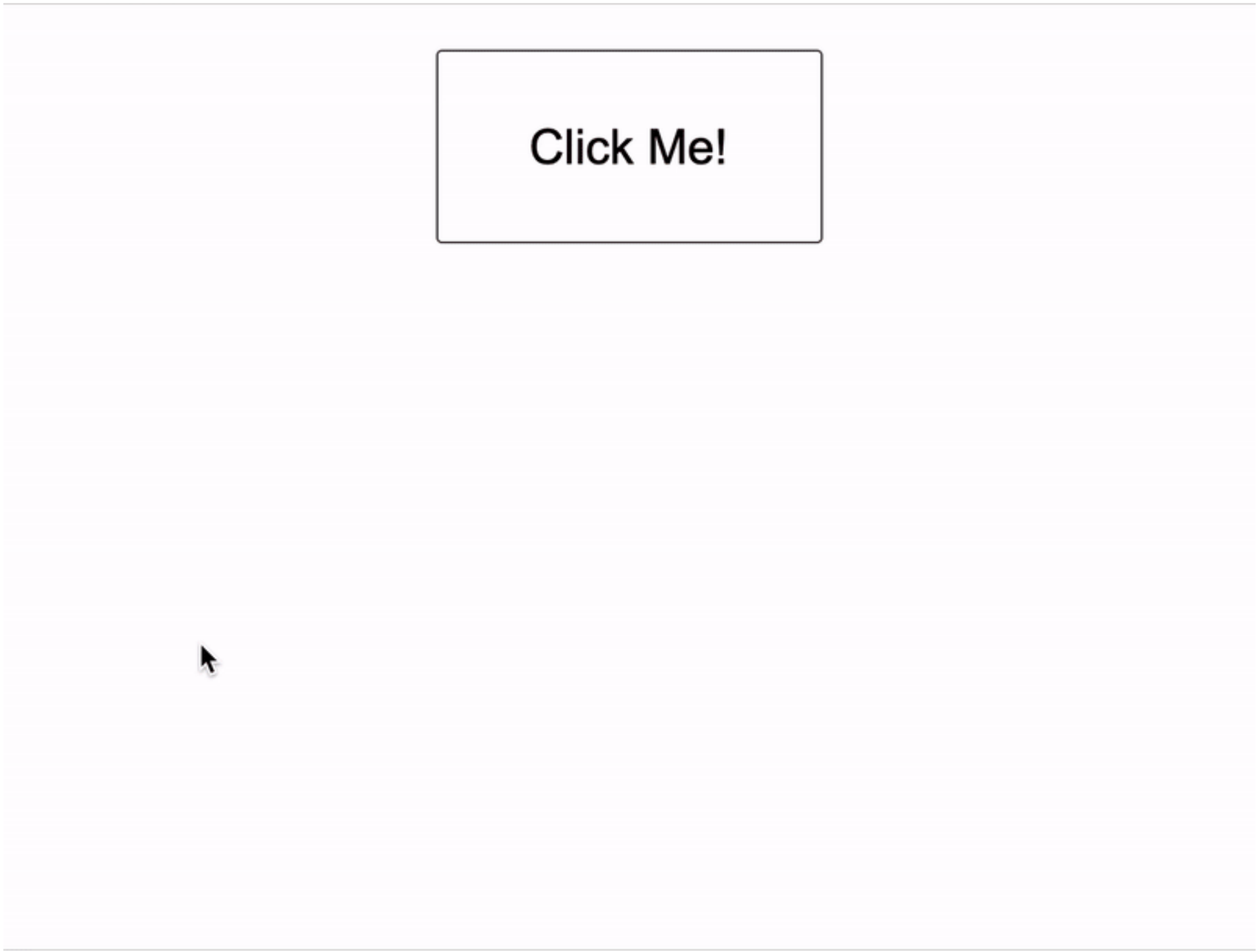




Redux DevTools



Layout Shift



Cumulative Layout Shift

성능

측정항목

First Contentful Paint	2.5 초	Time to Interactive	6.5 초
Speed Index	3.3 초	Total Blocking Time	400 밀리초
Largest Contentful Paint	3.2 초	Cumulative Layout Shift	0.127

값은 추정치이며 달라질 수 있습니다. 이러한 측정항목에서 성능 점수가 직접 계산됩니다. 계산기 보기

Layout Shift 원인

사이즈가 정해져 있지 않은 이미지

사이즈가 정해져 있지 않은 광고

동적으로 삽입된 콘텐츠

Web font (FOIT, FOUT)

useSelector 동작 원리

Component

```
useSelector(state => ({
    a: state.a,
    b: state.b
}));
```

새로운 State 비교

값이 다르면,

rendering

useSelector 문제 해결 방법

1. Object를 새로 만들지 않도록 State 쪼개기

2. 새로운 Equality Function 사용

정리

Layout Shift 피하기

↓

이미지 지연(lazy) 로딩

↓

useSelector 렌더링 문제 해결

↓

Redux Reselect를 통한 렌더링 최적화

↓

병목 함수에 memoization 적용

↓

병목 함수 로직 개선하기

정리

Layout Shift 피하기

↓

이미지 지연(lazy) 로딩

↓

useSelector 렌더링 문제 해결

↓

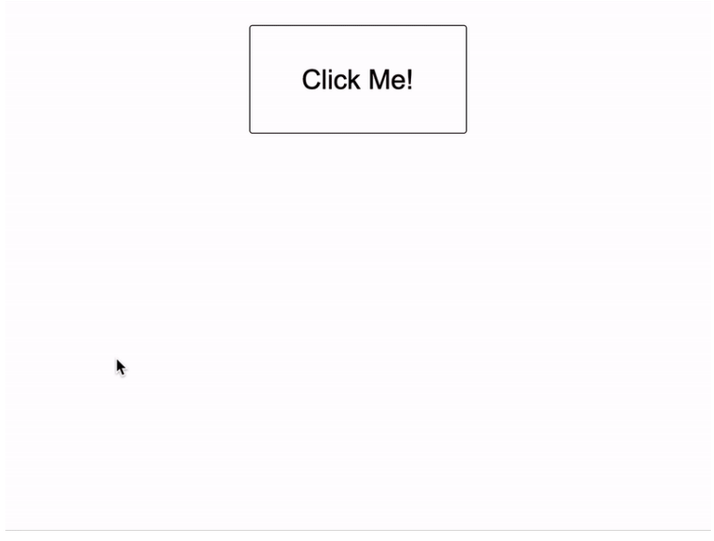
Redux Reselect를 통한 렌더링 최적화

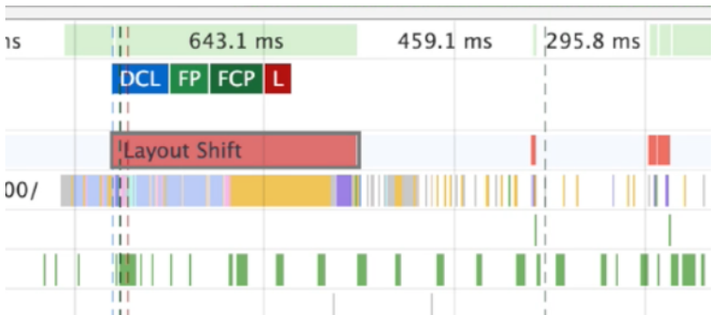
↓

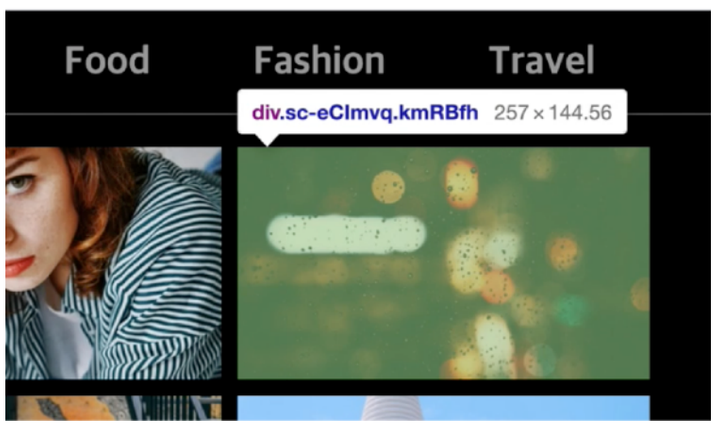
병목 함수에 memoization 적용

↓

병목 함수 로직 개선하기







정리

Layout Shift 피하기

↓

이미지 지연(lazy) 로딩

↓

useSelector 렌더링 문제 해결

↓

Redux Reselect를 통한 렌더링 최적화

↓

병목 함수에 memoization 적용

↓

병목 함수 로직 개선하기

```
import { useDispatch } from 'react-redux';
import LazyLoad from 'react-lazyload';
import { showModal } from '../redux/imageModal';

function PhotoItem({ photo: { id, urls, alt } }) {
  const dispatch = useDispatch();

  const openModal = () => {
    dispatch(showModal({ src: urls.full, alt, id }));
  };

  return (
    <ImageWrap>
      <LazyLoad offset={1000}>
        <Image
          crossOrigin="*"
          id={id}
          src={urls.small}
          alt={alt}
          onClick={openModal}
        />
      </LazyLoad>
    </ImageWrap>
  );
}
```

정리

Layout Shift 피하기

↓

이미지 지연(lazy) 로딩

↓

useSelector 렌더링 문제 해결

↓

Redux Reselect를 통한 렌더링 최적화

↓

병목 함수에 memoization 적용

↓

병목 함수 로직 개선하기

```
const { photos, loading } = useSelector(state => ({
  photos:
    state.category.category === 'all'
      ? state.photos.data
      : state.photos.data.filter(
          photo => photo.category === state.category.category
        ),
  loading: state.photos.loading,
}));
```

```
const { category, allPhotos, loading } = useSelector(
  state => ({
    category: state.category.category,
    allPhotos: state.photos.data,
    loading: state.photos.loading,
  }),
  shallowEqual
);
```

```
const category = useSelector(state => state.category.category);
const allPhotos = useSelector(state => state.photos.data);
const loading = useSelector(state => state.photos.loading);
```

정리

Layout Shift 피하기

↓

이미지 지연(lazy) 로딩

↓

useSelector 렌더링 문제 해결

↓

Redux Reselect를 통한 렌더링 최적화

↓

병목 함수에 memoization 적용

↓

병목 함수 로직 개선하기

```
import { createSelector } from 'reselect';

export default createSelector(
  [state => state.photos.data, state => state.category.category],
  (photos, category) =>
    category === 'all'
      ? photos
      : photos.filter(photo => photo.category === category)
);
```

```
const photos = useSelector(selectFilteredPhotos);
const loading = useSelector(state => state.photos.loading);
```

정리

Layout Shift 피하기

↓

이미지 지연(lazy) 로딩

↓

useSelector 렌더링 문제 해결

↓

Redux Reselect를 통한 렌더링 최적화

↓

병목 함수에 memoization 적용

↓

병목 함수 로직 개선하기

```
const cache = {};

export function getAverageColorOfImage(imgElement) {
  if (cache.hasOwnProperty(imgElement.src)) {
    return cache[imgElement.src];
  }
```

.
.
.

```
  cache[imgElement.src] = averageColor;

  return averageColor;
}
```

정리

Layout Shift 피하기

↓

이미지 지연(lazy) 로딩

↓

useSelector 렌더링 문제 해결

↓

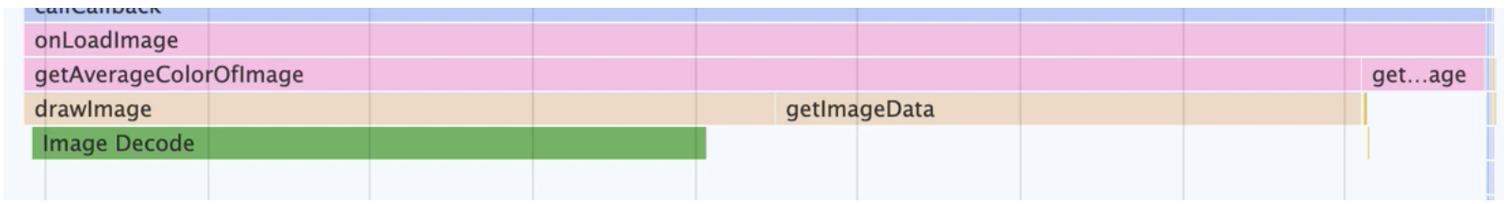
Redux Reselect를 통한 렌더링 최적화

↓

병목 함수에 memoization 적용

↓

병목 함수 로직 개선하기




```
const onLoadImage = e => {
  const averageColor = getAverageColorOfImage(
    document.querySelector(`#${id}`)
  );
  dispatch(setBgColor(averageColor));
};
```

```
canvas.width = width / 3;
canvas.height = height / 3;

context.drawImage(imgElement, 0, 0, canvas.width, canvas.height);
```

```
for (let i = 0; i < length; i += 40) {
  averageColor.r += imageData[i];
  averageColor.g += imageData[i + 1];
  averageColor.b += imageData[i + 2];
}
```

```
background-color: ${({ bgColor }) =>
  `rgba(${bgColor.r}, ${bgColor.g}, ${bgColor.b}, 0.8)`};
transition: background-color 1s ease;
```

4장 끝

수고하셨습니다