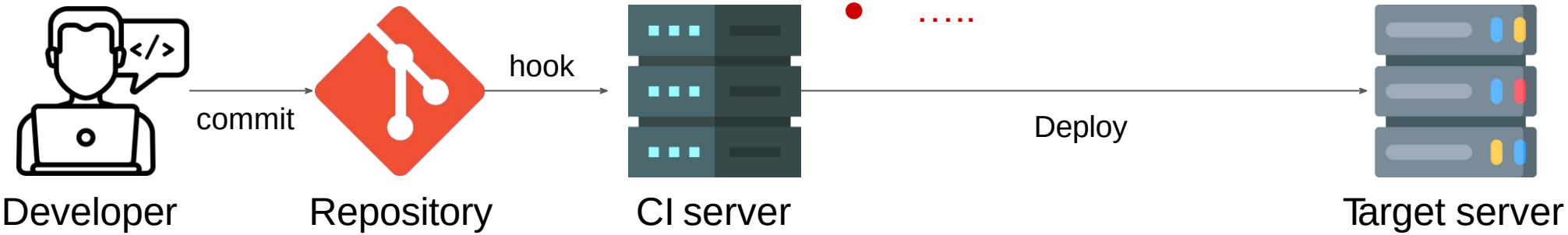
react-testing-library 를 사용하여 TDD 개발

CI Workflows

- TDD



- Compile
- **Run unit/integration test -> Report!**
- Static Analyses
- …
- …..

TDD

Test Driven Development

테스트 주도 개발

우리는 주로 테스트를 코딩을 다 하고 나서 해 왔다.
테스트 주도 개발이라는 말의 의미가 잘 와닿지 않는다.

목표 주도 개발

목표를 달성 했다고 하는 것은 프로그래밍 상에서는 테스트를 통과 했는지로 알 수가 있을 것이다.

테스트 시나리오에는 달성 해야 할 목표가(기능이) 정의 되어져 있다.

사용자 중심 개발

테스트 시나리오는 사용자 입장에서 작성되어져야 할 것이다.

테스트를 통과 한다는 것은 사용자가 원하는(목표하는) 결과인지를 확인 하는 것일 것이다.

인터페이스 중심 개발

우리가 코딩을 하다 보면 내가 만든 클래스 또는 남이 만든 클래스 등 여러 객체를 불러서 사용하게 된다.

올바르게 작동

TDD를 하는 이유는

1. 목표를 달성하는지를 확인 하는 것이고, 이는 올바르게 작동 하는 지를 확인 하는 것이다.

깔끔한 코드

2. 사용함에 있어 좀더 사용하기 좋은 코드(인터페이스 중심 관점)를 만드는 것이다.

올바르게 작동하는
깔끔한 코드

다시 말 해 목표한 대로(올바르게) 작동하는 사용하기 좋은(깔끔한 코드) 코드를 만드는 것이 TDD의 목표이다.

리팩토링
(Refactoring)

TDD는 어떻게 하는 것인가?

1 올바르게 작동하는 코드를 만든 후

2 깔끔하게 작동하는 코드로 리팩토링을 한다.

리팩토링이란? 작동하는 결과는 그대로 이면서 내부 구현을 수정 하는 것을 말함

테스트 코드
(Test Code)

올바르게 작동 한다는 것을 보장 해 주는 테스트가 필수적으로 필요 해 진다.
테스트 코드를 언제 작성 할 것인가? 여러 의견들이 있을 수 있겠지만

테스트 먼저
(Test First)

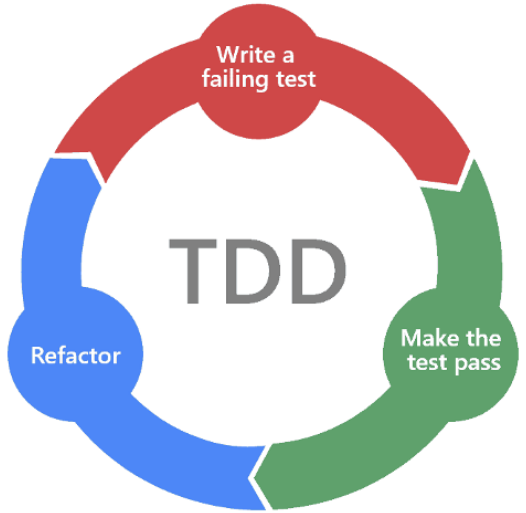
테스트 코드를 먼저 작성 하자는 의견이 테스트 주도 개발의 주요 내용이다.

테스트 주도 개발
TDD

테스트 코드를 먼저 작성하면서 개발 하자는 것이 TDD 이다.

레드
그린
리팩토링
(Red, Green, Refactoring)

TDD의 절차는 3 단계를 거치면서 수행을 하게 된다.
TDD 사이클 이라고 함.



레드
그린
리팩토링
(Red, Green, Refactoring)

1 레드 : 실패하는 테스트를 말함.

테스트를 코드를 먼저 작업을 함. 1 + 1을 하는 프로그램을 만든다고 가정을 하면

우선 1+1의 결과와 add 메소드의 리턴 값이 같은지 확인 하는 테스트 코드만 먼저 작업한다

이 테스트는 당연히 실패 할 것이다, 아직 add 메소드를 안 만든 상황이니 테스트가 실패 할 것이다.

레드
그린
리팩토링
(Red, Green, Refactoring)

2 그린 : add 메소드를 만들고 2의 결과를 리턴 하게끔 코딩해서 테스트를 통과 할 수 있을 것이다.

레드
그린
리팩토링
(Red, Green, Refactoring)

3 리팩토링 : 앞에 만든 메소드에 1, 3을 입력하면 테스트가 실패 한다는 것을 알 수 있다, 테스트가 실패 하기 전에 현재의 테스트 코드를 기준으로 add 메소드의 코드를 x, y 파라미터를 더한 결과를 리턴 하도록 리팩토링 하면 현재의 테스트 코드를 기준으로 테스트를 통과 할 수 있을 것이다.

첫번째 테스트 코드

yarn test (혹은 npm test) 명령어를 실행해서 작성한 테스트가 잘 통과하는지 확인

react-testing-library 에서 컴포넌트를 렌더링 할 때에는 render() 라는 함수를 사용합니다.

이 함수가 호출되면 그 결과물 에는 DOM 을 선택 할 수 있는 다양한 쿼리들과 container 가 포함되어있는데요, 여기서 container 는 해당 컴포넌트의 최상위 DOM 을 가르킵니다.

그리고, 그 하단의 getByText 는 쿼리함수라고 부르는데요 이 함수를 사용하면 텍스트를 사용해서 원하는 DOM 을 선택 할 수 있습니다.

스냅샷 테스팅

Snapshot Test

Snapshot test란 UI가 바뀌지 않았다는 것을 증명하는 유용한 도구이다.

snapshot 은 UI컴포넌트를 렌더링 한 후, 이를 다음 테스트와 함께 저장된 스냅샷과 비교하며 테스트가 진행된다. snapshot테스트는 두개의 스냅샷이 일치 하지 않은 경우, 실패하게 됩니다. 이 경우, 새롭게 적용한 컴포넌트의 UI가 의도치 않게 변경되었거나, 컴포넌트가 버전이 변경되었음을 시사합니다.

Profile branch Merge

- **git add .**
- **git commit -m 'Profile test ok'**
- **git switch main**
- **git merge profile**
- **git status**
- **git log**
- **git branch –d profile**

다양한 쿼리

render 함수를 실행하고 나면 그 결과물 안에는 다양한 쿼리 함수들이 있는데요, 이 쿼리 함수들은 react-testing-library 의 기반인 dom-testing-library 에서 지원하는 함수들입니다.

이 쿼리 함수들은 Variant 와 Queries 의 조합으로 네이밍이 이루어져있는데요, 우선 Varient 에는 어떤 종류들이 있는지 봅시다.

Variant

getBy

getBy* 로 시작하는 쿼리는 조건에 일치하는 DOM 엘리먼트 **하나**를 선택합니다. **만약에 없으면 에러가 발생합니다.**

getAllBy

getAllBy* 로 시작하는 쿼리는 조건에 일치하는 DOM 엘리먼트 **여러개**를 선택합니다. **만약에 하나도 없으면 에러가 발생합니다.**

queryBy

queryBy* 로 시작하는 쿼리는 조건에 일치하는 DOM 엘리먼트 **하나**를 선택합니다. 만약에 존재하지 않아도 **에러가 발생하지 않습니다.**

Variant

queryAllBy

queryAllBy* 로 시작하는 쿼리는 조건에 일치하는 DOM 엘리먼트 **여러개**를 선택합니다. 만약에 존재하지 않아도 **에러가 발생하지 않습니다.**

findBy

findBy* 로 시작하는 쿼리는 조건에 일치하는 DOM 엘리먼트 하나가 나타날 때 까지 기다렸다가 해당 DOM 을 선택하는 Promise 를 반환합니다. 기본 timeout 인 4500ms 이후에도 나타나지 않으면 **에러가 발생합니다.**

findAllBy

findBy* 로 시작하는 쿼리는 조건에 일치하는 DOM 엘리먼트 여러개가 나타날 때 까지 기다렸다가 해당 DOM 을 선택하는 Promise 를 반환합니다. 기본 timeout 인 4500ms 이후에도 나타나지 않으면 **에러가 발생합니다.**

Queries

ByLabelText

ByLabelText 는 label 이 있는 input 의 label 내용으로 input 을 선택합니다.

```
<label for="username-input">아이디</label>
<input id="username-input" />

const inputNode = getByLabelText('아이디');
```

Queries

ByPlaceholderText

ByPlaceholderText 는 placeholder 값으로 input 및 textarea 를 선택합니다.

```
<input placeholder="아이디" />;
```

```
const inputNode = getByPlaceholderText('아이디');
```

Queries

ByText
ByText는 엘리먼트가 가지고 있는 텍스트 값으로 DOM 을 선택합니다.

Hello World! ;

const div = getByText('Hello World!');
참고로, 텍스트 값에 정규식을 넣어도 작동합니다.

const div = getByText(/^Hello/);

Queries

ByAltText

ByAltText 는 alt 속성을 가지고 있는 엘리먼트 (주로 img) 를 선택합니다.

```
<img src="/awesome.png" alt="awesome image" />;
```

```
const imgAwesome = getByAltText('awesomse image');
```

Queries

ByTitle

ByTitle 은 title 속성을 가지고 있는 DOM 혹은 title 엘리먼트를 지니고있는 SVG 를 선택 할 때 사용합니다.

title 속성은 html 에서 툴팁을 보여줘야 하는 상황에 사용하곤 합니다.

```
<p>
  <span title="React">리액트</span>는 짱 멋진 라이브러리다.
</p>

<svg>
  <title>Delete</title>
  <g><path/></g>
</svg>

const spanReact = getByTitle('React');
const svgDelete = getByTitle('Delete');
```

Queries

ByDisplayValue

ByDisplayValue 는 input, textarea, select 가 지니고 있는 현재 값을 가지고 엘리먼트를 선택합니다.

```
<input value="text" />;
```

```
const input = getByDisplayValue('text');
```

Queries

ByRole

ByRole은 특정 role 값을 지니고 있는 엘리먼트를 선택합니다.

```
<span role="button">삭제</span>;
```

```
const spanRemove = getByRole('button');
```

Queries

ByTestId

ByTestId 는 다른 방법으로 못 선택할때 사용하는 방법인데요, 특정 DOM 에 직접 test 할 때 사용할 id 를 달아서 선택하는 것을 의미합니다.

흔한 div ;

const commonDiv = getByTestId('commondiv');

!> 주의: camelCase 가 아닙니다. 값을 설정할때 data-testid="..." 이렇게 설정하셔야합니다. 추가적으로, ByTestId 는 다른 방법으로 선택할 수 없을때에만 사용해야합니다.

어떤 쿼리를 사용해야 할까?

쿼리의 종류가 정말 많죠? 그렇다면, 어떤 쿼리를 우선적으로 사용해야 할까요? 매뉴얼 에서는 다음 우선순위를 따라서 사용하는것을 권장하고있습니다.

1 **getByLabelText**
2 **getByPlaceholderText**
3 **getByText**
4 getByDisplayValue
5 getByAltText
6 getByTitle
7 getByRole
8 getByTestId

어떤 쿼리를 사용해야 할까?

그리고, DOM 의 querySelector 를 사용 할 수도 있는데요, 이는 지양해야합니다.

차라리 data-testid 를 설정하는것이 좋습니다.

```
const utils = render(<MyComponent />);
const element = utils.container.querySelector('.my-class');
```