



ADVANCED TESTING CONCEPTS

Another CodingWithCallum™ Session

WHAT ARE WE GOING TO TALK ABOUT?

- Async rendering in React Testing Library (RTL)
 - wtf is act
- Ordering queries in RTL (a11y, eg. getByRole > getByDataId)

WHAT.ANZ.COM

Abbrevs:

- Async = Asynchronous
- RTL = React Testing Library
- MSW = Mock Service Worker
- Abbrev = Abbreviation

ASYNCRENDERING IN REACT TESTING LIBRARY

Wtf is this act warning and why do i see it all the time?

When testing, code that causes React `state` updates should be wrapped in `act()`

```
act(() => {  
  /* fire events that update state */  
});  
/* assert on the output */
```

This ensures that you're testing the behavior the **user** would s

BUT WHY?

React is trying to warn us something happened when we expected nothing to happen at all

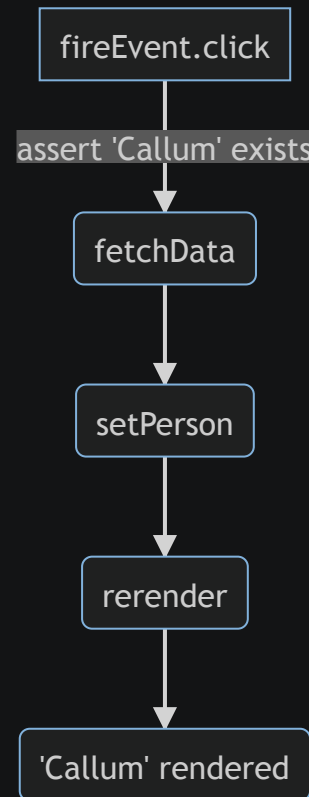
This can happen, before, during or after (most likely) the test

EXAMPLE 1: ASYNC UPDATES

```
const ShowName = () => {
  const [person, setPerson] = useState();
  const handleFetch = useCallback(async () => {
    const { data } = await fetchData(); // Returns { name: "Ca
    setPerson(data.person); // <- Asynchronous update
  }, []);
  return (
    <button type="button" onClick="handleFetch">
      {person ? person.name : "Fetch"}
    </button>
  );
};
```

```
it("should fetch persons name", () => {
  render(<ShowName ></ShowName>);
  fireEvent.click(screen.getByText("Fetch"));
  expect(screen.getByText("Callum")).toBeInTheDocument();
});
```

EXAMPLE 1: SEQUENCE



EXAMPLE 1: SOLUTION

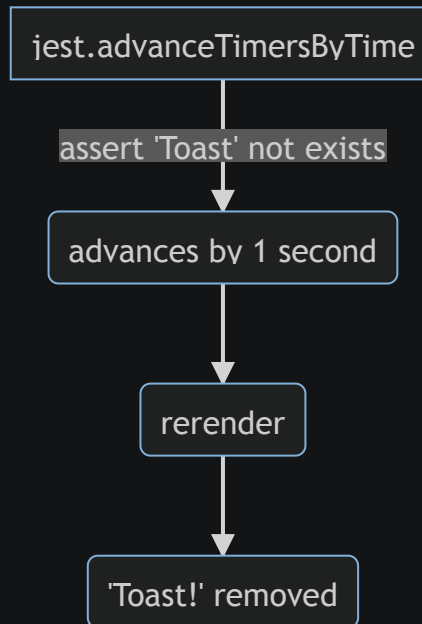
```
it("should fetch persons name", async () => {  
  render(<ShowName ></ShowName>);  
  fireEvent.click(getByText("Fetch"));  
  expect(screen.getByText("Fetch names")).toBeInTheDocument();  
  expect(await screen.findByText("Callum")).toBeInTheDocument(  
  });
```

EXAMPLE 2: TIMERS

```
const Toast = () => {  
  const [isVisible, setIsVisible] = useState(true);  
  useEffect(() => {  
    setTimeout(() => { setIsVisible(false); }, 1000); // hide  
  }, []);  
  return isVisible && <div>Toast!</div>;  
};
```

```
it("should display Toast for 1 second", () => {  
  jest.useFakeTimers();  
  render(<Toast ></Toast>);  
  jest.advanceTimersByTime(1000);  
  expect(screen.queryByText("Toast!")).not.toBeInTheDocument()  
});
```

EXAMPLE 2: SEQUENCE



EXAMPLE 2: SOLUTION

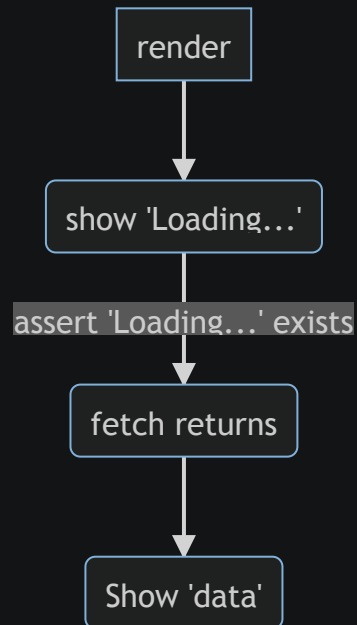
```
it("should display Toast for 1 second", () => {  
  jest.useFakeTimers();  
  render(<Toast ></Toast>);  
  act(() => { // Actually wrap it in an act 😅  
    jest.advanceTimersByTime(1000);  
  });  
  expect(screen.queryByText("Toast!")).not.toBeInTheDocument()  
});
```

EXAMPLE 3: PREMATURE EXIT

```
const ShowName = () => {  
  const { loading, data } = await fetchData(); // Returns { na  
  return loading && data?.name ? (  
    <div>Loading...</div>  
  ) : (  
    <div>{data.name}</div>  
  );  
};
```

```
it("should display loading state", () => {  
  const render(<ShowName ></ShowName>);  
  expect(screen.getByText("Loading ...")).toBeInTheDocument();  
});
```

EXAMPLE 3: SEQUENCE



EXAMPLE 3: SOLUTION

```
it("should display loading state", () => {  
  const render(<ShowName ></ShowName>);  
  expect(getByText("Loading ...")).toBeInTheDocument();  
  await waitFor(() => {  
    expect(queryByText("Loading ...")).not.toBeInTheDocument();  
  });  
});
```


even better...

```
it("should display loading state", () => {  
  const render(<ShowName ></ShowName>);  
  expect(getByText("Loading ...")).toBeInTheDocument();  
  await waitForElementToBeRemoved(() => queryByText("Loading .  
});
```

PHEW 🥵

That was fun, questions?

ORDERING OF QUERIES IN RTL

- What should I use in what circumstance?
- How do I remember all these queries?
- Whats the difference?
- Water break 

TYPES OF QUERIES

At high level there's 3 types

returns match but fails if none found

```
ByText( "Callum" )
```

returns match but `null` if none found

```
ect( queryByText( "Callum" ) .not.toBeInTheD
```

returns promise that resolves with match found or errors on

```
it findByText( "Callum" )
```

SINGLE VS MULTIPLE

- Single (by)
 - getBy
 - queryBy
 - findBy
- Multiple (allBy)
 - getAllBy
 - queryAllBy
 - findAllBy

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	Return	Return	No

Query Type	0 Match	1 Match	>1 Match	Async/Await
	error	array	array	
queryAllBy...	Return []	Return array	Return array	No
findAllBy...	Throw error	Return array	Return array	Yes

ORDER OF QUERYING

1. Focus on accessibility (reflect the experience of visual/assistive users)
2. Semantic Queries (HTML5 and ARIA compliant selectors)
3. Test IDs (last resort, user cannot see or hear these)

- `getByRole` - can be used to query every element that is exposed in the [accessibility tree](#)
 - name option can filter the returned elements by their [accessible name](#)
 - Check the [list of accessible roles](#)
 - eg. `getByRole('button', {name: /submit/i})`
- `getByLabelText`
 - Navigating form fields
- `getByText`
 - Outside of forms, text content is the main way users find elements
- `getByDisplayValue`

getDisplayValue

- Navigate a page with filled in values

2. SEMANTIC QUERIES

- `getByAltText` - If your element is one which supports alt text (img, area, input, and any custom element), then you can use this to find that element.
- `getByTitle` - The title attribute is not consistently read by screenreaders, and is not visible by default for sighted users

3. TEST IDS

- `getByIdTestId`: 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).

FINALLY

- use `await findByRole()` to avoid rendering problems and be as accessible as possible
- check our console, act errors are painful and cause

FURTHER READING

- [Fix the "not wrapped in act\(...\)" warning](#)
- [React testing library - queries](#)
- [The accessibility tree](#)
- [Accessible names](#)
- [List of accessible roles](#)

