API와 SPA를 활용한 음원검색 및 재생목록

프론트 프로젝트

서민서

이젠아카데미강남
KDT 풀스택

목차

프로젝트 배경

개발 동기 및 배경

1. 사용자 경험의 개선

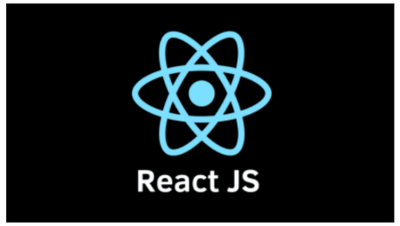
유명인이 듣는 음원에 대한 관심이 높지만, 매번 검색하고 노래를 하나하나 확인하는 것은 번거롭습니다. 유명인의 차트를 직관적으로 확인하고, 여러 음원 사이트의 데이터를 종합적으로 제공함으로써 더 풍부하고 빠른 음악 경험을 제공할 필요성이 커졌습니다.

2. 차트의 분산성 문제

현재 음원 사이트마다 차트가 달라, 통합된 인기 음원을 파악하기 어려운 상황입니다. 사이트별 1위가 아닌 종합적인 인기 음원을 제공할 필요성이 커졌습니다.

개발 환경

기술 스택



라이브러리

- axios
- react
- react-router-dom
- react-icons
- styled-components

협업 툴

수행절차

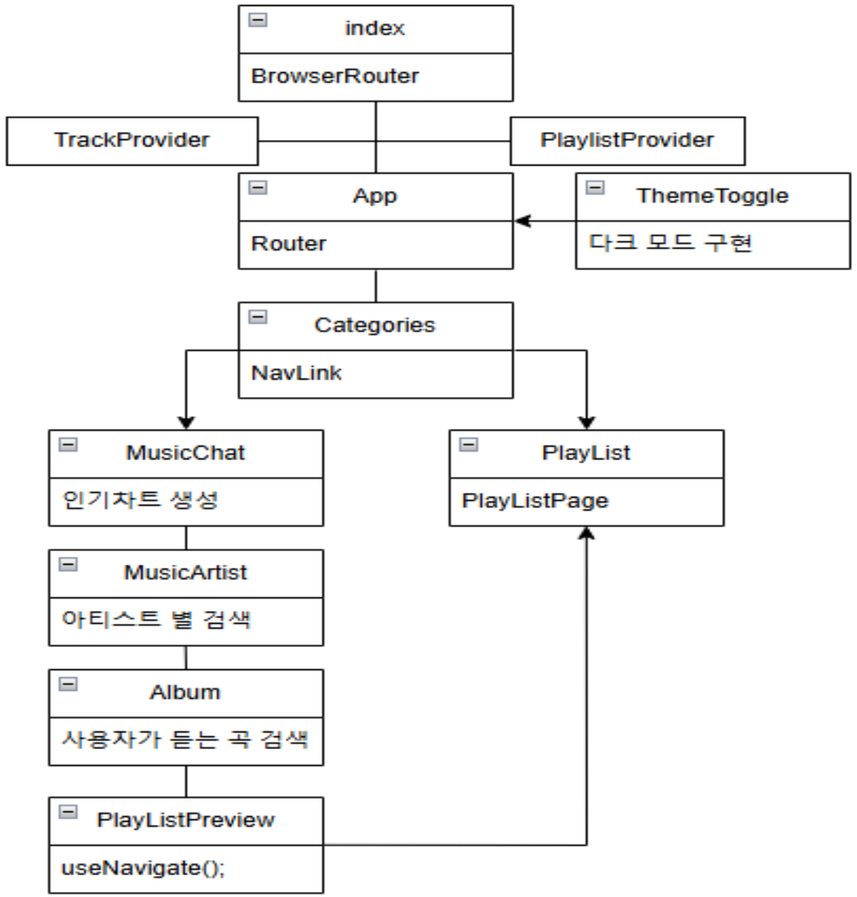
수행절차 별 날짜

주제 선정 및 spa 기본 구상	API 연결 및 기본 코드 작성	API 연결 후 오류 확인	CRUD 구현	성능개선
API 존재 여부 및 필요성 확인	API주소 확인 키 발급 및 연결 후 오류 확인	오류부분 확인 후 수정 진행	CRUD구현 및 세부 기능 구현	마지막 세부 오류 확인 및 성능 측정 후 개선
04/11~04/14	04/15~04/16	04/17~04/18	04/21~04/22	04/23

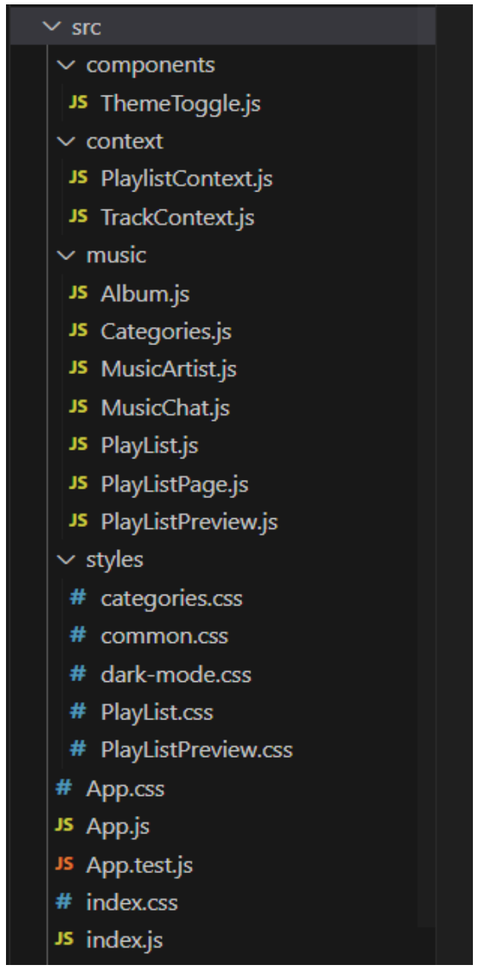
구현 결과

music 웹 서비스 구현을 위한 FLOW CHART

FLOW CHART



폴더 목록



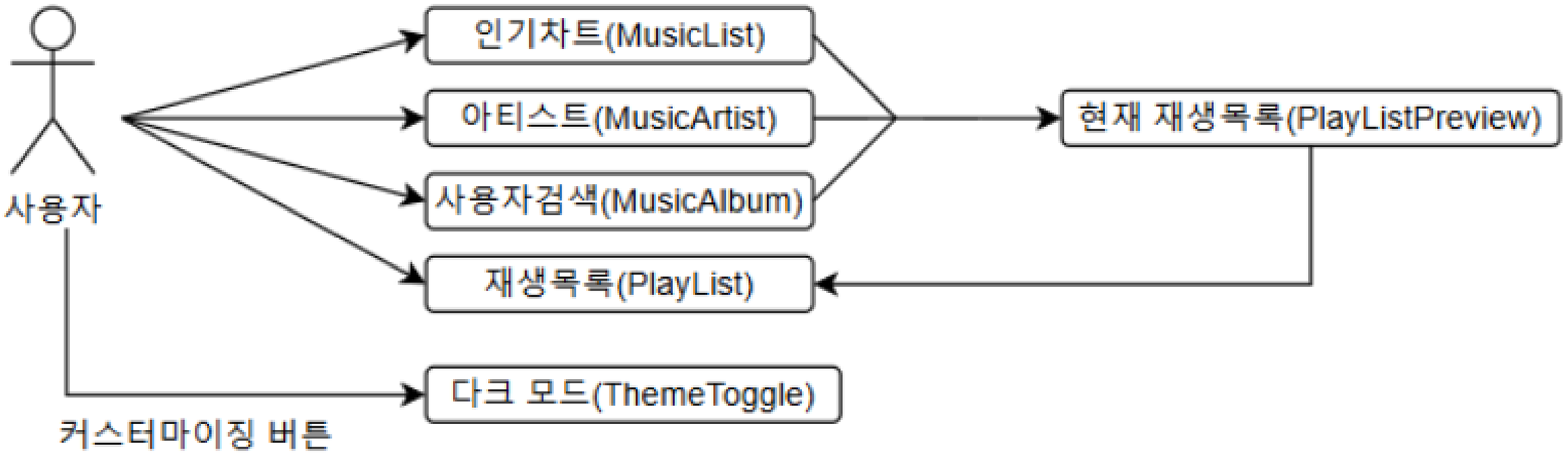
components: 사용자가 커스터마이징할 수 있는 다양한 UI 컴포넌트를 포함하는 디렉토리

context: 재생목록 등 애플리케이션의 상태 관리 및 기능 관련 로직을 담당하는 디렉토리

music: 사용자가 직접 상호작용하는 음악 관련 UI 요소들이 포함된 디렉토리

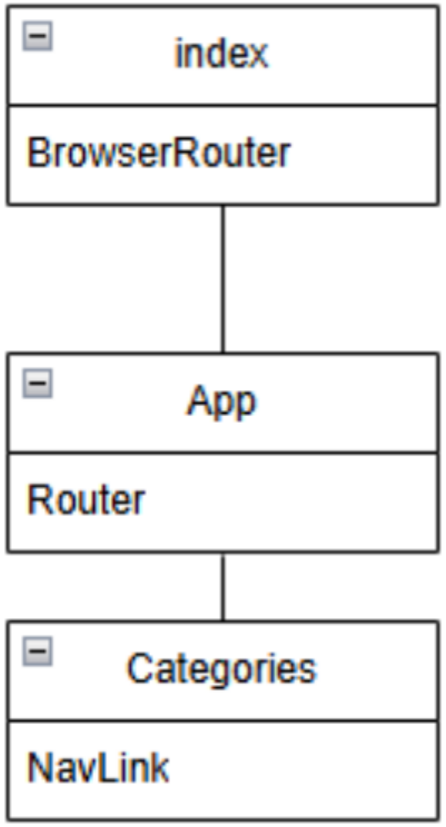
styles: 공통적으로 사용되는 CSS 및 스타일 파일을 정리한 디렉토리

구현 결과 Usecase



구현 결과 spa구성

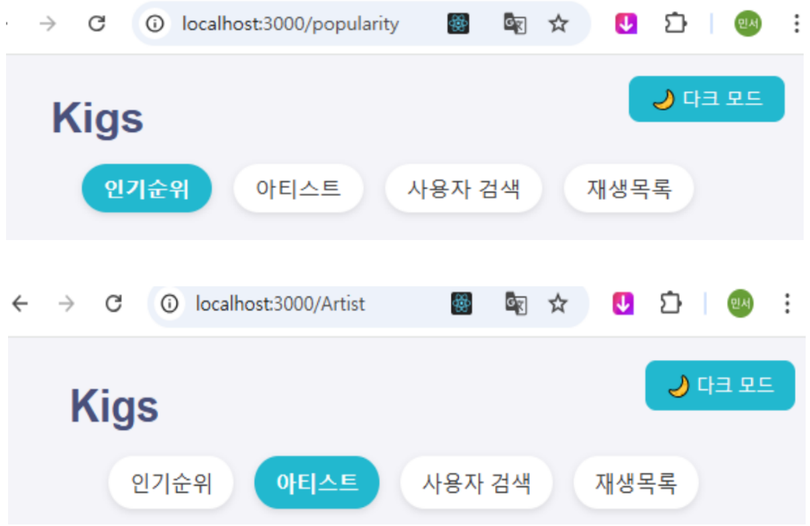
- FLOWCHART(기본 구성)



- Router Hook을 이용하여 SPA구성(App.js)

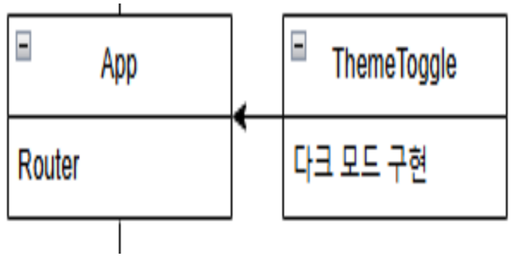
```jsx
<Routes>
  <Route path="/" element={<Navigate to="/popularity" replace />}
  <Route path="/popularity" element={<MusicChat />} />
  <Route path="/Artist" element={<MusicArtist />} />
  <Route path="/Album" element={<Album />} />
  <Route path="/PlayList" element={<PlayListPage />} />
</Routes>
```

- 각 카테고리에 따른 경로구성



구현 결과 다크 모드spa구성

- 다크 모드 플로우차트



- 다크 모드 구현 버튼 구성

```
function App() {
  const [darkMode, setDarkMode] = useState(() => {
    const stored = localStorage.getItem("darkMode");
    return stored === "true";
  });
```

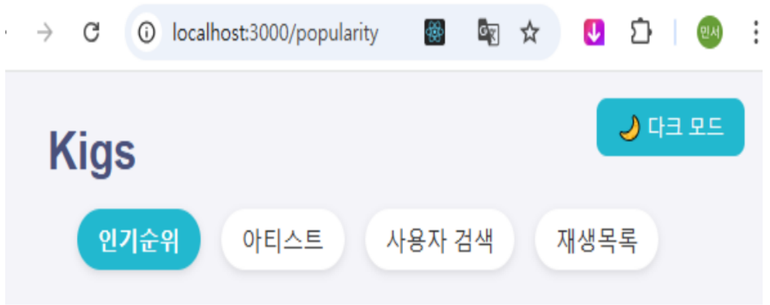
- 다크 모드 상태 전달 함수

```
return (
  <TrackProvider>
    <PlaylistProvider darkMode={darkMode}>
      <AppContent darkMode={darkMode} toggleDarkMode={toggleDarkMode} />
    </PlaylistProvider>
  </TrackProvider>
);
```

- 사용자의 새로고침시 다크 모드가 풀림 방지

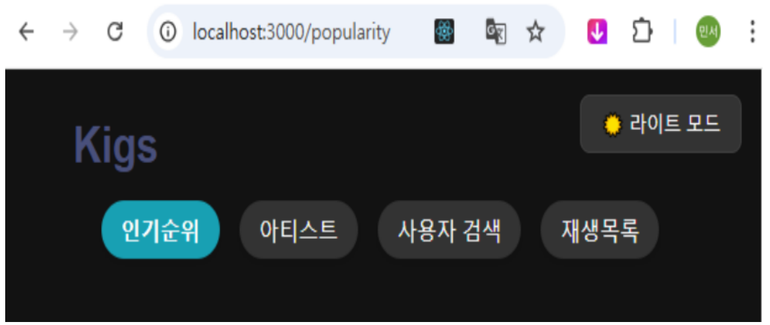
```
useEffect(() => {
  document.body.classList.toggle("dark", darkMode);
}, [darkMode]);
```

- 화이트 모드



다크 모드 적용 후

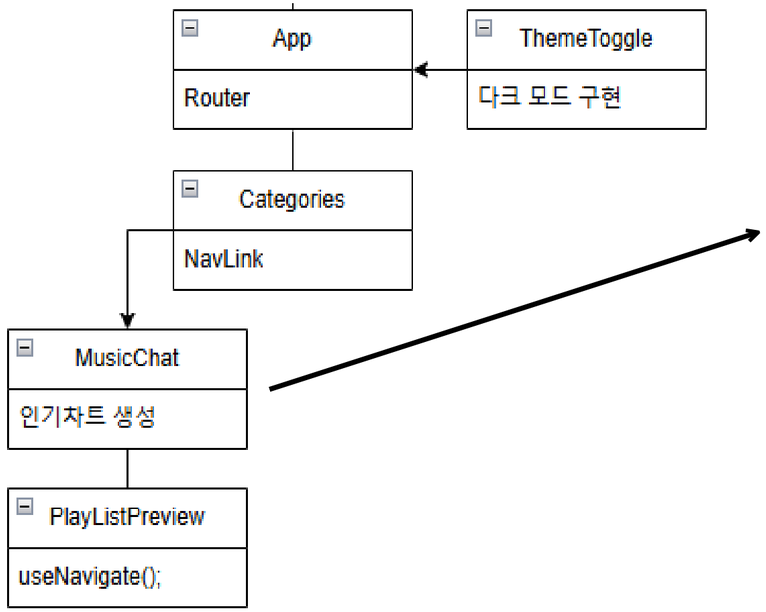
- 다크 모드



구현 결과

API연결 및 무한스크롤 구현

인기차트 FlowChart



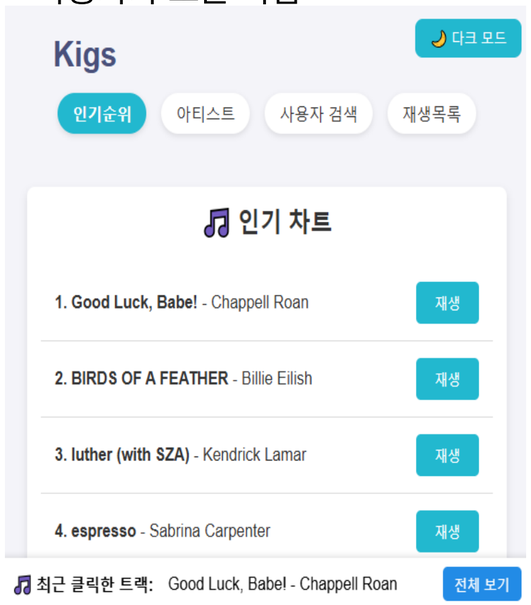
API연결

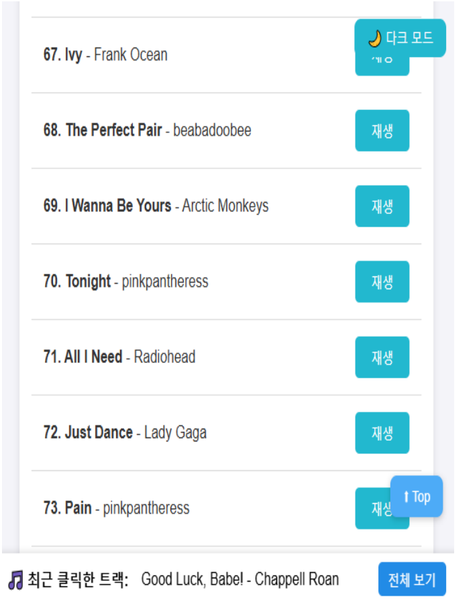
```
const fetchTracks = useCallback(async () => {
  try {
    const limit = 30;
    const apiKey = process.env.REACT_APP_LASTFM_API_KEY;
    const response = await fetch(
      `https://ws.audioscrobbler.com/2.0/?method=chart.gettoptracks&api_key
    );
    const data = await response.json();
    const newTracks = data.tracks.track;
```

무한스크롤 구현

```
useEffect(() => {
  const observer = new IntersectionObserver(
    (entries) => {
      if (entries[0].isIntersecting && hasMore) {
        setPage((prev) => prev + 1);
      }
    },
    { threshold: 1 }
```

사용자가 보는 시점



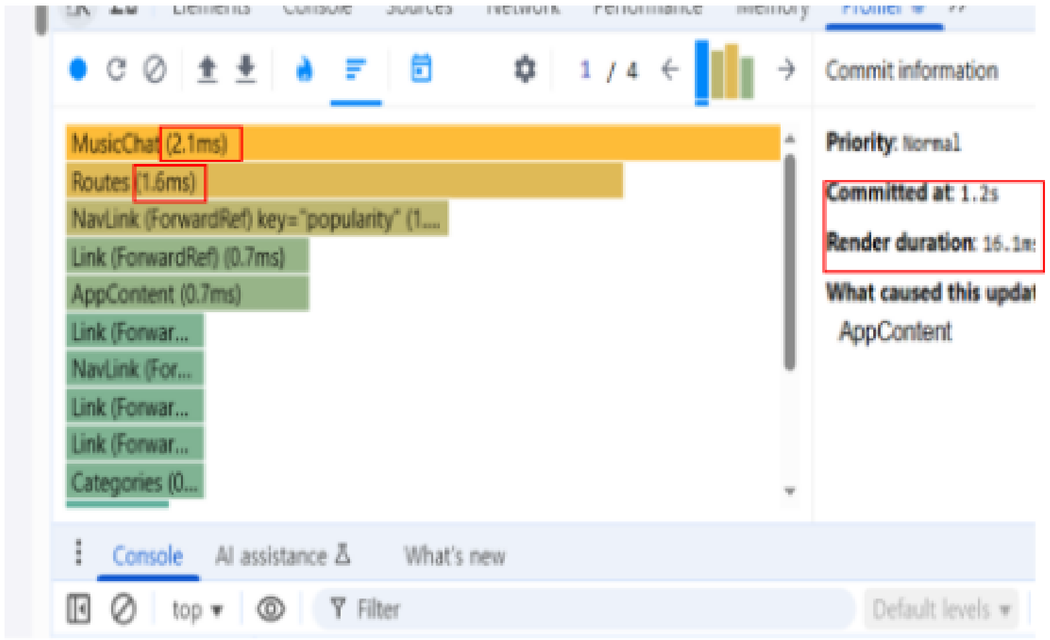


구현 결과

무한스크롤에 따른 성능 비교

BEFORE

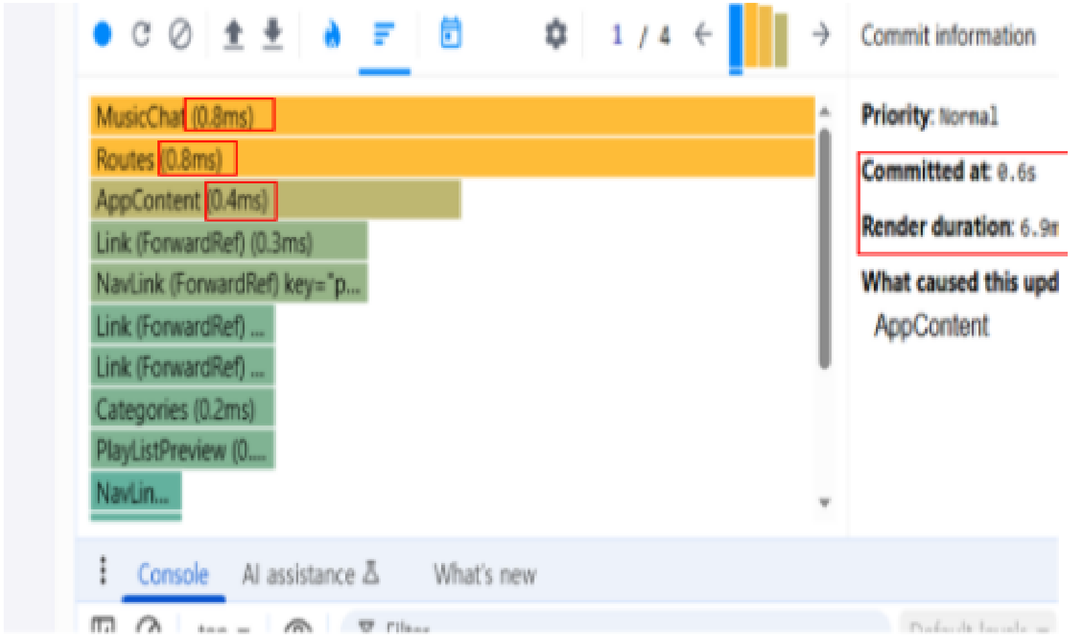
라우터로 성능 개선 전



VS

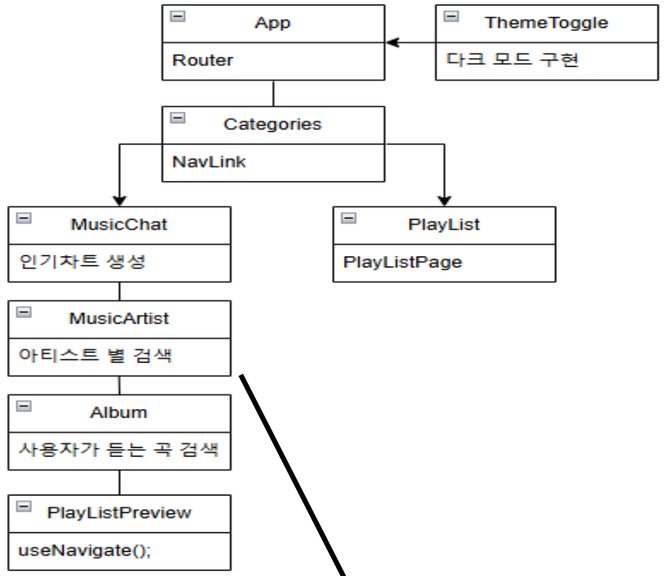
AFTER

라우터로 성능 개선 후



구현 결과 API연결

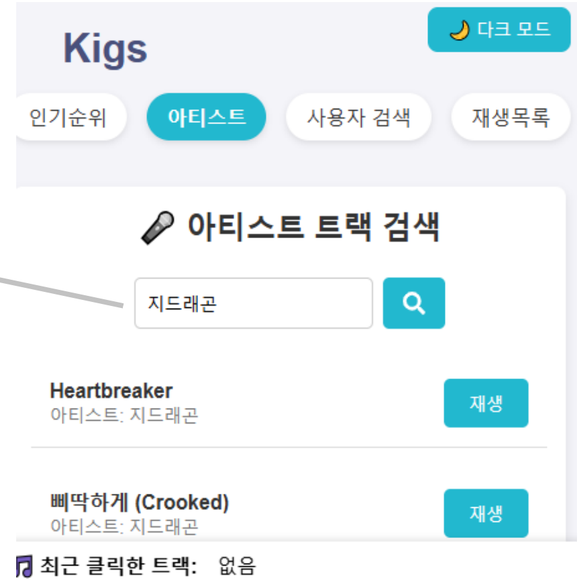
인기차트 FlowChart




```
const handleSearch = () => {
  const trimmed = inputValue.trim();
  if (!trimmed) return;
  setTracks([]);
  setPage(1);
  setHasMore(true);
  fetchTracks(trimmed, 1);
};
```

검색어 입력

변수 전달



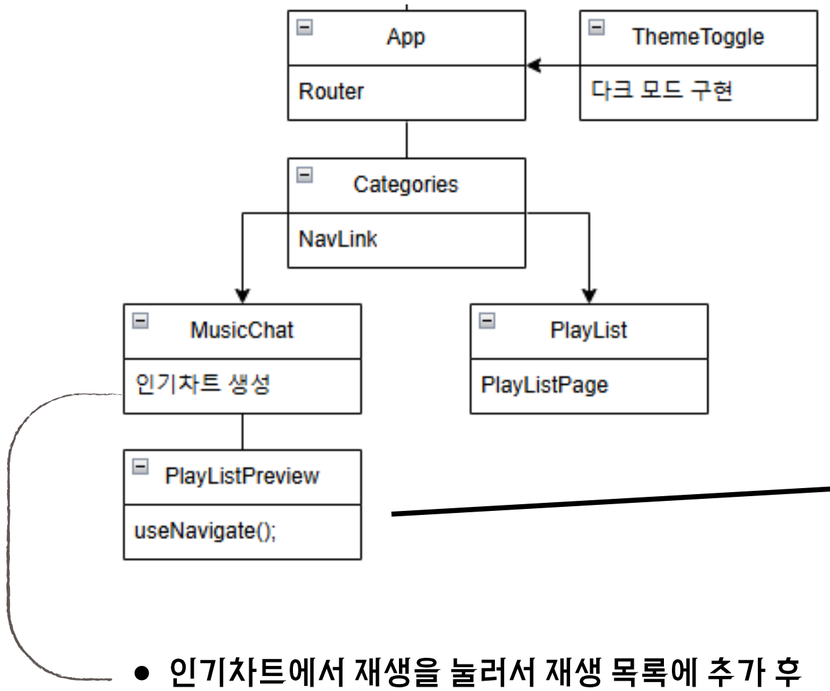
API출력

```
const fetchTracks = useCallback(async (currentArtist, currentPage) => {
  if (!currentArtist) return;
  try {
    const API_URL = `https://ws.audioscrobbler.com/2.0/?method=artist.gettoptracks&artist=${currentArtist}&api_key=${API_KEY}&format=json&limit=2
```

- Value에 검색어 값 전달
- trim에 변수 API에 전달
- 링크에서 API출력

method=artist.gettoptracks&artist=${currentArtist}&api_key=${API_KEY}&format=json&limit=20&page=${currentPage}`KEY

구현 결과 CRUD구현



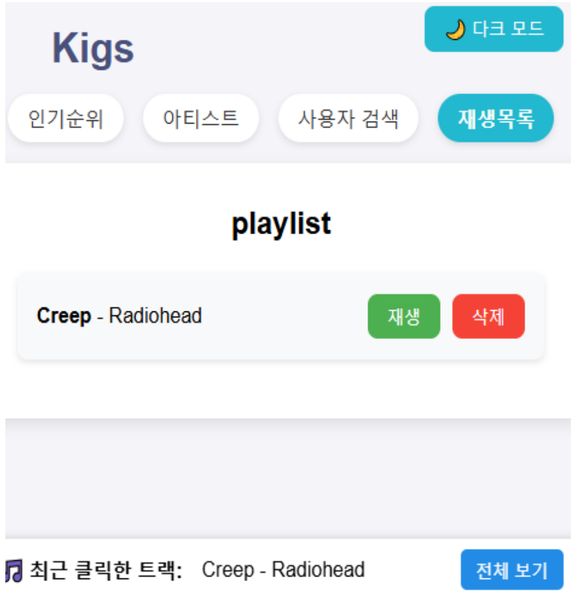
- 인기차트에서 재생을 눌러서 재생 목록에 추가 후

```
const handleTrackClick = (track) => {
  console.log("Track clicked:", track);
  addTrack({
    name: track.name,
    artist: { name: track.artist.name },
    url: track.url,
  });
```

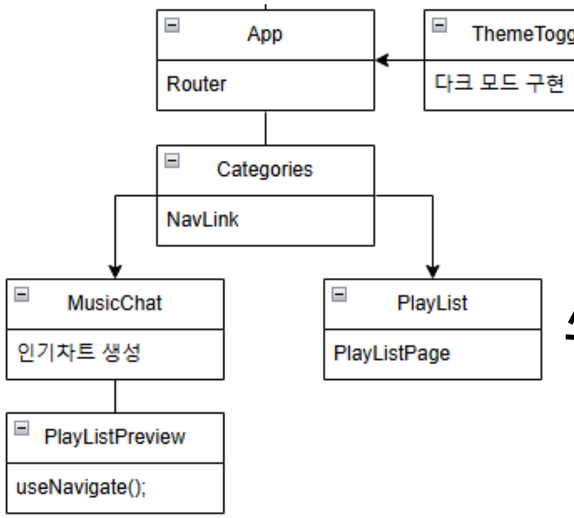
- 인기차트에서 재생을 눌렀을때 가장 최근에 클릭한 트랙 조회

```
return (
  <div className="playlist-preview">
    {" "}
    <strong className="playlist-preview__title">🎵 최근 클릭한 트랙:</strong>
    {lastClickedTrack ? (
      <span>
        {lastClickedTrack.name} -{" "}
        {typeof lastClickedTrack.artist === "string"
          ? lastClickedTrack.artist
          : lastClickedTrack.artist?.name}
      </span>
    ) : (
      <span>없음</span>
    )}
    {playlist.length > 0 && (
      <button
        onClick={() => navigate("/PlayList")}
        className="playlist-preview__button"
      >
        전체 보기
      </button>
```

- 사용자가 보는 화면



구현 결과 CRUD구현



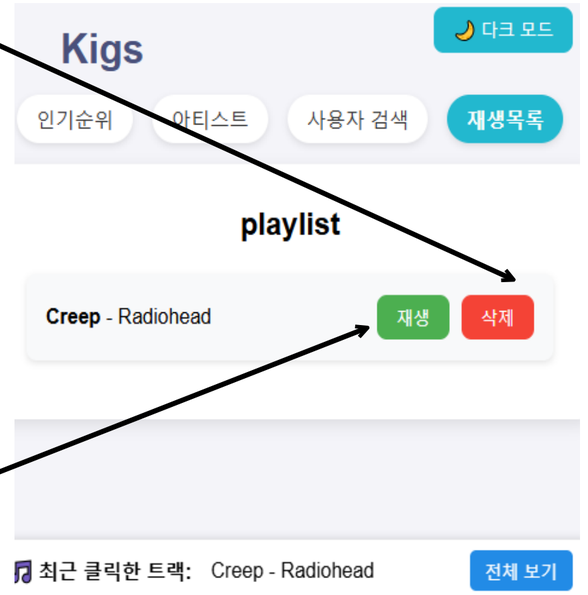
- 재생목록에서 삭제 구현

```
const handleRemove = (index) => {
  removeTrack(index);
};
```

- 재생을 눌렀을때의 재생 구현

```
const handlePlay = (track) => {
  const previewUrl = track?.url;
  if (previewUrl) {
    setCurrentTrackUrl(previewUrl);
    if (audioRef.current) {
      audioRef.current.pause();
      audioRef.current.load();
      audioRef.current.play().catch((e) => {
        console.warn("재생이 차단됨:", e);
      });
    }
  } else {
    alert("재생 가능한 미리듣기 URL이 없습니다.");
  }
```

- 사용자가 보는 화면



오류 수정 문제 해결

	문제점	해결 방법	처리 결과
성능 하락	재생목록 컴포넌트를 만들어서 연결하니 서버가 다운	성능 향상 적용(무한스크롤 적용) 공통으로 들어가는 css는 하나로 묶어서 적용	재생목록을 연결하고 재생, 삭제를 눌러도 잘 구현완료
기능 오류	PlayListPreview로 가장 최근에 선택한 노래가 뜨고 재생목록에 노래가 삭제되면 없음으로 업데이트 되야하는데 적용 x	playlist.length === 0일 경우 " 재생목록이 비어 있습니다."라는 텍스트가 나타나도록 else 분기를 추가 및 모든 트랙을 삭제했을 때 lastClickedTrack이 null로 변경	업데이트 적용 완료
기능오류	F5를 눌렀을때 다크 모드가 풀려버리는 현상 발생	확실하게 동기화를 보장하기 위해 다크 모드 초기값 설정 → 토글 → body 클래스 적용	사용자가 F5를 눌렀을때도 다크모드가 풀리는 것이 아닌 다크 모드랑 버튼은 그대로 유지

느낀점

구분	내용
느낀 점	이번 음악 웹앱 개발을 통해 전역 상태 관리의 필요성과 컴포넌트 간 데이터 흐름의 중요성을 실감했다. 재생목록 기능 구현과 다크모드 대응, 무한스크롤 처리 등 실제 사용자 환경을 고려한 설계 경험이 개발 역량을 한층 성장시키는 계기가 되었다.
어려운 점	이번 개발에서는 사용자 입장에서 바라보던 서비스를 직접 구현하며 기대했던 기능들이 쉽게 구현되지 않아 어려움을 느꼈다. 오류가 반복되며 원인을 정확히 파악하지 못해 불필요한 수정을 하기도 했고, 생각만큼 실력이 따라주지 않아 아쉬움과 혼란을 겪었다.
잘한점	기획 단계부터 직접 참여하고 구현, 오류 해결까지 주도적으로 진행하면서 전체 개발 과정을 스스로 이끌었다. 사용자 중심의 UI 구성, 재생목록 기능, 무한스크롤, 전역 상태 관리 등 핵심 기능을 성공적으로 적용해낸 점이 가장 잘한 부분이다.
아쉬운 점	사용자의 트랙 검색 시, 다른 사용자에게 공유하고 싶은 곡만 보이게 하는 기능을 구현하고 싶었지만, 백엔드 기술 부족으로 적용하지 못해 아쉬웠다. 앞으로 로그인 기능과 사용자 맞춤 음악 추천 기능을 구현하며 백엔드 영역까지 확장할 것이다.

감사합니다!

프로젝트 개발자: 서민서