

```
1  * RTL vs Jest (Testing Library).....
2
3  1) Jest
4
5      * Testing Framework: Jest is a JavaScript testing framework developed by Facebook.
6        It's designed to be a complete solution for testing JavaScript code, including React applications.
7
8      * General Focuses on JavaScript testing....
9
10     * Emphasis on functional and unit tests....
11
12     * Built-in mocking capabilities...
13
14     * Provides a wide range of utilities for testing JavaScript code
15
16
17  2) RTL(React Testing Library):-
18
19     * Testing Utilities: RTL, on the other hand, is not a testing framework but a set of utilities for testing React components.
20       It encourages testing components in a way that closely resembles how users interact with the application.
21
22     * Focuses on React Component testing....
23
24     * Emphasis on user-centric and integration tests...
25
26     * Supports manual mocking; can be used with Jest....
27
28     * Focuses more on component interaction and behavior testing....
29
30     * Provides utilities specifically designed for testing React components...
31
32     * Encourages testing based on user behavior and interactions....
33
34
35     Note :- Jest provides an excellent testing framework and can be enhanced with RTL to improve the testing of React components,
36             especially when focusing on user interactions and behaviors.
37
38
39
40  * React Testing
41
42  * Package.json
43
44      "@testing-library/jest-dom": "^5.17.0",
45      "@testing-library/react": "^13.4.0",
46      "@testing-library/user-event": "^13.5.0",
47
48  Note :- This Above all Library Covers Unit and Integration Test Cases....
49
50
```

```
51
52 * To Test a Particular File....
53
54 ---> npm test src/App.test.js
55
56
57
58 //Syntax for a Function...
59
60 #1
61 test("testing for sum function", ()=>{
62   expect(sum(10,20)).toBe(30);
63 });
64
65
66 #2
67 test("testing for Sum Function With Different method", ()=> {
68
69   let a=10;
70   let b=20;
71   let output=30
72   expect(sum(a,b)).toBe(output)
73 });
74
75
76
77 * React Testing Structure....
78
79   * What needs to import....
80   * How to render the component....
81   * How tests working with RTL....
82   * How test finding UI elements....
83
84
85
86 * Important Things to import in App.test.js
87
88   * import {render , screen} from '@testing-library/react';
89
90   * render :- It is used to render a Components....
91   * screen :-
92
93   * Can we Test Multiple Components in a Single File...?
94   => We Can Test Multiple Componnets in a Single Test File...
95
96
97 * Writing First React Test
98
99   * Make New Test function...
100  * Write test case for check text on screen...
```

```

101      * Write test for case-insensitive text...
102      * Test title for image...
103      * Write multiple expect in the same test function....
104
105
106  * Test Input Box
107
108      * Make Input box in App Component
109      * Write Test case Function
110      * The test Input box is present or not.
111
112      * Test input box.
113
114          * name ,
115          * Placeholder,
116          * Id,
117          * Value,
118          * Type,
119
120
121  * Test Case Run Options....
122
123      * How to run specific test files...
124      * What is watch Mode...? -----> (IMP)
125      * How to run the failed test case.... -----> (IMP)
126      * How to run call test cases... -----> (IMP)
127      * How to quit watch mode... -----> (IMP)
128      * How to filter test files for run.... -----> (IMP)
129      * How to Filter Test case.....
130
131
132  * Test Grouping With Describe
133
134      * What is Describe?
135      * How to make testcases Group?
136      * Run test case with Describe?
137      * Skip in Describe?
138      * Only in Describe...
139      * Nested Describe...
140
141
142  Syntax for describe :-
143
144  describe("Name of Test Case ", () =>
145      {
146          Write Your Test Cases....
147      }
148  )
149
150

```

```

151 * Example :-
152
153 describe("API test case group", ()=> {
154
155     test("api case 1 ",()=> {
156         render(<InputText/>);
157         let checkInput1 = screen.getByRole("textbox");
158         expect(checkInput1).toHaveAttribute("name","username")
159     })
160     test("api case 2 ",()=> {
161         render(<InputText/>);
162         let checkInput1 = screen.getByRole("textbox");
163         expect(checkInput1).toHaveAttribute("name","username")
164     })
165     test("api case 3 ",()=> {
166         render(<InputText/>);
167         let checkInput1 = screen.getByRole("textbox");
168         expect(checkInput1).toHaveAttribute("name","username")
169     })
170 })
171
172
173 * To Skip any of The Test Case....
174
175 // To Skip Test Case....
176
177 describe("Name of Test Case ", () =>
178     {
179         Write Your Test Cases....
180     }
181 )
182
183
184 // To Execute This Test Cases Only....
185
186 describe.only("Name of Test Case ", () =>
187     {
188         Write Your Test Cases....
189     }
190 )
191
192
193 // To Skip This Group of Test Cases Only....
194
195 describe.skip("Name of Test Case ", () =>
196     {
197         Write Your Test Cases....
198     }
199 )
200

```

```

201 // Nested Describe
202
203 describe("",()=>{
204
205     test.skip("",()=>{
206
207         })
208
209         describe("",()=>{
210
211             test("",()=>{
212
213
214                 })
215
216             })
217         })
218
219
220
221 # Test OnChange Event with Input Box...
222
223 * Make Input box in the component.
224 * Define state and use with on change event.
225 * Import Component in Test File.
226 * Write Code for test case.
227 * Run test case.
228
229 describe("UI Testing for OnChange Event",()=>{
230
231     test("Input Box Testing",()=>{
232         render(<Input/>);
233         let inputtest = screen.getByRole("textbox");
234         fireEvent.change(inputtest,{target:{value:"a"}});
235         expect(inputtest.value).toBe("a");
236     });
237 });
238
239
240 * Test Click Event with Button...
241
242 * Make Button and State in the Component...
243 * Update state with button click event...
244 * Import Component in test file...
245 * Write code for test click event...
246 * Run Test Case....
247
248
249 import { fireEvent, render , screen } from "@testing-library/react";
250 import Button from "../Components/Button";

```

```

describe(" UI Testing Button",()=>{
    test("test case on the button",()=>{
        render(<Button/>);
        const TestButton = screen.getByRole("button");
        fireEvent.click(TestButton);
        expect(screen.getByText("Welcome World ReactJs World Atharva Deelip Deshmukh")).toBeInTheDocument();
    })
})

```

#### \* File and Folder naming Convention....

- \* What file name we can use for test case file?
- \* Folder name for testing files...
- \* Run test case with naming convention

Example :- test\_name.test.js

#### Different Naming Convention....

- \* file\_name.test.js
- \* fie\_name.spec.js
- \* file\_name.spec.jsx
- \* \_\_test\_\_ ---> Folder Name
- \* Inside \_\_tests\_\_ Folder We can Write Simply text.js (no need to write text.test.js)....

#### \* Before and After Hooks....

---> They all are Simply Jest Hooks and not a React Hooks....

- \* use of before and after hooks...
- \* beforeAll and beforeEach....
- \* AfterAll and afterEach...
- \* Example....

Uses :-

- 1) When we Want to Run Any Of the Function...
- 2) To Clean the Database...
- 3) To set a Environment...
- 4) for Uni Testing if we want to set the constant....

```

301
302
303 * beforeAll :- It will Run Only Once all the Cases...
304
305 * beforeEach :- It will Run as Many Test Cases are Present...
306
307 *   AfterAll :- When All the Test Cases are Runned then also it will run once...
308
309 *   afterEach :- When All the Test Cases are Runned then also it will run as Many Test Cases are Present...
310
311
312
313
314
315
316 *   SnapShot Testing....
317
318     *   What is SnapShot testing...?
319     *   Example ?
320     *   When this is usefull?
321     *   How to update snapshots?
322
323
324
325 *   What is SnapShot Testing....?
326
327 *   Snapshot tests are useful when you want to make sure your UI does not change unexpectedly.
328
329 import { render , screen } from "@testing-library/react";
330 import Data from "../Components/Data";
331
332 describe("UI testing",()=>
333 {
334     test("testing a Data",()=>
335     {
336         let Data1=render(<Data/>);
337         expect(Data1).toMatchSnapshot();
338     })
339 })
340
341
342
343 *   Important Points for Testing
344
345     *   What we Should Test ?
346     *   What things we should not test ?
347     *   Important Points....?
348
349
350     *   What we Should Test ?

```

```
351
352 *   Testing component rendering
353 *   UI Elements that we write
354 *   Functions which we write    ---> To Check a Validation....
355 *   API Testing
356 *   Event Testing    ----> Button and Input Testing and Many More....
357 *   Props and States
358 *   UI Condition testing | UI State Testing...
359
360
361 *   Avoid Testing for
362
363 *   External UI Library code...
364 *   No Need to test default function of JS and React...
365 *   Sometimes we should mock function rather than testing it in details...
366
367
368
369 *   Important Points
370
371 *   Do not write snapshots in starting of the project...
372 *   Run test case after completing your functionality...
373 *   Make a standard for code coverage...
374
375
376
377
378 *   Class Component Method Testing
379
380
381 *   Make Class Component....
382 *   Install React test renderer....
383 *   Test Class Component Method....
384
385
386
387 *   Generally We Should Not Do a Function Testing Because We are reside with the Output...
388
389
390 *   Test Render Package Creates the Instance of that Class Components....
391
392
393 To Install the react-test-renderer using the Below Following Command....
394
395 > npm i react-test-renderer
396
397
398 In the Testing Component Testing File Import Below Package
399
400 > import renderer from 'react-test-renderer';
```



```

401
402
403 import Users from "../Components/UsersClassComp";
404 import renderer from 'react-test-renderer';
405
406
407 test("Class Components method testing",()=>{
408
409     const componentData = renderer.create(<Users/>).getInstance();
410     expect(componentData.getUserList()).toMatch("user List")
411
412 })
413
414
415 * Functional Component Method Testing...
416 * Discuss Possible case for method testing
417 * Define the button, click event and method...
418 * Test method with event...
419 * Test method without event....
420
421
422
423 * First method to Test a Functional Component....
424
425
426 UserFunctionalComp.js
427
428 import React, { useState } from 'react'
429
430 function UsersFunctionalComp() {
431
432     const [data,SetData] = useState("");
433
434     const handleData = () => {
435         SetData("Hello World");
436     }
437     return (
438         <div>
439             <h1>Testing a Functional Components</h1>
440             <button data-testid="btn1" onClick={handleData}>Update Data</button>
441             <h2>{data}</h2>
442         </div>
443     )
444 }
445
446 export default UsersFunctionalComp;
447
448
449
450

```

```
451 UserFunctionalComp.test.js
452
453
454 import { fireEvent, render, screen } from "@testing-library/react"
455 import UsersFunctionalComp from "../Components/UsersFunctionalComp";
456
457 describe("UI Testing",()=> {
458     test("Functional Components Testing",()=>{
459         render(<UsersFunctionalComp/>);
460         const btn = screen.getByTestId("btn1");
461         fireEvent.click(btn);
462         expect(screen.getByText("Hello World")).toBeInTheDocument();
463     })
464 })
465
466 > Developer's Standard Method...
467
468
469 * Second method to Test a Functional Component....
470
471
472
473 > Take a particular Function in Different Component then Access Those Component in UserFunctionalComp.test.js
474 and then Test that Component Which Contain a Particular Function...
475
476
477 > handle.js
478
479 const handleData1 = () =>
480 {
481     console.log("Atharva Deshmukh");
482     return "Atharva Deshmukh";
483 }
484
485 export default handleData1;
486
487
488 > UserFunctionalComp.test.js
489
490
491 import { fireEvent, render, screen } from "@testing-library/react"
492 import handleData1 from "../Components/handle";
493 describe("UI Testing 2",()=>{
494
495     test("Functional Component Test 3",()=>{
496
497         expect(handleData1()).toMatch("Atharva Deshmukh");
498
499     })
500
```

```
501     })
502
503
504
505 *   RTL Query    ---> (IMP)...
506
507
508 *   What is RTL Query ?
509 *   Why need RTL Query ?
510 *   Steps in Testing UI ?
511 *   How RTL Query finds elements ?
512 *   Type of RTL Queries....
513
514
515
516 *   RTL Query :- RTL Query is Used to find Our UI Elements So that We Can Test That Elements...
517
518
519 Steps in Testing UI
520
521 *   Render Component...
522 *   Find Element and action...
523 *   Assertions...
524
525
526
527 *   How RTL Find Elements
528
529 *   By Element Type
530 *   By Element name
531 *   By Element id
532 *   By Test id
533
534
535 *   Type of RTL Queries
536
537 *   Find Single Element
538
539 *       getBy
540 *       queryBy
541 *       findBy
542
543 *   Find Multiple Elements
544
545 *       getAllBy
546 *       queryAllBy
547 *       findAllBy
548
549
550
```

```

551 #
552
553 *   getByRole Query  -----> Most Used Query....
554
555
556
557 *   What is the Role in getByRole ?
558 *   What is semantic elements ?
559
560 *   Button, heading tags and table are semantic element...
561 *   Div and span are not semantic elements
562
563 *   Test textbox with getByRole
564
565 *   text box present or not....
566 *   text box value...
567 *   text box disabled or not...
568
569
570 *   Test button with getByRole....
571
572
573 *   getByRole Comes in the Catgory of getBy....
574
575 *   What is the Role in getByRole ?
576
577 >   In Our UI There are Many Semantic Tags Where Each of Them Role is Already Defined....
578
579
580 *   What is Semantic Tags ?
581
582 > Semantic tags are those tags Which Tells Themselves and Browser and us that What are their Specific Working...
583
584 > button
585 > heading tags
586 > table
587
588
589 *   Example of getByRole Testing on TextBox
590
591 FirstInput.js
592
593 import React from 'react'
594
595 export default function First() {
596   return (
597     <div >
598       <h1>getByRole</h1>
599       <input type="text" />
600     </div>

```

```

601     )
602   }
603
604   FirstInput.test.js
605
606
607   import { render, screen } from "@testing-library/react"
608   import First from "../Components/getByRole/FirstInput";
609
610   describe ("UI Testing Part",()=>{
611
612     test("Testing Input Box ", ()=> {
613       render(<First/>);
614       let input=screen.getByRole("textbox");
615       expect(input).toBeInTheDocument();
616     })
617   })
618
619
620   *   Note :- To Set a Value in a Input TextBox We Can Give Through Two Approach
621
622   *   1) Pass the defaultValue="Atharva Deshmukh";
623   *   2) Pass a Value Through OnChange Using Different Function....
624
625
626

```

Type To Check Whether It is a Input TextBox is Disbaled or Not...

```

628   Input.js
629
630
631   import React from 'react'
632
633   export default function First() {
634     return (
635       <div >
636         <h1>getByRole</h1>
637         <input type="text" defaultValue={"Atharva"} disabled/>
638       </div>
639     )
640
641
642   Input.test.js
643
644   import { render, screen } from "@testing-library/react"
645   import First from "../Components/getByRole/FirstInput";
646
647   describe ("UI Testing Part",()=>{
648
649     test("Testing Input Box ", ()=> {
650       render(<First/>);

```

```

651         let input=screen.getByRole("textbox");
652         expect(input).toBeInTheDocument();
653         // expect(input).toHaveValue("Enter You Name :- ");
654     })
655
656 })
657
658
659 Button.test.js
660
661     test ("Button Testing test-1",()=>
662     {
663         render(<First/>);
664         let btn=screen.getByRole("button");
665         fireEvent.click(btn);
666         expect(btn).toBeInTheDocument();          // To Check a Button is Present or Not...
667         expect(screen.getByText("Button UI Testing")).toBeInTheDocument(); // To Check Whether a Text is
        printed on Screen or Not While on the OnClick Functionality....
668
669     })
670
671     *   Types in expect
672
673     *   toBeInTheDocument();
674
675         *   To Check Whether My InputFields are Present or Not....
676
677         *   expect(input).toBeInTheDocument();
678
679     *   toHaveValue(); ----> It Contains a Value those Same value or Not....
680
681         *   expect(input).toHaveValue("Enter You Name :- ");
682
683     *   toBeDisabled(); ----> To Check Whether Button is Enabled or Disbaled....
684
685         *   expect(input).toBeDisabled();
686
687
688     *   toHaveAttribute("key","value");
689
690         *   To Check Whether The key value pair are Present in the Code or not...
691
692         *   expect(input).toHaveAttribute("name","username");
693
694
695
696
697 #
698 *   Multiple elements with Role Custom Role....
699

```

```
700 * Multiple elements with the same role issue...
701 * Multiple buttons with role...
702 * Multiple Input box with role...
703 * Custom Role....
704
705
706
707 * Multiple elements with the same role issue...
708
709 Solution :-
710
711
712 Button.js
713
714 import React from "react";
715
716 function Box() {
717   return (
718     <div>
719       <button>Click 1</button>
720       <button>Click 2</button>
721     </div>
722   )
723 }
724
725 export default Box;
726
727
728
729
730 // * It Will Show the Error TestingLibraryElementError: Found multiple elements with the role "button"....
731
732
733 Button.test.js
734
735 import { render, screen } from "@testing-library/react"
736 import Box from "../Box";
737
738 describe("UI Testing",()=>{
739   test("Button Testing test 1",()=>{
740     render(<Box/>);
741     let btn=screen.getByRole("button");
742     let btn1=screen.getByRole("button");
743     expect(btn).toBeInTheDocument();
744     expect(btn1).toBeInTheDocument();
745   })
746 })
747
748
749 // Solution for Above Error...
```

```

750
751 * We Can use Multiple Attributes in getByRole("button",{name:"Click 2"});
752
753
754 import { render, screen } from "@testing-library/react"
755 import Box from "../Box";
756
757 describe("UI Testing",()=>{
758     test("Button Testing test 1",()=>{
759         render(<Box/>);
760         let btn=screen.getByRole("button",{name:"Click 1"});
761         let btn1=screen.getByRole("button",{name:"Click 2"});
762         expect(btn).toBeInTheDocument();
763         expect(btn1).toBeInTheDocument();
764     })
765 })
766
767
768
769

```

Multiple Input TextBox in Same test Check....

Input.js

```

775 import React from 'react'
776
777 function Box1() {
778     return (
779         <div>
780             <label htmlFor="input1">User Name</label>
781             <input type="text" id='input1'/>
782             <label htmlFor="input2">User Name 2</label>
783             <input type="text" id='input2'/>
784         </div>
785     )
786 }
787
788 export default Box1;
789
790

```

Input.test.js

```

794 import { render , screen } from "@testing-library/react"
795 import Box1 from "../Box1";
796
797 describe("UI Testing ",()=>{
798     test('Multiple InputBox Testing', () => {
799

```



```

800         render(<Box1/>);
801         let input1=screen.getByRole("textbox",{name:"User Name"});
802         let input2=screen.getByRole("textbox",{name:"User Name 2"});
803
804         expect(input1).toBeInTheDocument();
805         expect(input2).toBeInTheDocument();
806     })
807
808 })
809
810 // How to Use Custom Role.... -----> (IMP)....
811
812
813 *   How to Test a Non Semantic Elements....
814
815
816
817
818 Error :- TestingLibraryElementError: Unable to find an accessible element with the role ""...
819
820
821 div.js
822
823 <div>
824     <h1>Hello Atharva It Your Time You Achive Something in Life..</h1>
825 </div>
826
827 div.test.js
828
829 const dv1=screen.getByRole("");
830 expect(dv1).toBeInTheDocument();
831
832
833 // To Slove Above Error.... Define Manually Role in div.js....
834
835
836 div.js
837
838 <div role='dummy'> ----> Define manually Role.... in Html File...
839
840     <h1>Hello Atharva It Your Time You Achive Something in Life..</h1>
841 </div>
842
843
844
845 div.test.js
846
847 const dv1=screen.getByRole("dummy");
848 expect(dv1).toBeInTheDocument();
849

```

```
850
851 #
852 * RTL Query : getAllByRole
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877 * React Testing Using Typescript
878
879 * Test Driven Development :- * Write a First test and to Pass That Case We Should Write Code later...
880
881
882 * Concept :- Props Object With a Name....
883
884
885 First.tsx
886
887 import { render, screen } from "@testing-library/react"
888 import Second from "../Second";
889
890 describe("UI Testing",()=>
891 {
892     test("Text Testing Second",()=> {
893
894         render(<Second name='Atharva'/>);
895         let Element=screen.getByText("Hello Atharva");
896         expect(Element).toBeInTheDocument();
897     })
898 })
899
```

```

First.test.tsx

import React from 'react'
type GreetProps = {
  name?: string
}

export default function Second(props:GreetProps) {
  return (
    <div>
      <h1>Hello {props.name}</h1>
    </div>
  )
}

```

# Jest Watch Mode

- \* Watch mode is an **option** that we can pass to Jest asking to watch files that have changed since the last commit and execute tests related only to those changed files...
- \* An Optimization designed to make your tests run fast regardless **of** How many tests you have....

\* Note :- In ReactJs With Typescript We Can use it **with** the replace **of** test... They Both Are Declared Globally...

Syntax :-

```

describe("",()=>{

  it("",()=>{

  })

  it("",()=>{

  })
})

```

\* To Replicate test.only we Can Use

```
test.only("",()=>{
```

```
})
```

```
replace it with
```

```
fit("",()=>{
```

```
})
```

```
and to Exclude a test use
```

```
xit("",()=>{
```

```
})
```

#### # Code Coverage

- \* A metric that can help you understand how much **of** your Software code is tested...
- \* Statement Coverage :- How many **of** the statements **in** the Software code have been executed
- \* Branches Coverage:- How many **of** the Branches **of** the Controll Structures(**if** statements **for** instance) have been executed
- \* **function** coverage :- How many **of** the functions defined have been called and **finally..**
- \* Line Coverage: How many **of** lines **of** source code have been tested...
- \* Important Command To See Whole Coverage

```
Add in Package.json
```

```
*   "scripts": {  
      "test:coverage": "react-scripts test --coverage --watchAll"  
    }  
}
```

```
> Run The Command in CMD
```

```
1000 *   npm run test:coverage
1001
1002
1003
1004 *   Add in Package.json for the Whole Coverage
1005
1006 Case-1 :- If we Want to Access to Test one Folder files...
1007
1008 "test:coverage": "react-scripts test --coverage --watchAll --collectCoverageFrom='src/Components/**/*.{ts,tsx}"
1009
1010 or
1011
1012 "test:coverage": "react-scripts test --coverage --watchAll
--collectCoverageFrom='!src/components/**/*.{types,stories,constants,test,spec}.{ts,tsx}'"
1013
1014
1015 (IMP)
1016
1017 *   CoverageThreshold :- With Jest it is Possible to Specify Minimum threshold Inforcement for Coverage Reports...
1018
1019
1020 "jest":{
1021   "coverageThreshold":{
1022     "global":{
1023       "branches": 100,
1024       "functions": 100,
1025       "lines": 100,
1026       "statements": 100
1027     }
1028   }
1029 }
1030
1031
1032 * Assertions
1033
1034 * When writing tests, we often need to check that values
1035   meet certain conditions...
1036
1037 * Assertions decide if a test passes or fails...
1038
1039 * expect(value)
1040
1041 *   The argument should be the value that your code produces...
1042 *   Typically, you will use expect along with a "matcher
1043   function to assert something about a value...
1044 *   A Matcher can optionally accept an argument which is the correct expected value...
```

```
1048 #
1049 *   What is Test...?
1050
1051     > Test Component renders...
1052     > Test Component renders with props...
1053     > Test Component renders in different States...
1054     > Test Component reacts to events...
1055
1056 *   What not to test?
1057
1058     > Implementation details....
1059     > Third Party Code...
1060     > Code That is not Important from a User Point of view...
1061
1062
1063
1064 #   RTL Queries
1065
1066 *   Every test we write generally involves the following basic steps...
1067
1068     1. render the component
1069     2. Find an element rendered by the component
1070     3. Assert against the element found in step 2 which will pass
1071         or fail the test
1072
1073     To render the component, we use the render method from RTL
1074
1075     *   For assertion, we use expect passing in a value and combine it with a matcher
1076         function from jest or jest-dom..
1077
1078
1079 *   Queries are the Methods that testing Library provides to find elements on the page..
1080
1081 *   To Find a Single element on the page, We Have
1082
1083     *   getBy...
1084     *   queryBy...
1085     *   findBy...
1086
1087 *   To find multiple elements on the page, we Have
1088
1089     *   getAllBy...
1090     *   queryAllBy...
1091     *   FindAllBy...
1092
1093     The Suffix can be one of Role, LabelText PlaceholderText, text, DisplayValue, AltText, Title and Finally TestId...
1094
1095     *   getBy.. class of queries return the matching node for a query, and throw a descriptive error if no elements
1096         match
1097         or if more than one match is found...
```

```

* The Suffix can be one of Role, LabelText PlaceholderText, text, DisplayValue, AltText, Title and Finally
TestId...

* By default, many semantic elements in HTML have role....

* Button element has a button role, anchor element has link role, h1 to h6
elements have a heading role, checkboxes have a checkbox role, radio buttons have a radio role and so on...

* If you are Working with elements that do not have a default role or if you
want to specify a different role, the role attribute can be used to add the desired role..

* To use an anchor element as a button in the navbar, you can add role='button'...

* getByRole

* getByRole Options ----> (IMP)

* It is UseFull When Multiple Elements Have Same Role....

1) name
2) level
3) hidden
4) selected
5) checked
6) pressed

```

Third.js

```

import React from 'react'

export default function Third() {
  return (
    <div>
      <form action="">
        <div>
          <label htmlFor="name">Name</label>
          <input type="text" id='name' />
        </div>
        <div>
          <label htmlFor="job-location">Job Location</label>
          <select value="">
            <option value="">Select a Country</option>
            <option value="US">United States</option>
            <option value="GB">United Kingdom</option>
            <option value="CA">Canada</option>
          </select>
        </div>
      </form>
    </div>
  )
}

```

```

1146         <option value="AU">Australia</option>
1147         <option value="FR">France</option>
1148         <option value="DE">Germany</option>
1149     </select>
1150 </div>
1151 <div>
1152     <label>
1153         <input type="checkbox" id='terms' /> I agree to the terms and conditions
1154     </label>
1155 </div>
1156 <button>Submit</button>
1157 </form>
1158 </div>
1159 )
1160 }

```

Third.test.js

```

1166
1167 import { render , screen} from "@testing-library/react";
1168 import Third from "../Third";
1169
1170 describe('Application', () => {
1171     test('Renders Correctly', () => {
1172
1173         render(<Third/>);
1174         let Input=screen.getByRole("textbox");
1175         expect(Input).toBeInTheDocument();
1176
1177         let jobLocationElement = screen.getByRole("combobox");
1178         expect(jobLocationElement).toBeInTheDocument();
1179
1180         const termsElement = screen.getByRole("checkbox");
1181         expect(termsElement).toBeInTheDocument();
1182
1183         const submitButtonElement = screen.getByRole("button");
1184         expect(submitButtonElement).toBeInTheDocument();
1185     })
1186
1187 })

```

\* To Test a Multiple TextBox....

Third.js



```
1196     <div>
1197         <label htmlFor="name">Name</label>
1198         <input type="text" id='name' />
1199     </div>
1200     <div>
1201         <label htmlFor="bio">Bio</label>
1202         <textarea name="bio" id="bio"></textarea>
1203     </div>
```

```
1204
1205
1206 Third.test.js
```

```
1207
1208
1209 let Input=screen.getByRole("textbox" , {name:"Name"});
1210 expect(Input).toBeInTheDocument();
1211
1212 let BioName=screen.getByRole("textbox" , {name:"Bio"});
1213 expect(BioName).toBeInTheDocument();
1214
1215
```

```
1216
1217 *   To Test a Multiple Heading...
```

```
1218
1219 Third.js
1220
1221 <h1>Job Application Form</h1>
1222 <h2>Section 1</h2>
```

```
1223
1224
1225
1226 Third.test.js
1227
1228 let Head = screen.getByRole("heading" , {name:"Job Application Form"});
1229 expect(Head).toBeInTheDocument();
1230
1231 let Head1 = screen.getByRole("heading" , {name:"Section 1"});
1232 expect(Head1).toBeInTheDocument();
1233
```

```
1234
1235 *   Note :- To Test Multiple Same Field With Different Name Assign to Them...
```

```
1236
1237 * We Can Differentiate Heading By There Level Also...
```

```
1238
1239 example :- level:2;
```

```
1240
1241
1242
1243 #
1244 *   getByLabelText
```

```
*   getByLabelText will search for the label that matches the given text, then find
    the element associated with the label....
```

```
Third.js
```

```
<label htmlFor="name">Name</label>
<input type="text" id="name" />
```

```
Third.test.js
```

```
let nameElement = screen.getByLabelText('Name');
expect(nameElement).toBeInTheDocument();
```

```
(IMP)
```

```
Note :- * Use Selector Also To Differentiate Between Input and textarea or any...
```

```
Third.js
```

```
<div>
  <label htmlFor="name">Name</label> ----> (*)
  <input type="text" id="name" />
</div>
```

```
Third.js
```

```
<div>
  <label htmlFor="job-location">Name</label> ---> (*)
  <select value="">
    <option value="">Select a Country</option>
    <option value="US">United States</option>
    <option value="GB">United Kingdom</option>
    <option value="CA">Canada</option>
    <option value="AU">Australia</option>
    <option value="FR">France</option>
    <option value="DE">Germany</option>
  </select>
</div>
```

```
Third.test.js
```

```
let Input=screen.getByRole("textbox" , {name:"Name"});
```

```

1296     expect(Input).toBeInTheDocument();
1297
1298
1299     let nameElement = screen.getByLabelText('Name',{selector:"input"});
1300     expect(nameElement).toBeInTheDocument();
1301
1302
1303
1304 #
1305 *   getByPlaceholderText
1306
1307     *   getByPlaceholderText will search for all elements with a placeholder
1308         attribute and find one that matches the given text...
1309
1310
1311 #
1312 *   getByText
1313
1314
1315     Fourth.js
1316
1317     <p>This Section is Mandatory</p>
1318
1319     Fourth.text.js
1320
1321     let paragraphText = screen.getByText("This Section is Mandatory");
1322     expect(paragraphText).toBeInTheDocument();
1323
1324
1325 #
1326 *   getByDisplayValue
1327
1328     *   getByDisplayValue returns the input, textarea, or select element that has the matching display value...
1329
1330
1331     Fifth.tsx
1332
1333     <div>
1334         <label htmlFor="name">Name</label>
1335         <input type="text" id="name" placeholder="Fullname" value="Atharva" onChange={()=> {}}/>
1336     </div>
1337
1338     // OnChange Handler is Used to Remove the Warning....
1339
1340     FiFth.test.tsx
1341
1342     let nameElement2=screen.getByDisplayValue("Atharva");
1343     expect(nameElement2).toBeInTheDocument();
1344
1345

```

```
1346 #
1347 *   getByAltText
1348
1349 *   getByAltText will return the element that has the given alt text...
1350 *   This method only supports elements which accept an alt attribute like <img>,
1351 <input>,<area> or custom HTML elements...
1352
1353
1354 Sixth.js
1355
1356 <img src="" alt="A Natures Image" />
1357
1358 Sixth.test.js
1359
1360 let ImageElement=screen.getByAltText("A Natures Image");
1361 expect(ImageElement).toBeInTheDocument();
1362
1363
1364 #
1365 *   getByTitle
1366
1367 *   getByTitle returns the element that has the matching title attribute...
1368
1369
1370 Seventh.js
1371
1372 <span title="close"></span>
1373
1374 Seventh.test.js
1375
1376 let closeElement=screen.getByTitle("close");
1377 expect(closeElement).toBeInTheDocument();
1378
1379
1380 #
1381 *   getByTestId
1382
1383 *   getByTestId returns the element that has the matching data-testid attribute...
1384
1385
1386 Eight.js
1387
1388 <div data-testid="custom-element">
1389   Custom HTML element
1390 </div>
1391
1392
1393 Eight.test.js
1394
1395 let CustomElement = screen.getByTestId("custom-element");
```

```
1396     expect(CustomElement).toBeInTheDocument();
1397
1398
1399
1400
1401 #
1402 *   Priority Order for Queries
1403
1404 *   Note :- Your test should resemble how users interact with your code
1405         (Component , page ) as much as possible....
1406
1407
1408 1) getByRole
1409
1410 2) getByLabelText
1411
1412 3) getByPlaceholderText
1413
1414 4) getByText    ---> Outside a Form we can Simply find Elements like div , span , paragraph ....
1415
1416 5) getByDisplayValue
1417
1418 6) getByAltText :- When Your Element is one Which supports all text such Image , area , Input or any Custom Element...
1419
1420 7) getByTitle
1421
1422 8) getByTestId
1423
1424
1425
1426 #
1427 *   RTL getAllBy Queries
1428
1429 *   Find multiple elements in the DOM...
1430
1431 *   getAllBy returns an array of all matching nodes for a query, and throws
1432         an error if no elements match...
1433
1434
1435 *   getAllByRole
1436
1437 *   getAllByLabelText
1438
1439 *   getAllByPlaceholderText
1440
1441 *   getAllByText
1442
1443 *   getAllByDisplayValue
1444
1445 *   getAllByAltText
```

```
1446
1447     *   getAllByTitle
1448
1449     *   getAllByTestId
1450
1451
1452
1453     #
1454     *   Query Multiple Elements
1455
1456
1457     > skills.types.ts
1458
1459     export type SkillsProps = {
1460         skills: string[];
1461     }
1462
1463
1464     > skills.tsx
1465
1466     import { SkillsProps } from "./skills.types";
1467     import React from 'react'
1468
1469     export const Skills = (props: SkillsProps) => {
1470         const {skills} = props;
1471         return (
1472             <>
1473                 <ul>
1474                     {skills.map((skill)=>{
1475                         return <li key={skill}>{skill}</li>;
1476                     })}
1477                 </ul>
1478             </>
1479         )
1480     }
1481
1482
1483
1484     > skills.test.tsx
1485
1486     import { render , screen } from "@testing-library/react";
1487     import { Skills } from "./skills";
1488
1489     describe('Skills', () => {
1490         const skills = ["HTML","CSS","JavaScript"];
1491
1492         test('renders Correctly', () => {
1493             render(<Skills skills={skills} />);
1494             const listElement = screen.getByRole("list");
1495             expect(listElement).toBeInTheDocument();
```

```

1496     })
1497
1498     test("renders a list of skills",()=>{
1499         render(<Skills skills={skills} />);
1500         const listItemElements = screen.getAllByRole("listItem");
1501         expect(listItemElements).toHaveLength(skills.length);
1502     })
1503 })
1504
1505
1506 #
1507 *   Text Match
1508
1509 *   TextMatch represents a Type which can be either a
1510
1511 *   String...
1512 *   regex...
1513 *   function...
1514
1515
1516 *   String
1517
1518   <div>Hello World</div>
1519
1520   screen.getByText("Hello World"); // full String match...
1521
1522   screen.getByText('llo Worl', {exact:false}) // substring match...
1523
1524   screen.getByText('hello world',{exact:false}) // ignore case
1525
1526
1527 *   TextMatch - regex
1528
1529   <div>Hello World</div>
1530
1531   screen.getByText(/World/) // substring match
1532
1533   screen.getByText(/word/i) // substring match, ignore case...
1534
1535   screen.getByText(/^hello world$/i) // full string match, ignore Case...
1536
1537
1538 *   TextMatch - custom function
1539
1540   (content?: string, element?: Element | null) => boolean
1541
1542   <div>Hello World </div>
1543
1544   screen.getByText((content)=> content.startsWith('Hello'))
1545

```

```
1546
1547 #
1548 * queryBy
1549
1550
1551
1552 Tenth.types.js
1553
1554 export type SkillsProps = {
1555   skills: string[];
1556 }
1557
1558 Tenth.js
1559
1560 import { SkillsProps } from "../skills.types";
1561 import React from 'react'
1562
1563 export const Skills = (props: SkillsProps) => {
1564   const { skills } = props;
1565   return (
1566     <>
1567       <ul>
1568         {skills.map((skill) => {
1569           return <li key={skill}>{skill}</li>;
1570         })}
1571       </ul>
1572     </>
1573   )
1574 }
1575
1576
1577 Tenth.test.js
1578
1579
1580 import { render, screen } from "@testing-library/react";
1581 import { Skills } from "../skills";
1582
1583 describe('Skills', () => {
1584   const skills = ["HTML", "CSS", "JavaScript"];
1585
1586   test('renders Correctly', () => {
1587     render(<Skills skills={skills} />);
1588     const listElement = screen.getByRole("list");
1589     expect(listElement).toBeInTheDocument();
1590   })
1591
1592   test("renders a list of skills", () => {
1593     render(<Skills skills={skills} />);
1594     const listItemElements = screen.getAllByRole("listItem");
1595     expect(listItemElements).toHaveLength(skills);
1596   })
1597 }
```



```
1596         })
1597     })
1598
1599
1600
1601
1602
1603
1604
1605 Testing a Linear Gradient....
1606
1607
1608 // MyComponent.tsx
1609 import React from 'react';
1610
1611 const MyComponent = () => {
1612     return (
1613         <div style={{ backgroundImage: 'linear-gradient(to right, red, blue)' }}>
1614             This is my component
1615         </div>
1616     );
1617 };
1618
1619 // MyComponent.test.tsx
1620 import React from 'react';
1621 import { render } from '@testing-library/react';
1622 import { expect } from '@jest/globals';
1623 import MyComponent from './MyComponent';
1624
1625 describe('MyComponent', () => {
1626     it('should render the expected linear gradient', () => {
1627         const { container } = render(<MyComponent />);
1628         expect(container).toMatchInlineSnapshot(); // Capture the rendered component
1629     });
1630 });
1631
1632
1633
1634 Mocking: Mock the component that provides the text or the function that generates it. Update the mock data to simulate the text change.
1635
1636
1637     // "@testing-library/jest-dom": "^6.4.2",
1638
1639
1640
1641
1642
1643
1644
```

1645  
1646  
1647  
1648  
1649  
1650  
1651  
1652  
1653  
1654  
1655  
1656  
1657  
1658  
1659  
1660  
1661  
1662  
1663  
1664  
1665  
1666  
1667  
1668  
1669  
1670  
1671  
1672  
1673  
1674  
1675  
1676  
1677  
1678  
1679  
1680  
1681  
1682  
1683  
1684  
1685  
1686  
1687  
1688  
1689  
1690  
1691  
1692  
1693  
1694

1695  
1696  
1697  
1698  
1699  
1700  
1701  
1702  
1703  
1704  
1705  
1706  
1707  
1708  
1709  
1710  
1711  
1712  
1713  
1714  
1715  
1716  
1717  
1718  
1719  
1720  
1721  
1722  
1723  
1724  
1725  
1726  
1727  
1728  
1729  
1730  
1731  
1732  
1733  
1734