

the Master Course

React Testing Library

{CODENATION}

Learning Objectives

Describe why testing is important, and the benefits of it

Get started with the react-testing-library

Write basic tests for a react component

Why test?

"Please don't break my code"

Have a quick think / brainstorm on what the advantages there could be in testing

react-testing-library

Benefits of Testing

- Confidence in your code
- Catches potential bugs
- Speeds up QA time
- Can also be documentation

react-testing-library

What have we seen before?

We are going to be using jest again

This means that you will recognise some of the tools you have use previously

Jest is our test runner, meaning we use it for creating, running and structuring of our tests



```
1  const greet = (name) => {  
2    return `hello ${name}`  
3  }  
4
```

react-testing-library

What have we seen before?

```
1  const greet = (name) => {  
2    return `hello ${name}`  
3  }  
4
```



From the naming of our files (App.test.js)

To our matchers (toBe, toEqual etc)

```
1  const functions = require('./functions')  
2  
3  test('greet function returns hello', () => {  
4    const greet = functions.greet  
5    expect(greet('leon')).toEqual('hello leon')  
6  })
```

Even the test block is similar

PASS

react-testing-library

What have we seen before?



We just have to combine it with React Testing Library to simulate the user experience

```
1  const functions = require('./functions')
2
3  test('greet function returns hello', () => {
4    const greet = functions.greet
5    expect(greet('leon')).toEqual('hello leon')
6  })
```

```
1  import { render, screen } from '@testing-library/react';
2  import App from './App';
3
4  test('renders learn react link', () => {
5    render(<App />);
6    const linkElement = screen.getByText(/learn react/i);
7    expect(linkElement).toBeInTheDocument();
8  });
```

PASS

react-testing-library

Tends to be 3 type of testing

- **Unit**
- **Intergration**
- **End-to-End**

react-testing-library

Unit

Tests a piece of code in isolation

**Can focus on a component,
making sure it renders correct and
uses any props it could receive**

react-testing-library

Integration

**Interacting with multiple components,
how they are working together and
are working properly together**

react-testing-library

End-2-End

Real time procedure on how the app behaves from one point to the end (but we won't be looking at that)

react-testing-library

Lets create a new project

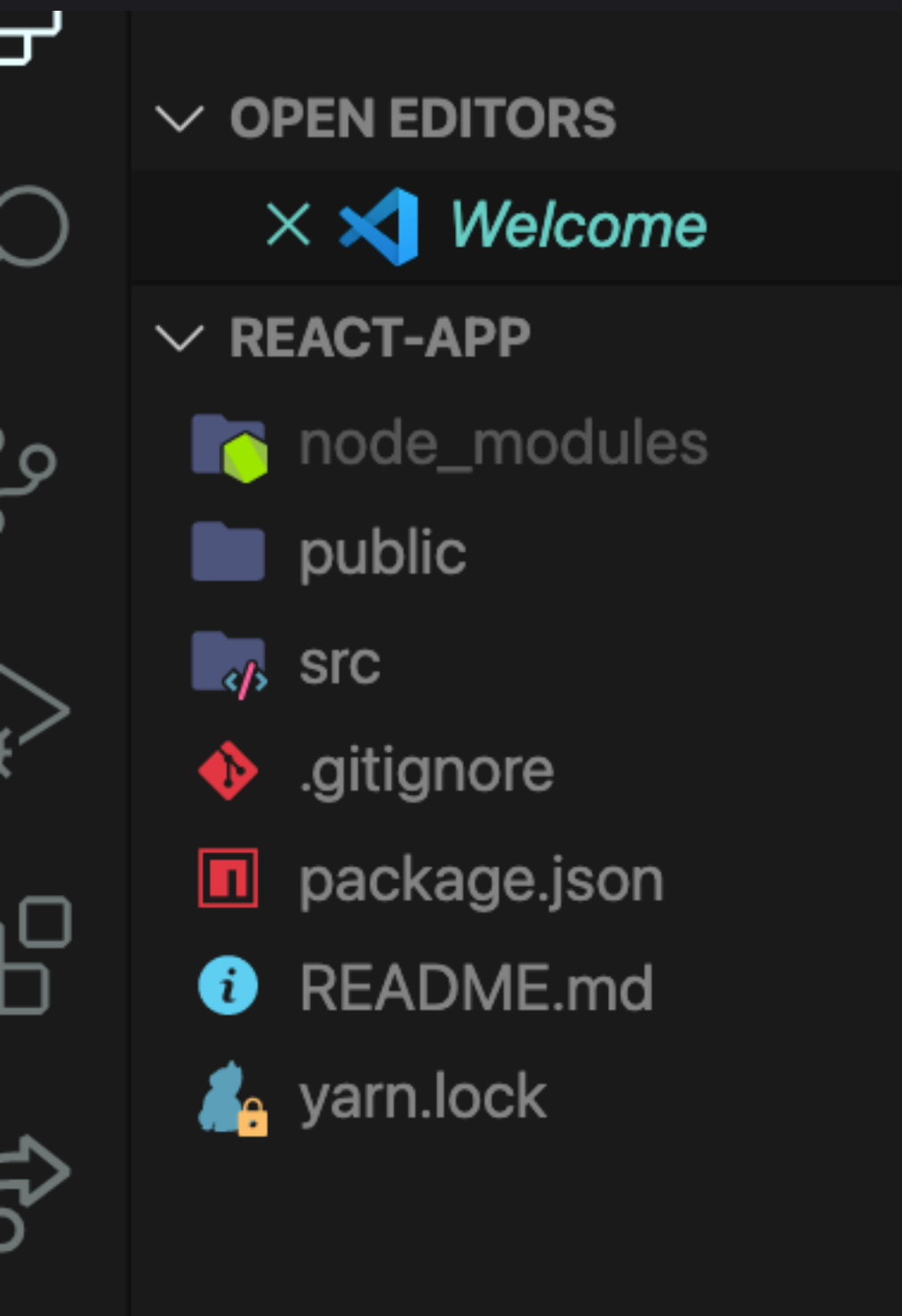
npx create-react-app app-name

So this is where we're going to start.



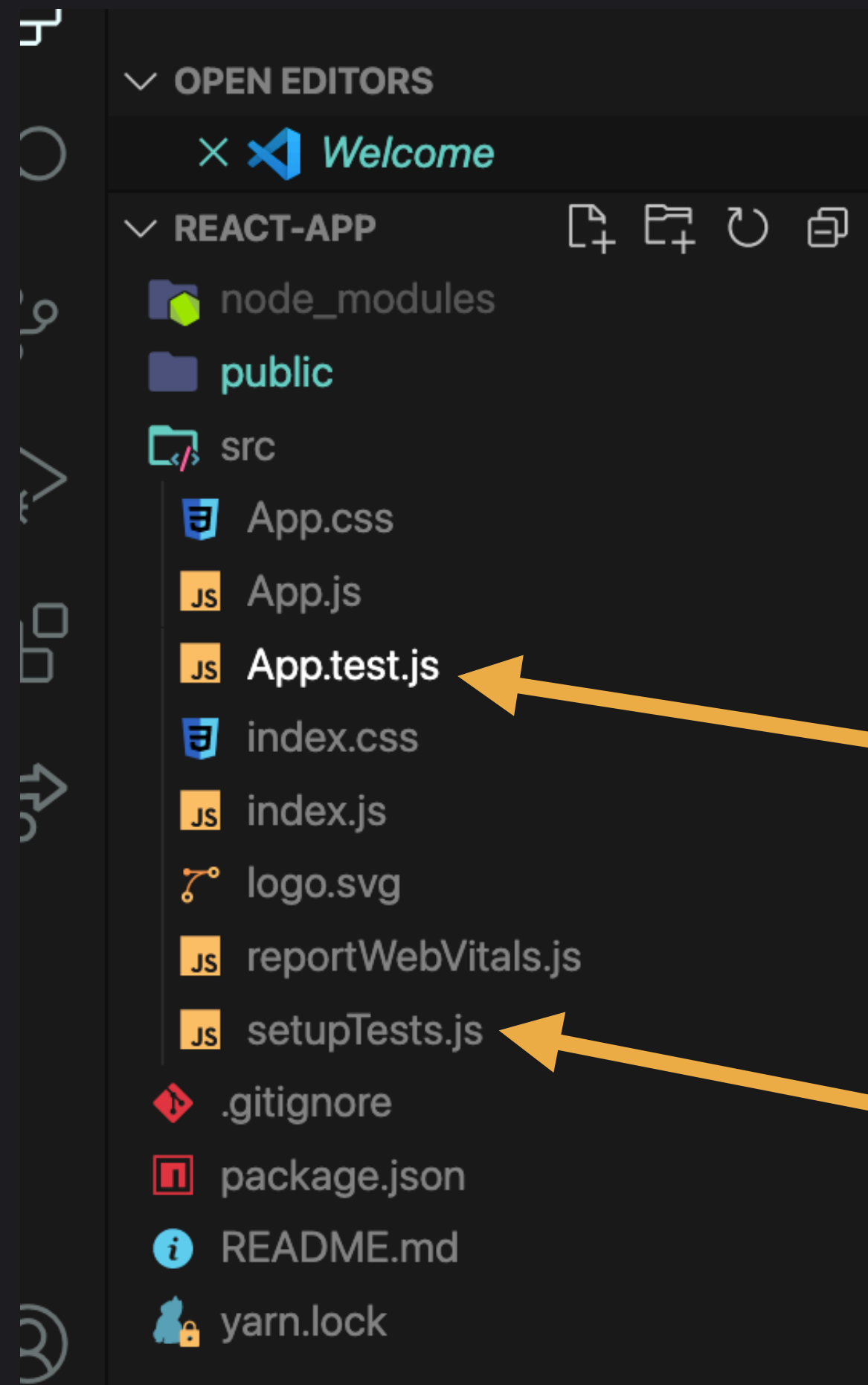
react-testing-library

Before we “clean up” lets have a look in our explorer



react-testing-library

With CRA it actually already gives us the tools and files to get testing, you have probably seen them before right?



Test file

Importing something to help us test





react-testing-library

setupTests.js

With this we are importing the testing library jest, you've used it before

makes testing for us a lot easier



react-testing-library

App.test.js

This is a test file for App.js

**Lets just check a couple of things before we
dive into there... to package.json**



react-testing-library

package.json

```
"dependencies": {  
  "@testing-library/jest-dom": "^5.11.4",  
  "@testing-library/react": "^11.1.0",  
  "@testing-library/user-event": "^12.1.10",  
  "react": "^17.0.2",  
  "react-dom": "^17.0.2",  
  "react-scripts": "4.0.3",  
  "web-vitals": "^1.0.1"  
},
```

react-testing-library

**The library is already installed
with CRA**

**If we didn't use CRA then we
would have to have installed it
using npm**



react-testing-library

package.json

```
"scripts": {  
  "start": "react-scripts start",  
  "build": "react-scripts build",  
  "test": "react-scripts test",  
  "eject": "react-scripts eject"  
},
```

These are our scripts

We use "npm start" to run our app.

We run our tests with "npm run test"

Lets run test

react-testing-library

We did it...didn't we?

Well sort of, we ran our test script, it found a test file and that test, luckily for us, **PASSED!**

RTL catches the file name and runs that test file

Test file naming convention

fileName.test.js

App.test.js



```
PASS src/App.test.js
  ✓ renders learn react link (39 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        2.248 s, estimated 3 s
Ran all test suites related to changed files.

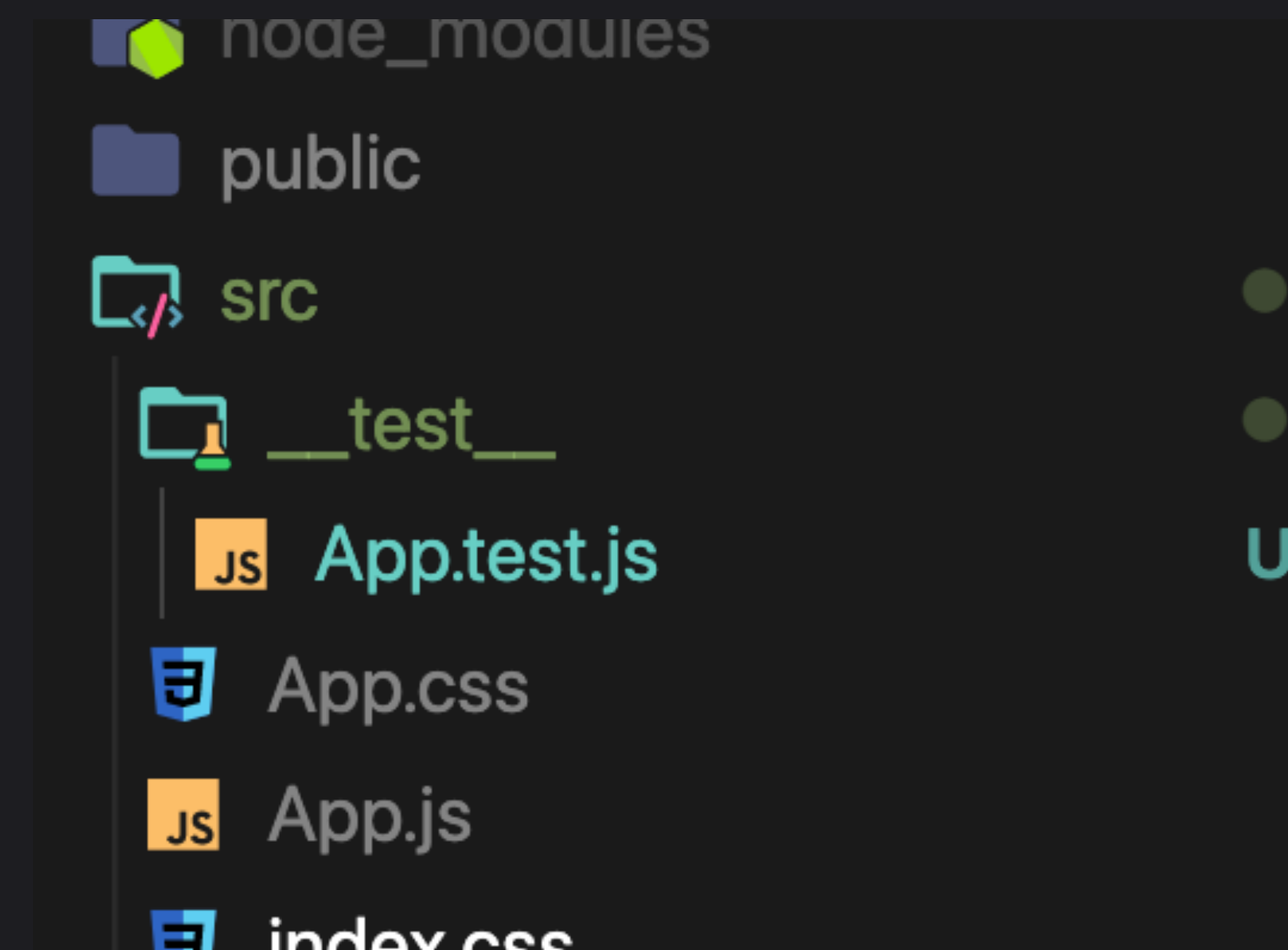
Watch Usage
  > Press a to run all tests.
  > Press f to run only failed tests.
  > Press q to quit watch mode.
  > Press p to filter by a filename regex pattern.
  > Press t to filter by a test name regex pattern.
  > Press Enter to trigger a test run.
```

react-testing-library

__test__


We can also put our tests inside folders named `__test__`, RTL will still find them

It helps us out and makes it look neat and tidy



react-testing-library

App.test.js

```
src >  App.test.js > ...
1  import { render, screen } from '@testing-library/react';
2  import App from './App';
3
4  test('renders learn react link', () => {
5    render(<App />);
6    const linkElement = screen.getByText(/learn react/i);
7    expect(linkElement).toBeInTheDocument();
8  });
```

**So what have we got,
and how is it passing?**

What's going on here?

react-testing-library

This is a test block, the structure of a test

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

test("description of the test" , = () => {

render the component we want to test

find the element we want to interact with

interact with the element / fire events

assert the result that is expected

)}

react-testing-library

This is a test block, the structure of a test

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

test("description of the test" , = () => {

render the component we want to test

find the element we want to interact with

interact with the element / fire events

assert the result that is expected

)}

The component we want to test

We are using the render method from the testing library

(Don't forget to import the component)

Tells our test block what component we are testing

```
test("description of the test" , = () => {
```

render the component we want to test

find the element we want to interact with

interact with the element / fire events

assert the result that is expected

```
})
```


Find the element

```
test("description of the test" , = () => {
```

render the component we want to test

find the element we want to interact with

interact with the element / fire events

assert the result that is expected

```
})
```

There is a lot of elements in App.js.

We declare a new variable and we use screen to look at the DOM and use a method to get a certain element

(We'll look at these methods later)

Interaction

```
test("description of the test" , = () => {
```

render the component we want to test

find the element we want to interact with

interact with the element / fire events

assert the result that is expected

```
})
```

Depending on the element, here we would cause our events.

Click, type, change...

(Not in our test though, we will fire some later)

Expectation / Assertion

```
test("description of the test" , = () => {
```

render the component we want to test

find the element we want to interact with

interact with the element / fire events

assert the result that is expected

```
})
```

Expecting the element that we have chosen, to...

The first test expects that element toBeInTheDocument()

(other methods to expect different results)

react-testing-library

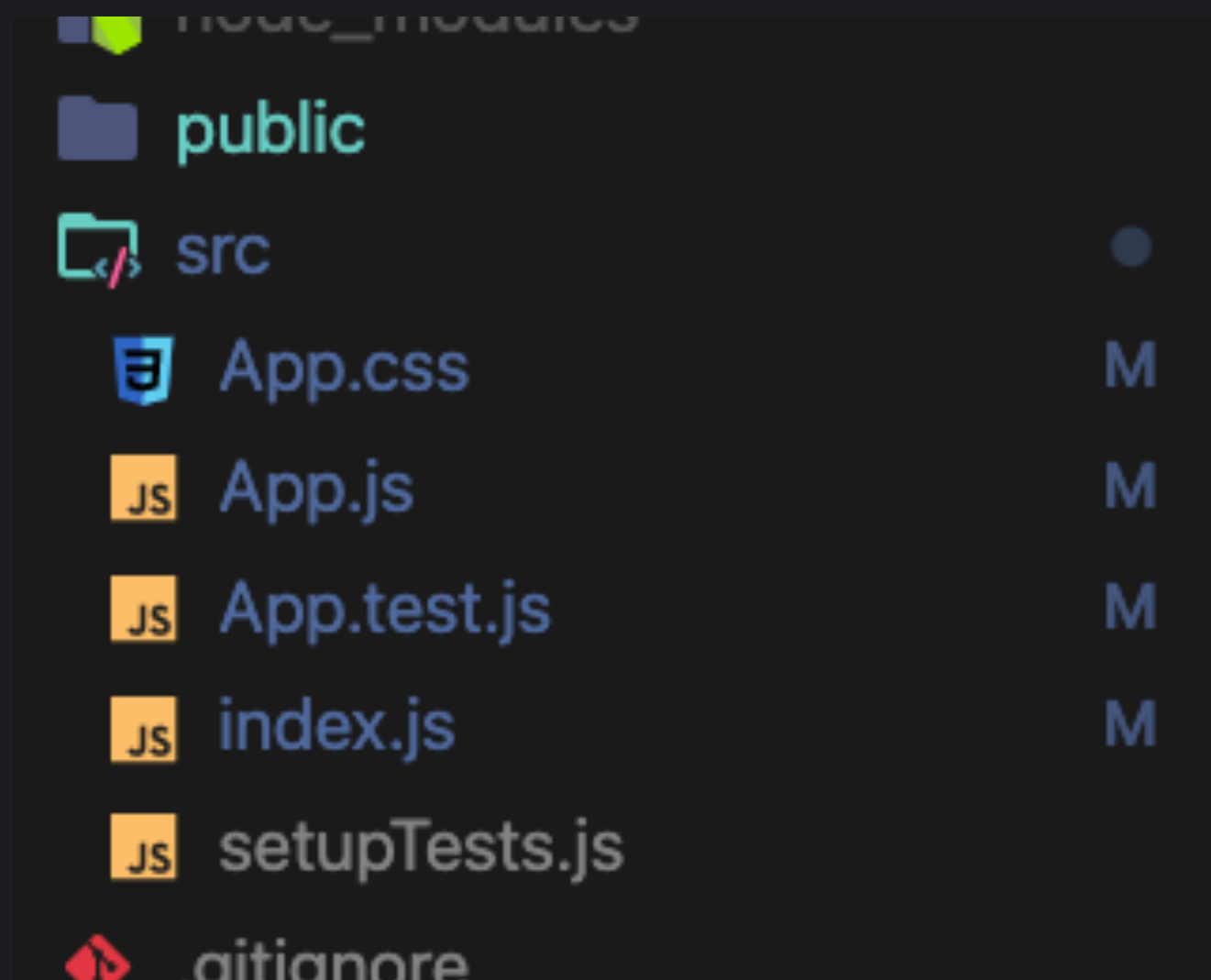
We won't be keeping some of the files made with CRA

Lets have a cleanup and delete some of the files (Deleted logo, web vitals etc, as well as change my App.js to return just a header)



react-testing-library

Now looks something like this



```
1 import './App.css';
2
3 const App = () => {
4   return <h1>react app</h1>
5 }
6
7 export default App
```

react-testing-library

But wait...

```
FAIL src/App.test.js
  ✕ renders learn react link (24 ms)

  ● renders learn react link

    TestingLibraryElementError: Unable
    to find an element with the text:
    learn react link. This could be
    because the text is broken up by multiple
    elements. Try using a matcher that is
    more flexible.

    <body>
      <div>
```

Our test has now failed...sigh

Well of course it has, that element is no longer in the document!

In fact, lets go ahead and make our first test



App.test.js

react-testing-library

```
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);
  // interaction
  expect(linkElement).toBeInTheDocument();
});
```

**What do we
need to change?**



react-testing-library

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

We have the correct component

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

**But our element doesn't exist,
and we expect it to be in the
document...spoiler...it definitely
isn't in the document**

react-testing-library

```
const App = () => {  
  return <h1>react app</h1>  
}
```

```
export default App
```

In our App.js we do have an element, but we need to find it in our test

Lets look at how we can find that element



react-testing-library

Query Methods

This is what we use to get the element in a component

There are several we could use ("get", "find", "query")

Its up to you to decide on the query that is most appropriate

Take a quick look, see what you have

react-testing-library

Query Methods

Theres a lot isn't there...

findByText vs findAllByText

get vs query...vs find... 🙄

react-testing-library

Query Methods

	getBy	findBy	queryBy	getAllBy	findAllBy	queryAllBy
No Match	error	error	null	error	error	array
1 Match	return	return	return	array	array	array
1+ Match	error	error	error	array	array	array
Await	no	yes	no	no	yes	no

Query Methods

They also have some attributes

`getBy`**AltText**

`getBy`**Text**

`getBy`**TestId**

We have to think like the user to decide which to use

Query Methods

Top Priority

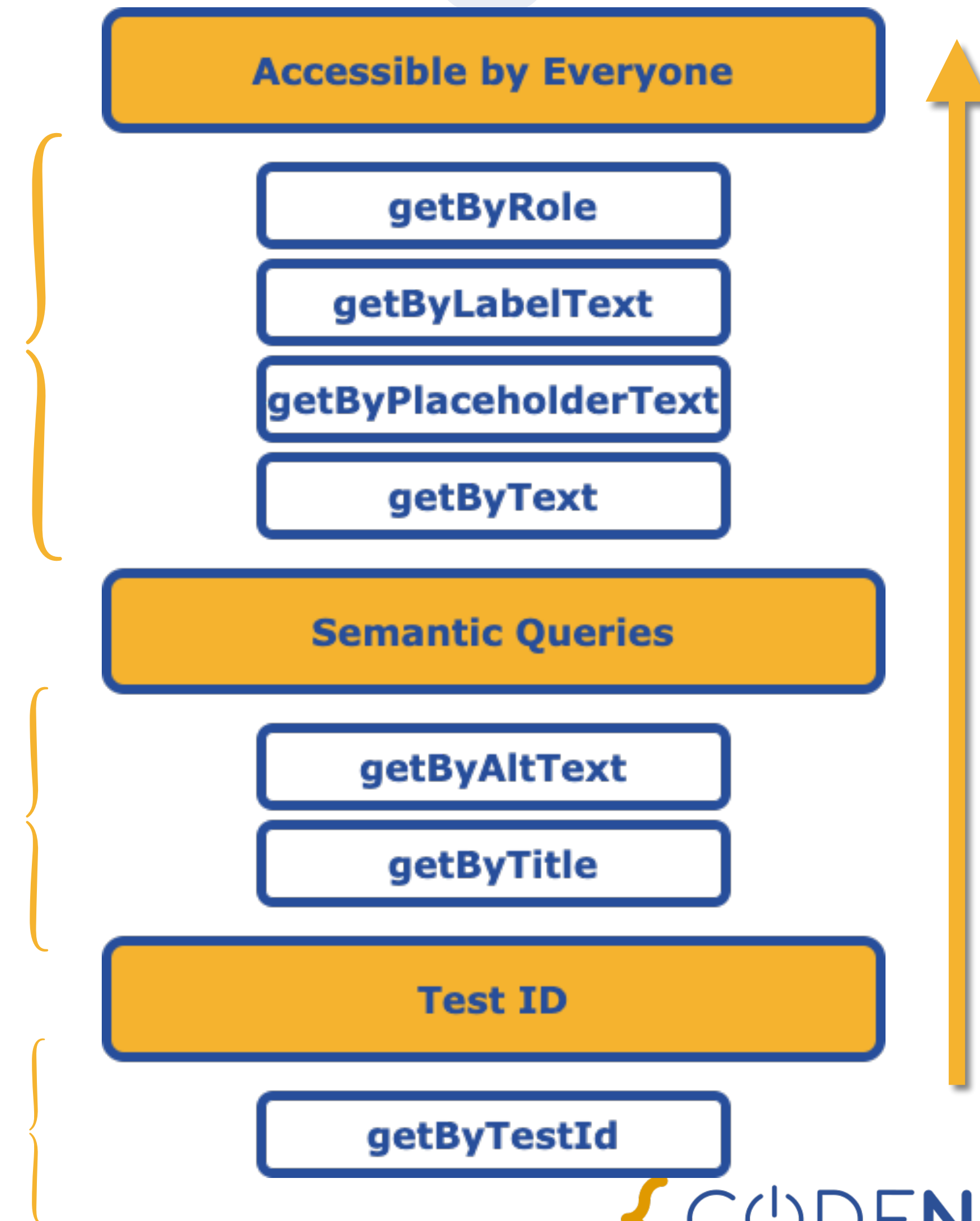
Mimicking users directly

Second Priority

Something our user won't be seeing but can be read by screen readers

Last Priority

Users will not see and neither will screen readers, last priority, last resort



react-testing-library

Lets look back a this and make a test

```
const App = () => {  
  return <h1>react app</h1>  
}  
  
export default App
```

**Go through some of those query methods
and lets make sure that it is there and in
the document**

For now lets keep
`expect(element).toBeInTheDocument();`



react-testing-library

Don't forget about Test Driven Development

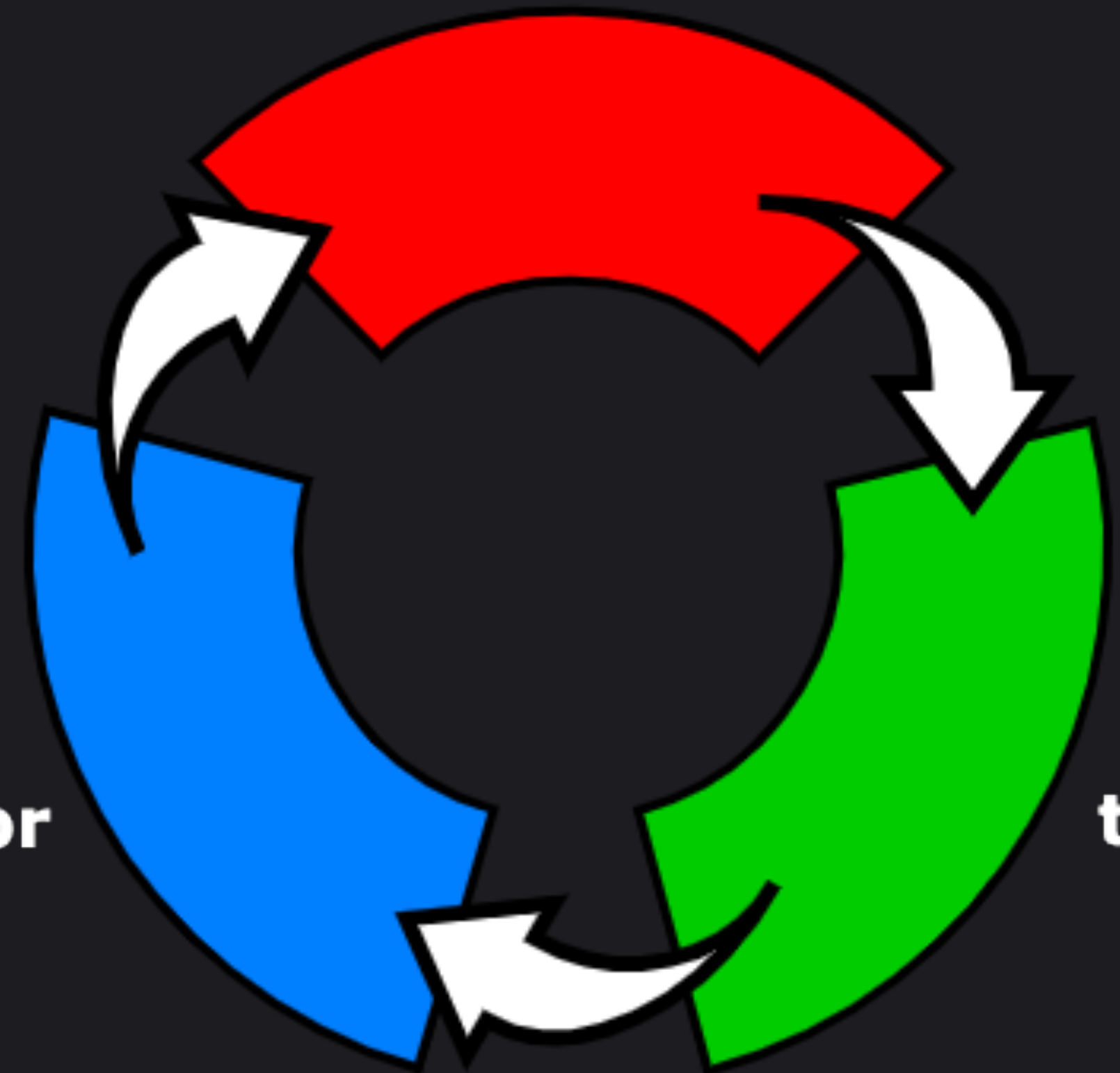
Make a failed test first



Refactor

**Write a
failed test**

**Make
the test
pass**



react-testing-library

```
test('renders the h1 element', () => {  
  render(<App />);  
  const headElement = screen.getByRole("paragraph");  
  expect(headElement).toBeInTheDocument();  
});
```

A failed attempt

“Unable to find element with the role paragraph”

App.js

react-testing-library

```
test('renders the h1 element', () => {  
  render(<App />);  
  const headElement = screen.getByRole("heading");  
  expect(headElement).toBeInTheDocument();  
});
```

PASS



react-testing-library

Extra

We may have a fair few tests, so we could group them together in a describe block (a place to put multiple tests in)



Extra

react-testing-library

Instead of having this...

```
test('finds the h1 element', () => {
  render(<App />);
  const headElement = screen.getByRole("heading");
  expect(headElement).toBeInTheDocument();
});

test('find the h1 element by text', () => {
  render(<App />);
  const headElement = screen.getByText(/react app/i);
  expect(headElement).toBeInTheDocument();
})
```

We can have this instead...

```
describe("appjs tests", () => {
  test("finds the h1 element", () => {
    render(<App />);
    const headElement = screen.getByRole("heading");
    expect(headElement).toBeInTheDocument();
  });

  test("find the h1 element by text", () => {
    render(<App />);
    const headElement = screen.getByText(/react app/i);
    expect(headElement).toBeInTheDocument();
  });
});
```



This can help keep out test organised, you can even have a describe block within a describe block, splitting your tests even more

Revisiting Learning Objectives

Describe why testing is important, and the benefits of it

Get started with the react-testing-library

Write basic tests for a react component