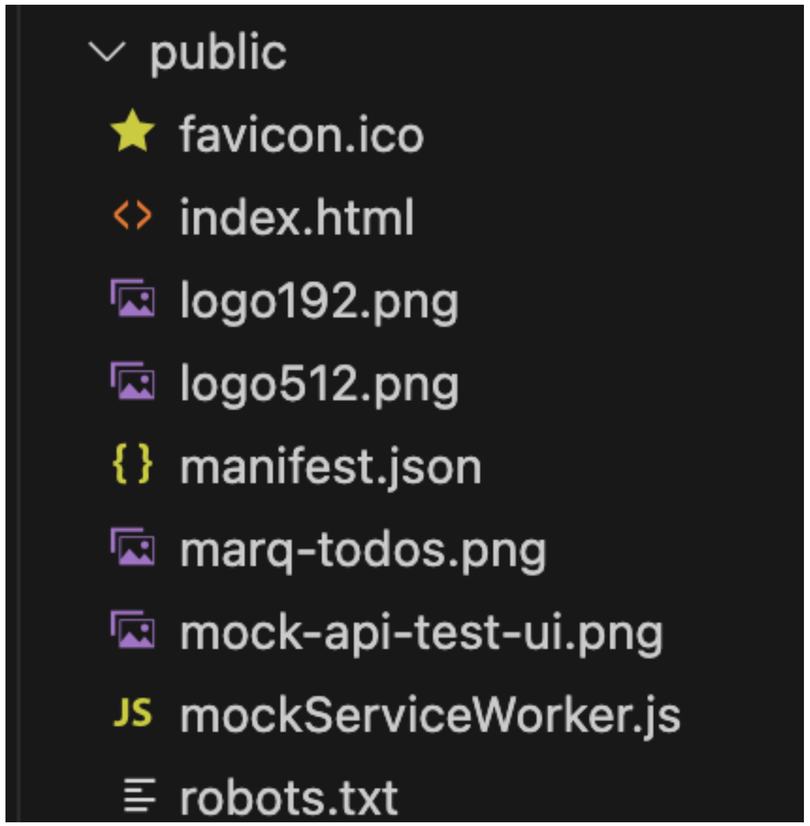
TODO

멋사5기
강남스터디 강은초

```
src
 ├ apis
 │ ├ base
 │ │ ├ addTodos.ts
 │ │ ├ deleteTodos.ts
 │ │ ├ editTodos.ts
 │ │ ├ getTodos.ts
 │ │ └ url.ts
 │ └ instance
 │ │ └ index.ts
 ├ container
 │ └ todolist
 │ │ ├ components
 │ │ │ ├ common
 │ │ │ │ └ statusBar
 │ │ │ │ │ ├ components
 │ │ │ │ │ │ └ Chip.tsx
 │ │ │ │ │ ├ StatusBar.tsx
 │ │ │ │ │ └ index.ts
 │ │ │ ├ utils
 │ │ │ │ └ formatDate.ts
 │ │ │ ├ Todo.tsx
 │ │ │ └ TodoForm.tsx
 │ │ ├ contexts
 │ │ │ └ StatusContext.tsx
 │ │ ├ TodoList.tsx
 │ │ └ index.ts
 ├ mocks
 │ ├ browser.ts
 │ └ handlers.ts
 ├ models
 │ └ todo.ts
 ├ queries
 │ ├ useAddTodos.ts
 │ ├ useDeleteTodos.ts
 │ ├ useEditTodos.ts
 │ └ useGetTodos.ts
 ├ App.tsx
 ├ index.css
 ├ index.tsx
 ├ react-app-env.d.ts
 ├ reportWebVitals.ts
 └ setupTests.ts
```

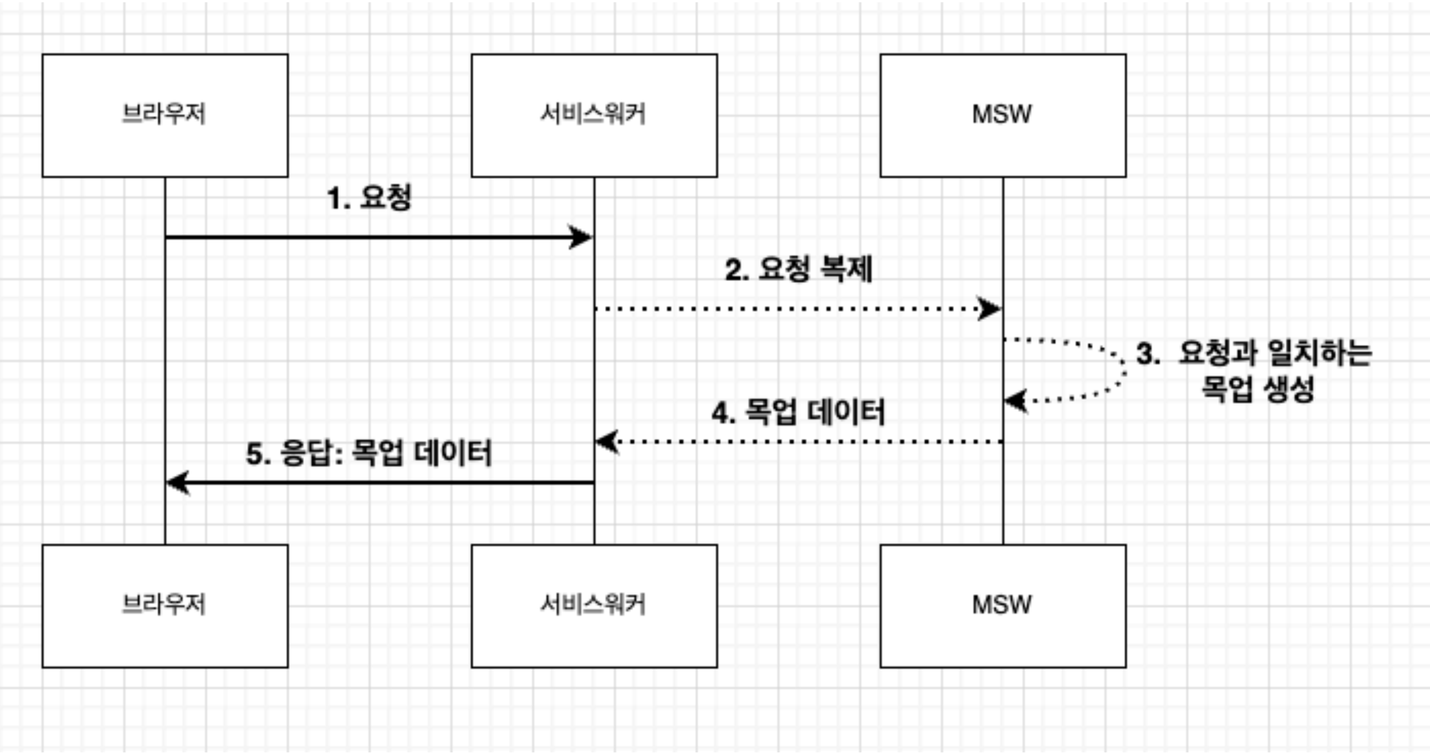
폴더구조



public

- mockServiceWorker.js : msw 설치후 자동 생성
- robots.txt

MSW란?



- Mock Service Worker: https://mswjs.io/
- 서비스워커 함수를 이용하여 네트워크 레벨에서 서버 요청(request)을 가로채 mock 데이터를 생성
- 타입스크립트 기반 프로젝트
- Front-end 개발에서 필요
  - api의 개발이 늦어지는 경우 mockup 데이터를 이용해서 테스트 가능 -> 개발 일정 단축
  - 실제 서버를 구축하지 않아도 네트워크 서비스 이용 가능
  - 잘못된 api를 호출하거나 에러처리를 잘못했을 때 이슈가 발생하는데, mock api로 디버깅을 해보면 도움이 됨

robot.txt

- 크롤링에 대한 지침
  - 어떤 페이지에 접근하거나 정보수집을 해도 되는지
  - 크롤러가 웹사이트에 접속했을 경우 정보수집요청으로 인해 사이트가 과부하되는 것을 방지
  - 크롤러로 인해 일어나는 Crawl-buget(크롤링 예산) 방지
  - 사이트맵의 위치를 제공하여 웹사이트의 콘텐츠가 검색엔진에 더 잘 발견될 수 있도록 함

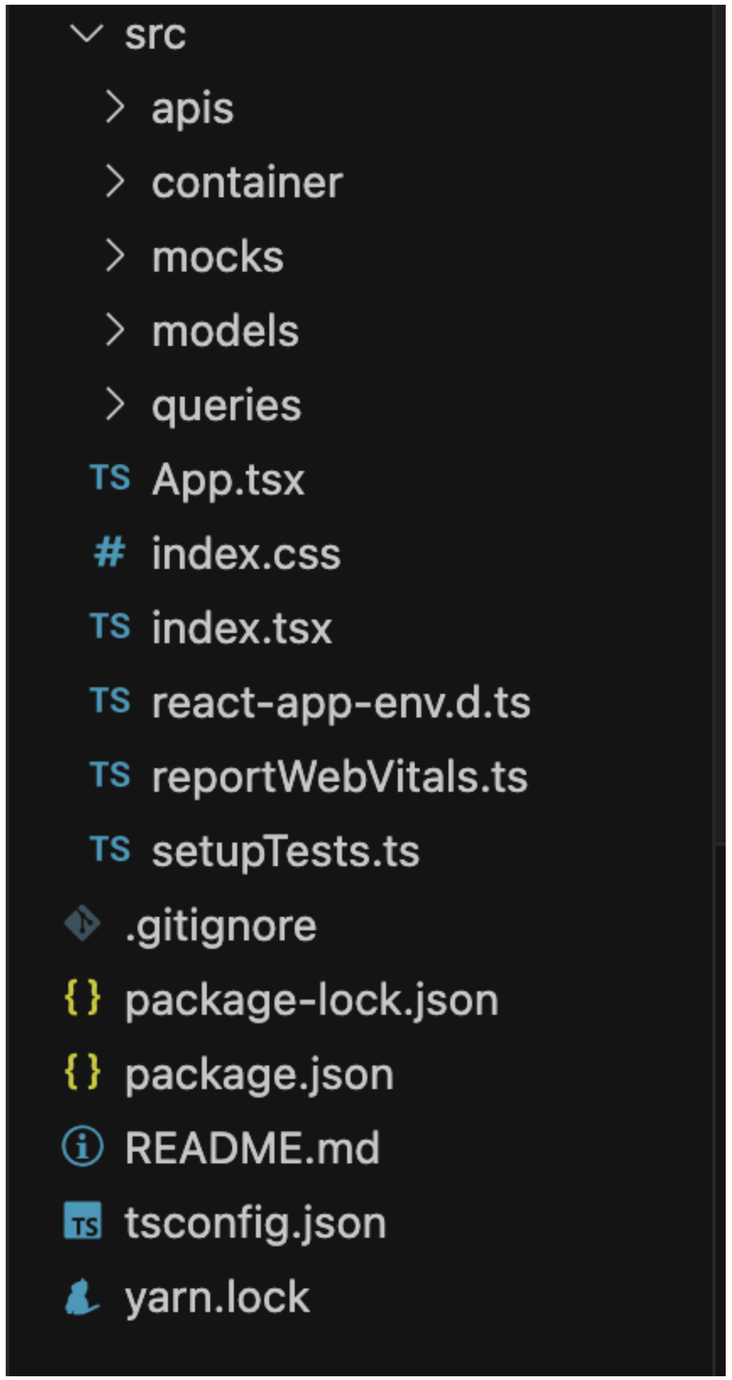
robot.txt

```
#
#    We thought you'd never make it!
#    We hope you feel right at home in this file...unless you're a disallowed subfolder.
#    And since you're here, read up on our culture and team: https://www.airbnb.com/careers/departments/engineering
#    There's even a bring your robot to work day.
```

위: 에어비앤비 **robots.txt** 파일 / 아래: 나이키 **robots.txt** 파일

두 브랜드 모두 주석 처리된 문자열을 이용하여 브랜드 창의적으로 브랜드 로고를 노출하고 있으며,

에어비앤비의 경우 위트 있는 웹 엔지니어 구인 문구를 포함하고 있습니다.



src

- apis: api 통신을 위한 파일들
- container: 컴포넌트, 자주쓰이는 양식(날짜), context api
- mocks: msw 함수, msw 통신
- models: api 모델에 필요한 요소 타입 모음 (타입스크립트)
- queries: react-query(useQuery, useMutation을 사용하여 서버로부터 데이터 조회/변경)

React-Query

- TanStack Query: https://tanstack.com/query/v4
- React에서 비동기 로직을 리액트스럽게 다룰 수 있게 해주는 라이브러리
- 기존의 isLoading, isError, refetch, 데이터캐싱 등 개발자가 구현하기 귀찮거나 까다로운 기능을 자체적으로 제공
- 기존 상태관리 라이브러리인 redux, mobX가 클라이언트 상태에는 적합하지만 비동기 또는 서버 상태 작업에서는 그다지 좋지않음 -> react-query는 클라이언트 상태와 서버 상태를 명확히 구분하기위해 만들어짐

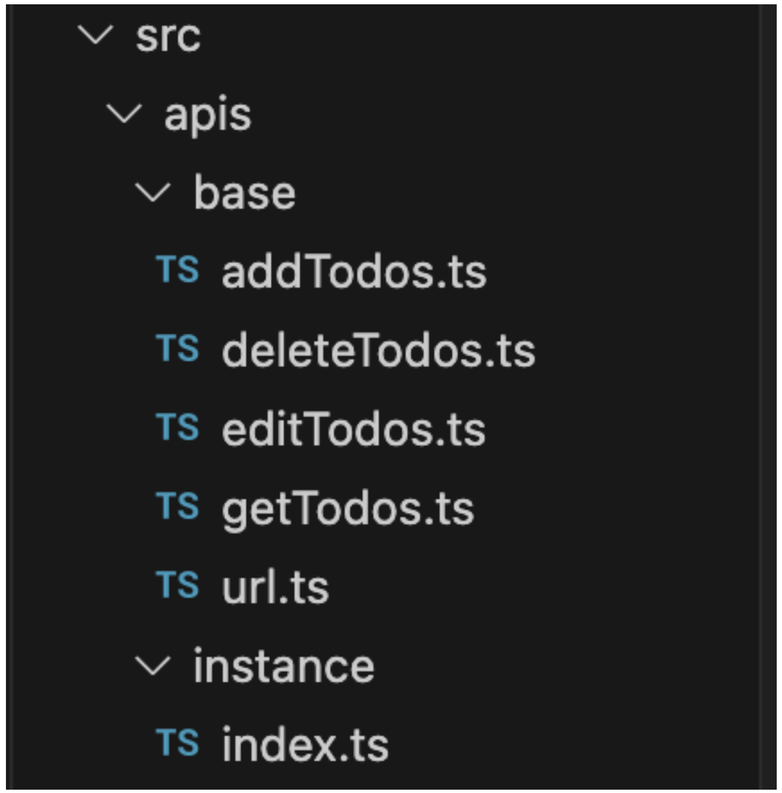
*클라이언트 상태(Client State)는 각각의 input 값으로 예를 들 수 있고, 서버 상태(Server State)는 데이터베이스에 저장되어 있는 데이터로 예를 들 수 있다.

React-Query

src > TS index.tsx > ...

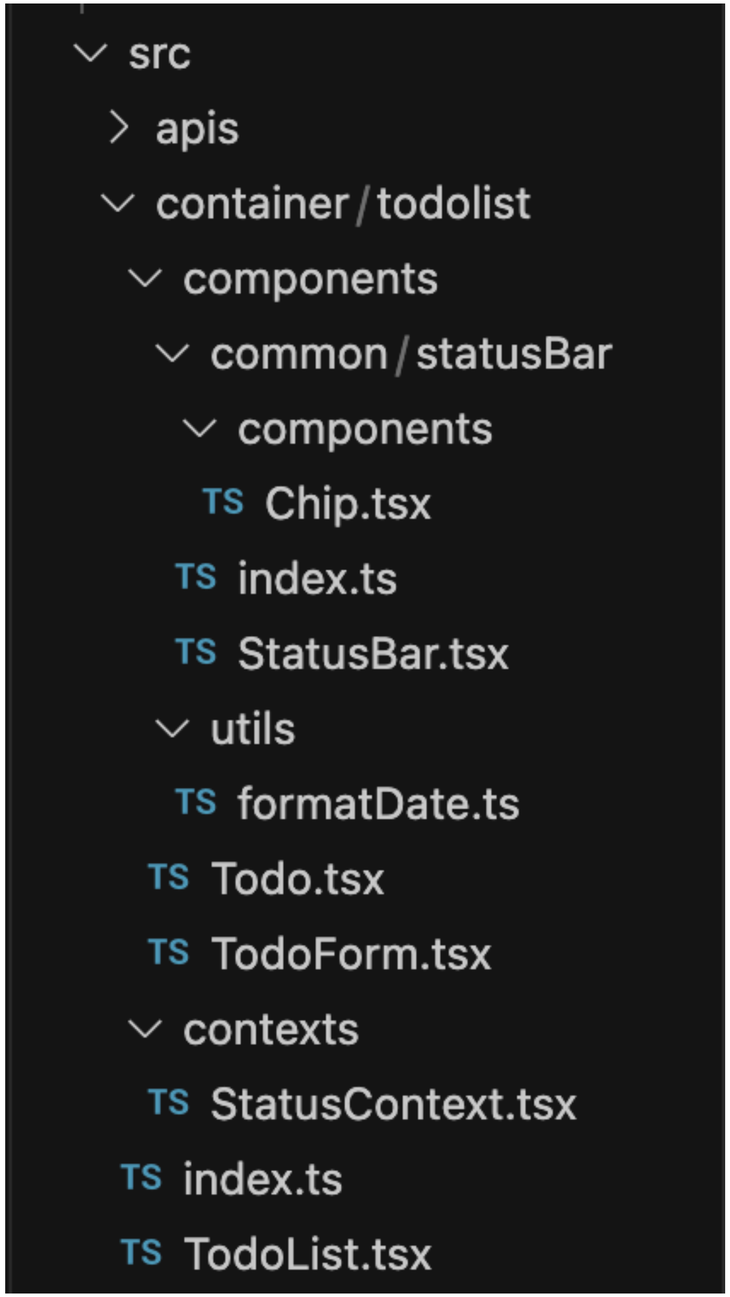
```tsx
import React from "react";
import ReactDOM from "react-dom/client";
import "./index.css";
import App from "./App";
import reportWebVitals from "./reportWebVitals";
import { QueryClientProvider } from "react-query";
import { queryClient } from "./apis/instance";

if (process.env.NODE_ENV === "development") {
  const { worker } = require("./mocks/browser");
  worker.start();
}

const root = ReactDOM.createRoot(
  document.getElementById("root") as HTMLElement
);
root.render(
  <React.StrictMode>
    <QueryClientProvider client={queryClient}>
      <App />
    </QueryClientProvider>
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();

```

- index.ts
- Method
  - Queryclient: 캐시를 다룰 때 사용
  - useQuery: GET
  - useMutation: POST, PUT, DELETE



apis

- base: HTTP통신을 하는 CRUD
  - addTodos.ts
  - deleteTodos.ts
  - editTodos.ts
  - getTodos.ts
  - url.ts : ‘/todos’
- instance
  - index.ts: baseUrl, timeout, headers 등을 axios의 instance로 생성-> 여러곳에서 반복해서 작성하지 않아도 됨

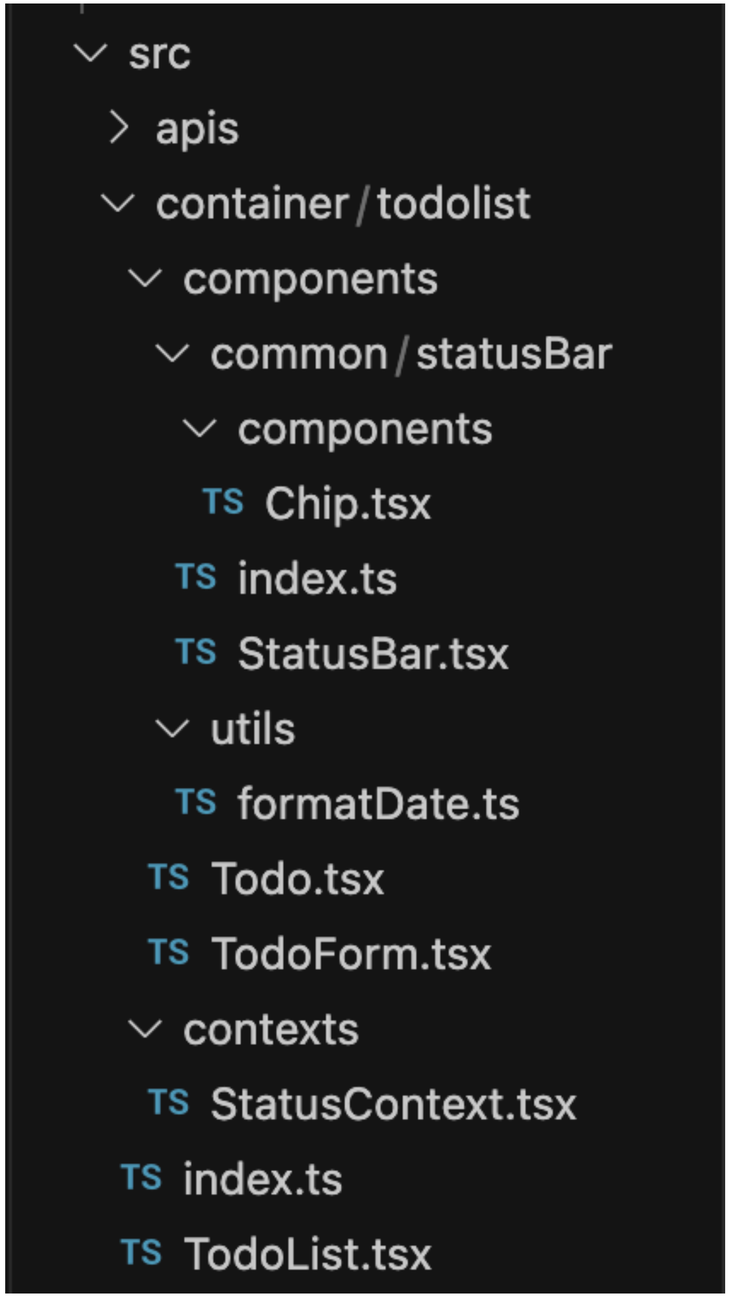


container(todolist) - 1

- components
  - common
    - statusBar
      - components
        - Chip.tsx
      - index.ts
      - StatusBar.tsx
  - utils
    - formDate.ts : 날짜 형식
  - Todo.tsx: Todo component
  - TodoForm.tsx: Todo Form component

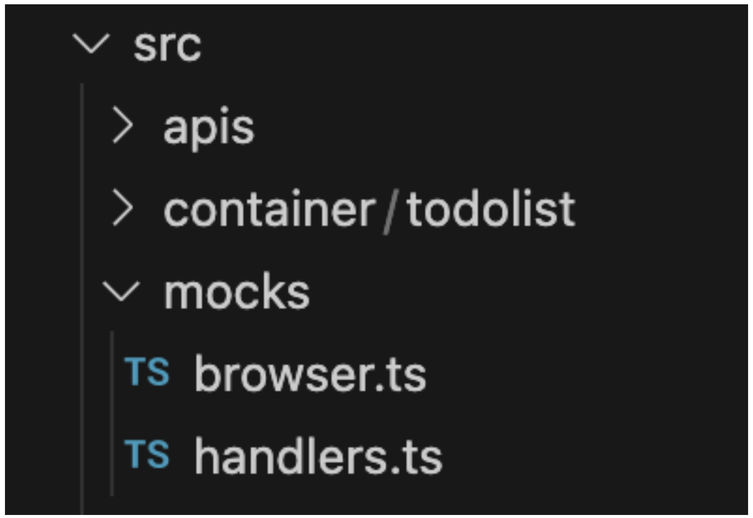
Chip(Component api)

- Chip: https://mui.com/material-ui/api/chip/
- Props, css
- styled-component로 css작성 -> 다른 컴포넌트에서 사용



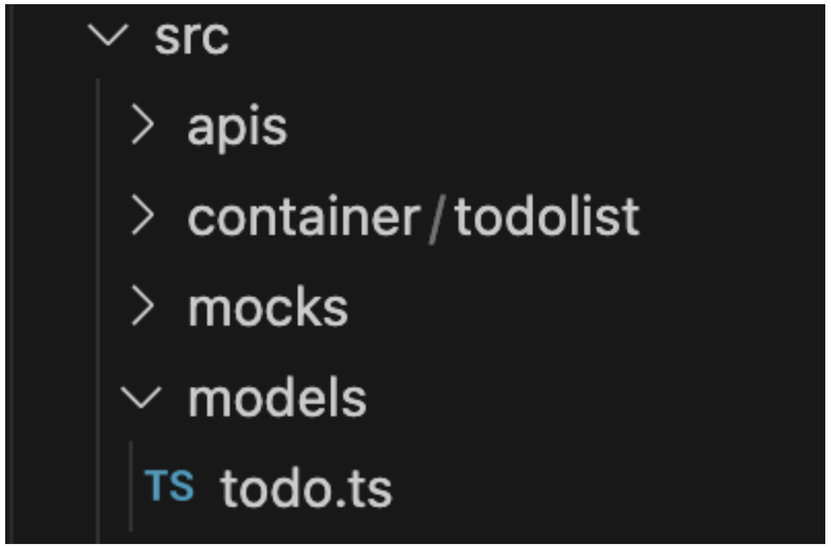
container(todolist) - 2

- Contexts
  - StatusContext.tsx : React context api
- index.ts
- TodoList.tsx



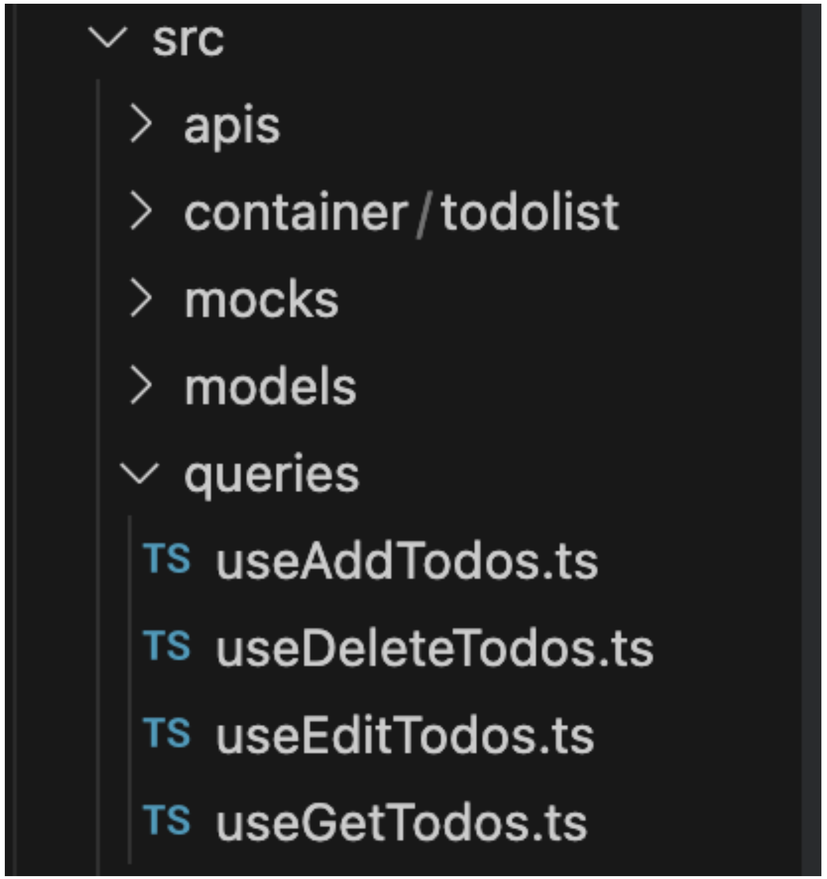
mocks

- mocks
  - browser.ts : msw 서비스워커 정의하고 시작 파일
  - handlers.ts : ‘/todos’ API 요청을 보내 서버의 데이터를 가져고(get), 서버에 데이터를 저장하고(post), 수정하고(put), 삭제(delete)



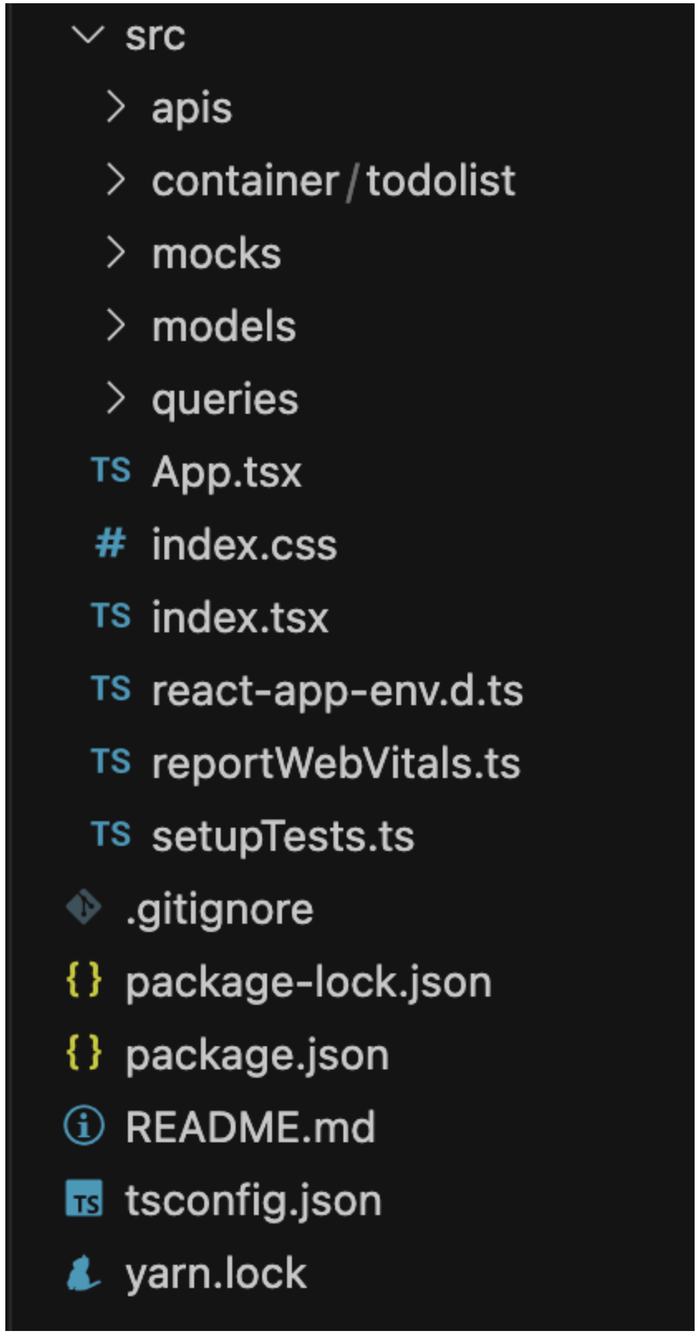
models

- todo.ts : todo 컴포넌트 작업시 필요한 항목 및 타입 정리



queries

- useQuery, useMutation을 사용하여 서버로부터 데이터를 조회/변경할 때 사용
  - useAddTodos.ts
  - useDeleteTodos.ts
  - useEditTodos.ts
  - useGetTodos.ts



그 외

- App.tsx
- index.css
- index.tsx
- react-app-env.d.ts :
  - typeScript 타입 선언을 참조.
  - bmp, git, jpeg, jpg, png 등의 리소스 파일 가져오기에 대한 지원 추가
  - .module.css, .module.scss확장자를 가진 css모듈 가져오기에 대한 지원 추가
- reportWebVitals.ts: React 프로젝트의 성능을 측정하기 위한 파일
- setupTests.ts: 테스트를 실행하기 위한 설정 파일

App.tsx

src > TS App.tsx > ...

```tsx
import { Suspense } from "react";
import { ErrorBoundary } from "react-error-boundary";
import Todolist from "./container/todolist";
import { StatusProvider } from "./container/todolist/contexts/StatusContext";

function App() {
  return (
    <>
      <ErrorBoundary fallback={<div>에러가 발생했습니다.</div>}>
        <Suspense fallback={<div>로딩 중입니다.</div>}>
          <StatusProvider>
            <Todolist />
          </StatusProvider>
        </Suspense>
      </ErrorBoundary>
    </>
  );
}

export default App;

```

- Suspense
- ErrorBoundary: 에러가 있을 경우 fallback이 실행
- Suspense: 데이터를 가져오지 못한경우 fallback이 실행
- StatusContext: context api

index.tsx

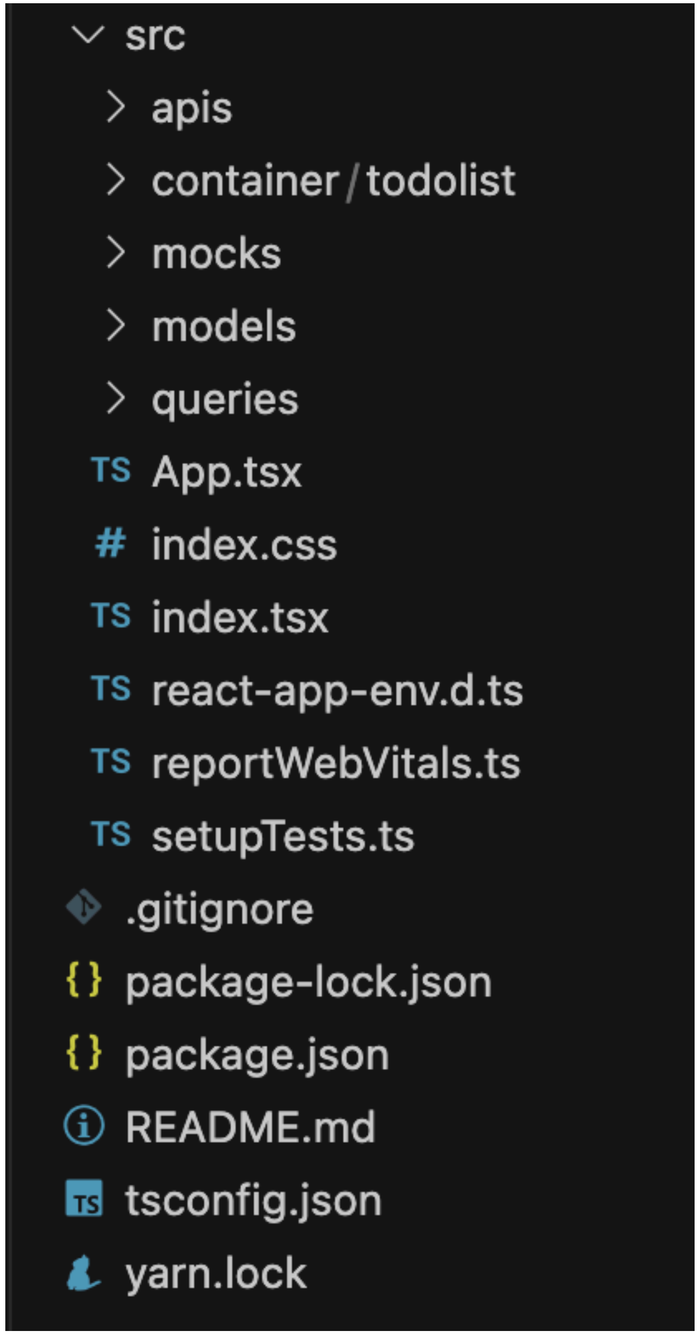
src > TS index.tsx > ...

```tsx
import React from "react";
import ReactDOM from "react-dom/client";
import "./index.css";
import App from "./App";
import reportWebVitals from "./reportWebVitals";
import { QueryClientProvider } from "react-query";
import { queryClient } from "./apis/instance";

if (process.env.NODE_ENV === "development") {
  const { worker } = require("./mocks/browser");
  worker.start();
}

const root = ReactDOM.createRoot(
  document.getElementById("root") as HTMLElement
);
root.render(
  <React.StrictMode>
    <QueryClientProvider client={queryClient}>
      <App />
    </QueryClientProvider>
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();

```

uery 사용

sx를 QueryClientProvider 컴포넌
싸준다

Client값을 props로 넣어준다



그 외

- .gitignore: 레포지토리에 올라가지 않았으면 하는 파일 지정
- package-lock.json
- package.json: 설치한 패키지들의 정보가 담긴 파일 (npm i로 설치 후 프로젝트 시작)
- README.md
- tsconfig.json: 타입스크립트를 컴파일하는데 필요한 루트파일과 컴파일 옵션을 지정하는 파일
- yarn.lock: 이 파일이 생성된 시점의 의존성 트리에대한 정보. Yarn 패키지를 사용한 경우 yarn.lock, npm 패키지를 사용한 경우 package-lock.json파일이 생성됨