React CLI 환경

CLI 환경은 Single Page Application을 만들기 쉽도록 Component 및 개발 환경에 대해 설정을 통해 프로젝트를 구조화한 환경을 제공한다.

- React CLI 설치
  - ◆ 직접 설치하기

1. 폴더 생성 및 초기화

   test 프로젝트 파일을 생성하고 pakage.json 파일을 초기화한다.

```
npm init -y// pakage.json 파일 초기화
```

2. 리액트 핵심 패키지들(애플리케이션 동작과 연관된) 설치(dependencies)

   react, react-dom 설치

```
npm install react react-dom
```

   package.json에 설정을 확인하면 다음과 같다.

```
test-cli > package.json > ...
{
  "name": "test-cli",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "react": "^19.1.0",
    "react-dom": "^19.1.0"
  }
}
```

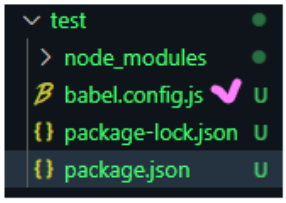
   - package.json script 부분에서 test를 웹팩 사용을 위해 dev로 변경해 준다. 설정한 dev 명령어로 사용 가능하다.
     --mode 옵션은 번들링 진행 상태를 보여준다. > --mode development --open --hot

```
Debug
"scripts": {
  "dev": "webpack serve --mode development --open --hot",
  "build": "webpack --mode production"
},
```

3. 개발에 필요한 라이브러리들(애플리케이션 동작과는 직접연관이 없지만 개발할 때 필요한)설치 (devDependencies)
   바벨(Babel) 설치

```
npm install @babel/core @babel/preset-react @babel/preset-env -D
```

babel.config.js 파일을 생성하고 프리셋들을 설정한다.



- babel.config.js

```
module.exports = {
  presets: ['@babel/preset-react', '@babel/preset-env'],
};
```

4. 웹팩(Web pack) 설치

   모듈 번들러인 웹팩의 핵심 패키지들 설치

```
npm install webpack webpack-cli webpack-dev-server -D
```

- webpack : 웹팩의 코어
- webpack-cli : 웹팩을 커맨드라인에서 사용
- webpack-dev-server : 웹팩을 메모리 상에 빌드하여 개발 서버를 구동

5. 로더 설치

   웹팩 번들링에 필요한 로더들을 설치

```
npm install babel-loader style-loader css-loader file-loader -D
```

- babel-loader : JSX 및 ES6+ 문법을 트랜스파일링

- style-loader : 변환된 CSS 파일을 <style> 태그로 감싸서 삽입
- css-loader : CSS 파일을 자바스크립트가 이해할 수 있도록 변환
- file-loader : 이미지 및 폰트 등의 파일 로딩

6. 플러그인 설치

   웹팩 번들링 후 적용할 플러그인 설치

```
npm install html-webpack-plugin clean-webpack-plugin -D
```

- html-webpack-plugin : HTML 파일에 번들링된 자바스크립트 파일을 삽입해주고 번들링된 결과가 저장되는 폴더에 옮김.
- clean-webpack-plugin : 번들링을 할 때마다 이전 번들링 결과를 제거.
- mini-css-extract-plugin : css 파일로 변환해주는 플러그인.

7. 웹팩 설정

   루트 경로에 webpack.config.js 파일을 생성한다.

test-cli > webpack.config.js > <unknown> > module > rules

```
1  const path = require("path");
2  const HtmlWebpackPlugin = require("html-webpack-plugin");
3  module.exports = {
4    mode: "development",
5    entry: "./src/index.js",
6    output: {
7      path: path.resolve(__dirname, "dist"),
8      filename: "bundle.js",
9    },
10   module: {
11     rules: [
12       {
13         test: /\.jsx?$/, // JS와 JSX 파일을 처리
14         exclude: /node_modules/,
15         use: {
16           loader: "babel-loader",
17         },
18       },
19       {
20         test: /\.css$/, // CSS 파일을 처리
21         use: ["style-loader", "css-loader"],
22       },
23     ],
24   },
```

```
  resolve: {
    extensions: [".js", ".jsx"],
  },
  plugins: [
    new HtmlWebpackPlugin({
      template: "./public/index.html",
    }),
  ],
  devServer: {
    static: {
      directory: path.join(__dirname, "dist"),
    },
    compress: true,
    port: 3000,
    hot: true,
    open: true,
  },
};

```

8. 리액트 컴포넌트 생성

8.1 public/index.html 생성

```
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>CRA 없이 리액트 개발 환경 구축</title>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

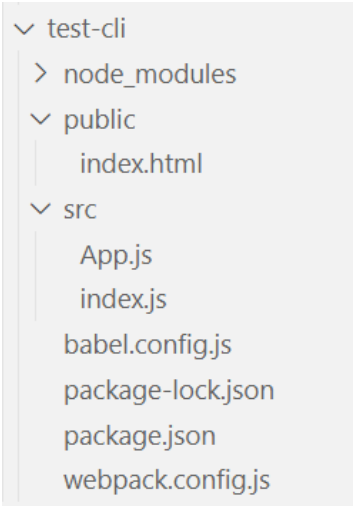
8.2 src/App.js 생성

```
test-cli > src > App.js > ...
import React from "react";

const App = () => {
  return <div>Hello React!!!</div>;
};

export default App;
```

8.3 src/index.js 생성

```jsx
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<App />);
```

완성 후 폴더 구조



9. 실행

npm run dev

■ CRA로 react App 생성하기

CLI 환경을 node.js를 이용해서 설치하기 (node.js 설치가 안된 경우 미리 node.js를 설치한다)

형식] npm install –g create-react-app

```
C:\uplus\06_react>npm i -g create-react-app
```

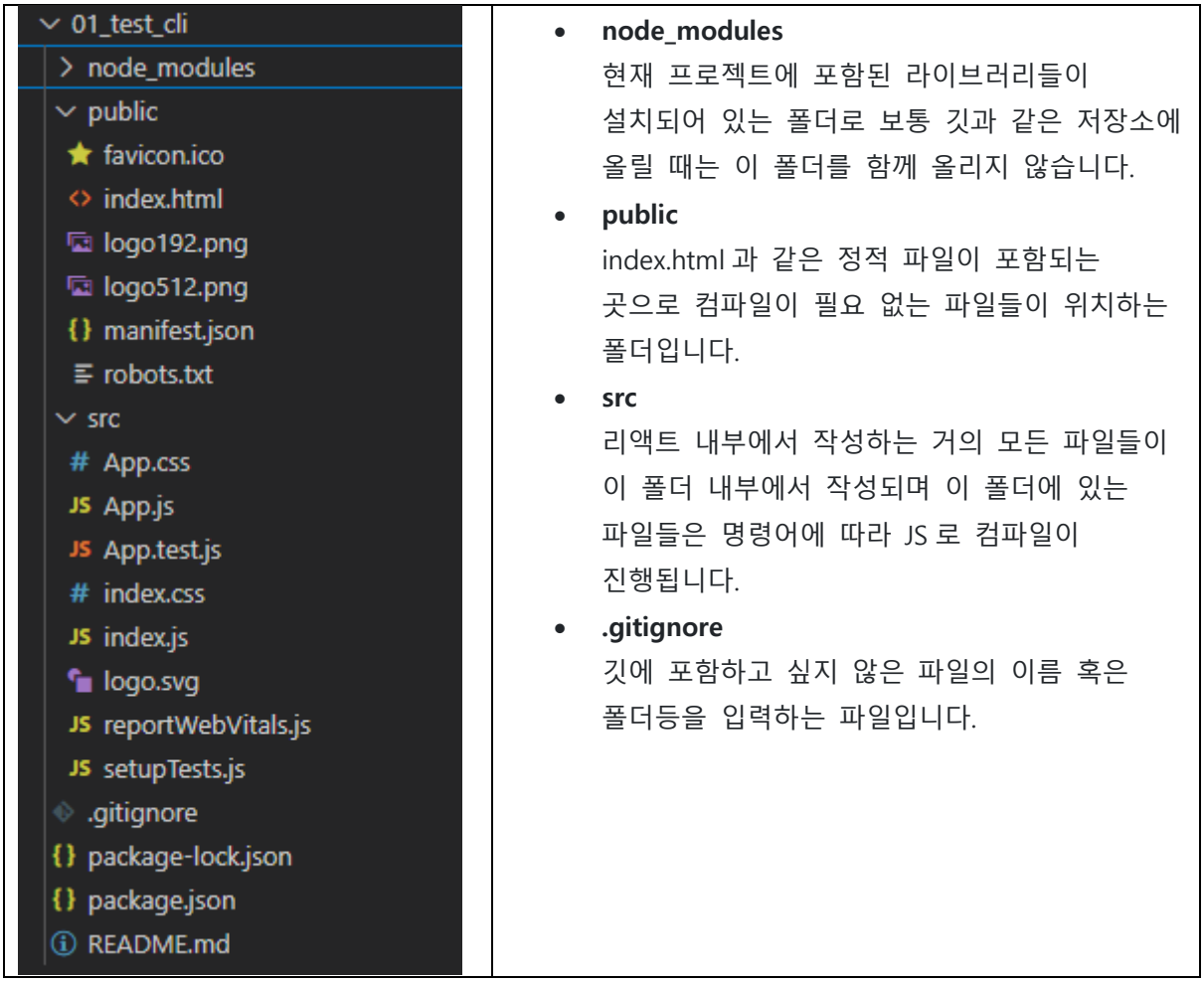
■ React CLI 프로젝트 생성하기

형식] create-react-app 프로젝트명

```
C:\uplus\06_react>create-react-app 01_test_cli
```

```
C:\uplus\06_react>create-react-app 01_test_cli
create-react-app is deprecated.

You can find a list of up-to-date React frameworks on react.dev
For more info see:https://react.dev/link/cra

This error message will only be shown once per install.

Creating a new React app in C:\uplus\06_react\01_test_cli.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...


added 1325 packages in 38s

268 packages are looking for funding
```

생성된 프로젝트 구조

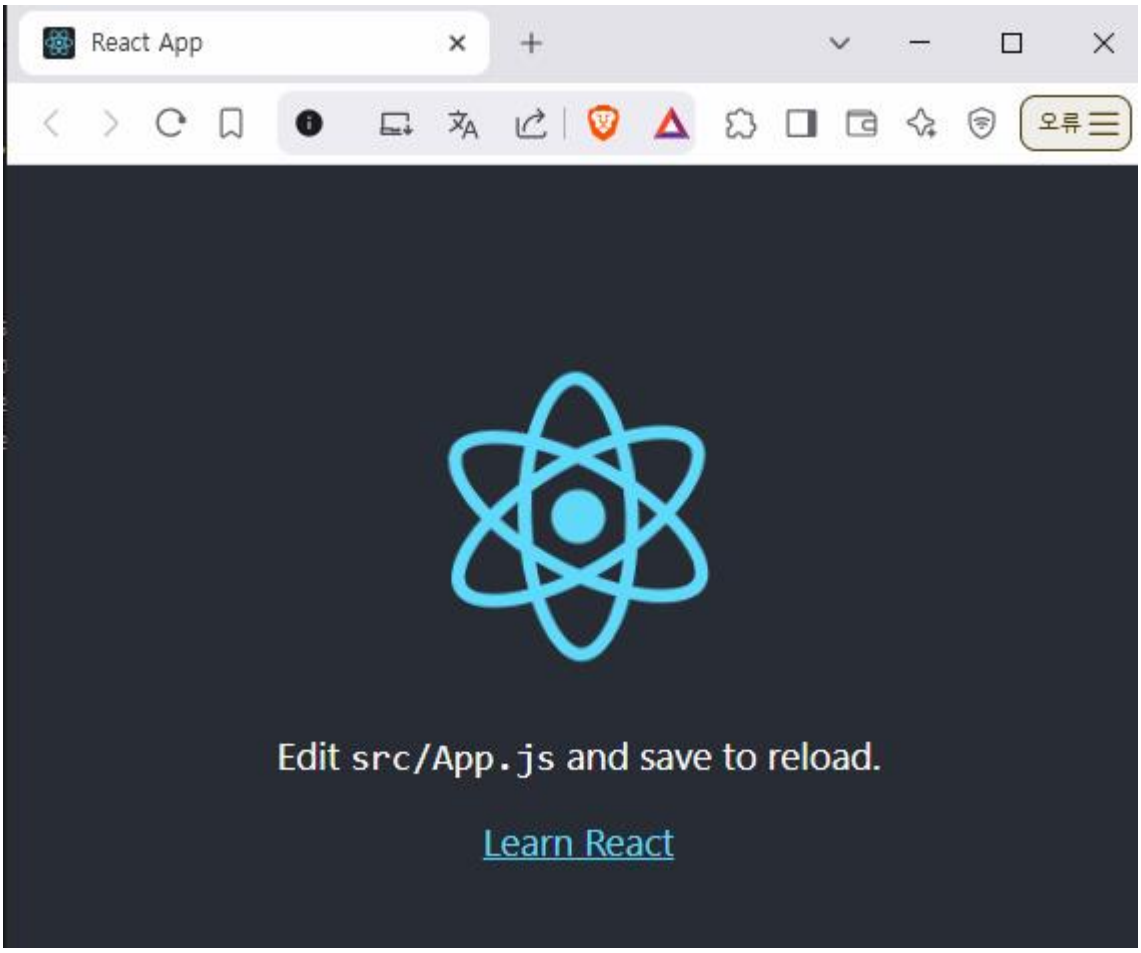


- **node_modules**
  현재 프로젝트에 포함된 라이브러리들이 설치되어 있는 폴더로 보통 깃과 같은 저장소에 올릴 때는 이 폴더를 함께 올리지 않습니다.
- **public**
  index.html 과 같은 정적 파일이 포함되는 곳으로 컴파일이 필요 없는 파일들이 위치하는 폴더입니다.
- **src**
  리액트 내부에서 작성하는 거의 모든 파일들이 이 폴더 내부에서 작성되며 이 폴더에 있는 파일들은 명령어에 따라 JS 로 컴파일이 진행됩니다.
- **.gitignore**
  깃에 포함하고 싶지 않은 파일의 이름 혹은 폴더등을 입력하는 파일입니다.

- **package.json**
  프로젝트에 관련된 기본적인 내용(프로젝트의 이름, 버전 등)과 라이브러리들의 목록이 포함되어 있습니다. 라이브러리가 설치된 node_modules 대신에 이 package.json 을 깃에 포함하여 올리게 되며 후에 누군가가 프로젝트를 클론할 때 이 package.json 에 적혀있는 라이브러리의 목록을 기준으로 npm 에서 설치하게 됩니다.
- **README.md**
  보통 깃과 같은 저장소에 올릴 때 프로젝트에 대한 설명을 작성하는곳으로 해당 저장소에 진입하면 가장 먼저 띄워집니다.

실행] npm start

실행하면 CLI에서 미리 작성한 기본 화면이 나온다.



참고 : package.json에서 버전 관리

틸드(~) : 현재 지정한 버전의 마지막 자리 내의 범위에서만 자동으로 업데이트한다.

- ~0.0.1 : >=0.0.1 <0.1.0
- ~0.1.1 : >=0.1.1 <0.2.0
- ~0.1 : >=0.1.0 <0.2.0
- ~0 : >=0.0 <1.0

캐럿(^) : Node.js 모듈이 이 SemVer의 규약을 따른다는 것을 신뢰한다는 가정하에서 동작한다. 그래서 MINOR나 PATCH버전은 하위호환성이 보장되어야 하므로 업데이트를 한다.

- ^1.0.2 : >=1.0.2 <2.0
- ^1.0 : >=1.0.0 <2.0
- ^1 : >=1.0.0 <2.0

SemVer :

> 1. MAJOR version when you make incompatible API changes,
> 2. MINOR version when you add functionality in a backwards-compatible manner, and
> 3. PATCH version when you make backwards-compatible bug fixes.

즉, MAJOR 버전은 API의 호환성이 깨질 만한 변경사항을 의미하고 MINOR 버전은 하위호환성을 지키면서 기능이 추가된 것을 의미하고 PATCH 버전은 하위호환성을 지키는 범위 내에서 버그가 수정된 것을 의미한다.