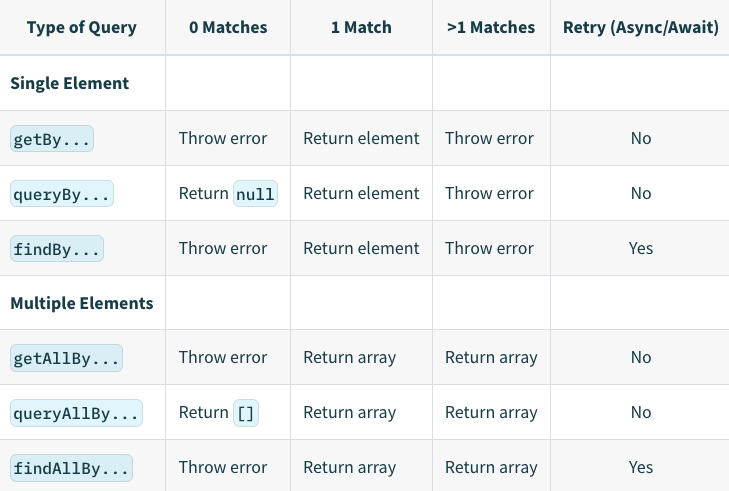
Slide 1: Introduction

* What is testing?
* (Dev: unit testing) 🡪SDLC at multiple stages  
  - Briefly introduce the topic of unit testing in React.js  
  - Mention the importance of unit testing for developing high-quality applications  
    
  Slide 2: What is unit testing?  
  - - Mention that unit testing involves testing individual components or functions in isolation  
  Slide 3: Tools for unit testing in React.js  
  - Mention some of the popular tools for unit testing in React.js, such as Jest, Enzyme, and React Testing Library  
  - Briefly describe each tool and its features
* Plugins like:
* **eslint-plugin-testing-library** (already included in CRA)
* **eslint-plugin-jest-dom**

RTL vs Enzyme (talk about why we need together)

Slide 4: Short Introduction to Query Methods in RTL



**findBy\*:**

* When Match Is Found: Returns a resolved Promise.
* When Match Is Not Found: Returns a rejected Promise

**queryBy\*:**

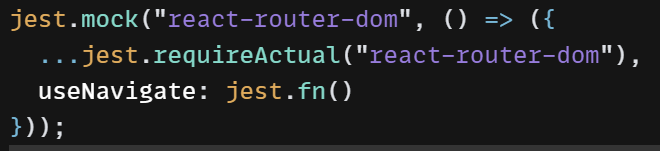
* When Match Is Found: Returns the node that matches the query
* When Match Is Not Found: Returns null

**getBy\*:**

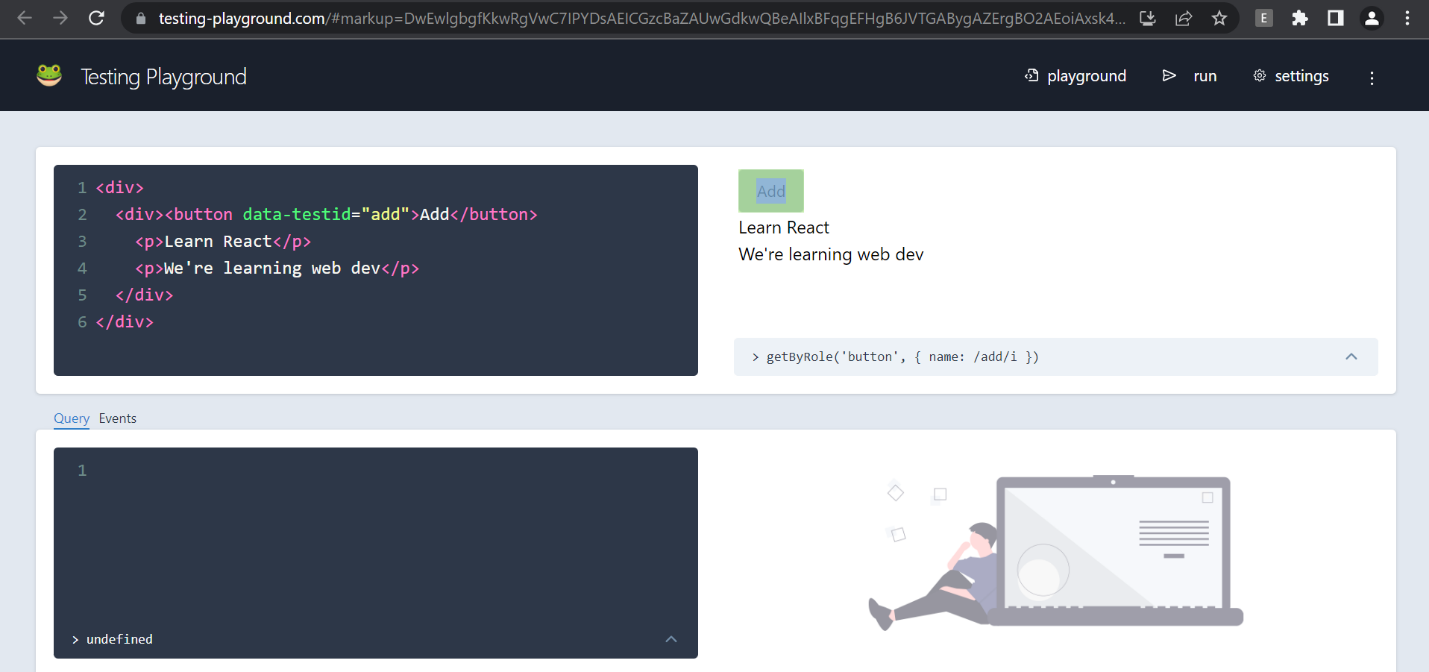
* When Match Is Found: Returns the node that matches the query
* When Match Is Not Found: Throws an error

queryBy\* methods can be useful for asserting an element that is not present (for example, **expect(screen.queryByText('foo')).not.toBeInTheDocument())**

Slide 5: Best practices for unit testing in React.js  
List the best practices for unit testing in React.js, such as:  
 - Test small units of code

* Each React component should ideally be unit tested (in isolation) and any/every external or child components. If required, we can add the integration tests to test interactions b/w components.
* A goal of unit test case should be outcome dependent.  
   - Use a testing framework
* Naming convention  
   - Test both positive and negative scenarios
* Add a unit test case for edge cases as well  
   - Use mock data
*   
   - Test asynchronous code  
   - Keep tests fast and isolated[  
   - Run tests frequently
* Avoid destructuring from render method instead use screen object

(The benefit of using screen is you no longer need to keep the render call destructure up to date as you add/remove the queries you need)

* Using @testing-library/jest-dom (provides a set of custom jest matchers that you can use to extend jest. These will make your tests more declarative, clear to read and to maintain)
* Using @testing-library/user-event
* Only use the query\* variants for asserting that an element cannot be found.
* Use findBy\* in conjunction with await instead of using waitFor (use find\* any time you want to query for something that may not be available right away)
* Await findBy\* >> waitFor +getBy\*
* screen.logTestingPlaygroundURL(); //add a snippet
* 
* Top Priority Queries
* 
* getByRole
* getByLabelText
* getByPlaceholderText
* getByText
* getByDisplayValue
* byTestId

Slide 8: Conclusion  
- Summarize the key points of the presentation  
- Emphasize the importance of unit testing for developing high-quality React applications  
  
Slide 9: Additional resources  
- Provide links to additional resources for learning more about unit testing in React.js, such as official documentation and tutorial videos.

<https://infinum.com/handbook/frontend/react/testing-best-practices>

<https://kentcdodds.com/blog/common-mistakes-with-react-testing-library>

<https://testing-library.com/docs/queries/about/#priority>

<https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/ARIA_Techniques#roles>