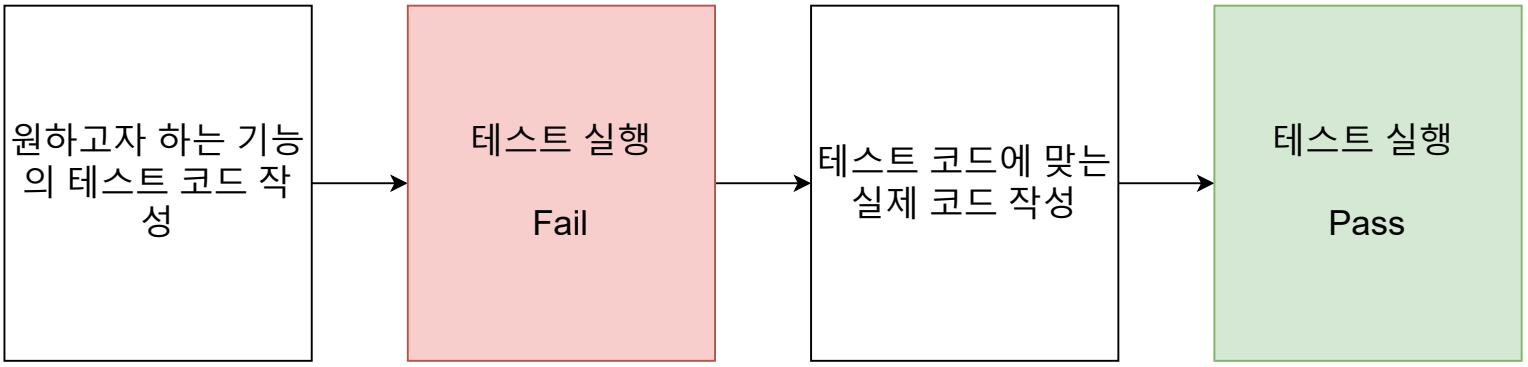
# 테스트 주도 개발(Test Driven Development)이란 무엇인가?

## Test Driven Development 란 무엇인가요?

실제 코드를 작성하기 전에 테스트 코드를 먼저 작성합니다.

테스트 코드를 작성한 후 그 테스트 코드를 Pass 할 수 있는 실제 코드를 작성합니다.




## TDD를 하면 좋은 점

1. TDD를 하므로 인해 많은 기능을 테스트하기에 소스 코드에 안정감이 부여된다.

2. 실제 개발하면서 많은 시간이 소요되는 부분은 디버깅 부분이기에 TDD를 사용하면 디버깅 시간이 줄어들고 실제 개발 시간도 줄어듭니다.

3. 소스 코드 하나하나를 더욱 신중하게 짤 수 있기 때문에 깨끗한 코드가 나올 확률이 높습니다.

# React Testing Library 란?

Create React App 로 리액트 앱을 생성하면 기본적으로 테스팅 할 때 React Testing Library를 사용하는 것을 볼 수 있습니다. 그럼 이 React Testing Library가 무엇일까요?


## React Testing Library란 무엇인가요?

React Testing Library는 React 구성 요소 작업을 위한 API를 추가하여 DOM Testing Library 위에 구축됩니다.

DOM Testing Library란 Dom 노드(Node)를 테스트하기 위한 매우 가벼운 솔루션입니다.

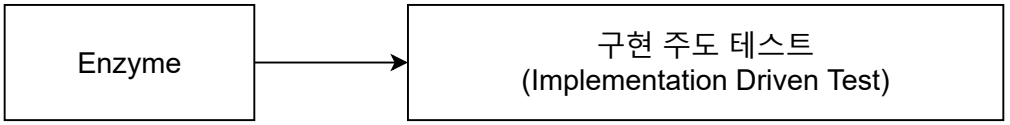
Create React App으로 생성된 프로젝트는 즉시 React Testing Library를 지원합니다. 그렇지 않은 경우 다음과 같이 npm을 통해 추가할 수 있습니다.

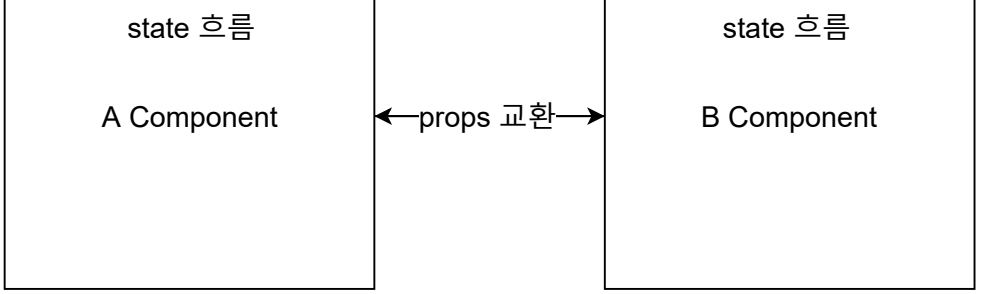
ddfsdf

```
npm install --save-dev @testing-library/react
```

리액트 컴포넌트를 테스트하는 가벼운 솔루션!

React Testing Library는  에어비앤비에서 만든 Enzyme을 대처하는 솔루션 입니다.
Enzyme이 구성 요소의 구현 세부 정보를 테스트하는 대신
React Testing Library는 개발자를 React 애플리케이션의 사용자 입장에 둡니다.




```
<p>
Edit <code>src/App.js</code> and save to
reload....
</p>
```

구현 주도 테스트에서는 위의 UI를 테스트할 때 주로 <p> 태그가 쓰였고 Edit 등의 문자가 들어갔다는것을 테스트합니다. 그래서 만약 <p>태그를 <h2> 태그로 바뀌면 에러가 날 것입니다.

하지만 행동 주도 테스트에서는 사용자의 입장에서 테스트 하기에 <p>태그가 쓰이던 <h3> 태그가 쓰여서 글을 표현하는지가 중요한지 보다 어떠한 이벤트를 발생시켰을때 화면이 어떻게 변화가 되는지 같은 테스트가 더 주를 이루게 됩니다.

이렇게 가볍게 알아보고 실제 사용하면서 더 자세히 알아보겠습니다

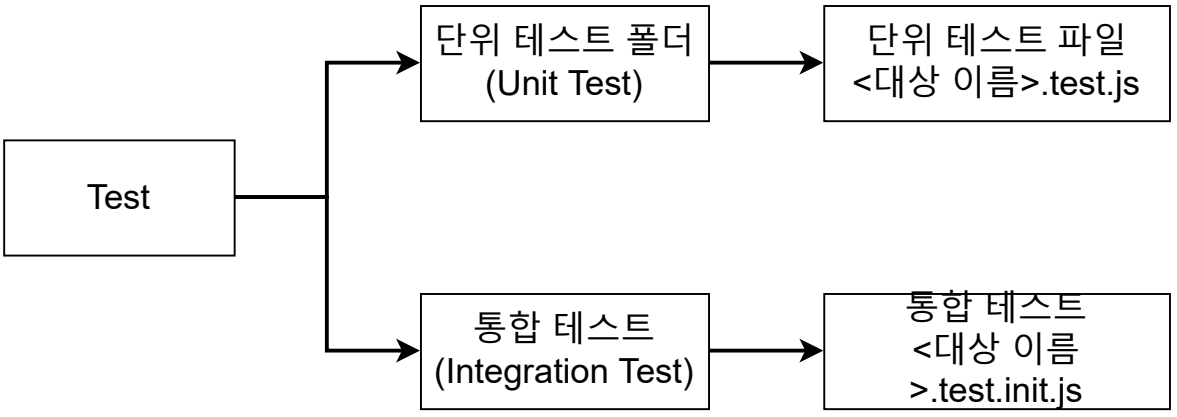
# Jest 란?

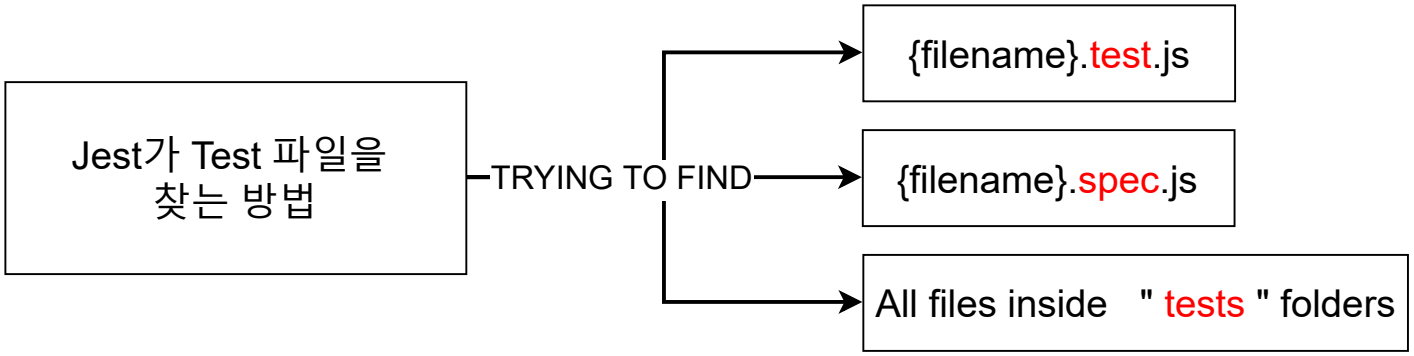
## Jest란 무엇인가요?

FaceBook에 의해서 만들어진 테스팅 프레임 워크입니다.
최소한의 설정으로 동작하며 Test Case 를 만들어서 어플리케이션 코드가
잘 돌아가는지 확인해줍니다.
단위 (Unit) 테스트를 위해서 이용합니다.

## Jest 시작하기

1. Jest 라이브러리 설치 npm install jest --save-dev
2. Test 스크립트 변경 "test" : "jest" or "jest --watchAll
3. 테스트 작성할 폴더 및 파일 기본 구조 생성





Jest가 Test 파일을
찾는 방법
TRYING TO FIND
{filename}.test.js
{filename}.spec.js
All files inside " tests " folders

# Jest 파일 구조 및 사용법

## Jest 파일 구조

```
describe("Product Controller create", () => {
  beforeEach(() => {
    req.body = newProduct;
  });
  it("should have a createProduct function", () => {
    expect(typeof productController.createProduct).toBe("function");
  });
  it("should call Product.create", async () => {
    await productController.createProduct(req, res, next);
    expect(Product.create).toBeCalledWith(newProduct);
  });
  it("should return 201 response code", async () => {
    await productController.createProduct(req, res, next);
    expect(res.statusCode).toBe(201);
    expect(res._isEndCalled()).toBeTruthy();
  });
  it("should return json body in response", async () => {
    Product.create.mockReturnValue(newProduct);
    await productController.createProduct(req, res, next);
    expect(res._getJSONData()).toStrictEqual(newProduct);
  });
  it("should handle errors", async () => {
    const errorMessage = { message: "Done property missing" };
    const rejectedPromise = Promise.reject(errorMessage);
    Product.create.mockReturnValue(rejectedPromise);
    await productController.createProduct(req, res, next);
    expect(next).toBeCalledWith(errorMessage);
  });
});
```

**describe**

**test (it)**

**test (it)**

**test (it)**

## Explain

**"describe"**
***argument ( name, fn )***

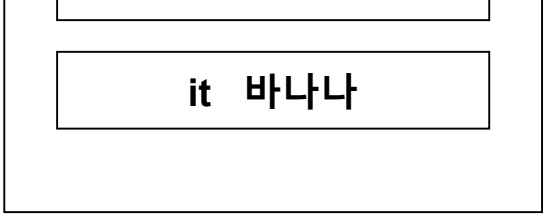
여러 관련 테스트를
**그룹화**하는 블록을 만듭니다.

**"it"** same as **test**
***argument ( name, fn, timeout )***

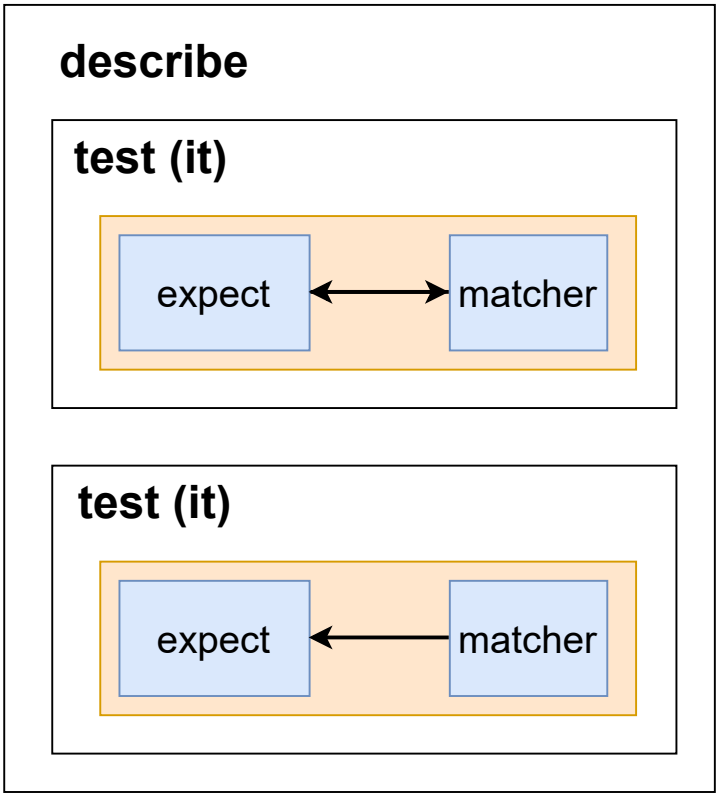
**개별 테스트**를 수행하는 곳.
각 테스트를 작은 문장처럼
설명합니다.

**Describe (과일)**

**it 사과**




```
describe("Product Controller create", () => {
  beforeEach(() => {
    req.body = newProduct;
  });
  it("should have a createProduct function", () => {
    expect(typeof productController.createProduct).toBe("function");
  });
  it("should call Product.create", async () => {
    await productController.createProduct(req, res, next);
    expect(Product.create).toBeCalledWith(newProduct);
  });
  it("should return 201 response code", async () => {
    await productController.createProduct(req, res, next);
    expect(res.statusCode).toBe(201);
    expect(res._isEndCalled()).toBeTruthy();
  });
  it("should return json body in response", async () => {
    Product.create.mockReturnValue(newProduct);
    await productController.createProduct(req, res, next);
    expect(res._getJSONData()).toStrictEqual(newProduct);
  });
  it("should handle errors", async () => {
    const errorMessage = { message: "Done property missing" };
    const rejectedPromise = Promise.reject(errorMessage);
    Product.create.mockReturnValue(rejectedPromise);
    await productController.createProduct(req, res, next);
    expect(next).toBeCalledWith(errorMessage);
  });
});
```




## Explain

**"expect"**

expect 함수는 값을
테스트할 때마다 사용됩니다.
그리고 expect 함수 혼자서는
거의 사용 되지 않으며 matcher와
함께 사용됩니다.

**"matcher"**

다른 방법으로 값을 테스트 하도록
"매처"를 사용합니다.

```
test('two plus two is four', () => {
  expect(2 + 2).toBe(4);
});
```

matcher

```
PASS test/unit/products.test.js
  ✓ two plus two is four (2 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.279 s, estimated 1 s
Ran all test suites.

Watch Usage: Press w to show more.
```

```
test('two plus two is not five', () => {
  expect(2 + 2).not.toBe(5);
});
```

matcher

```
PASS test/unit/products.test.js
  ✓ two plus two is four (2 ms)
  ✓ two plus two is not five (1 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        0.281 s, estimated 1 s
Ran all test suites.

Watch Usage: Press w to show more.
```

React Testing Library 주요 API

npx create-react-app react-testing-app

앱을 키고 기본 테스트 진행 → a

q quit

App.test.js → 기본 테스트가 진행되는 곳

```
import { render, screen } from '@testing-library/react';
import App from './App';

test('renders learn react link', () => {
  render(<App />);
  const linkElement = screen.getByText(/learn react/i);
  expect(linkElement).toBeInTheDocument();
});
```

**"render" 함수**

DOM에 컴포넌트를 랜더링하는 함수

인자로 랜더링할 React 컴포넌트가 들어감

Return은 RTL에서 제공하는 쿼리 함수와 기타 유틸리티 함수를 담고 있는 객체를 리턴(Destructuring 문법으로 원하는 쿼리 함수만 얻어올 수 있다.)
====> 소스 코드가 복잡해지면 비추천 !!! screen 객체를 사용하기
왜냐면 사용해야 할 쿼리가 많아질수록 코드가 복잡해질 수 있음

```
test('renders learn react link', () => {
  const { getByText } = render(<App />);
  const linkElement = getByText(/learn react/i);
  expect(linkElement).toBeInTheDocument();
});
```

쿼리 함수에 대해서

공식 문서 → https://testing-library.com/docs/queries/about/

## 쿼리 함수란 ?

쿼리는 페이지에서 요소를 찾기 위해 테스트 라이브러리가 제공하는 방법입니다. 여러 유형의 쿼리("get", "find", "query")가 있습니다. 이들 간의 차이점은 요소가 발견되지 않으면 쿼리에서 오류가 발생하는지 또는 Promise를 반환하고 다시 시도하는지 여부입니다. 선택하는 페이지 콘텐츠에 따라 다른 쿼리가 다소 적절할 수 있습니다.

```
import {render, screen} from '@testing-library/react'

test('should show login form', () => {
  render(<Login />)
  const input = screen.getByLabelText('Username')
  // Events and assertions...
})
```

## get, find, query 의 차이점

쿼리는 페이지에서 요소를 찾기 위해 테스트 라이브러리가 제공하는 방법입니다. 여러 유형의 쿼리("get", "find", "query")가 있습니다. 이들 간의 차이점은 요소가 발견되지 않으면 쿼리에서 오류가 발생하는지 또는 Promise를 반환하고 다시 시도하는지 여부입니다. 선택하는 페이지 콘텐츠에 따라 다른 쿼리가 다소 적절할 수 있습니다.

| "getBy..." | "queryBy..." | "findBy..." |
| --- | --- | --- |
| 쿼리에 대해 일치하는 노드를 반환하고 일치하는 요소가 없거나 둘 이 | 쿼리에 대해 일치하는 노드를 반환하고 일치하는 요소가 없으면 null을 반환합니다. 이것은 존재하지 않 | 주어진 쿼리와 일치하는 요소가 발견되면 Promise를 반환합니다. 요소가 발견되지 않거나 기본 제한 시간 |

상의 일치가 발견되면 실행 오류를 발생시킵니다(둘 이상의 요소가 예상되는 경우 대신 getAllBy 사용).

는 요소를 어설션하는 데 유용합니다. 둘 이상의 일치 항목이 발견되면 오류가 발생합니다(확인된 경우 대신 queryAllBy 사용).

인 1000ms 후에 둘 이상의 요소가 발견되면 약속이 거부됩니다. 둘 이상의 요소를 찾아야 하는 경우 findAllBy를 사용하십시오.

## getBy + waitFor = findBy

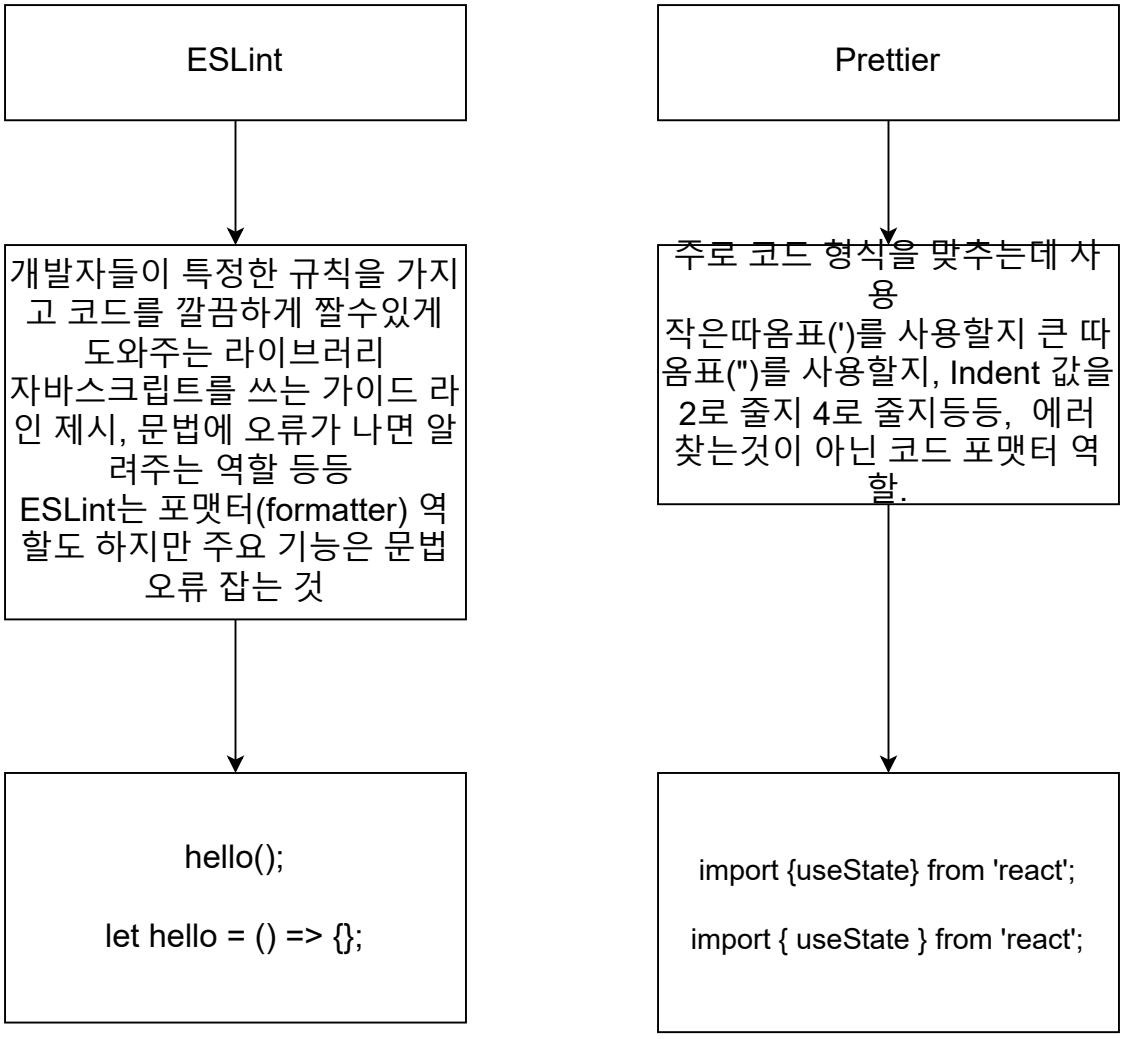
| Type of Query | 0 Matches | 1 Match | >1 Matches | Retry (Async/Await) |
|---|---|---|---|---|
| **Single Element** | | | | |
| `getBy...` | Throw error | Return element | Throw error | No |
| `queryBy...` | Return `null` | Return element | Throw error | No |
| `findBy...` | Throw error | Return element | Throw error | Yes |
| **Multiple Elements** | | | | |
| `getAllBy...` | Throw error | Return array | Return array | No |
| `queryAllBy...` | Return `[]` | Return array | Return array | No |
| `findAllBy...` | Throw error | Return array | Return array | Yes |

**"waitFor"**

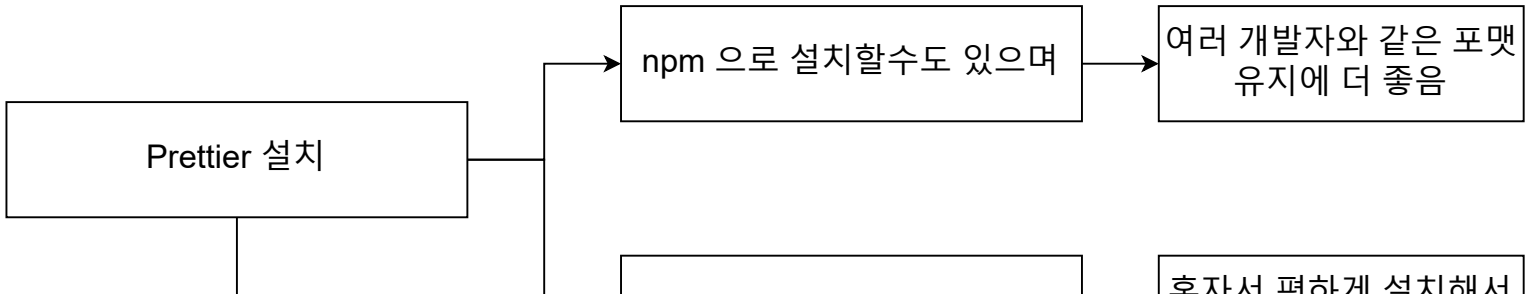
일정 기간 동안 기다려야 할 때 waitFor를 사용하여 기대가 통과할 때까지 기다릴 수 있습니다.

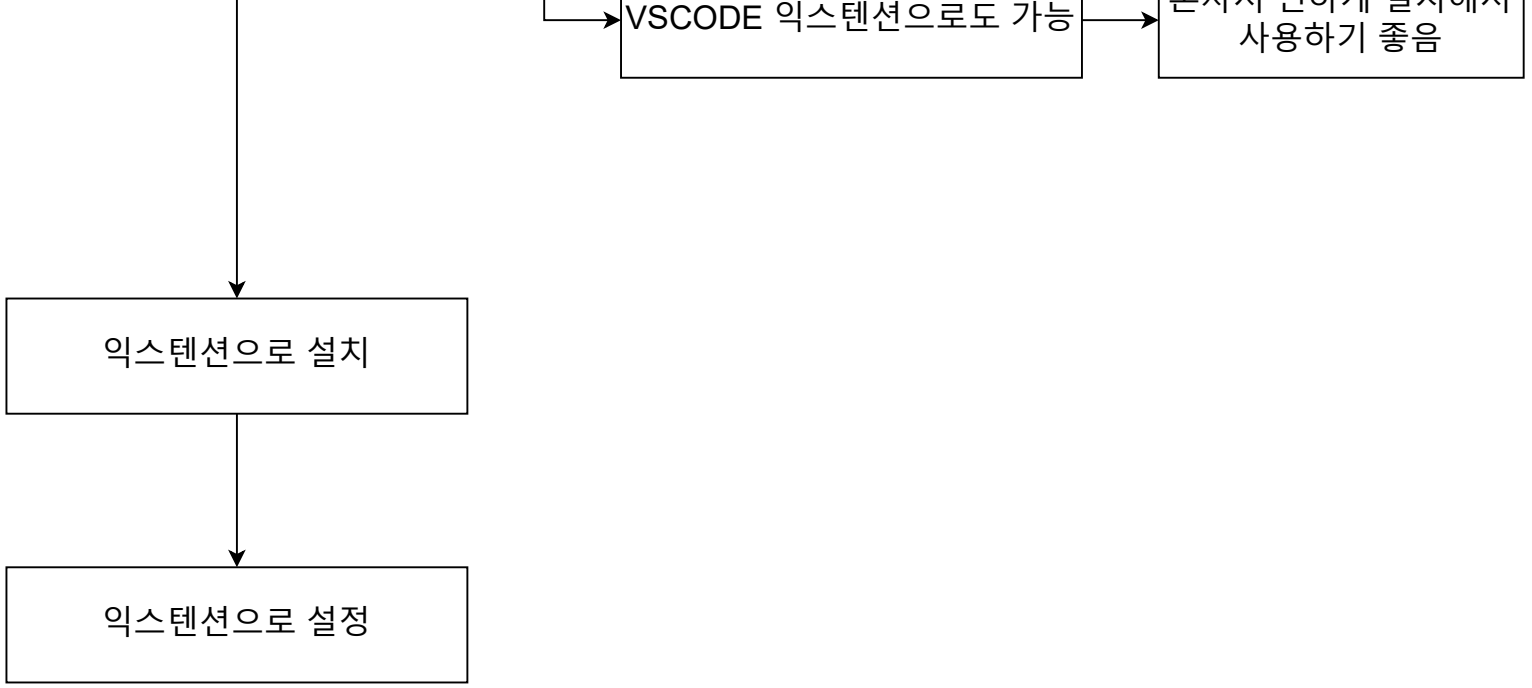
# Prettier 설치 및 설정

테스팅할 때 matcher를 알맞게 쓰는지 확신이 들지 않을 때가 있으며, 코드의 형식이나 자바스크립트 문법 등을 올바르게 쓰지 못할 때가 있습니다. 그러한 부분을 도와주는 모듈을 설치해주는 시간을 갖겠습니다.




Prettier는 코드 형식을 맞추는데 사용을 합니다. 테스팅을 위해서 특화된 것은 아니지만 ESLint와 함께 자주 사용하기에 Prettier도 설치해보겠습




VSCODE 익스텐션으로도 가능
사용하기 좋음
익스텐션으로 설치
익스텐션으로 설정

# ESLint 설치 및 설정하기

ESLint 익스텐션 설치

이미 Create React App 으로 리액트를 설치할 때 기본 eslint가 설정되어 있습니다. 하지만 이 상태로는 VS Code에서 바로 에러 확인 할 수 없고 앱을 시작했을 때 터미널 상에서 볼 수 있습니다.

eslint 설정 파일 생성, package.json에 eslintConfig 부분 지우고 .eslintrc.json 파일 생성

이렇게 하면 PROBELMS 탭에서 ESLint 에서 주는 기본적인 경고들을 확인 할 수 있습니다.

```
{
    "extends": [
        "react-app",
        "react-app/jest"
    ]
}
```

이제는 Testing을 도와주는 ESlint를 설치해보겠습니다.

ESLint Testing Plugins 설치

Plugins 란 ?

eslint에서 기본으로 제공하지 않는 다양한 규칙을 플러그인을 통해 사용할 수 있습니다.
예를 들어서 react에 관련된 린트설정을 위해서는 eslint-plugin-react를 사용하면 되며, react hooks에 관련된 규칙을 적용시켜주려면
eslint-plugin-react-hooks를 사용하면 됩니다.

npm install
eslint-plugin-testing-library
eslint-plugin-jest-dom

testing-library
render로 Dom 그리는 부분

jest-dom
expect-matcher로 테스트

내부 설정해주기

plugins 항목 : 플러그인 추가
추가할 때,
eslint-plugin- 부분 생략가능

extends 항목 :
플러그인을 추가 한 후에 규칙을 정해줘야 사용가능합니다.
그래서 extends 항목에 사용하고자 하는 규칙을 설정합니다.
vue, angular, react 중에 react 를 위한 규칙
recommended는 추천이 되는 걸 사용

만약 규칙을 변경하고자 할 때는 rule 항목 추가

```
{
    "plugins": [
        "testing-library",
        "jest-dom"
    ],
    "extends": [
        "react-app",
        "react-app/jest",
        "plugin:testing-library/react",
        "plugin:jest-dom/recommended"
    ]
}
```

lint가 잘 작동하는지 확인

```
const lintTest = screen.getByRole('button', {
  name: 'lintTest',
});

expect(lintTest.textContent).toBe('lintTest');
```