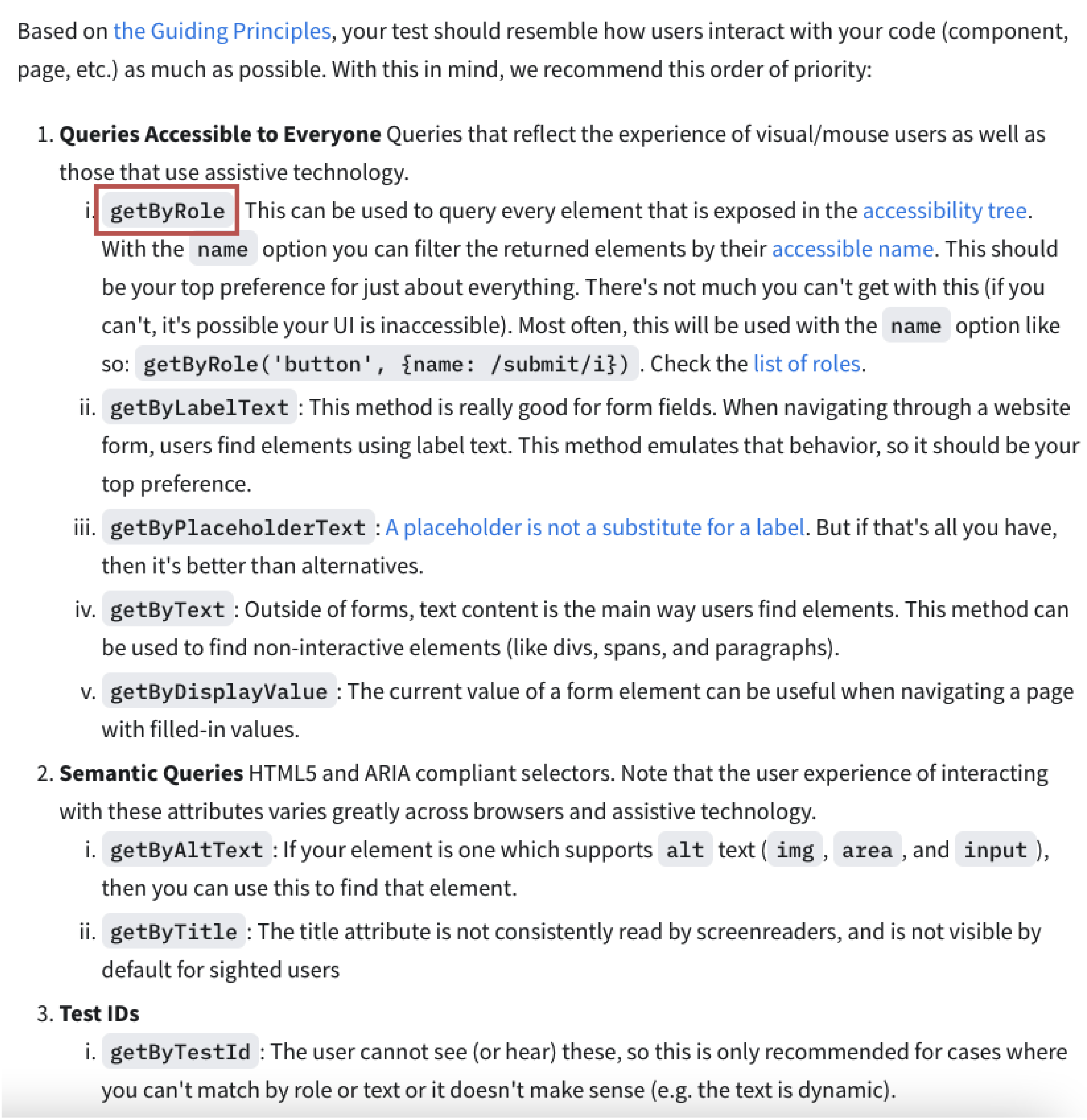
Query 사용 우선 순위

참고 문서 → https://testing-library.com/docs/queries/about/#priority

screen.getByTestId()

현재까지는 getByTestId 쿼리를 이용해서 엘레멘트에 접근해서 테스트를 진행했습니다. 이 방법이 편리하긴 하지만 testing library에서 추천하는 쿼리 사용 우선순위가 있기 때문에 한번 보고 가겠습니다.



Based on the Guiding Principles, your test should resemble how users interact with your code (component, page, etc.) as much as possible. With this in mind, we recommend this order of priority:

1. **Queries Accessible to Everyone** Queries that reflect the experience of visual/mouse users as well as those that use assistive technology.
   i. `getByRole` This can be used to query every element that is exposed in the accessibility tree. With the `name` option you can filter the returned elements by their accessible name. This should be your top preference for just about everything. There's not much you can't get with this (if you can't, it's possible your UI is inaccessible). Most often, this will be used with the `name` option like so: `getByRole('button', {name: /submit/i})`. Check the list of roles.
   ii. `getByLabelText`: This method is really good for form fields. When navigating through a website form, users find elements using label text. This method emulates that behavior, so it should be your top preference.
   iii. `getByPlaceholderText`: A placeholder is not a substitute for a label. But if that's all you have, then it's better than alternatives.
   iv. `getByText`: Outside of forms, text content is the main way users find elements. This method can be used to find non-interactive elements (like divs, spans, and paragraphs).
   v. `getByDisplayValue`: The current value of a form element can be useful when navigating a page with filled-in values.
2. **Semantic Queries** HTML5 and ARIA compliant selectors. Note that the user experience of interacting with these attributes varies greatly across browsers and assistive technology.
   i. `getByAltText`: If your element is one which supports `alt` text (`img`, `area`, and `input`), then you can use this to find that element.
   ii. `getByTitle`: The title attribute is not consistently read by screenreaders, and is not visible by default for sighted users
3. **Test IDs**
   i. `getByTestId`: The user cannot see (or hear) these, so this is only recommended for cases where you can't match by role or text or it doesn't make sense (e.g. the text is dynamic).

userEvent > fireEvent

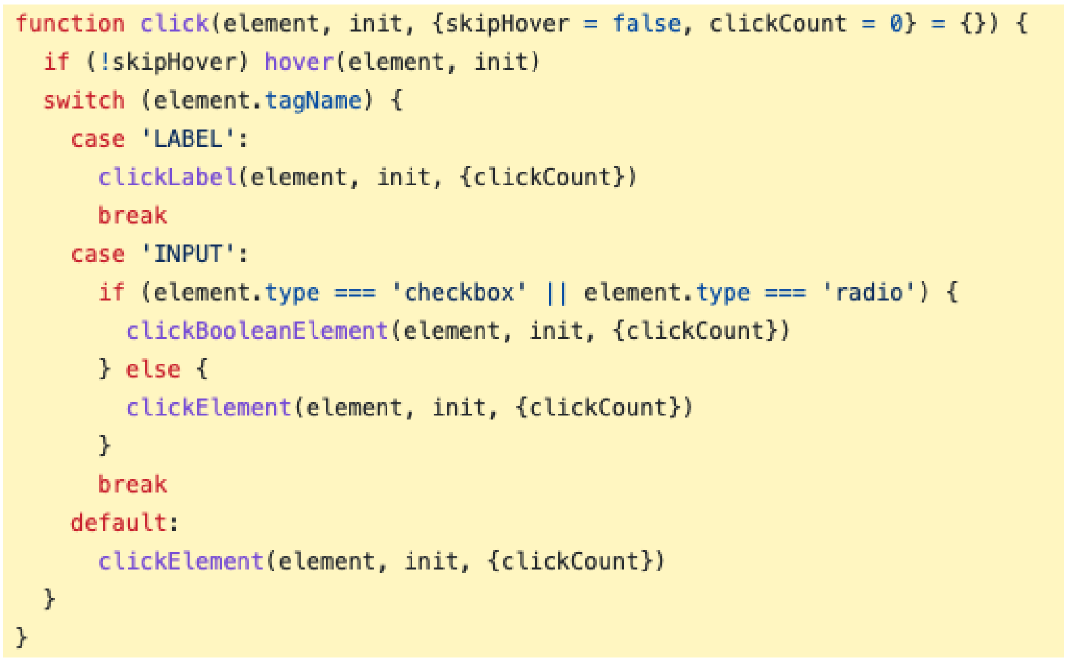
이전에 테스팅에서 버튼을 클릭했을 때 fireEvent API를 사용했습니다. 이때 fireEvent를 사용해서 잘 처리를 해줬지만 userEvent API를 사용하는 게 더 좋은 방법입니다.   fireEvent.click(element)   <

userEvent

userEvent는 fireEvent 를 사용해서 만들어졌습니다. userEvent의 내부 코드를 보면 fireEvent를 사용하면서 엘리먼트의 타입에 따라서 Label을 클릭했을 때, checkbox, radio 을 클릭했을 때 그 엘리먼트 타입에 맞는 더욱 적절한 반응을 보여줍니다.

예를 들어서 fireEvent 로 버튼을 클릭하면 fireEvent.click(button) 버튼이 focus 되지 않습니다. 하지만 userEvent로 클릭하면 userEvent.click(button) 버튼이 focus 가 됩니다. 이렇게 실제 사용하는 유저가 보기에 실제 버튼을 클릭하는 행위가 더 잘 표현되기에 userEvent를 사용하는 게 더 추천되는 방법입니다.

https://github.com/testing-library/user-event/blob/5feaa942f46bb37d96c2f2fbeb4b33e8beff75ad/src/click.js#L87-L103



```
function click(element, init, {skipHover = false, clickCount = 0} = {}) {
  if (!skipHover) hover(element, init)
  switch (element.tagName) {
    case 'LABEL':
      clickLabel(element, init, {clickCount})
      break
    case 'INPUT':
      if (element.type === 'checkbox' || element.type === 'radio') {
        clickBooleanElement(element, init, {clickCount})
      } else {
        clickElement(element, init, {clickCount})
      }
      break
    default:
      clickElement(element, init, {clickCount})
  }
}
```