**Notebook of React.js**

**What is React?**

* React is a front-end JavaScript library developed by Facebook in 2011.
* It follows the component-based approach which helps in building reusable UI components.
* It is used for developing complex and interactive web and mobile UI.
* Even though it was open sourced only in 2015, it has one of the largest communities supporting it.

**Write some important commands in React?**

npx create-react-app my-app

npm install --save react-router-dom

npm install --save redux

npm install --save react-redux

npm install --save redux-thunk

npm install @reduxjs/toolkit

npm install @material-ui/core

npm i @material-ui/icons -s

npm install axios –save

npm install --save enzyme

npm install --save enzyme react-test-renderer enzyme-adapter-react-16

**What are the features of React?**

Major features of React are listed below:

1. It uses the **virtual DOM** instead of the real DOM.
2. It uses **server-side rendering**.
3. It follows **uni-directional data flow** or data binding.

**What are the props in React?**

“Props” is a special keyword in React, which stands for properties and is being **used for passing data from one component to another**. Furthermore, props data is read-only, which means that data coming from the parent should not be changed by child components.

### ****What is JSX?****

JSX is a shorthand for JavaScript XML.

JSX is NOT HTML but it looks a lot like it. Differences can be seen when looking closely though (for example className in JSX vs class in "normal HTML"). JSX is just syntactic sugar for JavaScript, allowing you to write HTMLish code instead of nested React.createElement(...) calls.

### ****What do you understand by Virtual DOM? Explain its working.****

A virtual DOM is a lightweight JavaScript object which originally is just the copy of the real DOM. It is a node tree that lists the elements, their attributes and content as Objects and their properties. React’s render function creates a node tree out of the React components. It then updates this tree in response to the mutations in the data model which is caused by various actions done by the user or by the system.  
This Virtual DOM works in three simple steps.

1. Whenever any underlying data changes, the entire UI is re-rendered in Virtual DOM representation.
2. Then the difference between the previous DOM representation and the new one is calculated.
3. Once the calculations are done, the real DOM will be updated with only the things that have actually changed. 

### ****Why can’t browsers read JSX?****

Browsers can only read JavaScript objects but JSX in not a regular JavaScript object. Thus, to enable a browser to read JSX, first, we need to transform JSX file into a JavaScript object using JSX transformers like Babel and then pass it to the browser.

### ****“In React, everything is a component.” Explain.****

Components are the building blocks of a React application’s UI. These components split up the entire UI into small independent and reusable pieces. Then it renders each of these components independent of each other without affecting the rest of the UI.

### ****How is React different from Angular?****

|  |  |  |
| --- | --- | --- |
| **React vs Angular** | | |
| **TOPIC** | **REACT** | **ANGULAR** |
| 1. ARCHITECTURE | Only the View of MVC | Complete MVC |
| 2. RENDERING | Server-side rendering | Client-side rendering |
| 3. DOM | Uses virtual DOM | Uses real DOM |
| 4. DATA BINDING | One-way data binding | Two-way data binding |
| 5. DEBUGGING | Compile time debugging | Runtime debugging |
| 6. AUTHOR | Facebook | Google |

### ****What is the purpose of render() in React.****

Each React component must have a **render()**mandatorily. It returns a single React element which is the representation of the native DOM component. If more than one HTML element needs to be rendered, then they must be grouped together inside one enclosing tag such as **<form>, <group>,<div>** etc. This function must be kept pure i.e., it must return the same result each time it is invoked.

### ****What is a state in React and how is it used?****

States are the heart of React components. States are the source of data and must be kept as simple as possible. Basically, states are the objects which determine components rendering and behavior. They are mutable unlike the props and create dynamic and interactive components. They are accessed via **this.state().**

### ****What are the different phases of React component’s lifecycle?****

There are three different phases of React component’s lifecycle:

1. Initial Rendering Phase: This is the phase when the component is about to start its life journey and make its way to the DOM.
2. Updating Phase: Once the component gets added to the DOM, it can potentially update and re-render only when a prop or state change occurs. That happens only in this phase.
3. Unmounting Phase: This is the final phase of a component’s life cycle in which the component is destroyed and removed from the DOM.

Timeline

Description automatically generated

#### **Mounting**

These methods are called in the following order when an instance of a component is being created and inserted into the DOM:

* [**constructor()**](https://reactjs.org/docs/react-component.html#constructor)
* [static getDerivedStateFromProps()](https://reactjs.org/docs/react-component.html#static-getderivedstatefromprops)
* [**render()**](https://reactjs.org/docs/react-component.html#render)
* [**componentDidMount()**](https://reactjs.org/docs/react-component.html#componentdidmount)

**constructor**:

The constructor for a React component is called before it is mounted. When implementing the constructor for a React.Component subclass, you should call super(props) before any other statement. Otherwise, this.props will be undefined in the constructor, which can lead to bugs.

Typically, in React constructors are only used for two purposes:

* Initializing [local state](https://reactjs.org/docs/state-and-lifecycle.html) by assigning an object to this.state.
* Binding [event handler](https://reactjs.org/docs/handling-events.html) methods to an instance.

Constructor is the only place where you should assign this.state directly. In all other methods, you need to use this.setState() instead.

Avoid introducing any side-effects or subscriptions in the constructor. For those use cases, use componentDidMount() instead.

Render Method:

The render() function should be pure, meaning that it does not modify component state, it returns the same result each time it’s invoked, and it does not directly interact with the browser.

ComponentDidMount():

ComponentDidMount () is invoked immediately after a component is mounted (inserted into the tree). Initialization that requires DOM nodes should go here. If you need to load data from a remote endpoint, this is a good place to instantiate the network request.

This method is a good place to set up any subscriptions. If you do that, don’t forget to unsubscribe in componentWillUnmount()

**Updating Phase:**

### componentDidUpdate

componentDidUpdate() is invoked immediately after updating occurs. This method is not called for the initial render.

Use this as an opportunity to operate on the DOM when the component has been updated. This is also a good place to do network requests as long as you compare the current props to previous props (e.g. a network request may not be necessary if the props have not changed).

Unmounting Phase

**componentWillUnmount()** is invoked immediately before a component is unmounted and destroyed. Perform any necessary cleanup in this method, such as invalidating timers, canceling network requests, or cleaning up any subscriptions that were created in componentDidMount().

You **should not call setState()** in componentWillUnmount() because the component will never be re-rendered. Once a component instance is unmounted, it will never be mounted again.

### ****What is an event in React?****

In React, events are the triggered reactions to specific actions like mouse hover, mouse click, key press, etc. Handling these events are similar to handling events in DOM elements. But there are some syntactical differences like:

1. Events are named using camel case instead of just using the lowercase.
2. Events are passed as functions instead of strings.

The event argument contains a set of properties, which are specific to an event. Each event type contains its own properties and behavior which can be accessed via its event handler only.

### ****What are Higher Order Components(HOC)?****

Higher Order Component is an advanced way of reusing the component logic. Basically, it’s a pattern that is derived from React’s compositional nature. HOC are custom components which wrap another component within it. They can accept any dynamically provided child component but they won’t modify or copy any behavior from their input components. You can say that HOC are ‘pure’ components.

### ****What is the significance of keys in React?****

Keys are used for identifying unique Virtual DOM Elements with their corresponding data driving the UI. They help React to optimize the rendering by recycling all the existing elements in the DOM. These keys must be a unique number or string, using which React just reorders the elements instead of re-rendering them. This leads to increase in application’s performance.

### ****Explain Flux.****

Flux is an architectural pattern which enforces the uni-directional data flow. It controls derived data and enables communication between multiple components using a central Store which has authority for all data. Any update in data throughout the application must occur here only. Flux provides stability to the application and reduces run-time errors.

### ****What is Redux?****

Redux is one of the most trending libraries for front-end development in today’s marketplace. It is a predictable state container for JavaScript applications and is used for the entire applications state management. Applications developed with Redux are easy to test and can run in different environments showing consistent behavior.

### ****What are the three principles that Redux follows?****

1. ***Single source of truth:***The state of the entire application is stored in an object/ state tree within a single store. The single state tree makes it easier to keep track of changes over time and debug or inspect the application.
2. **State is read-only:**The only way to change the state is to trigger an action. An action is a plain JS object describing the change. Just like state is the minimal representation of data, the action is the minimal representation of the change to that data.
3. ***Changes are made with pure functions:*** In order to specify how the state tree is transformed by actions, you need pure functions. Pure functions are those whose return value depends solely on the values of their arguments.

**Redux Thunk Middleware:**

Redux Thunk is a middleware that allows **you to call the action creators that return a function(thunk) which takes the store’s dispatch method as the argument** and which is afterwards used to dispatch the synchronous action after the API or side effects has been finished.

**React Hooks:**

**useMemo():**

useMemo(() => computation(a, b), [a, b]) is the hook that lets you memorize expensive computations. Given the same [a, b] dependencies, once memorized, the hook is going to return the memorized value without invoking computation(a, b).

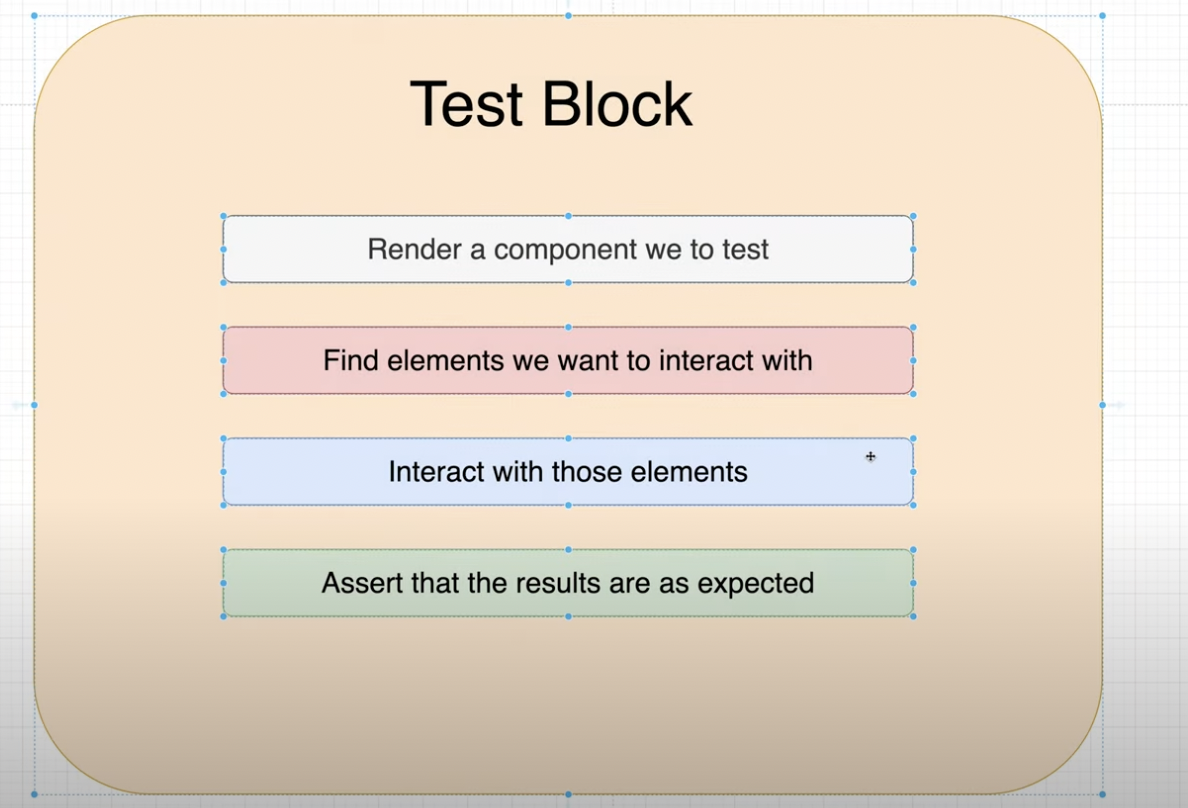
**useCallback():**

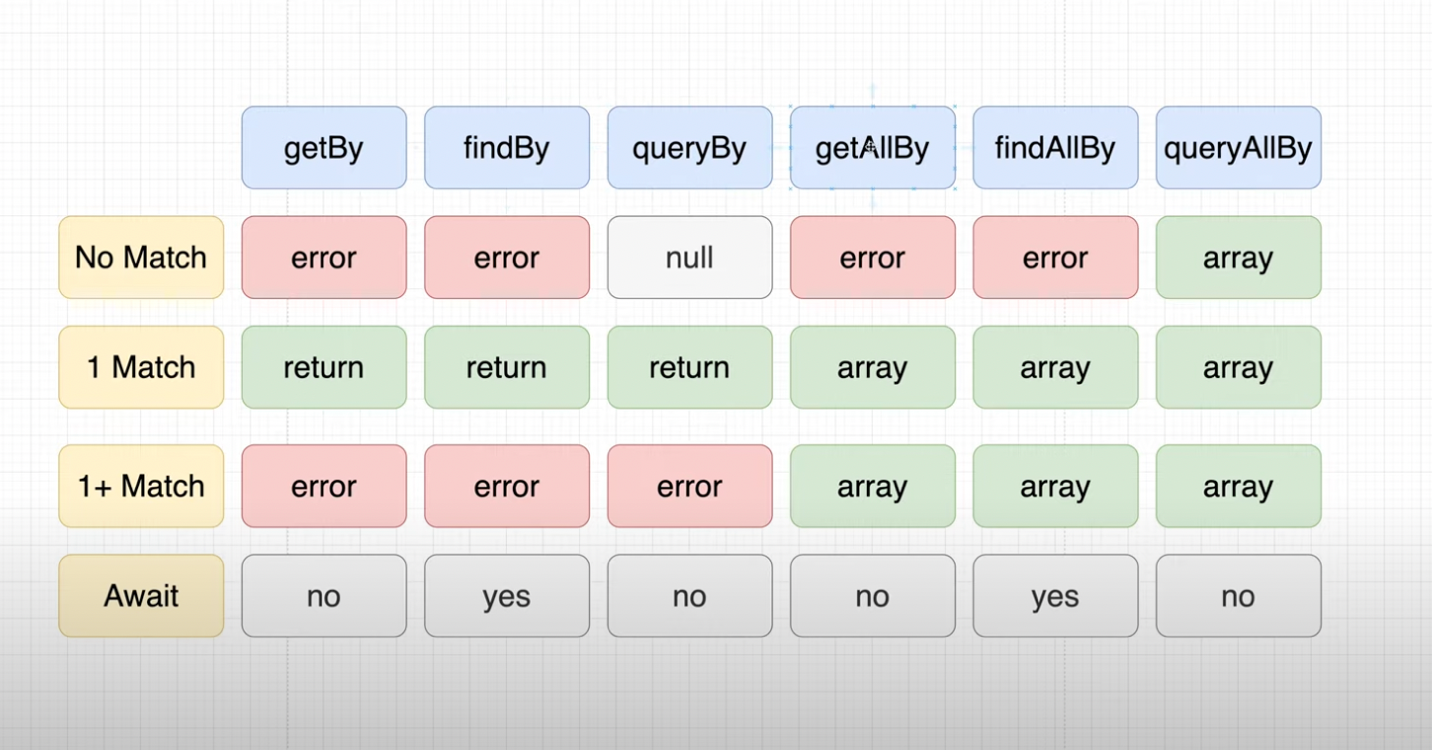
useCallback(() => {...}, [prop]) returns the same function instance as long as prop dependency is the same.

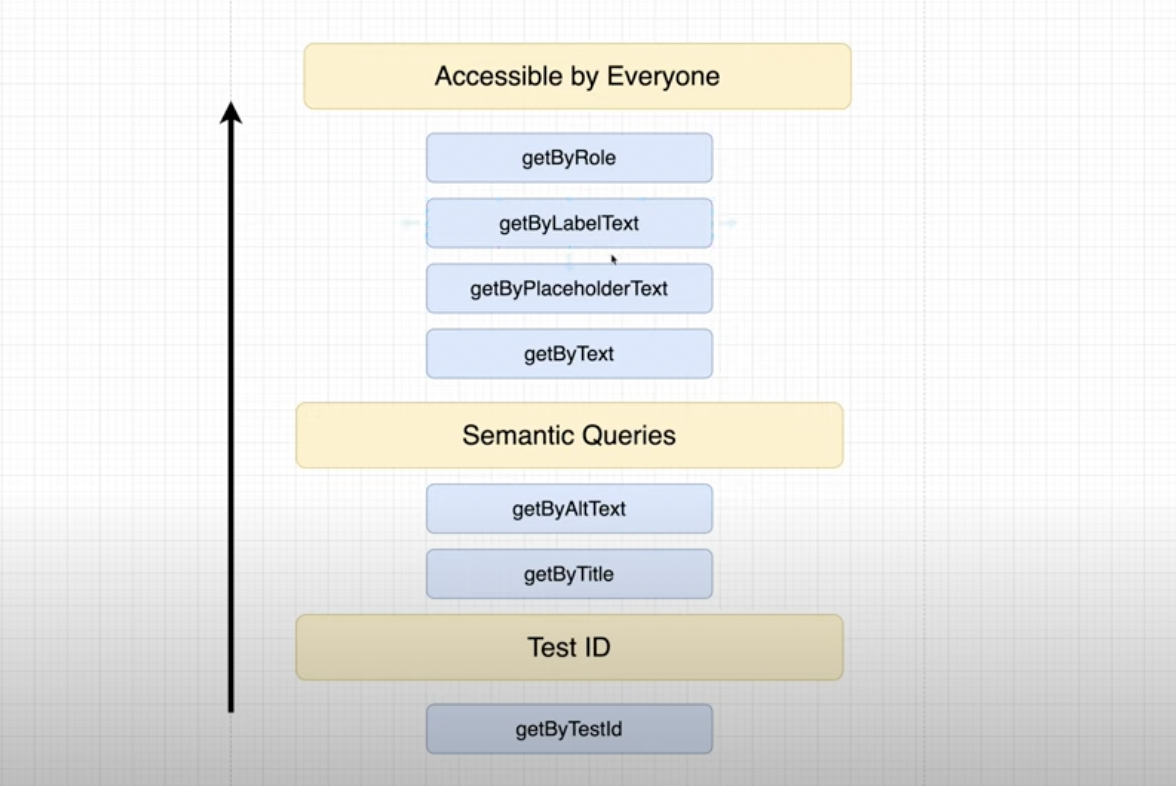
**Difference between useMemo() & useCallback():**

While both useMemo and useCallback remember something between renders until the dependencies change, the difference is just what they remember. **useMemo will remember the returned value from your function**. useCallback will remember your actual function.

**JEST**







**Node Interview Questions**

**What is difference between Callbacks and Promises?**

A key difference between the two is when using the callback approach, we'd normally just pass a callback into a function that would then get called upon completion in order to get the result of something. In promises, however, you attach callbacks on the returned promise object.

**JavaScript Interview questions**

**What is JavaScript?**

JavaScript is a programming language used to develop web applications.

At first JavaScript is mostly used in server side, later node.js came and JavaScript is now using in client side as well.

**What is this keyword in JavaScript?**

**It has different values depending on where it is used:**

In an object method, this refers to the owner object.

Alone, this refers to the global object.

In a function, this refers to the global object.

In a function, in strict mode, this is undefined.

In an HTML event, this refers to the element that received the event.

**What is var, let and const?**

var declarations are globally scoped or function scoped while let and const are block scoped. var variables can be updated and re-declared within its scope; let variables can be updated but not re-declared; const variables can neither be updated nor re-declared. They are all hoisted to the top of their scope.

**What is difference between function and a method in JavaScript?**

Method is which declared inside the object as like a property and function is declared outside the object.

**What are Classes in JavaScript?**

Classes are a feature which basically replace constructor functions and prototypes. You can define blueprints for JavaScript objects with them.

**What are the arrow functions?**

Arrow functions are a different way of creating functions in JavaScript. Besides a shorter syntax, they offer advantages when it comes to keeping the scope of the this keyword.

**Differences & Limitations:**

* Does not have its own bindings to this or super, and should not be used as methods.
* Does not have new.target keyword.
* Not suitable for call, apply and bind methods, which generally rely on establishing a scope.
* Cannot be used as constructors.
* Cannot use yield, within its body.

**JavaScript array functions:**

const a = [3,4,5];

* a.push(6,7) -> add the elements at the end.
* a.unshift(1,2) -> add the elements at the front.
* a.splice(2,0,”a”,”b”)-> add the elements in middle or to remove the elements.
* a.pop() ->To remove an element at the end.
* a.shift() ->To remove an element at the beginning.
* a.concat(numbers) -> to concat to arrays.

**What are Spread & Rest operators?**

The spread and rest operators use the same syntax: ...

Yes, that is the operator - just three dots. Its usage determines whether you're using it as the spread or rest operator.

Rest Operator: If three dots used in function parameter like function name (…args). The rest parameter syntax allows a function to accept an indefinite number of arguments as an array.

The spread operator allows you to pull elements out of an array (=> split the array into a list of its elements) or pull the properties out of an object.

The spread operator is extremely useful for cloning arrays and objects. Since both are reference types (and not primitives), copying them safely (i.e. preventing future mutation of the copied original) can be tricky. With the spread operator you have an easy way of creating a (shallow!) clone of the object or array.

. const oldArray = [1, 2, 3];

. const newArray = [...oldArray, 4, 5]; // This now is [1, 2,3, 4, 5];

Here's the spread operator used on an object:

. const oldObject = {

. name: 'Max'

. };

. const newObject = {

. ...oldObject,

. age: 28

. };

newObject would then be

. {

. name: 'Max',

. age: 28

. }

**What is Destructuring in JavaScript?**

Destructuring allows you to easily access the values of arrays or objects and assign them to variables.

Here's an example for an array:

. const array = [1, 2, 3];

. const [a, b] = array;

. console.log(a); // prints 1

. console.log(b); // prints 2

. console.log(array); // prints [1, 2, 3]

And here for an object:

. const myObj = {

. name: 'Max',

. age: 28

. }

. const {name} = myObj;

. console.log(name); // prints 'Max'

. console.log(age); // prints undefined

. console.log(myObj); // prints {name: 'Max', age: 28}