**React Questions**

1. **What is ReactJS?**

ReactJS is an open-source frontend JavaScript library which is used for building user interfaces especifically for single page applications. It is used for handling view layer for web and mobile apps. React was created by Jordan Walke, a software engineer working for Facebook. ReactJS was first deployed on Facebook’s newsfeed in 2011 and on Instagram.com in 2012.

1. **How does React work?**

React creates a virtual DOM. When state changes in a component it firstly runs a "diffing" algorithm, which identifies what has changed in the virtual DOM. The second step is reconciliation, where it updates the DOM with the results of diff.

1. **What are the advantages of using React?**

It is easy to know how a component is rendered, you just need to look at the render function.

JSX makes it easy to read the code of your components. It is also really easy to see the layout, or how components are plugged/combined with each other.

You can render React on the server-side. This improves SEO and performance.

It is easy to test.

You can use React with any framework you wish as it is only a view layer.

1. **What is the purpose of callback function as an argument of setState?**

The callback function is invoked when setState finished and the component gets rendered. Since setState is asynchronousthe callback function is used for any post action. Note: It is recommended to use lifecycle method rather this callback function.

setState({name: 'sudheer'}, () => console.log('The name has updated and component re-rendered'));

1. **What is the difference between a Presentational component and a Container component?**

Presentational components are concerned with how things look. They generally receive data and callbacks exclusively via props. These components rarely have their own state, but when they do it generally concerns UI state, as opposed to data state.

Container components are more concerned with how things work. These components provide the data and behavior to presentational or other container components. They define actions and provide these as callbacks to the presentational components. They are also often stateful as they serve as data sources.

1. **Why are String Refs legacy?**

If you worked with React before, you might be familiar with an older API where the ref attribute is a string, like "textInput", and the DOM node is accessed as this.refs.textInput. We advise against it because string refs have below issues, are considered legacy, and are likely to be removed in one of the future releases.

It requires that React keeps track of currently rendering component (since it can't guess this). This makes React a bit slower.

It is not composable, i.e. if a library puts a ref on the passed child, the user can't put another ref on it. Callback refs are perfectly composable.

1. **What is React Fiber?**

Fiber is the new reconciliation engine or reimplementation core algorithm in React 16. Its main goal is to enable incremental rendering of the virtual DOM.The goal of React Fiber is to increase its suitability for areas like animation, layout, gestures, ability to pause, abort, or reuse work and assign priority to different types of updates; and new concurrency primitives.

1. **What are the differences between a class component and functional component?**

Class components allows you to use additional features such as local state and lifecycle hooks. Also, to enable your component to have direct access to your store and thus holds state.

When your component just receives props and renders them to the page, this is a 'stateless component', for which a pure function can be used. These are also called dumb components or presentational components.

1. **What is the difference between state and props?**

State is a data structure that starts with a default value when a Component mounts. It may be mutated across time, mostly as a result of user events.

Props (short for properties) are a Component's configuration. They are received from above and immutable as far as the Component receiving them is concerned. A Component cannot change its props, but it is responsible for putting together the props of its child Components. Props do not have to just be data - callback functions may be passed in as props.

1. **What is Lifting State Up in ReactJS?**

When several components need to share the same changing data then it is recommended to lifting the shared state up to their closest common ancestor. For example, if two child components sharing the same data from its parent then move the state to parent instead of maintaining the local state inn both child components.

1. **Name the different lifecycle methods.**

**componentWillMount** (deprecated) - this is most commonly used for App configuration in your root component.

**componentDidMount** - here you want to do all the setup you couldn’t do without a DOM, and start getting all the data you need. Also if you want to set up eventListeners etc. this lifecycle hook is a good place to do that.

**componentWillReceiveProps** (deprecated) - this lifecyclye acts on particular prop changes to trigger state transitions.

**shouldComponentUpdate** - if you’re worried about wasted renders shouldComponentUpdate is a great place to improve performance as it allows you to prevent a rerender if component receives new prop. shouldComponentUpdate should always return a boolean and based on what this is will determine if the component is rerendered or not.

**componentWillUpdate** (deprecated) - rarely used. It can be used instead of componentWillReceiveProps on a component that also has shouldComponentUpdate (but no access to previous props).

**componentDidUpdate** - also commonly used to update the DOM in response to prop or state changes.

**componentWillUnmount** - enables you can cancel any outgoing network requests, or remove all event listeners associated with the component.

1. **Where in a React component should you make an AJAX request?**

componentDidMount is where an AJAX request should be made in a React component. This method will be executed when the component “mounts” (is added to the DOM) for the first time. This method is only executed once during the component’s life. Importantly, you can’t guarantee the AJAX request will have resolved before the component mounts. If it doesn't, that would mean that you’d be trying to setState on an unmounted component, which would not work. Making your AJAX request in componentDidMount will guarantee that there is a component to update.

1. **What is reconciliation?**

When a component’s props or state change, React decides whether an actual DOM update is necessary by comparing the newly returned element with the previously rendered one. When they are not equal, React will update the DOM. This process is called “reconciliation”.

1. **What are controlled components?**

In HTML, form elements such as <input>, <textarea>, and <select> typically maintain their own state and update it based on user input. When a user submits a form the values from the aforementioned elements are sent with the form. With React it works differently. The component containing the form will keep track of the value of the input in it's state and will re-render the component each time the callback function e.g. onChange is fired as the state will be updated. An input form element whose value is controlled by React in this way is called a "controlled component".

1. **What are refs used for in React?**

Refs are used to get reference to a DOM node or an instance of a component in React. Good examples of when to use refs are for managing focus/text selection, triggering imperative animations, or integrating with third-party DOM libraries. You should avoid using string refs and inline ref callbacks. Callback refs are advised by React.

1. **What is a higher order component?**

A higher-order component is a function that takes a component and returns a new component. HOC's allow you to reuse code, logic and bootstrap abstraction. The most common is probably Redux’s connect function. Beyond simply sharing utility libraries and simple composition, HOCs are the best way to share behavior between React Components. If you find yourself writing a lot of code in different places that does the same thing, you may be able to refactor that code into a reusable HOC.

**Exercises**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Write an HOC that reverses it’s input.

Write an HOC that supplies data from an API to it’s Passed Component.

Write an HOC that implements shouldComponentUpdate to avoid reconciliation.

Write an HOC that uses React.Children.toArray to sort the children passed to it's Passed Component.

1. **What advantages are there in using arrow functions?**

**Scope safety:** Until arrow functions, every new function defined its own this value (a new object in the case of a constructor, undefined in strict mode function calls, the base object if the function is called as an "object method", etc.). An arrow function does not create its own this, the this value of the enclosing execution context is used.

1. **Compactness: Arrow functions are easier to read and write.**

**Clarity:** When almost everything is an arrow function, any regular function immediately sticks out for defining the scope. A developer can always look up the next-higher function statement to see what the thisObject is.

1. **How error boundaries handled in React15?**

React15 provided very basic support for error boundaries using the method name "unstable\_handleError". Later this has been renamed as "componentDidCatch" starting from React16 beta release.

1. **Why is it advised to pass a callback function to setState as opposed to an object?**

Because this.props and this.state may be updated asynchronously, you should not rely on their values for calculating the next state.

1. **What is the alternative of binding this in the constructor?**

You can use property initializers to correctly bind callbacks. This is enabled by default in create react app. you can use an arrow function in the callback. The problem here is that a new callback is created each time the component renders.

1. **How would you prevent a component from rendering?**

Returning null from a component's render method means nothing will be displayed, but it does not affect the firing of the component's lifecycle methods.

If the amount of times the component re-renders is an issue, there are two options available. Manually implementing a check in the shouldComponentUpdate lifecycle method hook.

**shouldComponentUpdate(nextProps, nextState){**

**// Do some check here**

**return resultOFCheckAsBoolean**

**}**

Or using React.PureComponent instead of React.Component React.PureComponent implements shouldComponentUpdate()with a shallow prop and state comparison. This enables you to avoid re-rendering the component with the same props and state.

1. **When rendering a list what is a key and what is it's purpose?**

Keys help React identify which items have changed, are added, or are removed. Keys should be given to the elements inside the array to give the elements a stable identity. The best way to pick a key is to use a string that uniquely identifies a list item among its siblings. Most often you would use IDs from your data as keys. When you don't have stable IDs for rendered items, you may use the item index as a key as a last resort. It is not recommend to use indexes for keys if the items can reorder, as that would be slow.

1. **What is the purpose of super(props)?**

A child class constructor cannot make use of this until super() has been called. Also, ES2015 class constructors have to call super() if they are subclasses. The reason for passing props to super() is to enable you to access this.props in the constructor.

1. **What is JSX?**

JSX is a syntax extension to JavaScript and comes with the full power of JavaScript. JSX produces React "elements". You can embed any JavaScript expression in JSX by wrapping it in curly braces. After compilation, JSX expressions become regular JavaScript objects. This means that you can use JSX inside of if statements and for loops, assign it to variables, accept it as arguments, and return it from functions:

1. **What is equivalent of the following using React.createElement?**

**Question:**

const element = (

<h1 className="greeting">

Hello, world!

</h1>

);

**Answer:**

**const element = React.createElement(**

'h1',

{className: 'greeting'},

'Hello, world!'

**);**

1. **What is Children?**

In JSX expressions that contain both an opening tag and a closing tag, the content between those tags is passed to components automatically as a special prop: props.children.

There are a number of methods available in the React API to work with this prop. These include React.Children.map, React.Children.forEach, React.Children.count, React.Children.only, React.Children.toArray.

1. **What is state in react?**

State is similar to props, but it is private and fully controlled by the component. State is essentially an object that holds data and determines how the component renders and behaves.

Why would you eject from create-react-app?

Until you eject you are unable to configure webpack or babel presets.

1. **What is redux?**

The basic idea of redux is that the entire application state is kept in a single store. The store is simply a javascript object. The only way to change the state is by sending actions from your application and then writing reducers for these actions that modify the state. The entire state transition is kept inside reducers and should not have any side-effects.

1. **What is a store in redux?**

The store is a javascript object that holds application state. Along with this it also has the following responsibilities:

Allows access to state via getState();

Allows state to be updated via dispatch(action);

Registers listeners via subscribe(listener);

Handles unregistering of listeners via the function returned by subscribe(listener).

1. **What is an action?**

Actions are plain javascript objects. They must have a type indicating the type of action being performed. In essence, actions are payloads of information that send data from your application to your store.

1. **What is a reducer?**

A reducer is simply a pure function that takes the previous state and an action, and returns the next state.

1. **What is Redux Thunk used for?**

Redux thunk is middleware that allows you to write action creators that return a function instead of an action. The thunk can then be used to delay the dispatch of an action if a certain condition is met. This allows you to handle the asynchronous dispatching of actions.

1. **What is a pure function?**

A pure function is a function that doesn't depend on and doesn't modify the states of variables out of its scope. Essentially, this means that a pure function will always return the same result given same parameters.