**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.

**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.

#### 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.

#### 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.

**What are 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.

#### React Hooks

Hooks let you use more of React’s features without having to use classes. The first hook that you will most likely encounter is useState. useState is a Hook that lets you add React state to function components. It returns an array with a getter and a setter. The syntax looks like const [count, setCount] = React.useState(0);. The equivalent when using a class component would be.

this.state = {

count: 0

};

<button onClick={() => this.setState({ count: this.state.count + 1 })}>

Increase Count

</button>

The next hook you will most likely encounter is useEffect. The Effect Hook lets you perform side effects in function components. By passing an empty array as the second argument to useEffect is equivalent to using componentDidMount. If you pass a value to the array it will only call the useEffect function when the value in the array updates.

useEffect(() => {

// do stuff when the component mounts

}, []);

#### 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".

#### 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.

#### 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.

**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.
* 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 Object is.

#### 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.

#### 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.

#### 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:

#### 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.

#### 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.

**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.

**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).

**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.

**What is a reducer?**

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

#### 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.

#### 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.