1. **How do you decide when to split a component into subcomponents?**

- When I find a reusable section of code I decide to extract it into subcomponents to be shared by other components.
- When I find a component becomes too complex to handle multiple tasks, I decide to split a component into subcomponents, letting each focus on a single task.

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

- Props can be passed from parent component to child component, but the state can only be used within the same component.
- State is mutable by setState or useState, but props is immutable since the child component cannot change the props it received from the parent component.

3. **How to trigger rerender in a component?**

- setState method in class component and useState method in functional component to change current components' states or context value (if it exists), or props that passed (as a state in parent component) to the child components
- forceUpdate methods to force re-render a component

4. **Why do you like react over other front-end libraries and frameworks?**

- Angular use **Typescript** adding static types to Javascript make it steeper learning curve
- Angular uses a **Real Dom** but React uses a Virtual dom which gives better performance.
- Angular's **Component** consists of 4 separate files for HTML, CSS, Typescript, and Test file, but React's Component uses **JSX** allows for seamless integration of HTML and Javascript in a single file, leading to more boilerplate and more flexibility.
- Angular uses **NgModule** to import other modules and declare and inject components, directives, and services, and **DI** for allowing injection of services to other components, but React uses Javascript module (ES6 import/export) making less boilerplate and learning curve
- Angular use **Directive** (structural directive: *ngIf -> conditions logic, *ngFor -> looping logic, and attribute directive: ngClass -> add/remove class conditionally , ngStyle -> apply inline style conditionally), but React use JSX and conditional rendering to manipulate DOM, making it less learning curve
- Angular uses **Services** with RxJS for state management (introducing new concepts of observables, operators, and subscription management) but React uses 'useState', 'useContext' and 'useReducer' to follow the Javascript paradigm. Angular's RxJS for

asynchronous operations can be done with Promise in React, which has less learning curve.

5. **What's the difference between controlled components and uncontrolled components?**

In React, controlled and uncontrolled components are two approaches to handle form input elements. Controlled components use the component's state to manage the input value, updating it through event handlers like onChange. This allows for easy validation and manipulation because the value is stored in the state, making it straightforward to implement logic directly within the event handler. For example, you can validate the input, trim whitespace, or convert it to uppercase before processing further. Since the state and DOM are synced, this approach is more consistent and predictable. In contrast, uncontrolled components use refs to access the input value directly from the DOM. This means the value isn't synced with the component's state, leading to less predictability and more manual DOM manipulation."

6. **How to prevent components from unnecessary rerendering?**

- Use React.memo which is a higher order component that memorizes the rendered output of a functional component, which makes React to skip rendering the component if its props have not changed.
- Use shouldComponentUpdate for class component, and customize and override it to return false if the component need to skip rerendering
- Use React.PureComponent for class components. It is a base class in React that extends React.Component and comes with a built-in implementation of the shouldComponentUpdate lifecycle method that performs shallow comparison (check value of primitive and reference of state and props) of the components' props and state, if the state or props have not changed, the component re-render will be skipped.
- Use useCallback and useMemo hooks to memorize function and values, preventing re-creation on every renders

7. **Why are props needed to be immutable?**

- Unidirectional Data Flow: React let's data flow from parent component to child components, by making immutable props ensure that child components cannot alter the data, maintaining a clear and predictable data flow, making it clear about how data changes and evolves over time.

- Predictability: Since props are immutable, giving the same input props produce the same output, make debugging and reasoning about the application easier since we know data passed to a component will not be modified unexpectedly.
- Avoiding Side Effects: If a child component alters its props, it could inadvertently alter the state or behavior of its parent component, leading to unpredictable behaviors. Also allow functional components to be seen as pure functions that merely focus on transforming input data (props) into UI elements without side effects , making it easier to debug and reason about the application.

8. Explain the Virtual DOM and how React uses it to improve performance.

- Virtual dom is a representation of real dom, which is saved in memory, synced with real dom.
- Its workflow: Whenever updates happen (either state or props changes), it generates a new version of virtual dom, compare with it using diffing algorithms, and React will update the real dom based on the newer virtual dom, where changes happen only, then delete the older virtual dom when update completed.
- How it improves performance:
- 1. Diffing algorithms can quickly identify where changes happened.
- 2. The Reconciliation process only updates parts that are different instead of re-render the entire dom
- 3. Batch updates in React groups multiple state or props changes together and apply them in a single update cycle instead of updating the real DOm immediately after each change.

9. Can you explain the useMemo and useCallback hooks and provide examples of when you might use them?

- useMemo hook is used to memorize the result of a function so that it is only recomputed when its dependencies change, which helps to optimize performance by avoiding unnecessary calculations on every render. We use useMemo when we have a computationally expensive function and want to avoid recalculating the result on every render unless its dependencies have changed.
- useCallback hook is used to memorize a function reference so that it is only recreated when its dependencies change, which helps to prevent unnecessary re-renders when passing callbacks to child components. We use useCallback when we pass a function to the child component and want to prevent the child component from re-rendering unnecessary due to a new function reference being created on each render.

10. Explain the concept of Higher-Order Components (HOCs) and provide an example use case.

- A Higher-Order Component (HOC) is used for reusing component logic. HOCs are functions that take a component and return a new component with enhanced capabilities.
- Example use case:
-
- ```
  import React from 'react';
  
  const withSimpleMessage = (Component) => (props) => (

      This is a simple message!
      <Component {...props} />

  );
  export default withSimpleMessage;
  
  const SimpleComponent = () => I'm the wrapped component! ;
  export default SimpleComponent;
  
  import React from 'react';
  import withSimpleMessage from './withSimpleMessage';
  import SimpleComponent from './SimpleComponent';
  
  contact EnhancedComponent = withSimpleMessage(SimpleComponent);
  
  const App = () => (

      <EnhancedComponent />

  );
  
  export default App;
  ```
  ```

11. Discuss the differences between React's class components and functional components. Which one do you prefer and why?

- React's class components and functional components differ primarily in their syntax and how they manage state and lifecycle methods. Class components use ES6 syntax, manage state with this.state, and utilize lifecycle methods with componentDidMount, componentDidUpdate, and componentWillUnmount. Functional components are defined as palin Javascript functions, manage state using hooks like useState, and handle side effects with useEffect.
- Functional components are typically simpler and more concise, since they reduce the overhead associated with class syntax and lifecycle methods. I prefer functional components since:

- Functional components 'useState' hook API is more intuitive than setState in class components.
- Functional components avoid the usage of 'this' keyword and binding methods to component's instance leading to less boilerplate.
- Functional components use 'useEffect' to handle side effects while class components use 'componentDidMount', 'componentDidupdate', and 'componentWillUnMount' to handle side effect which are more larger, monolithic methods to handle, while useEffect can be break down into multiple effects.

12. How do you ensure your code is maintainable and scalable?

- Modular design: break down application into small and reusable components, making it easier to read and debug.
- Clear and Consistent naming convention: Use descriptive and consistent naming convention for variables, function, and components, making it better readability for others to understand the code.
- Consistent Styling Convention: ensure a uniform codebase and reduces stylistic discrepancies.
- Responsive Design: Ensure application is responsive on different devices.
- Documentation: Use clear documentation to explain complex logic and components logic
- Code Review: Participate in the code review in the early stage to catch potential errors early, making sure the coding standards are met and sharing knowledge with the team.
- Testing: Use unit tests and integration tests to ensure code correctness.
- Performance Optimization: use memorization or efficient state management technique