

react-router 심화



1. react-router 소개

❖react-router란?

- 리액트 애플리케이션의 요청 경로(주소)에 따라 각기 다른 컴포넌트를 렌더링하기 위한 라이브러리

❖Router 선택

- 컴포넌트 기반 Router
 - `<BrowserRouter />`, `<MemoryRouter />`, `<HashRouter />`
- Data Router API 최신 기법 요즘엔 이렇게 사용
 - `createBrowserRouter`, `createMemoryRouter`, `createHashRouter`
- 어떤 것을 선택할 것인가?
 - 새로운 Router API는 Data API를 지원함
 - 컴포넌트 기반 라우터는 Data API를 지원하지 않음

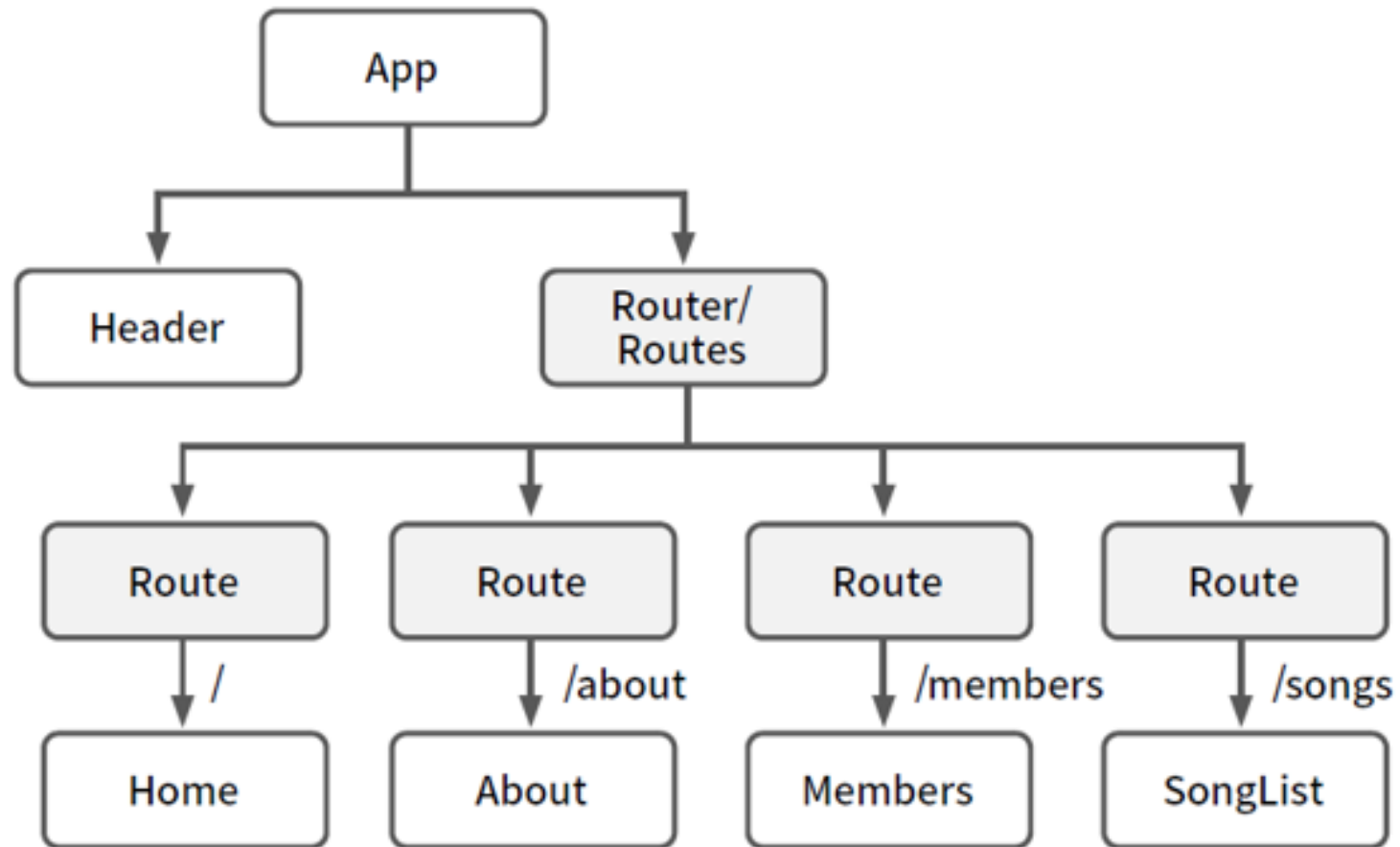
❖react-router Data API

- 기존의 react-router 기능에 데이터와 UI를 쉽게 동기화할 수 있는 데이터 추상화 기능을 추가한 새로운 API
- 6.4 버전 부터 지원

2. 컴포넌트 기반 라우터 예제 검토

❖ foxes-band-app-2 예제 검토

- 예제 아키텍처



2. 컴포넌트 기반 라우터 예제 검토

❖src/App.tsx 컴포넌트

- `<Router />` : Route 정보를 처리하는 방식을 지원하는 컴포넌트
- `<Route />` : 각 요청 경로에 대해 렌더링할 컴포넌트 지정

```
import { BrowserRouter as Router, Routes, Route } from "react-router-dom";  
.....  
const App = () => {  
  return (  
    <Router>  
      <div className="container">  
        <Header />  
        <Routes>  
          <Route path="/" element={<Home />} />  
          <Route path="/about" element={<About title={"여우와 늑다리들"} />} />  
          <Route path="/members" element={<Members />} />  
          <Route path="/songs" element={<SongList />} />  
          <Route path="/songs/:id" element={<SongDetail />} />  
        </Routes>  
      </div>  
    </Router>  
  );  
};  
export default App;
```

2. 컴포넌트 기반 라우터 예제 검토

❖src/BandProvider.tsx

- Context API를 사용하기 위한 Provider 컴포넌트
- songs, members 데이터를 상태로 보유 -> Context의 value 로 제공함

❖src/main.tsx

- Context API Provider(BandProvider) 를 컴포넌트 트리에 제공

```
import React from "react";
import ReactDOM from "react-dom/client";
import "bootstrap/dist/css/bootstrap.css";
import App from "./App";
import "./index.css";
import { BandProvider } from "./BandProvider";

ReactDOM.createRoot(document.getElementById("root")!).render(
  <React.StrictMode>
    <BandProvider>
      <App />
    </BandProvider>
  </React.StrictMode>
);
```

2. 컴포넌트 기반 라우터 예제 검토

❖src/components/Header.tsx

- <Link /> 컴포넌트 : 네비게이션을 위한 링크 기능 제공

```
import { Link } from "react-router-dom";

const Header = () => {
  return (
    <div className="card bg-light">
      <div className="card-heading">
        <h2 className="text-center m-3">Foxes And Fossils</h2>
        .....(생략)
      <div className="row">
        <div className="col-12">
          <Link className="btn btn-success menu" to="/">Home</Link>
          <Link className="btn btn-success menu" to="/about">About</Link>
          <Link className="btn btn-success menu" to="/members">Members</Link>
          <Link className="btn btn-success menu" to="/songs">Songs</Link>
        </div>
      </div>
    </div>
  );
};

export default Header;
```

2. 컴포넌트 기반 라우터 예제 검토

❖라우트 컴포넌트로 속성 전달

- <Route />에 의해 렌더링되는 컴포넌트에서 속성 전달

```
<Route path="/about" element={<About title={"여우와 늑다리들"} />} />
```

```
type Props = { title: string };

const About = (props: Props) => {
  return (
    <div className="card card-body">
      <h2>Abou {props.title}</h2>
    </div>
  );
};

export default About;
```

2. 컴포넌트 기반 라우터 예제 검토

❖ 동적 파라미터

- <Route />에 의해 렌더링되는 컴포넌트에서 속성 전달
- 예) /songs/10 으로 요청한 경우 --> id 파라미터 값으로 "10" 이 전달됨

```
<Route path="/songs/:id" element={<SongDetail />} />
```

```
.....(생략)
const SongDetail = () => {
  const value = useContext(BandContext);
  const { id } = useParams<SongParam>();
  ..... (생략)

  return (
    <div className="mt-5">
      .....(생략)
    </div>
  );
};

export default SongDetail;
```


3. Data Router API 적용하기

❖ Data Router API 적용시 주의 사항

- `<RouterProvider />` 컴포넌트를 이용해 최상위 컴포넌트부터 경로별로 렌더링하도록 작성해야 함
 - 내부적으로 Context API를 사용함
 - 중첩 라우트(nested route)를 사용하는 경우가 빈번함
- route 객체 배열 형태로 routes 정보를 작성해야 함.

❖ createBrowserRouter() 함수

- 브라우저의 DOM History API를 사용하여 URI 경로 리스트를 업데이트하고 관리하는 기능 제공
- `<BrowserRouter />` 컴포넌트와 유사함
- 중첩라우트는 상위 라우트에서 children 속성을 이용하여 정의함

3. Data Router API 적용하기

❖src/router/index.tsx 작성

```
import { createBrowserRouter } from "react-router-dom";
import App from "../App";
import Home from "../pages/Home";
import About from "../pages/About";
import Members from "../pages/Members";
import SongList from "../pages/SongList";
import SongDetail from "../pages/SongDetail";

const router = createBrowserRouter([
  {
    path: "/",
    element: <App />,
    children: [
      { index: true, element: <Home /> },
      { path: "about", element: <About title={"여우와 늑다리들"} /> },
      { path: "members", element: <Members /> },
      { path: "songs", element: <SongList /> },
      { path: "songs/:id", element: <SongDetail /> },
    ],
  },
]);

export default router;
```

3. Data Router API 적용하기

❖src/main.tsx 변경

- <RouterProvider /> 컴포넌트를 이용해 router 지정
- router 객체의 최상 경로별 컴포넌트가 <RouterProvider /> 컴포넌트 위치에 렌더링됨

```
import React from "react";
import ReactDOM from "react-dom/client";
import "bootstrap/dist/css/bootstrap.css";
import "./index.css";
import { BandProvider } from "./BandProvider";
import { RouterProvider } from "react-router-dom";
import router from "./router";

ReactDOM.createRoot(document.getElementById("root")!).render(
  <React.StrictMode>
    <BandProvider>
      <RouterProvider router={router} />
    </BandProvider>
  </React.StrictMode>
);
```

3. Data Router API 적용하기

❖ src/App.tsx 변경

- children 속성에 정의된 중첩 라우트에 의해 렌더링될 위치를 <Outlet /> 컴포넌트로 지정

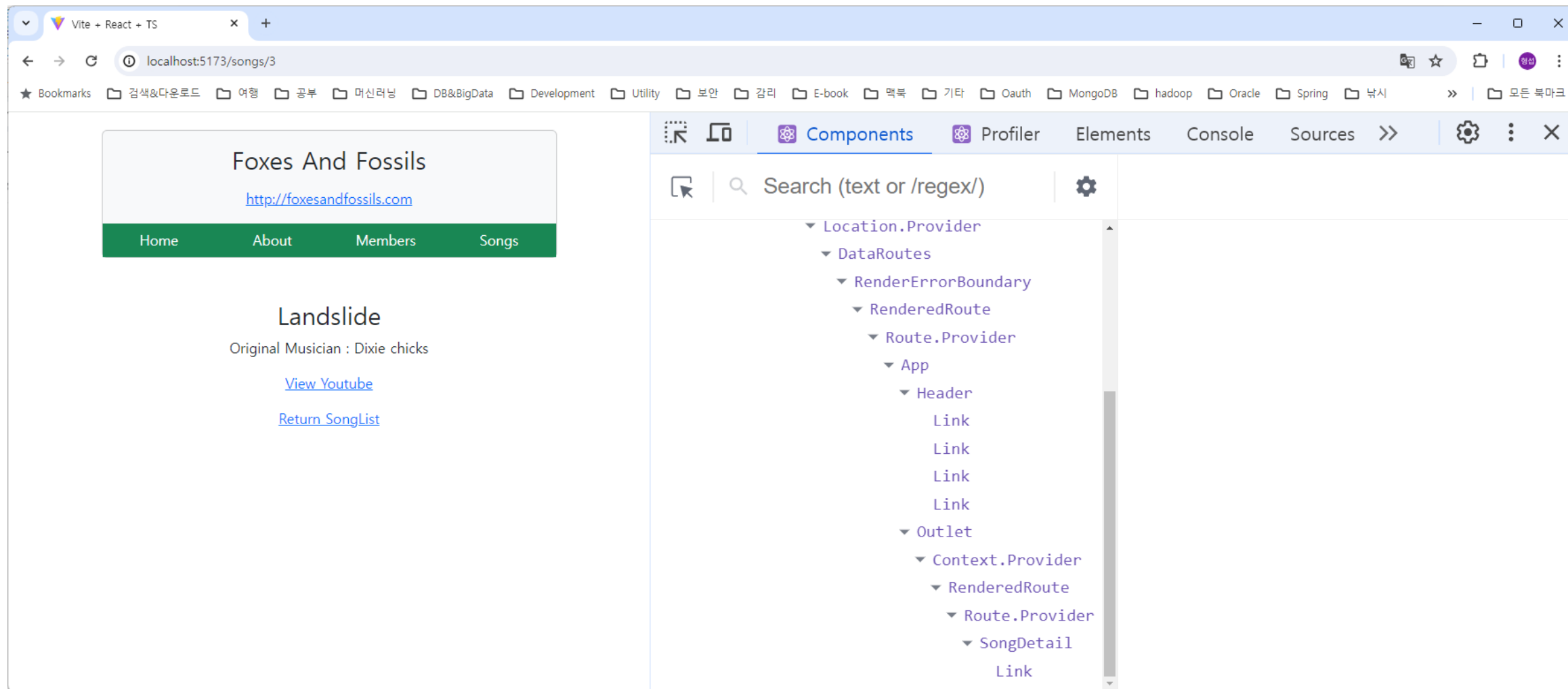
```
import { Outlet } from "react-router-dom";
import Header from "../components/Header";

const App = () => {
  return (
    <div className="container">
      <Header />
      <Outlet />
    </div>
  );
};

export default App;
```

3. Data Router API 적용하기

❖ 실행 결과



4. 중첩 라우트

❖ 중첩 라우트(nested route)

- 라우트 객체에 의해 렌더링된 컴포넌트에 기존 라우트의 children 속성의 라우트 컴포넌트가 렌더링 되도록 구성하는 방법

❖ 기존 예제 실행 결과 리뷰

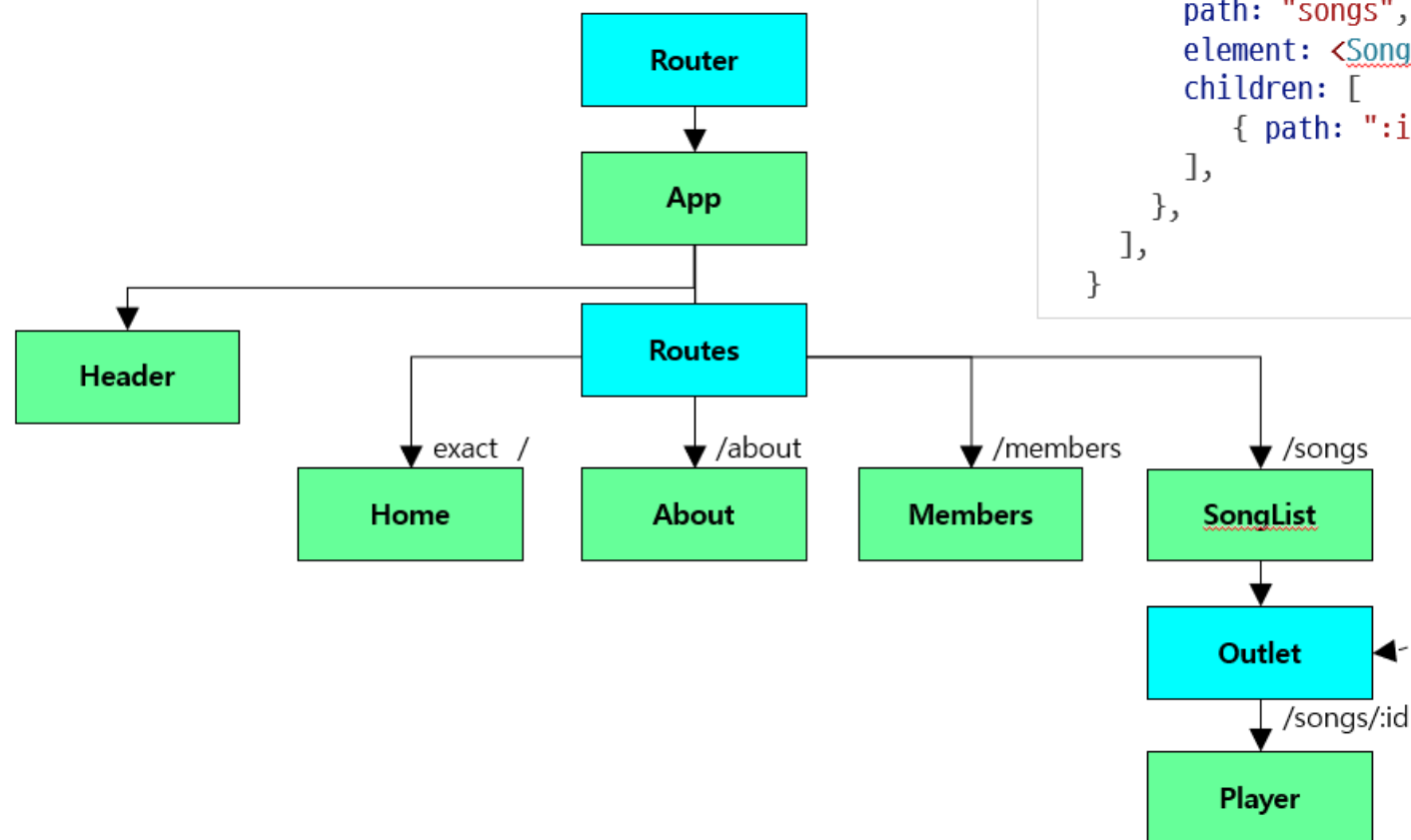
- /songs 로 요청 : SongList 컴포넌트 렌더링
- /songs/:id 로 요청 : SongDetail 컴포넌트 렌더링

❖ 중첩 라우트로 하려는 것

- /songs 로 요청 : SongList 컴포넌트 렌더링
- /songs/:id 로 요청 : SongList 컴포넌트 + 중첩된 라우트의 컴포넌트 렌더링
 - 중첩된 라우트의 컴포넌트를 렌더링하기 위해 상위 컴포넌트(SongList)에 <Outlet /> 컴포넌트가 필요함

4. 중첩 라우트

❖아키텍처



```
{
  path: "/",
  element: <App />,
  children: [
    .....
    {
      path: "songs",
      element: <SongList />,
      children: [
        { path: ":id", element: <Player /> }
      ],
    },
  ],
}
```

4. 중첩 라우트

❖ 기존 예제에 중첩 라우트 적용

- 실행중인 예제 중단하고 다음 명령어 실행하여 youtube 컴포넌트 다운로드
 - npm install react-youtube
- src/components/Player.tsx 추가

```
import { useContext, useState } from "react";
import { useParams, useNavigate } from "react-router";
import { Link } from "react-router-dom";
import Youtube from "react-youtube";
import BandContext from "../BandProvider";

type SongIdParam = { id: string };

const Player = () => {
  const navigate = useNavigate();
  const value = useContext(BandContext);
  const params = useParams<SongIdParam>();
  const id = params.id ? parseInt(params.id, 10) : 0;
  const song = value && value.songs.find((song) => song.id === id);
  if (!song) navigate("/songs");

  const [title] = useState<string>(song?.title ? song.title : "");
  const [youtubeLink] = useState<string>(song?.youtube_link ? song.youtube_link : "");
```


4. 중첩 라우트

- src/components/Player.tsx 추가(이어서)

[illegible]

4. 중첩 라우트

▪ src/pages/SongList.tsx 변경 : <Outlet />

```
.....(생략)
const SongList = () => {
  const value = useContext(BandContext);
  const list = value && value.songs.map((song) => {
    return (
      <li className="list-group-item" key={song.id}>
        <Link to={`/songs/${song.id}`} style={{ textDecoration: "none" }}>
          {song.title} ( {song.musician} )
          <span className="float-end badge bg-secondary">
            ▶
          </span>
        </Link>
      </li>
    );
  });
  return (
    <div>
      <h2 className="mt-4 mb-2">Song List</h2>
      <ul className="list-group">{list}</ul>
      <Outlet />
    </div>
  );
};
export default SongList;
```

영상을 플레이할 수 있다는 의미를 부여하기 위해
작성한 플레이어 버튼
한글 '▶' 입력후 '한자'키를 눌러서 아이콘 선택

중첩 라우트를 렌더링할 위치 지정

4. 중첩 라우트

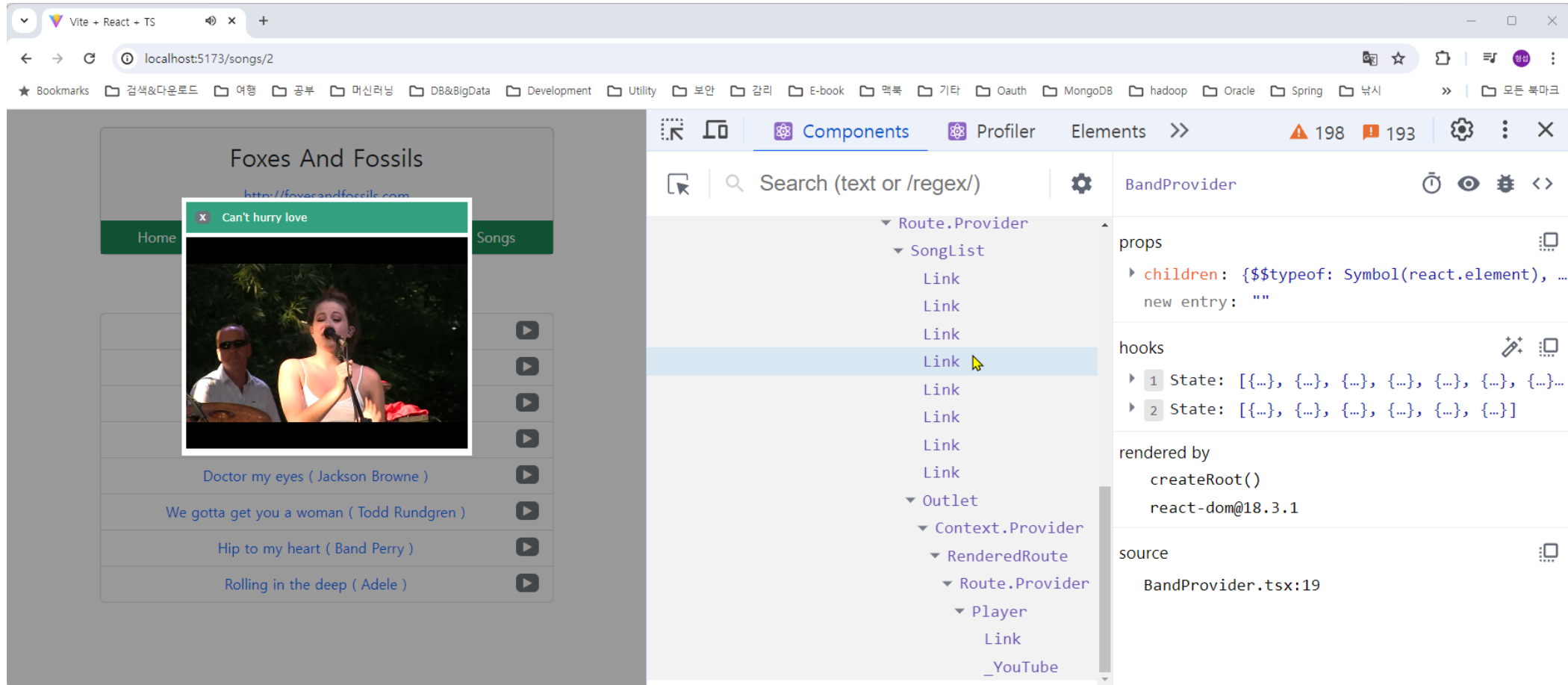
❖src/index.css 변경

- Player 컴포넌트를 모달로 보여주기 위한 CSS 스타일 지정

```
.....(기존 스타일 생략)
/* 모달을 위한 스타일 추가 */
.modal { display: block; position: fixed; z-index: 1;
  left: 0; top: 0; width: 100%; height: 100%;
  overflow: auto; background-color: rgb(0,0,0);
  background-color: rgba(0,0,0,0.4); }
.box { background-color: white; margin:100px auto;
  max-width: 330px; min-width: 100px; min-height: 250px;
  font: 12px "verdana"; padding: 5px 5px 5px 5px; }
.box div { padding: 0; display: block; margin: 5px 0 0 0; }
.box .heading { background: #33A17F; font-weight: 300; text-align: left;
  color: #fff; margin:0px; padding: 10px 10px 10px 10px; min-width:200px; max-width:360px; }
.box .player { background:white; }
.pointer { cursor:pointer; }
.play-button { width:15px; height:15px; }
.play-button-disabled { opacity:0.3 }
```

4. 중첩 라우트

❖ 실행 결과



5. loader

❖ 최근 트렌드

- 비동기로 데이터를 조회 후 -> 컴포넌트 렌더링
- 비동기 처리 과정에서 UI로 표현해야 할 것
 - 비동기 데이터 처리 진행 중에 fallback UI를 제공해야 함.
 - 비동기 데이터 처리가 완료된 후 리액트 컴포넌트가 렌더링해야 함
 - 처리과정 중 오류가 발생하면 오류 페이지를 보여줄 수 있어야 함

❖ react-router의 loader란?

- 각 라우트의 컴포넌트가 렌더링되기 전에 컴포넌트에 데이터를 제공하기 위한 기능
- 비동기 처리 가능
- 비동기 처리 진행중 상태를 반영해 fallback UI 제공
- loader + <Suspense> + <Await>

5. loader

❖ 기존 예제에 loader 적용하기 위해서 준비할 것

- Backend API 서버
 - 강사로부터 제공받음 : mock-server
 - 터미널에서 npm install 명령 실행 후 npm run dev 명령어로 서버 구동
 - 제공하는 데이터
 - members 데이터, songs 데이터
 - 사용 Endpoint
 - GET /songs_long : 곡 정보 전체 조회 + 2초의 지연시간
 - GET /songs/:id : 특정 한 곡 정보 조회
 - GET /members_long : 멤버 리스트 조회 + 2초의 지연 시간
 - 요청 테스트
 - http://localhost:3000/members_long
- 리액트 프로젝트 (foxes-band-app)
 - npm install axios react-csspin@0.0.4
 - React 18는 0.0.4 설치, React 19 최신 버전 설치
 - react-csspin은 처리 시간동안 보여줄 spinner UI 기능
 - <https://github.com/stepanowon/react-csspin>

5. loader

❖예제 리팩토링

- src/BandProvider.tsx 삭제

❖src/loaders/index.ts 작성

- loader 함수 작성 : /members, /songs, /songs/:id 라우트에서 사용하기 위한 loader
 - Promise 객체로 구성된 지연된 객체(deferred object)를 리턴해야 함
- /songs/:id 경로의 :id와 같은 동적 파라미터는 loader 함수의 아규먼트에서 params 속성을 이용하여 값을 받아낼 수 있음

```
import axios, { AxiosResponse } from "axios";  
import { defer } from "react-router-dom";  
  
axios.defaults.baseURL = "http://localhost:3000";  
  
export type SongType = { id: number; title: string; musician: string; youtube_link: string };  
export type MemberType = { id:number; name: string; photo: string };
```

5. loader

❖src/loaders/index.ts 작성 (이어서) - react-router v6 기준

```
export const membersLoader = async () => {
  const promiseMembers: Promise<MemberType[]> =
    axios.get<MemberType[], AxiosResponse<MemberType[]>>("/members_long")
      .then((response)=>response.data);
  return defer({ members: promiseMembers })
}

export const songListLoader = async () => {
  const promiseSongs: Promise<SongType[]> =
    axios.get<SongType[], AxiosResponse<SongType[]>>("/songs_long")
      .then((response)=>response.data);
  return defer({ songs: promiseSongs })
}

type ParamsType = { params: Partial<{ id: number }> }

export const playerLoader = async ({ params } : ParamsType) => {
  const promiseSong: Promise<SongType> =
    axios.get<SongType[], AxiosResponse<SongType>>(`/songs/${params.id}`)
      .then((response)=>response.data);
  return defer({ song: promiseSong })
}
```


5. loader

❖ react-router v7에서의 변경사항

- defer() 래퍼 함수를 제거하고 Promise를 직접 반환하는 방식으로 단순화

// react-router v6에서

```
export const membersLoader = async () => {  
  const promiseMembers: Promise<MemberType[]> =  
    axios.get<MemberType[], AxiosResponse<MemberType[]>>("/members_long")  
      .then((response)=>response.data);  
  return defer({ members: promiseMembers })  
}
```

// react-router v7에서

```
export const membersLoader = async () => {  
  const promiseMembers: Promise<MemberType[]> =  
    axios.get<MemberType[], AxiosResponse<MemberType[]>>("/members_long")  
      .then((response)=>response.data);  
  return { members: promiseMembers }  
}
```

5. loader

❖src/pages/Members.tsx 변경

- 기존 컴포넌트에서 useContext 혹은 Context Value를 획득하는 코드를 대신하여 useAsyncValue 혹은 이용해 비동기 처리된 값을 가져오도록 코드 변경
- MembersSuspense 컴포넌트 추가
 - <React.Suspense> 컴포넌트를 이용해 비동기 처리 진행 시간동안 fallback UI를 보여주도록 작성함
 - fallback UI로는 react-csspin 스피너 컴포넌트 사용
 - <https://github.com/stepanowon/react-csspin>
 - 비동기처리가 완료(resolve)되면 처리된 결과를 받을 수 있도록 <Await> 컴포넌트 이용

```
import React from "react";
import { MemberType } from "../loaders";
import { Await, useAsyncValue, useLoaderData } from "react-router-dom";
import { ReactCsspin } from "react-csspin";
import 'react-csspin/dist/style.css';
```

5. loader

❖src/pages/Members.tsx 변경 (이어서)

```
const Members = () => {  
  const members = useAsyncValue() as MemberType[];  
  
  const imgstyle = { width: 90, height: 80 };  
  const list = members.map((member) => {  
    return (  
      <div className="col-6 col-md-4 col-lg-3" key={member.name}>  
        <img src={member.photo} className="img-thumbnail" alt={member.name} style={imgstyle} />  
        <br />  
        <h6>{member.name}</h6>  
        <br /> <br />  
      </div>  
    );  
  });  
  
  return (  
    <div>  
      <h2 className="m-4">Members</h2>  
      <div className="container">  
        <div className="row">{list}</div>  
      </div>  
    </div>  
  );  
};
```

5. loader

❖src/pages/Members.tsx 변경 (이어서)

```
type DeferredMembersDataType = { members: Promise<MemberType[]> }

const MembersSuspense = () => {
  const data = useLoaderData() as DeferredMembersDataType;

  return (
    <React.Suspense fallback={<ReactCsspin />}>
      <Await resolve={data.members}>
        <Members />
      </Await>
    </React.Suspense>
  )
}

export { MembersSuspense };
export default Members;
```

5. loader

❖ src/pages/SongList.tsx, src/components/Player.tsx 변경

- src/pages/Members.tsx와 동일한 방법으로 변경함
- 완성된 예제 코드를 강사로부터 제공받아 검토하고 복사 후 붙여넣기

❖ src/router/index.tsx 변경

- ~Suspense 컴포넌트를 참조하여 각 라우트에서 렌더링
- ~Loader를 참조하여 각 라우트별 loader로 지정

```
import { createBrowserRouter } from "react-router-dom";  
import App from "../App";  
import Home from "../pages/Home";  
import About from "../pages/About";  
import { MembersSuspense } from "../pages/Members";  
import { SongListSuspense } from "../pages/SongList";  
import { PlayerSuspense } from "../components/Player";  
import { membersLoader, playerLoader, songListLoader } from "../loaders";
```

5. loader

❖src/router/index.tsx 변경 (이어서)

```
const router = createBrowserRouter([
  {
    path: "/",
    element: <App />,
    children: [
      { index: true, element: <Home /> },
      { path: "about", element: <About title={"여우와 늑다리들"} /> },
      { path: "members", element: <MembersSuspense />, loader: membersLoader },
      {
        path: "songs",
        element: <SongListSuspense />,
        loader: songListLoader,
        children: [
          { path: ":id", element: <PlayerSuspense />, loader: playerLoader }
        ],
      },
    ],
  },
]);

export default router;
```

5. loader

❖src/main.tsx 변경

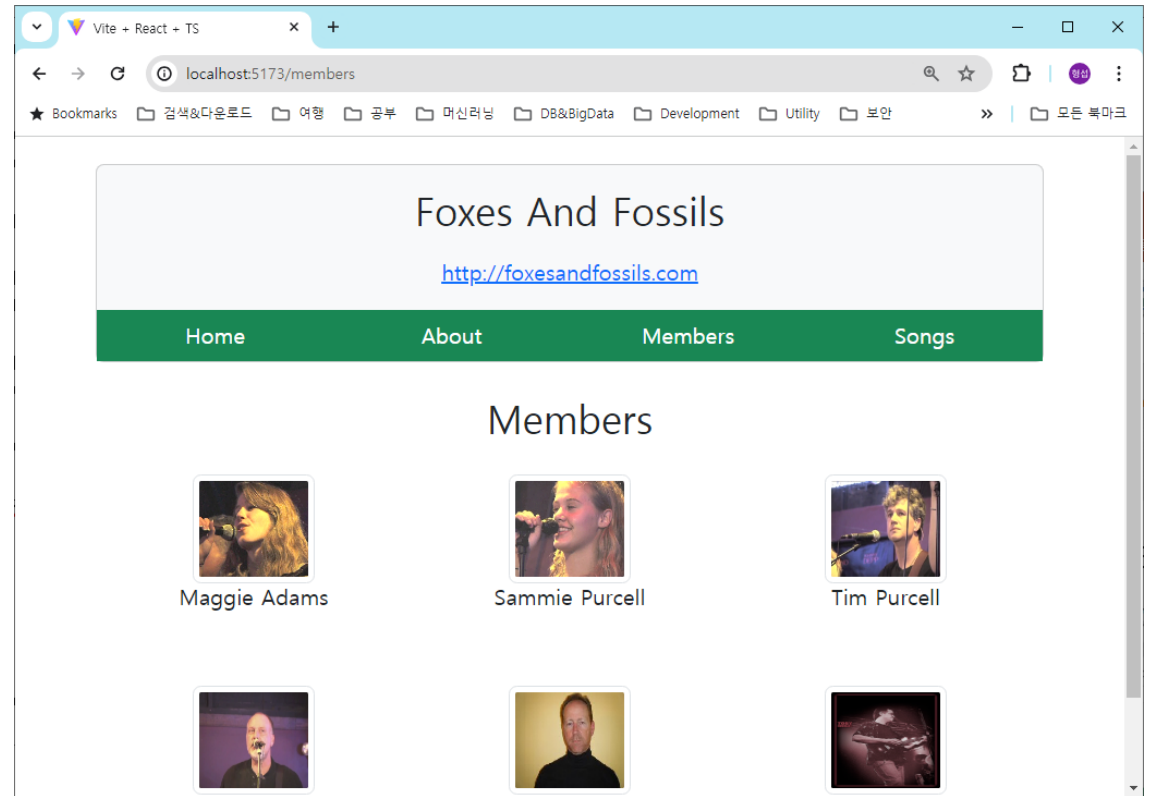
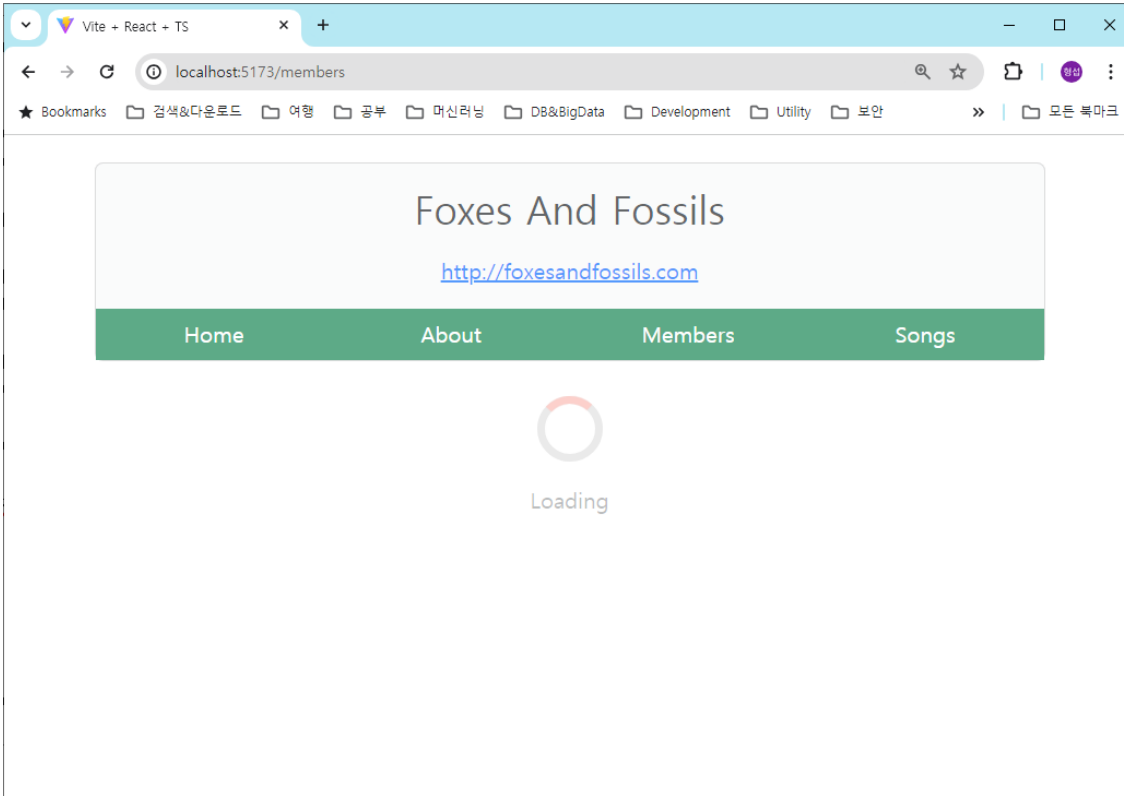
- Context API를 사용하는 BandProvider 컴포넌트 사용 코드 제거

```
import React from "react";
import ReactDOM from "react-dom/client";
import "bootstrap/dist/css/bootstrap.css";
import "./index.css";
import { RouterProvider } from "react-router-dom";
import router from "./router";

ReactDOM.createRoot(document.getElementById("root")!).render(
  <React.StrictMode>
    <RouterProvider router={router} />
  </React.StrictMode>
);
```

5. loader

❖loader 적용 결과

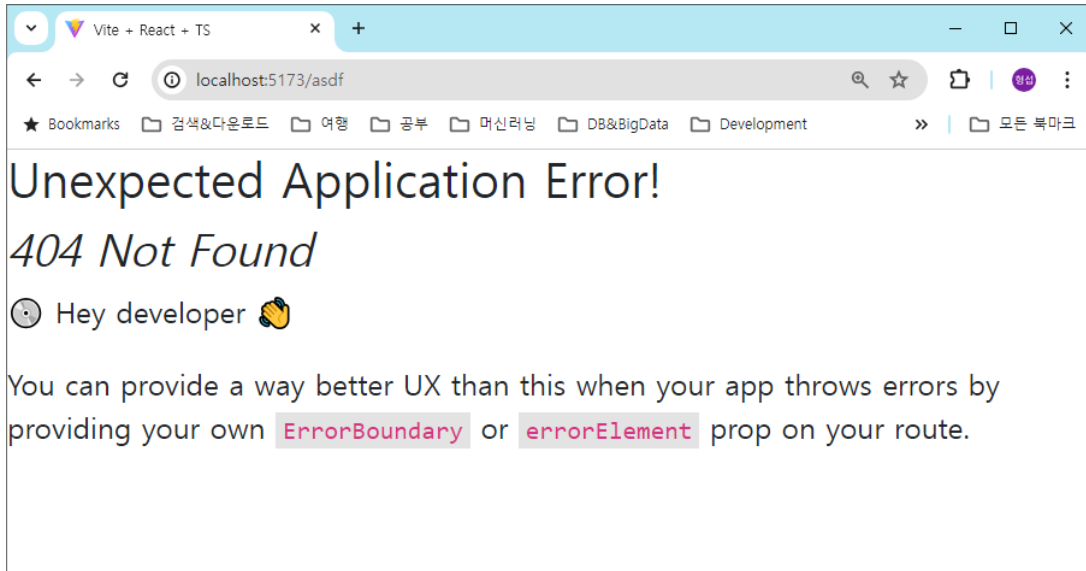


6. 404 Route와 ErrorBoundary

❖ 404 Route란?

- 존재하지 않는 경로로 요청했을 때 처리를 위한 라우트

❖ 존재하지 않는 경로에 대한 react-router 기본 에러 처리 페이지



❖ 개발자가 정의한 에러 페이지로 변경하려면?

- 404 Route
- ErrorBoundary

6. 404 Route와 ErrorBoundary

❖ 첫번째 방법 : 404 Route (Catch-All Route)

- 이전 react-router 버전에서 사용해오던 방법.
- 404 Route를 추가하는 방법

```
{ path: "*", element: <Error404 /> }
```

- src/components/Error404.tsx 추가

```
import { useLocation } from "react-router-dom";

const Error404 = () => {
  const location = useLocation();
  return (
    <div className="card card-body">
      <h2>존재하지 않는 경로입니다.</h2>
      요청 경로 : { location.pathname }
    </div>
  );
};

export default Error404;
```

6. 404 Route와 ErrorBoundary

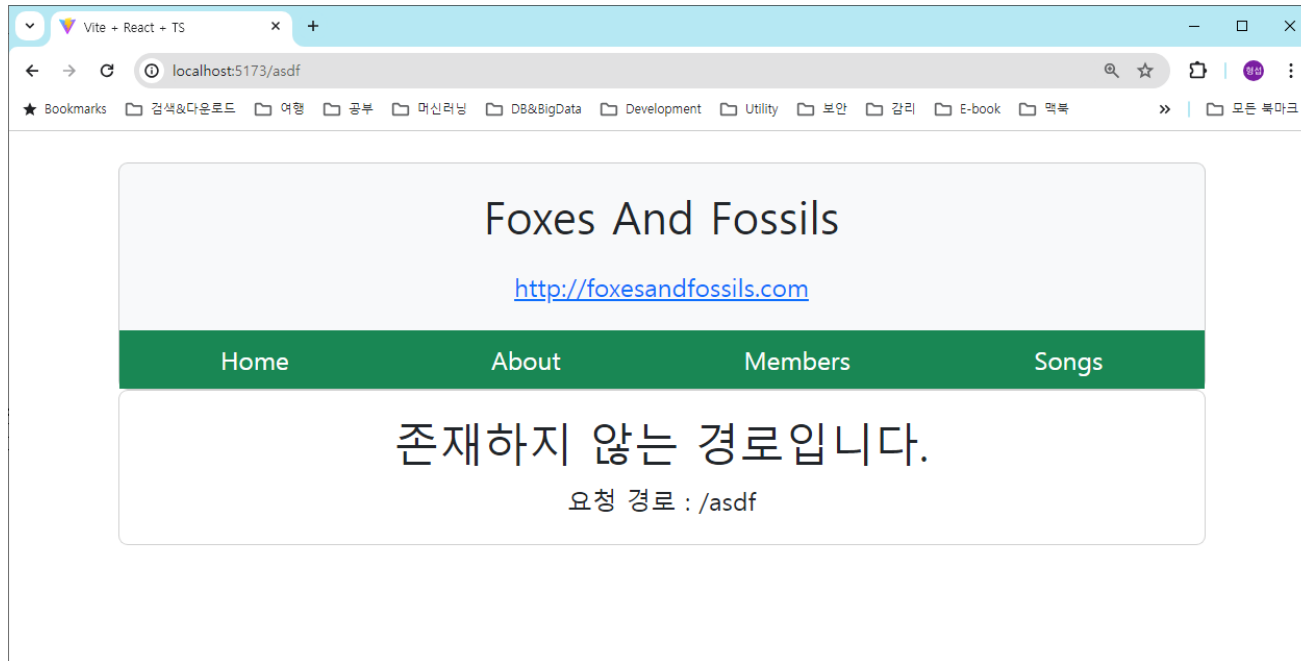
❖src/router/index.tsx 변경

```
.....(생략)
import Error404 from "../components/Error404";

const router = createBrowserRouter([
  {
    path: "/",
    element: <App />,
    children: [
      { index: true, element: <Home /> },
      { path: "about", element: <About title={"여우와 늑다리들"} /> },
      { path: "members", element: <MembersSuspense />, loader: membersLoader },
      {
        path: "songs",
        element: <SongListSuspense />,
        loader: songListLoader,
        children: [
          { path: ":id", element: <PlayerSuspense />, loader: playerLoader }
        ],
      },
    ],
  },
  { path: "*", element: <Error404 /> }
]);
.....(생략)
```

6. 404 Route와 ErrorBoundary

❖ 404 Route 실행 결과



❖ 404 Route의 문제점

- 중첩된 Route에 대해서도 모두 404 Route를 추가해야 함
- 그렇기 때문에 ErrorBoundary를 권장함.
 - 404뿐만 아니라 라우팅, 렌더링 과정에서 발생하는 다양한 오류를 처리함

6. 404 Route와 ErrorBoundary

❖ 두번째 방법 : ErrorBoundary

- 상위 라우트에서 에러 범위를 처리할 컴포넌트 사용
 - 상위 라우트의 중첩된 라우트에서는 이 컴포넌트를 사용해 에러 처리
- src/components/ErrorBoundary.tsx 추가

```
import { isRouteErrorResponse, useRouteError } from "react-router-dom";
import Header from "../Header";
import { AxiosError } from "axios";

const ErrorBoundary = () => {
  const error = useRouteError() as Error;
  let element: JSX.Element;
  if (isRouteErrorResponse(error)) {
    element = <p>{error.status} : {error.statusText}</p>;
  } else if (error instanceof AxiosError) {
    const axiosError = error as AxiosError;
    switch (axiosError.response && axiosError.response.status) {
      case 400:
        element = <p>백엔드 API : 잘못된 요청입니다.</p>;
        break;
      case 401:
        element = <p>백엔드 API : 접근이 불가능한 서비스입니다. 인가가 필요합니다.</p>;
        break;
    }
  }
  return <div>{element}</div>;
}
```

6. 404 Route와 ErrorBoundary

- src/components/ErrorBoundary.tsx 추가(이어서)

```
    case 404:
      element = <p>백엔드 API : 존재하지 않는 경로입니다.</p>;
      break;
    case 500:
      element = <p>백엔드 API : 백엔드 API 내부 오류입니다.</p>;
      break;
    default:
      element = <p>백엔드 API : 알 수 없는 오류가 발생했습니다.</p>;
  }
} else {
  element = <p>알 수 없는 오류가 발생했습니다.</p>;
}
return (
  <div className="container">
    <Header />
    <div className="card card-body">
      <h2>에러 발생</h2>
      {element}
    </div>
  </div>
);
};
export default ErrorBoundary;
```

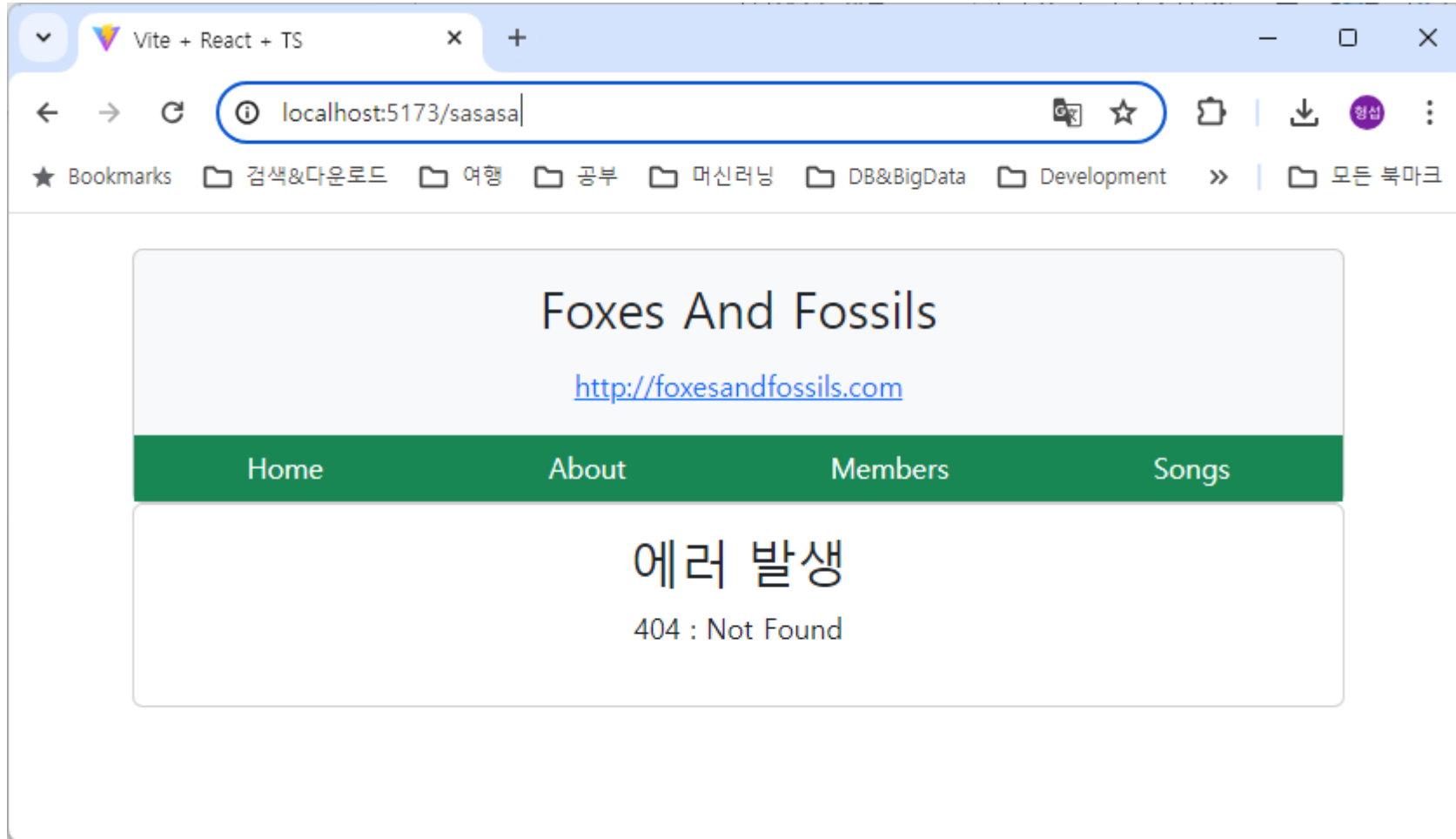
6. 404 Route와 ErrorBoundary

- src/route/index.tsx 변경 : 상위 라우트에 ErrorBoundary 적용 --> 중첩 라우트에도 적용됨

```
import ErrorBoundary from "../components/ErrorBoundary";
.....(생략)
const router = createBrowserRouter([
  {
    path: "/",
    element: <App />,
    errorElement: <ErrorBoundary />,
    children: [
      { index: true, element: <Home /> },
      { path: "about", element: <About title={"여우와 늑다리들"} /> },
      { path: "members", element: <MembersSuspense />, loader: fetchMembersLoader },
      {
        path: "songs",
        element: <SongListSuspense />,
        loader: fetchSongsLoader,
        children: [
          { path: ":id", element: <PlayerSuspense />, loader: fetchOneSongLoader }
        ],
      },
      //{ path: "*", element: <Error404 /> }
    ],
  },
]);
```

6. 404 Route와 ErrorBoundary

❖ ErrorBoundary 실행 결과



7. action

❖ react-router의 action이란?

- 라우트에서의 컴포넌트에서 Form 전송이 일어날 때 처리할 작업 기능을 제공
- 비동기 처리 + UI 제공 기능
- loader VS action
 - loader : 라우트에 필요한 데이터를 백엔드로부터 읽어오는 기능
 - action : 라우트에서 Submit이 일어날 때 백엔드로 전송하여 처리해주는 기능

❖ action 함수의 형식

```
const action: ActionFunction = async ({ request, params }: ActionFunctionArgs) => {  
  }  
}
```

- 인자 : ActionFunctionArgs
 - request : 요청 객체
 - formData : <Form />으로부터 전송된 입력값들.
 - 사용 예) request.formData.get("title") --> <input type="text" name="title" />
 - params : 라우트 경로 정보, /songs/:id -> params.id

7. action

❖ 기존 예제 action 적용 1

- 새로운 곡 추가기능, 편집 기능 추가

❖ 사전 작업

- src/loaders/index.ts의 axios baseURL 설정을 src/main.tsx 로 옮김
- src/main.tsx

```
import React from "react";
import ReactDOM from "react-dom/client";
import "bootstrap/dist/css/bootstrap.css";
import "./index.css";
import { RouterProvider } from "react-router-dom";
import router from "./router";
import axios from "axios";

axios.defaults.baseURL = "http://localhost:3000";

ReactDOM.createRoot(document.getElementById("root")!).render(
  <React.StrictMode>
    <RouterProvider router={router} />
  </React.StrictMode>
);
```

7. action

❖src/actions/index.ts 추가

```
import axios, { AxiosResponse } from "axios";
import { SongType } from "../loaders";
import { ActionFunction, ActionFunctionArgs, redirect } from "react-router-dom";

export type AddSongType = { title: string; musician: string; youtube_link: string };
export type AddSongReturnType = { status: string; message: string; item: SongType };
export type ActionError = { message: string };

export const addSongAction: ActionFunction = async ({ request }: ActionFunctionArgs) => {
  const formData = await request.formData();
  const requestData = {
    title: formData.get("title"),
    musician: formData.get("musician"),
    youtube_link: formData.get("youtube_link"),
  };

  const response = await axios.post<AddSongReturnType, AxiosResponse<AddSongReturnType>>("/songs_long", requestData);
  if (response.data.status === "success") {
    return redirect("/songs");
  } else {
    const error: ActionError = { message: response.data.message };
    return error;
  }
};
```


7. action

❖src/actions/index.ts 추가 (이어서)

```
type UpdateSongParam = { id: number };
export const updateSongAction: ActionFunction = async ({ request, params }: ActionFunctionArgs) => {
  const formData = await request.formData();
  const updateParam = params as unknown as UpdateSongParam;

  const requestData = {
    title: formData.get("title"),
    musician: formData.get("musician"),
    youtube_link: formData.get("youtube_link"),
  };

  const response = await axios.put<AddSongReturnType, AxiosResponse<AddSongReturnType>>(`/songs/${updateParam.id}`,
requestData);
  if (response.data.status === "success") {
    return redirect("/songs");
  } else {
    const error: ActionError = { message: response.data.message };
    return error;
  }
};
```



7. action

❖src/pages/AddSong.tsx 추가

- useActionData 훅
 - action 함수에서의 action data를 받아 옴
 - 앞서 작성했던 action에서는 error 객체
- useNavigation 훅
 - navigation 객체 리턴
 - navigation 객체의 속성
 - state : "idle", "submitting", "loading" --> navigation 진행 상태를 확인
 - location
 - formData
 - formAction
 - formMethod
 - state를 이용해서 백엔드 API로 POST 진행 중일 때 버튼들을 비활성화함
- useNavigate 훅
 - 직접 경로를 지정해 이동할 수 있는 navigate() 함수를 리턴
 - 사용 예) navigate("/songs")

7. action

❖src/pages/AddSong.tsx 추가 (이어서)

```
import { Form, useActionData, useNavigate, useNavigation } from "react-router-dom";
import { ActionError } from "../actions";

const AddSong = () => {
  const navigate = useNavigate();
  const navigation = useNavigation();
  const isSubmitting = navigation.state === "submitting";

  const error = useActionData() as ActionError;

  return (
    <div>
      <h2 className="m-5">새로운 곡 추가</h2>
      <Form method="post">
        <div className="form-floating mb-2">
          <input type="text" className="form-control" id="title" name="title" />
          <label htmlFor="title">곡 제목</label>
        </div>
        <div className="form-floating mb-2">
          <input type="text" className="form-control" id="musician" name="musician" />
          <label htmlFor="musician">원곡 가수</label>
        </div>
      </Form>
    </div>
  );
};
```

7. action

❖src/pages/AddSong.tsx 추가 (이어서)

```
<div className="form-floating mb-2">
  <input type="text" className="form-control" id="youtube_link" name="youtube_link" defaultValue={"PABUI_EX_hw"} />
  <label htmlFor="youtube_link">유튜브 링크</label>
</div>
<br />
<button type="submit" className="btn btn-primary m-1" disabled={isSubmitting}>
  {isSubmitting ? "저장 처리 중" : "추가"}
</button>
<button className="btn btn-primary" onClick={() => navigate("/songs")} disabled={isSubmitting}>
  취소
</button>
{error ? (
  <div className="card mt-5">
    <div className="card-body">{error.message}</div>
  </div>
) : ("")}
</Form>
</div>
);
};

export default AddSong;
```

7. action

❖src/pages/UpdateSong.tsx

- AddSong 컴포넌트와 유사하지만 loader를 추가적으로 사용함
 - loader를 이용해 일단 곡 한 건 정보를 읽어와서 화면으로 뿌려준 후 편집해야 함

```
import { Await, Form, useActionData, useAsyncValue, useLoaderData, useNavigate, useNavigation } from "react-router-dom";
import { ActionError } from "../actions";
import { SongType } from "../loaders";
import React from "react";
import { ReactCsspin } from "react-csspin";
import 'react-csspin/dist/style.css';

const UpdateSong = () => {
  const navigate = useNavigate();

  const error = useActionData() as ActionError;
  const song = useAsyncValue() as SongType;
  const navigation = useNavigation();
  const isSubmitting = navigation.state === "submitting";

  return (
    <div>
      <h2 className="m-5">곡 정보 변경</h2>
```


7. action

❖src/pages/UpdateSong.tsx (이어서)

```
<Form method="post">
  <input type="hidden" id="id" name="id" defaultValue={song.id} />
  <div className="form-floating mb-2">
    <input type="text" className="form-control" id="title" name="title" defaultValue={song.title} />
    <label htmlFor="title">곡 제목</label>
  </div>
  <div className="form-floating mb-2">
    <input type="text" className="form-control" id="musician" name="musician" defaultValue={song.musician} />
    <label htmlFor="musician">원곡 가수</label>
  </div>
  <div className="form-floating mb-2">
    <input type="text" className="form-control" id="youtube_link" name="youtube_link" defaultValue={song.youtube_link} />
    <label htmlFor="youtube_link">유튜브 링크</label>
  </div>
  <br />
  <button type="submit" className="btn btn-primary m-1" disabled={isSubmitting}>
    {isSubmitting ? "업데이트 처리중" : "업데이트"}
  </button>
  <button className="btn btn-primary" onClick={() => navigate("/songs")} disabled={isSubmitting}>
    취소
  </button>
```

7. action

❖src/pages/UpdateSong.tsx (이어서)

```
{error ? (
  <div className="card mt-5">
    <div className="card-body">{error.message}</div>
  </div>
) : ("")}
</Form>
</div>
);
};

type DeferredOneSongDataType = { song: Promise<SongType> };
const UpdateSongSuspense = () => {
  const data = useLoaderData() as DeferredOneSongDataType;
  return (
    <React.Suspense fallback={<ReactCsspin />}>
      <Await resolve={data.song}>
        <UpdateSong />
      </Await>
    </React.Suspense>
  );
};
export { UpdateSongSuspense };
export default UpdateSong;
```

7. action

❖src/pages/SongList.tsx 변경

- '새로운 곡 추가' 버튼 추가 -> /songs/new 로 네비게이션
- 기존 곡 '수정' 버튼 추가 -> /songs/update/:id 로 네비게이션

.....(생략)

```
const SongList = () => {
```

```
  const songs = useAsyncValue() as SongType[];
```

```
  const list = songs.map((song) => {
```

```
    return (
```

```
      <li className="list-group-item" key={song.id}>
```

```
        {song.title} ( {song.musician} )
```

```
        <Link to={`/songs/update/${song.id}`} style={{ textDecoration: "none" }}>
```

```
          <span className="float-end badge bg-secondary ms-2">수정</span>
```

```
        </Link>
```

```
        <Link to={`/songs/${song.id}`} style={{ textDecoration: "none" }}>
```

```
          <span className="float-end badge bg-secondary ms-2 me-2">
```

```
            <i className="fa fa-play"></i>
```

```
          </span>
```

```
        </Link>
```

```
      </li>
```

```
    );
```

```
  });
```

수정 버튼

플레이 버튼

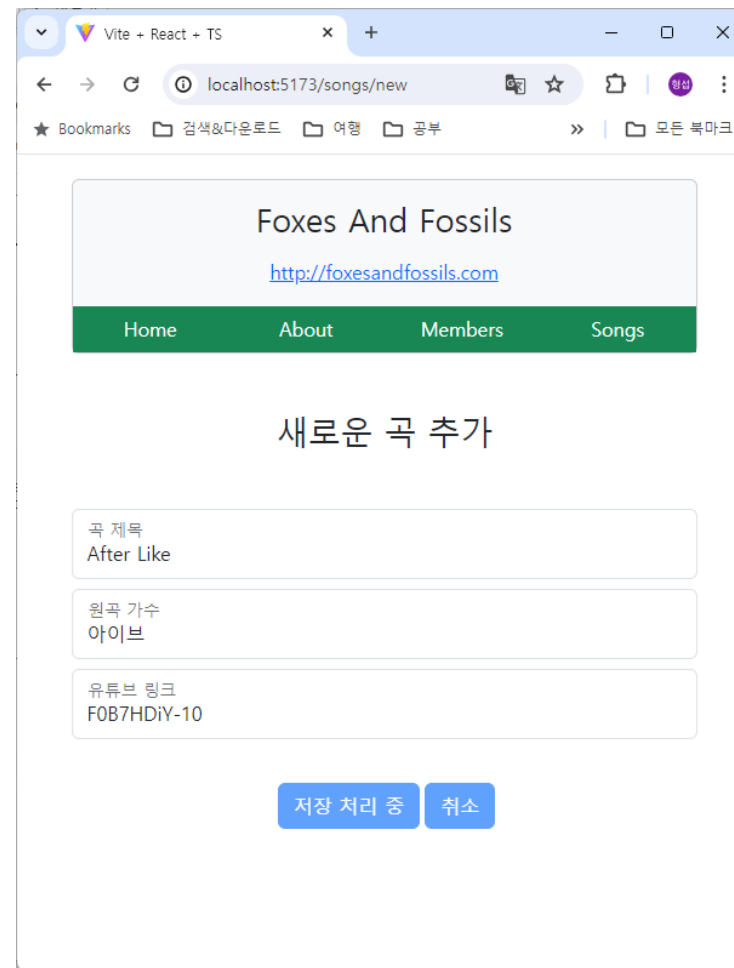
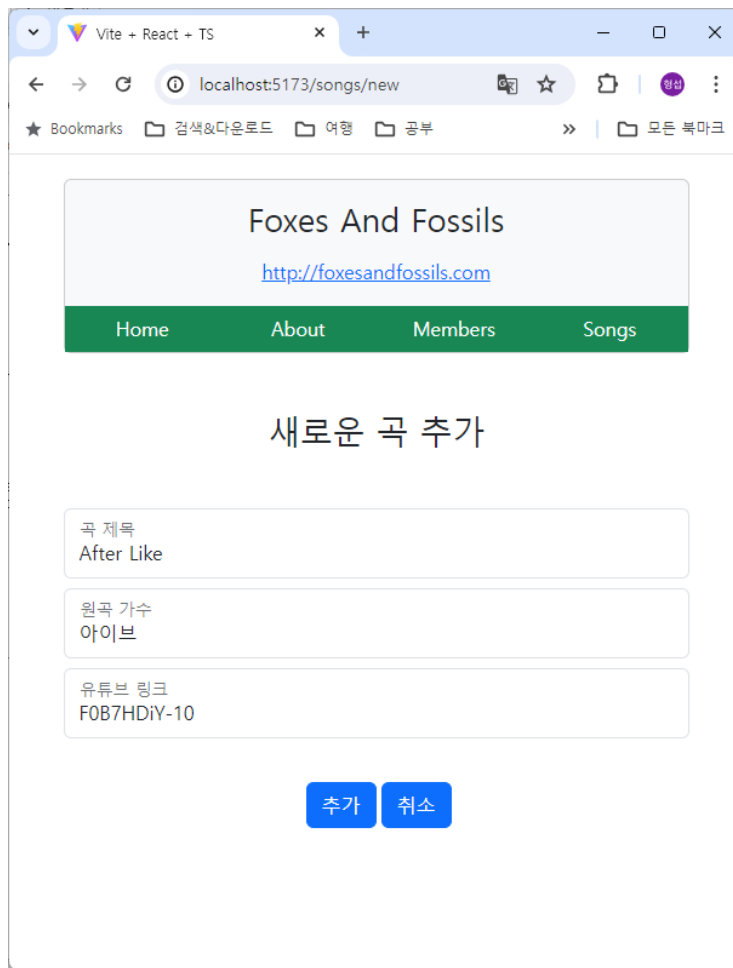
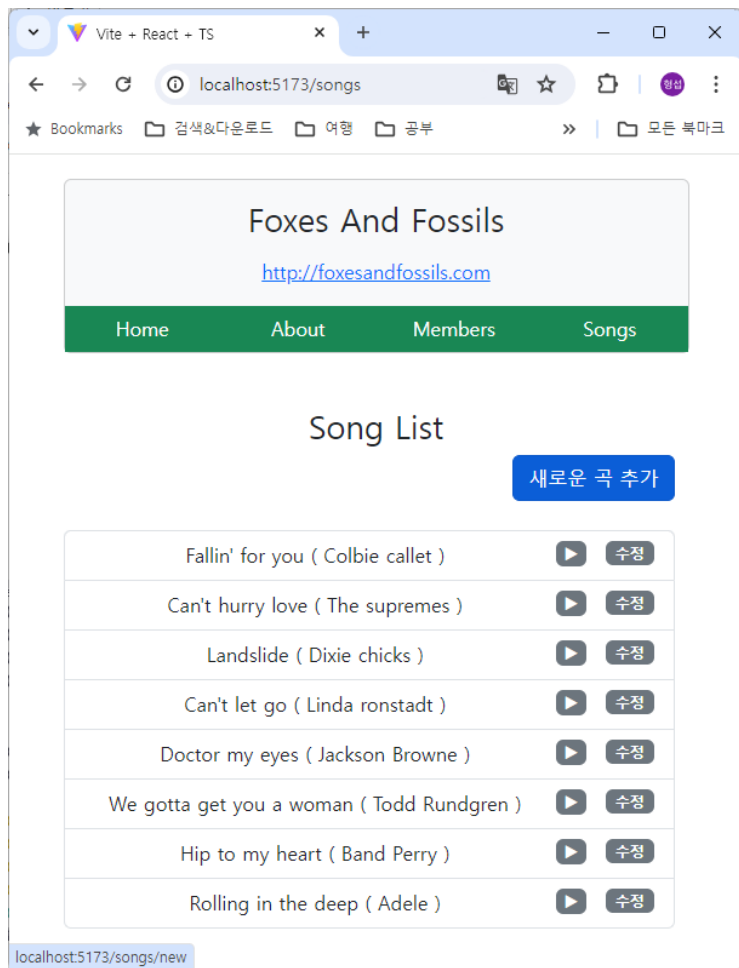
7. action

❖src/pages/SongList.tsx 변경

```
return (  
  <div className="container">  
    <div className="row">  
      <h2 className="mt-4 mb-2">Song List</h2>  
    </div>  
    <div className="row justify-content-end">  
      <div className="col-12">  
        <Link className="btn btn-primary float-end" to={"/songs/new"}>  
          새로운 곡 추가  
        </Link>  
      </div>  
    </div>  
    <br />  
    <div className="row">  
      <ul className="list-group">{list}</ul>  
    </div>  
    <Outlet />  
  </div>  
);  
};  
.....(생략)
```

7. action

❖ 실행 결과 1



7. action

❖ 실행 결과 2

Vite + React + TS

localhost:5173/songs/new

Bookmarks 검색&다운로드 여행 공부 >> 모든 북마크

Foxes And Fossils

<http://foxesandfossils.com>

Home About Members Songs

새로운 곡 추가

곡 제목

원곡 가수

유튜브 링크
PABUI_EX_hw

추가

취소

곡 추가 실패 : Error: 곡 타이틀과 유튜브링크를 입력하셔야 합니다.

Vite + React + TS

localhost:5173/songs/update/17155...

Bookmarks 검색&다운로드 여행 공부 >> 모든 북마크

Foxes And Fossils

<http://foxesandfossils.com>

Home About Members Songs

곡 정보 변경

곡 제목
After Like

원곡 가수
Ive

유튜브 링크
F0B7HDiY-10

업데이트

취소

Vite + React + TS

localhost:5173/songs/update/17155...

Bookmarks 검색&다운로드 여행 공부 >> 모든 북마크

Foxes And Fossils

<http://foxesandfossils.com>

Home About Members Songs

곡 정보 변경

곡 제목

원곡 가수
Ive

유튜브 링크
F0B7HDiY-10

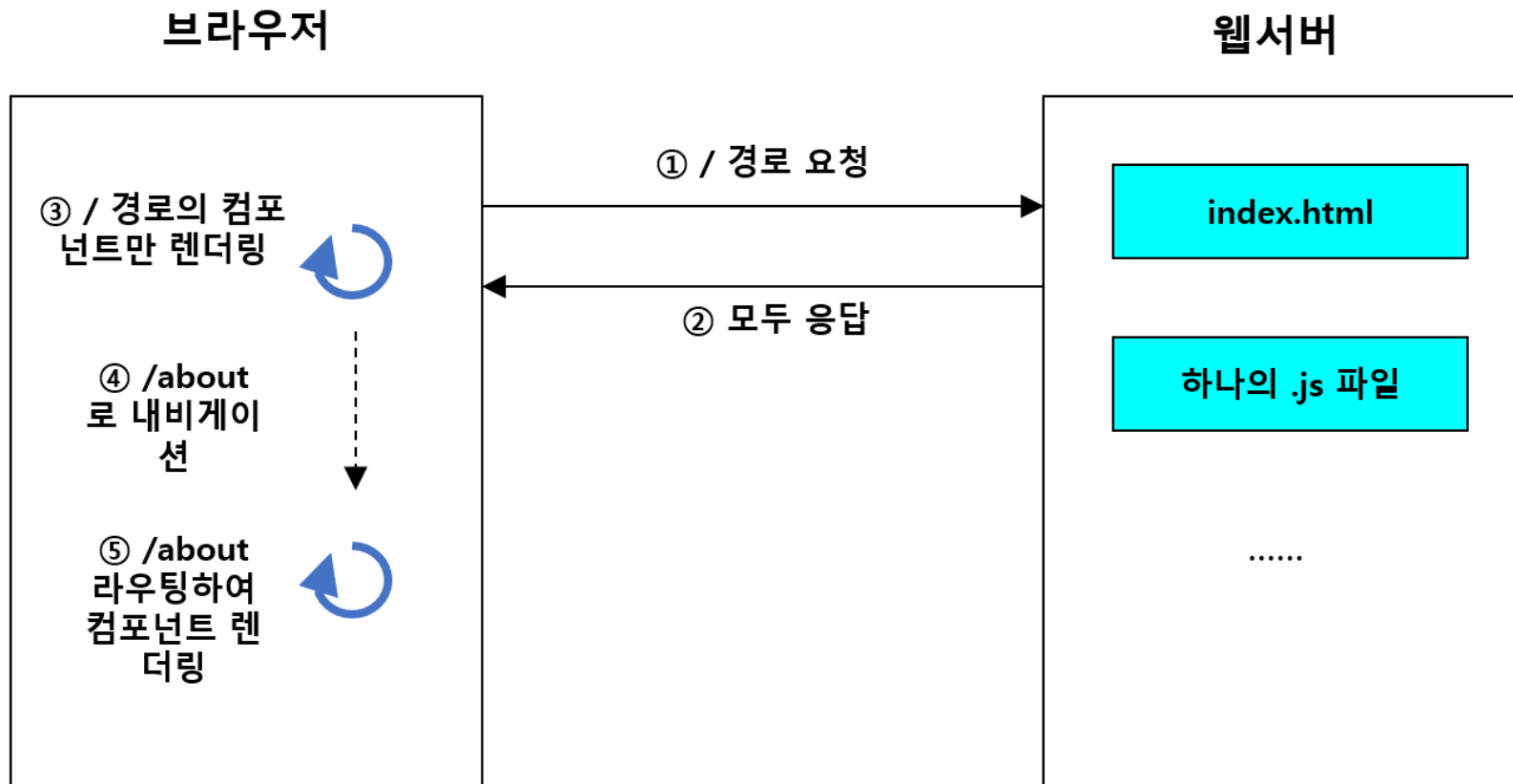
업데이트

취소

곡 변경 실패 : Error: 곡 타이틀과 유튜브링크를 입력하셔야 합니다.

8. Lazy Loading

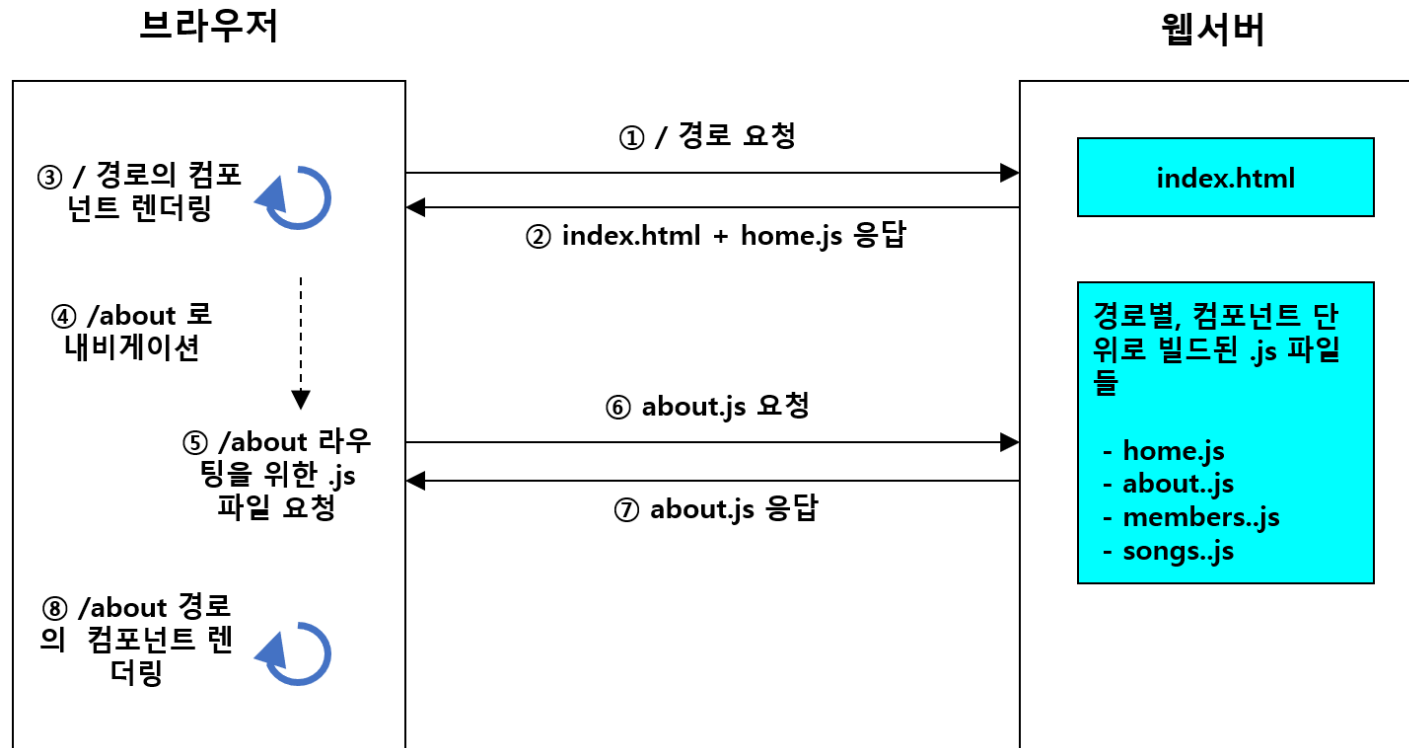
- ❖ 대규모 SPA 앱에서 첫 화면의 로딩 속도가 느린 이유
 - 요청, 응답 방식



8. Lazy Loading

❖ Lazy Loading의 의미

- 리액트 애플리케이션의 수많은 화면과 컴포넌트 코드를 적절히 구분하여 화면, 그룹 단위로 여러개의 파일로 빌드함
 - 이 여러개의 파일을 청크(Chunk)라고 부름
- 브라우저에서 컴포넌트가 필요한 시점에 서버에 요청해 청크를 받아온 후 렌더링하는 방법



8. Lazy Loading

❖ 적용 방법

```
//기존의 컴포넌트 import 방법
import Home from "../pages/Home"

//React.lazy()와 import 함수 사용
const Home = React.lazy(() => import("../pages/Home"));

//export default 되지 않은 컴포넌트를 lazy loading
//default 모듈로 로딩하도록 Promise 이용
const MembersSuspense =
  React.lazy(() => import("../pages/Members").then((module) => ({ default: module.MembersSuspense })));

//lazy loading된 컴포넌트는 반드시 <Suspense /> 내부에 배치해야 됨
// ** src/main.tsx에서 다음과 같이...
ReactDOM.createRoot(document.getElementById("root")!).render(
  <React.StrictMode>
    <Suspense fallback={<ReactCsspin />}>
      <RouterProvider router={router} />
    </Suspense>
  </React.StrictMode>
);
```

8. Lazy Loading

❖ 기존 예제에 lazy loading 적용 :

- npm install p-min-delay : 의도적 지연 시간 발생. 실무에서는 사용하지 말 것
- src/router/index.tsx 변경

```
/**아래 주석에 해당하는 부분을 찾아서 주석 처리한 후 주석 처리되지 않은 부분을 추가
//import Home from "../pages/Home";
//import About from "../pages/About";
//import AddSong from "../pages/AddSong";
//import { MembersSuspense } from "../pages/Members";
//import { SongListSuspense } from "../pages/SongList";
//import { PlayerSuspense } from "../components/Player";
//import { UpdateSongSuspense } from "../pages/UpdateSong";
const Home = React.lazy(() => import("../pages/Home"));
const About = React.lazy(() => pMinDelay(import("../pages/About"), 2000));    //import pMinDelay from 'p-min-delay';
const AddSong = React.lazy(() => import("../pages/AddSong"));
const MembersSuspense =
  React.lazy(() => import("../pages/Members").then((module) => ({ default: module.MembersSuspense })));
const SongListSuspense =
  React.lazy(() => import("../pages/SongList").then((module) => ({ default: module.SongListSuspense })));
const PlayerSuspense =
  React.lazy(() => import("../components/Player").then((module) => ({ default: module.PlayerSuspense })));
const UpdateSongSuspense =
  React.lazy(() => import("../pages/UpdateSong").then((module) => ({ default: module.UpdateSongSuspense })));
```

8. Lazy Loading

- src/router/index.tsx 변경 (이어서)

- 예제 작성 도중 다음의 경고 메시지가 나타나는 것을 볼 수 있음

Fast refresh only works when a file only exports components. Move your component(s) to a separate file.

- 경고 내용

- "한 모듈에서 컴포넌트와 함수를 동시에 작성해두면 Fast Refresh 기능을 사용할 없으니, 다른 분리된 파일로 코드를 옮겨라"
- ESLint가 경고해주는 것. 실행에는 영향이 없으나 경고가 나타난 모듈의 코드를 수정하고 저장하면 곧바로 브라우저에 반영되지 못하기 때문에 반드시 새로고침해주어야 함

- 경고가 보기 싫다면 .eslintrc.cjs 파일에서 다음을 주석 처리함

```
module.exports = {
  root: true,
  env: { browser: true, es2020: true },
  extends: ["eslint:recommended", "plugin:@typescript-eslint/recommended", "plugin:react-hooks/recommended"],
  ignorePatterns: ["dist", ".eslintrc.cjs"],
  parser: "@typescript-eslint/parser",
  plugins: ["react-refresh"],
  rules: {
    //"react-refresh/only-export-components": ["warn", { allowConstantExport: true }],
  },
};
```

8. Lazy Loading

❖src/main.tsx 변경

▪ <Suspense /> 추가

```
import React, { Suspense } from "react";
import ReactDOM from "react-dom/client";
import "bootstrap/dist/css/bootstrap.css";
import "./index.css";
import { RouterProvider } from "react-router-dom";
import router from "./router";
import axios from "axios";
import { ReactCsspin } from "react-csspin";
import "react-csspin/dist/style.css";

axios.defaults.baseURL = "http://localhost:3000";

ReactDOM.createRoot(document.getElementById("root")!).render(
  <React.StrictMode>
    <Suspense fallback={<ReactCsspin />}>
      <RouterProvider router={router} />
    </Suspense>
  </React.StrictMode>
);
```

8. Lazy Loading

❖ 빌드 결과

- npm run build 명령 실행
- 청크가 분할된 것을 확인할 수 있음

```
• PS C:\dev\workspace_react\react-deep\ch04\foxes-band-app> npm run build
```

```
> foxes-band-app@0.0.0 build
```

```
> tsc && vite build
```

```
vite v5.2.11 building for production...
```

```
✓ 134 modules transformed.
```

dist/index.html	0.46 kB	gzip: 0.30 kB
dist/assets/index-B6vsFp2-.css	243.55 kB	gzip: 32.67 kB
dist/assets/Home-CsRsU1tn.js	0.16 kB	gzip: 0.15 kB
dist/assets/About-CkrtZp6f.js	0.17 kB	gzip: 0.16 kB
dist/assets/Members-h4N_BxEI.js	0.69 kB	gzip: 0.37 kB
dist/assets/SongList-B_X3i9kF.js	1.17 kB	gzip: 0.55 kB
dist/assets/AddSong-DRenOgp0.js	1.20 kB	gzip: 0.57 kB
dist/assets/UpdateSong-CyQsMA20.js	1.50 kB	gzip: 0.66 kB
dist/assets/Player-DTVaKF3y.js	18.88 kB	gzip: 6.76 kB
dist/assets/index-Jt82uhux.js	245.97 kB	gzip: 82.30 kB

```
✓ built in 1.76s
```

8. Lazy Loading

❖ 실행 결과

- 처음 /about 으로 이동할 때 fallback UI가 나타남
- lazy loading이 일어나고 난 후에는 fallback UI가 나타나지 않음

