React Best Practice

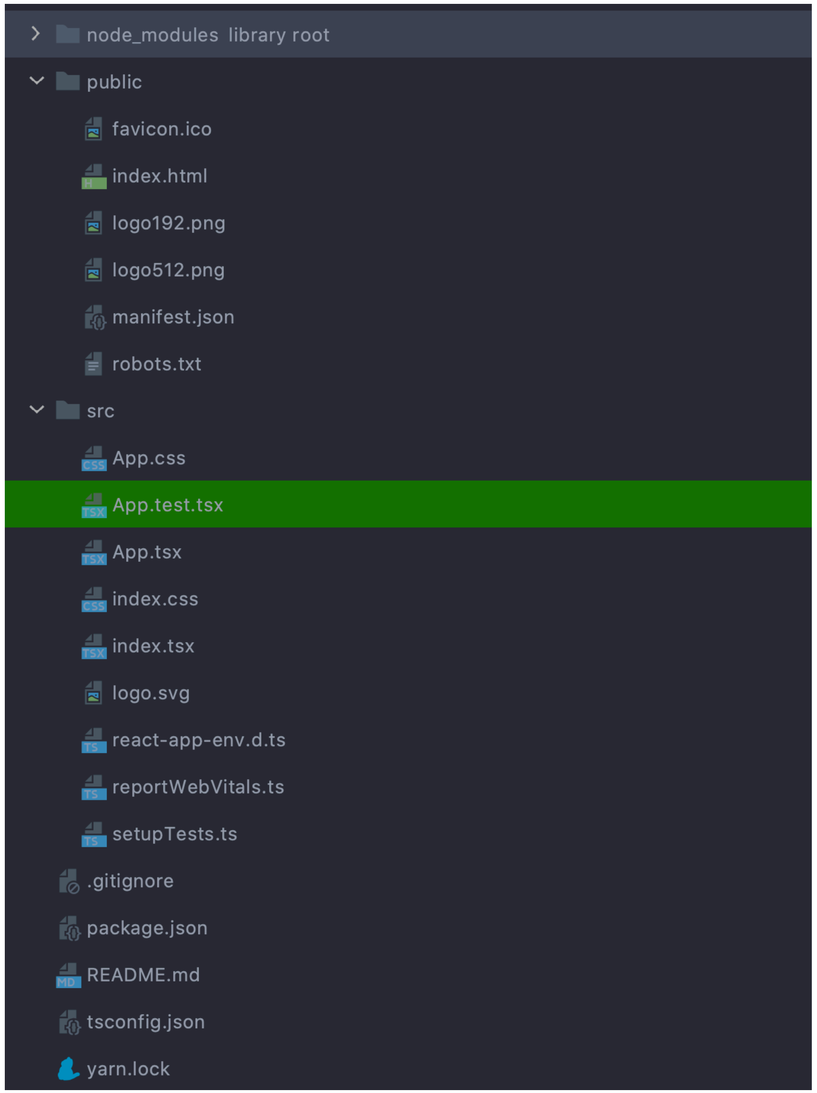
Project Setup

프로젝트 시작하기

- npx create-react-app
- Create-react-app은 react 웹 개발용 boilerplate (template)
- Global module은 자주 사용하는 모듈이 아니다
- 그러므로 오래된 버전을 사용할 위험이 있는데 늘 업데이트 된 버전(일회성)으로 사용하기 위해 npx를 쓴다
  - (npx: npm 패키지를 일회성으로 실행시켜주기 위한 도구)

- Typescript와 함께 사용할려면 npx create-react-app [프로젝트 명] —template typescript
  - (리액트 프로젝트에 타입스크립트를 적용하고 싶은 경우)

Project Setup



Public
- **robots.txt**
  - 웹 크롤러가 웹 페이지를 수집하는 규칙을 담고있는 파일

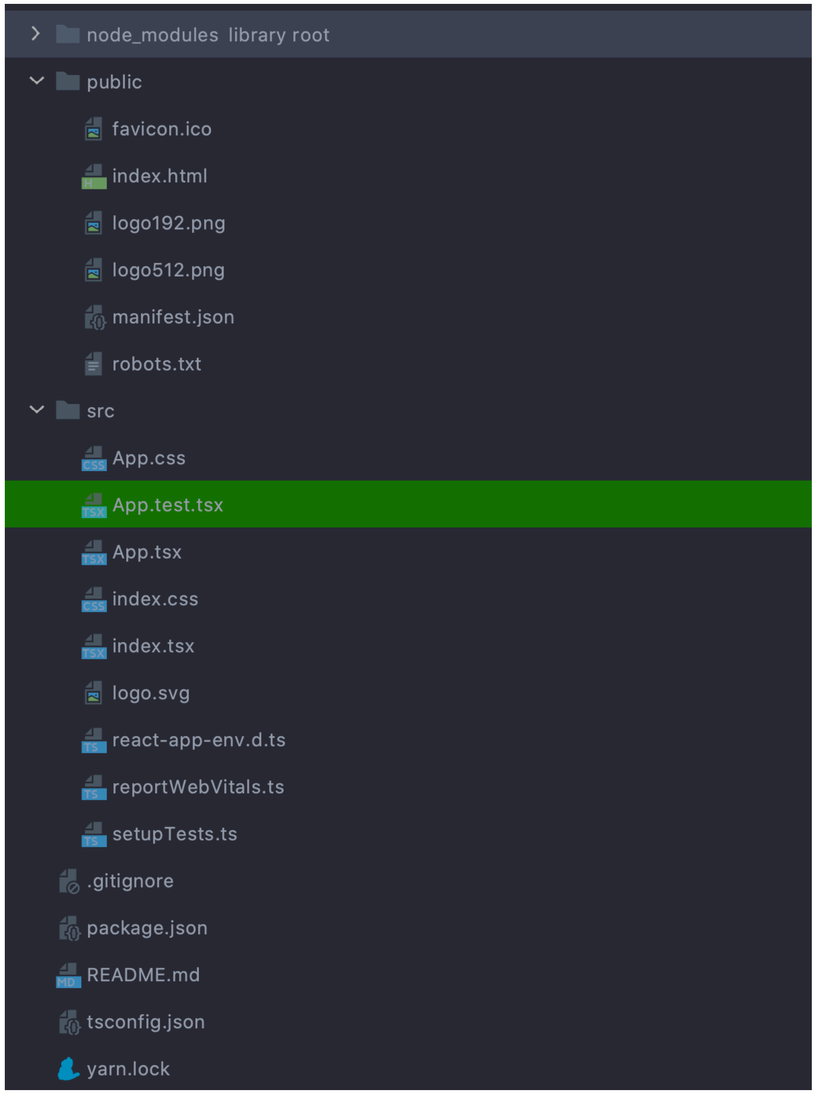
Src
- **reportWebVital.ts**
  - performance를 측정하도록 도와주는 파일
- **setUpTests.ts**
  - Jest라는 자바스크립트 테스트 프레임워크를 통해서 테스트를 할 때 사용하는 파일
- **react-app-env.d.ts**
  - global하게 타입을 확장해서 써야하는 경우

tsconfig.json
- typescript의 컴파일 옵션을 제어할 수 있다 .

- Example)

```
Interface Window {
  property: string
}
```

Project Setup



Public
- **robots.txt**
  - 웹 크롤러가 웹 페이지를 수집하는 규칙을 담고있는 파일

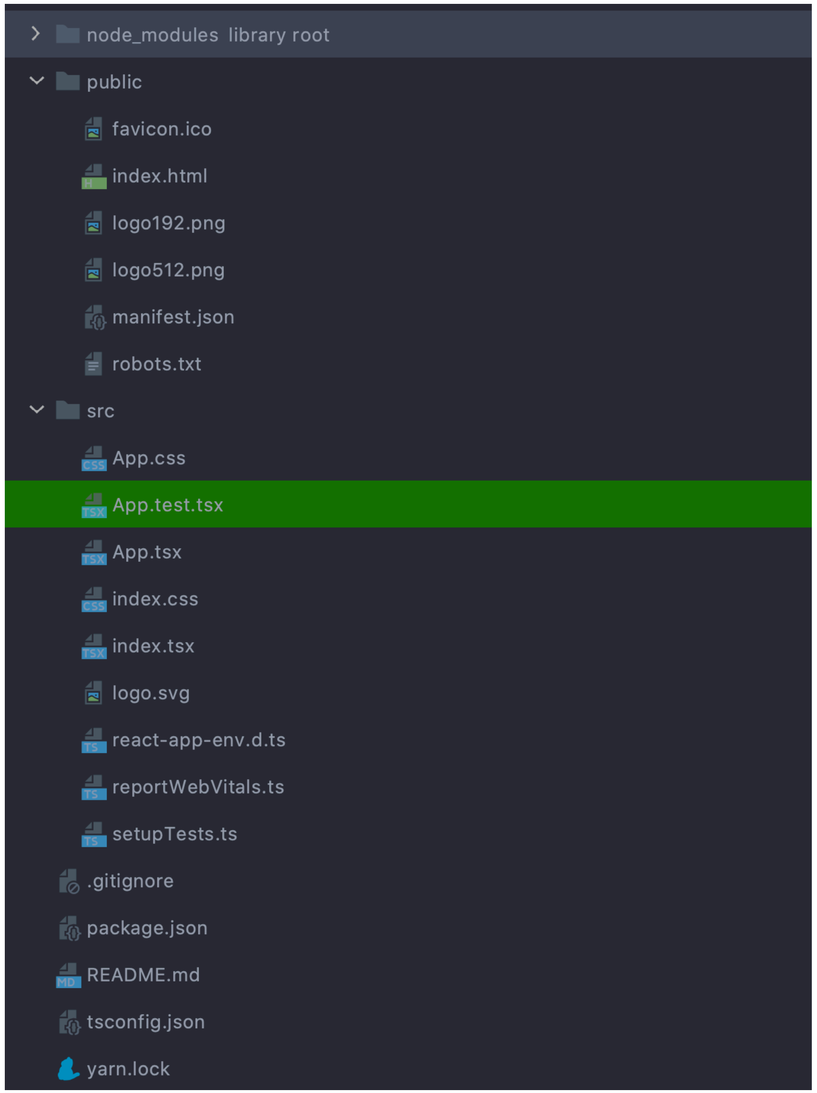
Src
- **reportWebVital.ts**
  - performance를 측정하도록 도와주는 파일
- **setUpTests.ts**
  - Jest라는 자바스크립트 테스트 프레임워크를 통해서 테스트를 할 때 사용하는 파일
- **react-app-env.d.ts**
  - global하게 타입을 확장해서 써야하는 경우

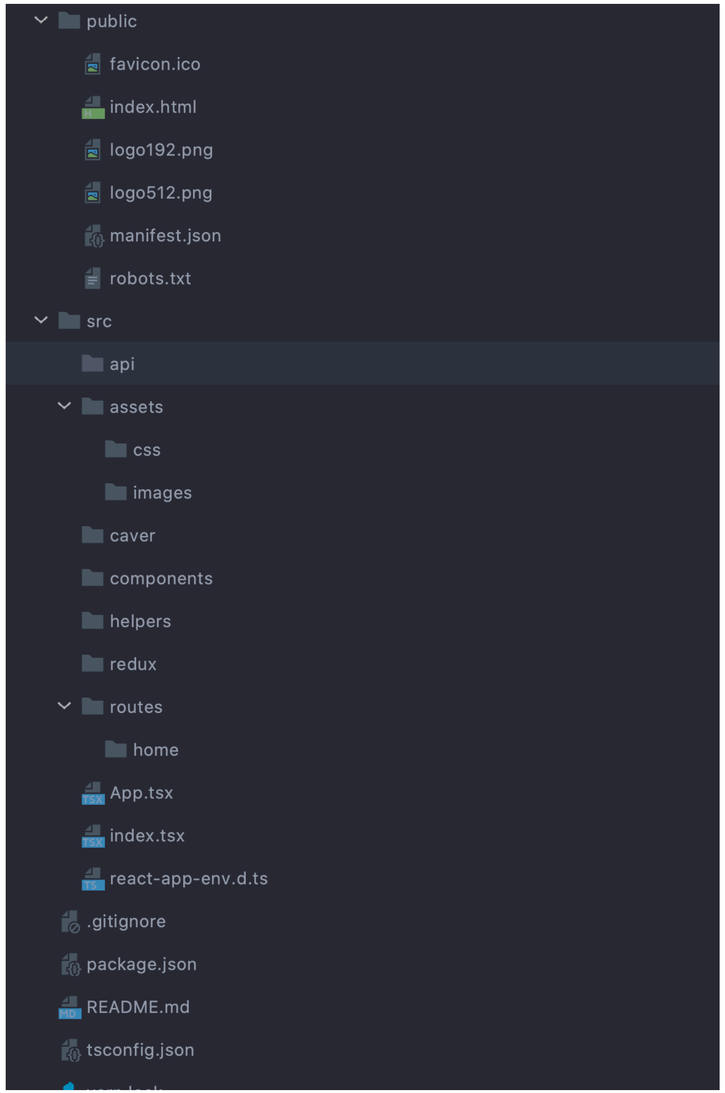
tsconfig.json
- typescript의 컴파일 옵션을 제어할 수 있다 .

- Example)

```
Interface Window {
  property: string
}
```

File Organization





React App File Organization

- Assets
  - images, css
- Helper
  - 외부 라이브러리
- Components
  - 재사용 가능한 컴포넌트들
- Routes
  - 라우트 페이지들
- Redux
  - Redux 상태 관리 라이브러리들
- Caver
  - Klaytn Caver SDK
- Api
  - 외부 요청 API들

Tiny and functional components

할 수 있다면 React Class Component 대신 React Functional Component와 React Hook을 쓰자

- 아무래도 작은 사이즈의 컴포넌트가 재활용될 가능성이 높다
- 라이프 사이클의 도움이 필요하지 않다면 클래스 컴포넌트가 아닌 함수형 컴포넌트를 써라
  - LifeCycle
    - Mounting (Birth of your component)
    - Update (Growth of your component)
    - Unmount (Death of your component)

LifeCycle - Mount

Mount 과정은 다음 순서대로 호출됩니다.

1. constructor()
- (이벤트 처리) 메소드를 바인딩하거나 State를 초기화 할 때 사용된다.
- super(props)를 호출해야 this.props가 생성자 내에 정의된다.
- constructor() 내부에서 setState()를 사용할 수 없다.

2. Static getDerivedStateFromProps()
- render() 메소드 전에 호출되는 라이프 사이클

3. render()
- JSX를 사용해서 생성된 React Element를 리턴한다
- render() 함수는 pure function 이어야 한다
  - 여기 안에서 값을 바꾸면 안된다
  - UI를 그리는 역할만 해야한다

4. componentDidMount()
- 컴포넌트가 마운트 되고나서 호출된다.

LifeCycle - Mount

Static getDerivedStateFromProps()

- **render() 메소드 전에 호출되는 라이프 사이클**
- **state가 props값에 의존하는 경우에 쓰인다.**

```
static getDerivedStateFromProps(props, state) {
    if (props.currentRow !== state.lastRow) {
      return {
        isScrollingDown: props.currentRow > state.lastRow,
        lastRow: props.currentRow,
      };
    }
    // Return null to indicate no change to state.
    return null;
  }
```

LifeCycle - Mount

componentDidMount()

- 컴포넌트가 마운트 되고나서 호출되는 라이프 사이클
- 여기서는 setState()를 할 수 있다.
  - 그러므로 사용할 때 조심해야한다
  - State 초기화를 여기서 하면 성능상의 이슈가 생길 수 있다.

- Api Call을 하기 딱 좋은 시점
- DOM이 다 Mounting되고나서 특정 Position에 위치한 요소들에 대한 무언가를 할 때 적합하다.

ex) 채팅 서비스

```
static getDerivedStateFromProps(props, state) {
    if (props.currentRow !== state.lastRow) {
      return {
        isScrollingDown: props.currentRow > state.lastRow,
        lastRow: props.currentRow,
      };
    }
    // Return null to indicate no change to state.
    return null;
  }
```

LifeCycle - Update

Update
- props 또는 state가 변경되면 호출되는 lifecycle

Update 과정은 다음 순서대로 호출됩니다.

1. **Static getDerivedStateFromProps()**

2. **ShouldComponentUpdate**
- **현재 state나 props의 변화가 컴포넌트의 출력 결과에 영향을 미칠지 결정하는 라이프 사이클**

3. **render()**

4. **getSnapshotBeforeUpdate()**
- **DOM 변화가 일어나기 직전의 DOM 상태를 가져옵니다.**

5. **componentDidUpdate()**
- **props와 state가 바뀌었을 때 호출되는 라이프 사이클**

LifeCycle - Update

ShouldComponentUpdate()

- 현재 state나 props의 변화가 컴포넌트의 출력 결과에 영향을 미칠지 결정하는 라이프 사이클
- Default는 UI에 영향을 주지 않는 State가 바껴도 render() 함수가 재호출된다.
  - 굳이 한번 더 호출될 필요는 없다.
  - 여기서 이제 성능 최적화를 위해서 render()이 호출되는 것을 막을 수 있다.
- shouldComponentUpdate()에서 false를 호출하면 이후의 라이프 사이클은 동작하지 않는다.

LifeCycle - Update

getSnapshotBeforeUpdate()

- **DOM 변화가 일어나기 직전의 DOM 상태를 가져옵니다**
- **DOM으로부터 스크롤 위치등과 같은 정보를 변경되기 전에 가지고 올 수 있다.**
- **여기서 리턴하는 값이 componentDidUpdate 값의 세번째로 들어간다.**

LifeCycle - Update

componentDidUpdate()

- **props와 state가 바뀌었을 때 호출되는 라이프 사이클**
- **최초 렌더링시에는 호출되지 않는다**
- **이전의 props와 비교해서 api 요청을 할 때 사용된다.**
- **여기서 setState를 호출할 수 있지만 사용할 때 조심해야한다.**

```
componentDidUpdate(prevProps) {
 //Typical usage, don't forget to compare the props
 if (this.props.userName !== prevProps.userName) {
   this.fetchData(this.props.userName);
 }
}
```

LifeCycle - unmount

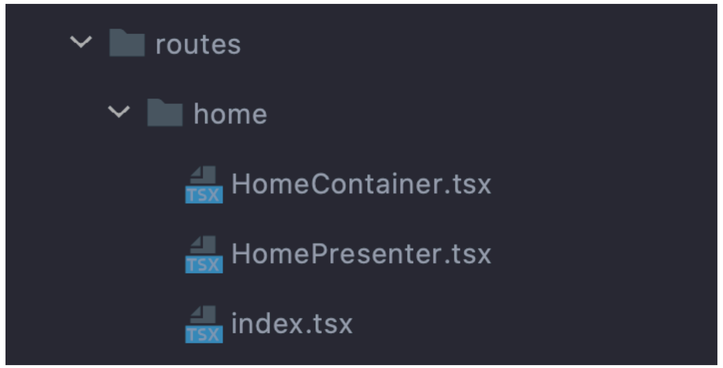
Unmount 과정은 컴포넌트가 DOM에서 제거될 때 호출된다.

- **componentWillUnmount()**
  - **정리작업을 수행할 때 사용된다.**
    - **Cache Storage를 비울 때 사용한다.**
    - **Socket 연결을 off 할 때 사용한다.**

LifeCycle 정리하자면

라이프 사이클에서 처리해야할 작업이 없다면 Functional Component + React Hook을 쓰고 그렇지 않다면 Class Component를 쓰자 !

Container + Presenter Pattern



React Routes 페이지를 만들 때 Container + Presenter 패턴을 사용한다.

Container에서 할 일
- **API Request, Exception Error, setState**

Presenter에서 할 일
- **only Props no logic**

이런 구조가 Container + Presenter 패턴이다.

왜 이렇게 하는가?
- **유지보수 때문에**
- **React에서 해줘야하는 역할이 State 관리, DOM 관리, 이벤트 관리 등 다양한 역할이 필요하다.**
- **그렇기 때문에 기능을 나눠서 역할을 한다.**

그 밖의 것들

Global styled, Typescript, API, Hook 등 할 얘기들은 많지만 너무 내용이 많아져서 다음 기회에 하겠습니다.