**REACT**

1. What is React? What are its features?
   * **React** is a JavaScript library for building user interfaces, primarily for single-page applications. It allows developers to create reusable UI components and manage the state of their applications efficiently.
   * **Features of React**
     1. **Component-Based Architecture**: React encourages building UI components that can be reused across the application. Each component manages its own state and can be composed to create complex UIs.
     2. **Virtual DOM**: React uses a virtual DOM to optimize updates and rendering. When the state of a component changes, React updates the virtual DOM first, then compares it with the real DOM, and only updates the parts of the DOM that have changed. This improves performance.
     3. **JSX (JavaScript XML)**: JSX is a syntax extension for JavaScript that looks similar to HTML. It allows developers to write HTML-like code within JavaScript, making it easier to create and understand the structure of UI components.
     4. **Unidirectional Data Flow**: React follows a unidirectional data flow, meaning data flows in one direction from parent to child components. This makes it easier to understand and debug the application state.
     5. **Declarative UI**: React allows developers to describe what the UI should look like based on the current state. React takes care of updating the UI when the state changes, making the code more predictable and easier to debug.
     6. **State Management**: React provides a way to manage the state of components using the useState hook for functional components and this.state for class components. For more complex state management, libraries like Redux or Context API can be used.
     7. **Lifecycle Methods**: React provides lifecycle methods (e.g., componentDidMount, componentDidUpdate, componentWillUnmount) that allow developers to run code at specific points in a component's lifecycle.
     8. **Hooks**: React Hooks (e.g., useState, useEffect, useContext) allow developers to use state and other React features in functional components, making it easier to write and manage component logic.
     9. **React Native**: React Native is a framework for building mobile applications using React. It allows developers to use the same principles and components to build native mobile apps for iOS and Android.
     10. **Ecosystem and Community**: React has a large ecosystem of libraries, tools, and a strong community. This makes it easier to find solutions, get support, and integrate with other technologies.

1. What are the Advantages of React?
2. **Declarative UI**: React's declarative approach makes it easier to understand and predict the UI state. Developers describe what the UI should look like based on the current state, and React handles the updates, reducing the complexity of manual DOM manipulation.
3. **Reusable Components**: React promotes the creation of reusable components, which can be shared across different parts of an application or even across different projects. This reduces redundancy and speeds up development.
4. **Strong Community and Ecosystem**: React has a large and active community, which means there are plenty of resources, tutorials, libraries, and tools available. This makes it easier to find solutions to problems and integrate with other technologies.
5. **SEO-Friendly**: React can be rendered on the server using server-side rendering (SSR) with frameworks like Next.js. This improves the SEO of web applications by providing fully rendered HTML to search engines.
6. **Unidirectional Data Flow**: React's unidirectional data flow makes it easier to understand how data changes in the application. This leads to more predictable and easier-to-debug code.
7. **Rich Ecosystem of Developer Tools**: React has a rich set of developer tools, including the React Developer Tools extension for browsers, which helps in inspecting React component hierarchies, state, and props.
8. **React Native**: React Native allows developers to build mobile applications using React. This enables code reuse between web and mobile applications, reducing development time and effort.
9. **Flexibility**: React is a library, not a full-fledged framework, which means it can be integrated with other libraries or frameworks as needed. This provides flexibility in choosing the best tools for specific tasks.
10. **Backward Compatibility**: React maintains a strong commitment to backward compatibility, ensuring that updates and new features do not break existing applications. This stability is crucial for long-term projects.
11. **Performance Optimization**: React provides various performance optimization techniques, such as code splitting, lazy loading, and memoization, to improve the performance of applications.
12. **Easy to Learn**: React's simple API and component-based architecture make it relatively easy to learn, especially for developers with a background in JavaScript.
13. What are SOLID principles of React?
    * The **SOLID principles** are a set of design principles that can be applied to object-oriented programming and software design to create more maintainable and scalable code. While these principles are not specific to React, they can be applied to React development to improve the quality of your components and application architecture.
      1. **Single Responsibility Principle (SRP):** A class or component should have only one reason to change, meaning it should have only one job or responsibility.
         + **Application in React:** Each React component should focus on a single piece of functionality. For example, a component that fetches data should not also handle UI rendering. Instead, separate concerns into different components.
      2. **Open/Closed Principle (OCP):** Software entities should be open for extension but closed for modification.
         + **Application in React**: Design components and functions in a way that allows them to be extended without modifying their existing code. This can be achieved through composition and higher-order components (HOCs).
      3. **Liskov Substitution Principle (LSP)**: Objects of a superclass should be replaceable with objects of a subclass without affecting the correctness of the program.
         + **Application in React:** Ensure that components can be replaced with other components that adhere to the same interface or contract. This is often achieved through prop types and consistent component APIs.
      4. **Interface Segregation Principle (ISP)**: Clients should not be forced to depend on interfaces they do not use.
         + **Application in React**: Design components with a clear and minimal API. Avoid passing unnecessary props and ensure that components only receive the data they need.
      5. **Dependency Inversion Principle (DIP)**: High-level modules should not depend on low-level modules. Both should depend on abstractions.
         + **Application in React**: Use dependency injection and context to manage dependencies. This allows components to be decoupled from specific implementations and makes them more testable.
14. What is the file structure for React?
15. **public/**: Contains static files like index.html, images, and other assets that need to be served directly.
16. **src/**: The main source directory for your React application.
    1. **assets/**: Contains static assets like images and stylesheets.
    2. **components/**: Contains reusable UI components. Each component can have its own directory if it includes related files like styles or tests.
    3. **pages/**: Contains components that represent different pages or views in your application. These are typically used with a router.
    4. **services/**: Contains modules for handling API calls or other services. This helps in separating business logic from UI components.
    5. **utils/**: Contains utility functions and helper modules that can be used throughout the application.
    6. **App.js**: The root component of your application. It typically contains the main layout and routing logic.
    7. **index.js**: The entry point of your application. It renders the App component into the DOM.
17. **.gitignore**: Specifies files and directories that should be ignored by Git.
18. **package.json**: Contains metadata about the project, including dependencies, scripts, and other configurations.
19. **README.md**: A markdown file that provides an overview of the project, how to set it up, and how to use it.
20. What are coding standard in React?

* Adhering to coding standards in React helps maintain consistency, readability, and maintainability of the codebase. Here are some common coding standards and best practices for React development:

1. **Component Naming**: Use PascalCase/CamelCase for component names.
2. **File and Folder Structure**: Organize files and folders logically, typically by feature or functionality.
3. **JSX Syntax**: Use self-closing tags for elements without children.
4. **Component Structure**: Keep components small and focused on a single responsibility.
5. **Prop Types**: Use PropTypes or TypeScript to define and validate props.
6. **State Management**: Use hooks like useState and useReducer for managing state in functional components.
7. **Effect Management**: Use useEffect for side effects like data fetching, subscriptions, or manually changing the DOM.
8. **Styling**: Use CSS Modules, styled-components, or other CSS-in-JS solutions for scoped and maintainable styles.
9. **Code Formatting**: Use a code formatter like Prettier to enforce consistent code style.
10. **Linting**: Use ESLint with a React-specific configuration to catch common issues and enforce coding standards.
11. **Testing**: Write unit tests for components using testing libraries like Jest and React Testing Library.
12. How is React different from React Native?

* React is used for building web applications, while React Native is used for building native mobile applications.
* React uses HTML and CSS for rendering and styling, whereas React Native uses native components and JavaScript objects for styling.
* Both share the same core principles and syntax, making it easier for developers to transition between web and mobile development.
* Some code can be shared between React and React Native using libraries like React Native Web.

|  |  |  |
| --- | --- | --- |
| **Features** | **React** | **React Native** |
| **Purpose** | * A JavaScript library for building user interfaces, primarily for web applications. * Focuses on creating reusable UI components for web development. | * A framework for building native mobile applications using React. * Allows developers to use React along with native platform capabilities to build mobile apps for iOS and Android. |
| **Platform** | * Runs in the browser. * Uses HTML, CSS, and JavaScript to build web applications. | * Runs on mobile devices (iOS and Android). * Uses native components instead of web components to build mobile applications. |
| **Rendering** | * Uses the virtual DOM to efficiently update the web page. * Renders HTML elements. | * Uses a bridge to communicate with native modules and render native components. * Renders native UI components (e.g., View, Text, Image). |
| **Styling** | * Uses CSS or CSS-in-JS solutions for styling. | * Uses a JavaScript object to define styles, similar to inline styles in web development. |
| **Navigation** | * Uses libraries like React Router for client-side routing. | * Uses libraries like React Navigation for navigation and routing. |
| **Development Tools** | * Uses tools like Webpack, Babel, and Create React App for development and build processes. | * Uses tools like Metro bundler and Expo for development and build processes. |
| **Code Sharing** | * Code is specific to web applications and cannot be directly used in mobile applications. | * Allows sharing of business logic and some UI components between iOS and Android applications. |

1. What is Babel?

* **Babel** is a JavaScript compiler that allows developers to write modern JavaScript code that can run in older browsers or environments that do not support the latest JavaScript features. It is an essential tool in the modern JavaScript development workflow, especially when using frameworks and libraries like React.
* **Key Features of Babel:**

1. **Transpiling Modern JavaScript**: Babel converts ECMAScript 2015+ (ES6+) code into a backward-compatible version of JavaScript that can run in older browsers and environments.
2. **JSX Transformation**: Babel can transform JSX syntax used in React into standard JavaScript.
3. **Plugins and Presets**: Babel's functionality can be extended using plugins and presets. Plugins are used to transform specific syntax, while presets are collections of plugins.
4. **Polyfills**: Babel can include polyfills for new JavaScript features that are not natively supported in older environments. This is often done using the @babel/polyfill package or core-js.
5. **Customizable**: Babel is highly customizable, allowing developers to configure it to suit their specific needs. This includes specifying which plugins and presets to use, and configuring how code should be transformed.
6. **Usage**: Babel can be used from the command line, integrated into build tools like Webpack, or used with task runners like Gulp.
7. What is JSX? How does the browser read it?

* **JSX (JavaScript XML)** is a syntax extension for JavaScript that allows you to write HTML-like code within JavaScript. It is primarily used with React to describe what the UI should look like. JSX makes it easier to write and understand the structure of React components by combining HTML and JavaScript in a single file.
* **Key Features of JSX**

1. **HTML-like Syntax**: JSX allows you to write HTML-like tags directly within JavaScript code.
2. **Embedding Expressions**: You can embed JavaScript expressions within JSX using curly braces {}.
3. **Attributes**: JSX allows you to use attributes similar to HTML attributes, but with camelCase naming for properties.
4. **Children**: JSX can contain children elements, allowing you to nest elements within each other.

* Browsers do not natively understand JSX. Therefore, JSX needs to be transformed into regular JavaScript before it can be executed by the browser. This transformation is typically done using a tool like Babel. This transformation process involves converting JSX into React.createElement calls, which React then uses to create and render elements in the DOM.

1. What are Fragments in React?

* **Fragments** in React provide a way to group multiple elements without adding extra nodes to the DOM. They help in returning multiple elements from a component's render method and avoiding unnecessary wrapper elements that can affect the layout and styling. React offers both a full syntax (React.Fragment) and a shorthand syntax (<>...</>) for using fragments, making it easy to integrate them into your components.

1. Explain the building blocks of React?

* The five main building blocks of React are:
  1. **Components**: These are reusable blocks of code that return HTML.
  2. **JSX**: It stands for JavaScript and XML and allows you to write HTML in React.
  3. **Props and State**: props are like function parameters and State is similar to variables.
  4. **Context**: This allows data to be passed through components as props in a hierarchy.
  5. **Virtual DOM**: It is a lightweight copy of the actual DOM which makes DOM manipulation easier.

1. What are the components in React? How are they different in components from others?

* **Components** are the building blocks of a React application. They allow you to split the UI into independent, reusable pieces that can be managed separately. Each component can have its own state and lifecycle methods, making it easier to build and maintain complex user interfaces.
* **Types of Components in React**
  1. **Functional Components:** Functional components are simple JavaScript functions that accept props as an argument and return React elements. They are also known as stateless components because they do not manage their own state (before the introduction of hooks).
  2. **Class Components (Older Method):** Class components are ES6 classes that extend from React.Component. They can manage their own state and have access to lifecycle methods.

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **React Components** | **Angular Components** | **Vue Components** |
| **Definition** | Functions or Classes | TypeScript Classes | Objects or .vue files |
| **Rendering** | JSX (JavaScript + HTML) | HTML Templates | HTML Templates |
| **State Mgmt.** | useState, useReducer, Context API | Services, RxJS, useState | data(), Vuex, Pinia |
| **Lifecycle** | useEffect, componentDidMount | ngOnInit, ngOnDestroy | created(), mounted() |
| **Reusability** | Highly reusable with hooks and props | Uses modules and dependency injection | Uses Single File Components (SFC) |
| **Performance** | Virtual DOM improves rendering | Two-way data binding may cause extra processing | Virtual DOM similar to React |

1. How to create components in React?

* **Components** in React are the fundamental building blocks used to create the user interface. They can be classified into two main types: functional components and class components. Functional components are simpler and can use hooks to manage state and side effects, while class components are more traditional and can manage state and lifecycle methods directly. Creating components in React involves defining them as functions or classes, optionally accepting props, and returning React elements that describe the UI.

1. Why we use keys when creating list of components?

* **Keys** are a special attribute you need to include when creating lists of elements in React. They help React identify which items have changed, are added, or are removed, which is crucial for efficiently updating the UI.
* Importance of Keys:
  1. **Efficient Updates**: Keys help React optimize the rendering process by allowing it to quickly determine which items in a list have changed. This minimizes the number of DOM operations and improves performance.
  2. **Stable Identity**: Keys provide a stable identity for each element in a list. This is important for maintaining the state of components across renders.
  3. **Avoiding Re-renders**: Without keys, React would re-render all items in a list whenever the list changes. With keys, React only re-renders the items that have changed, improving performance.

1. What are class components and functional components?

* **Class Components (Older Approach):** Class components are ES6 classes that extend React.Component. They use a render() method to return JSX. They have access to lifecycle methods and can manage their own state.
* **Key Features of Class Components**
  1. Can have state (this.state)
  2. Uses lifecycle methods (componentDidMount, componentDidUpdate)
  3. Requires this.props to access props
  4. More verbose compared to functional components
* **Functional Components (Modern Approach):** Functional components are JavaScript functions that return JSX. They are lighter, faster, and easier to read.
* **Key Features of Functional Components**
  1. Simpler syntax (just a function)
  2. Uses React Hooks for state (useState) and lifecycle (useEffect)
  3. No need for this.props or this.state

1. Which is better functional or class component? Why?

* **Functional component** is better as they are simpler, faster, and the standard approach in modern React.
* **Avoid Class Components** – They are more complex and are being replaced by Hooks.

1. What is Controlled and Uncontrolled component?

* **Controlled Component**: A controlled component is a component where React controls the form data. The form data is handled by the state within the component, and any changes to the form data are managed through event handlers.
  1. Form data is managed by React state.
* **Uncontrolled Component**: An uncontrolled component is a component where the form data is handled by the DOM itself. Instead of using state to manage the form data, you use a ref to access the form values directly from the DOM.
  1. Form data is managed by the DOM.

1. What is state-full and state-less component?

* **Stateful Component**: Manages its own state using useState or this.state.
* **Stateless Component**: Does not manage its own state; relies on props for data and behavior.

1. What are the different phases of the component lifecycle? (lifecycle methods)

* These phases can be categorized into three main stages: Mounting, Updating, and Unmounting.
  1. **Mounting**: This phase occurs when a component is being created and inserted into the DOM.
     + **constructor**(): Called before the component is mounted. Used to initialize state and bind event handlers.
     + **static getDerivedStateFromProps**(): Called right before rendering. Used to update the state based on props.
     + **render**(): The only required method in a class component. Returns the JSX to be rendered.
     + **componentDidMount**(): Called after the component is mounted. Used for side effects like fetching data.
  2. **Updating**: This phase occurs when a component is being re-rendered due to changes in props or state.
     + **static getDerivedStateFromProps**(): Called right before rendering. Used to update the state based on props.
     + **shouldComponentUpdate**(): Determines if the component should re-render. Used for performance optimization.
     + **render**(): Returns the JSX to be rendered.
     + **getSnapshotBeforeUpdate**(): Called right before the DOM is updated. Used to capture some information from the DOM.
     + **componentDidUpdate**(): Called after the component is updated. Used for side effects like fetching data based on previous props or state.
  3. **Unmounting**: This phase occurs when a component is being removed from the DOM.
     + **componentWillUnmount**(): Called right before the component is unmounted. Used for cleanup like removing event listeners or canceling network requests.
  4. **Error Handling**: These methods are called when there is an error during rendering, in a lifecycle method, or in the constructor of any child component.
     + **static getDerivedStateFromError(error)**: Called when an error is thrown in a descendant component. Used to update state to display an error boundary.
     + **componentDidCatch(error, info)**: Called when an error is thrown in a descendant component. Used to log error information.

1. How do you implement mount, update and unmount with functional components?

* **Mounting**: Use useEffect with an empty dependency array.
* **Updating**: Use useEffect with specific dependencies.
* **Unmounting**: Return a cleanup function from useEffect.

1. What are Higher Order Component (HOC)?

* **A higher-order component** is a function that takes a component as an argument and returns a new component that wraps the original component.
* **Key Points:**
  1. **Reusability**: HOCs allow you to reuse logic across multiple components.
  2. **Composition**: HOCs are a way to compose components for enhanced functionality.
  3. **Props**: HOCs pass props through to the wrapped component, allowing it to function.
* **Common Use Cases**:
  1. **Code Reuse**: Sharing common functionality between components.
  2. **Conditional Rendering**: Adding conditional logic to components.
  3. **Enhancing Props**: Modifying or adding props to components.
  4. **Handling State**: Managing state in a reusable way.

1. What are HOC and functional architecture? Please elaborate and explain it with example?

* **Higher Order Components (HOC)**: A function that takes a component and returns a new component with additional functionality.
* **Functional Architecture**: Building applications using functional components and hooks.
* **Combining HOC and Functional Architecture**: You can enhance functional components with HOCs to add reusable logic.

1. What is React.Memo? Where we use React.Memo?

* **React.memo()** is a higher-order component (HOC) that optimizes functional components by preventing unnecessary re-renders. It memoizes the component, meaning it only re-renders when its props change.
* **Why Use React.memo()?**
  1. By default, React re-renders a component every time its parent re-renders, even if the props are the same. React.memo() skips re-rendering if the props remain unchanged, improving performance.
* **When to Use React.memo()?**
  1. When a component always receives the same props.
  2. When a component is expensive to render (e.g., complex UI, charts, large lists).
  3. When optimizing performance in large applications.

1. What will be the benefits of using functional component in dev vs prod mode?

* **Benefits in Development Mode**
  1. **Simplicity and Readability**: Functional components are typically simpler and more concise than class components, making the code easier to read and understand.
  2. **Easier Debugging**: With fewer lifecycle methods and a more straightforward structure, functional components can be easier to debug.
  3. **Hooks**: Hooks like useState, useEffect, and custom hooks provide a powerful and flexible way to manage state and side effects, leading to cleaner and more maintainable code.
  4. **Stateless by Default**: Functional components are stateless by default, which encourages a more functional programming approach and reduces the likelihood of bugs related to state management.
  5. **Hot Reloading**: Functional components work well with hot reloading, allowing developers to see changes in real-time without losing the component's state.
* **Benefits in Production Mode**
  1. **Performance**: Functional components can be more performant than class components because they avoid the overhead of class instantiation and the this binding.
  2. React can optimize functional components better, especially with the use of hooks and memoization (React.memo).
  3. **Smaller Bundle Size**: Functional components can lead to a smaller bundle size, which improves load times and overall performance of the application.
  4. **Tree Shaking**: Functional components are more amenable to tree shaking, a technique used to eliminate dead code, resulting in smaller and more efficient bundles.
  5. **Consistency**: Using functional components consistently across the codebase can lead to more predictable and maintainable code, reducing the risk of bugs and improving long-term maintainability.

1. What is States and Props?

* **State**
  1. **Definition**: State is a built-in object that stores property values that belong to a component. When the state object changes, the component re-renders.
  2. **Usage**: State is used to manage data that can change over time and affect the rendering of the component.
  3. **Mutability**: State is mutable, meaning it can be changed using the setState method in class components or the useState hook in functional components.
* Props
  1. **Definition**: Props (short for properties) are read-only attributes passed from a parent component to a child component. They allow data to flow from parent to child.
  2. **Usage**: Props are used to pass data and event handlers to child components.
  3. **Immutability**: Props are immutable, meaning they cannot be changed by the child component.

1. What is difference b/w useState and Props?

* **State**:
  1. Managed within the component.
  2. Can be changed by the component itself.
  3. Used for data that changes over time.
* **Props**:
  1. Passed from parent to child components.
  2. Cannot be changed by the receiving component.
  3. Used to pass data and event handlers to child components.

1. How to pass data between Components?
2. How to pass state data from one component to another component?

* Parent to Child (via props)
* Child to Parent (via callback functions)
* Sibling to Sibling
* Using Context API
* Using State Management Libraries

1. What is prop drilling? What is its disadvantage and how to fix?

* **Prop drilling** is a term used to describe the process of passing data from a parent component to a deeply nested child component through multiple layers of intermediate components. This often involves passing props down through several levels of the component tree, even if some of those intermediate components do not need the data themselves.

1. Differentiate between real DOM and virtual DOM?
2. What is virtual DOM? How it works? What is the purpose of creating?
3. What is shadow DOM? How it works?
4. What are hooks in React? Give a brief example? What are rules of hooks?
5. What is custom hooks in React?
6. What are life cycle hooks and life cycle methods?
7. What are class based and function based life cycle hooks?
8. What are two most used lifecycle method in hooks?
9. What is State management?
10. What State management we used in react and explain why we use them?
11. What are State Management Libraries and how we use them?
12. What is the use of ref in React?
13. What is useState. Why we can't use let or var instead of useState?
14. What is the difference b/w useState and useReducer?
15. What is difference b/w useRef, createRef in React?
16. What is useEffect in React?
17. What are the use of arguments for useEffect, how they work?
18. What dependency is passed in useEffect?
19. If we pass an object to useEffect how does it behave.
20. What is Pure Component?

* A **Pure Component** in React is a class component that automatically implements shouldComponentUpdate() for performance optimization. It only re-renders when there is a change in state or props, preventing unnecessary updates.
* **Key Features of Pure Component:**
  1. **Shallow Comparison**: Pure Component performs a shallow comparison of the current and next props and state. If there are no changes, the component will not re-render.
  2. **Performance Optimization**: By avoiding unnecessary re-renders, Pure Component can improve the performance of your application, especially when dealing with complex or frequently updated components.
  3. **Usage with Functional Components**: For functional components, you can achieve similar behavior using React.memo, which memoizes the component and performs a shallow comparison of props.
* **Advantages of Pure Components**
  1. **Improves Performance**: Avoids unnecessary re-renders.
  2. **Automatic Optimization**: No need to manually implement shouldComponentUpdate().
  3. **Useful for Large Applications**: Reduces the number of renders in complex UI.
* **Limitations of PureComponent**
  1. **Shallow Comparison**: PureComponent performs a shallow comparison, which means it only compares the first level of properties. If your props or state contain nested objects, changes to those nested objects may not trigger a re-render.
  2. **Custom Comparison Logic**: If you need more control over the comparison logic, you can implement the shouldComponentUpdate lifecycle method in a regular React.Component.
* **When Should You Use PureComponent?**
  1. Use for performance optimization when a component re-renders unnecessarily.
  2. Best for components with the same props/state over time.
  3. Avoid for deeply nested objects/arrays (as it only does a shallow comparison).

1. What is useMemo? Why we use it? Explain with an example?

* **useMemo** is a React Hook that allows you to memoize the result of a computation, preventing unnecessary recalculations on every render. It is used to optimize performance by caching the result of expensive calculations and only recomputing them when their dependencies change.
* **Why Use useMemo?**
  1. **Performance Optimization**: useMemo helps optimize performance by memoizing the result of expensive computations, ensuring that they are only recalculated when necessary.
  2. This can be particularly useful in scenarios where a component re-renders frequently, and the computation is resource-intensive.
  3. **Avoiding Unnecessary Recalculations**: By memoizing the result, useMemo prevents unnecessary recalculations, which can help reduce the rendering time and improve the overall performance of the application.
* **Example of useMemo**
  1. Consider a scenario where you have a component that performs an expensive calculation based on some input data. Without useMemo, the calculation would be performed on every render, even if the input data hasn't changed.

1. What is difference b/w useEffect and useMemo?
2. Why we use Memoization in React?
3. What are the techniques uses in Memoization, what problems we face what are their principles?
4. If we are reutilizing the value with Memoization then why we don't use cache?
5. What is useContext in React?
6. In useContext we have variable a, b, c and all these are siblings to each other. after made a change in parent for c only, what will happen on these component, will it re-render?
7. What are interceptors in React?
8. What are synthetic events in React JS?

* **Synthetic Events** in React are a cross-browser wrapper around the browser's native event system. They provide a consistent interface for handling events in a way that works identically across different browsers. Synthetic events are part of React's event handling system and are designed to mimic the behavior of native events while providing additional features and optimizations.

1. What are error boundaries?

* **Error boundaries** in React are components that catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI. They help in handling errors gracefully and preventing the entire application from crashing. Error boundaries are created using class components with the getDerivedStateFromError and componentDidCatch lifecycle methods. They are useful for isolating components that are more likely to throw errors and for top-level error handling in the application.
* **Key Features of Error Boundaries:** 
  1. **Catching Errors**: Error boundaries catch errors during rendering, in lifecycle methods, and in constructors of the whole tree.
  2. **Fallback UI**: When an error is caught, an error boundary can render a fallback UI to inform the user that something went wrong.
  3. **Logging Errors**: Error boundaries can also log errors to an error reporting service.
* **Limitations of Error Boundaries**
  1. **Event Handlers**: Error boundaries do not catch errors inside event handlers. You need to use regular JavaScript try/catch blocks for those.
  2. **Asynchronous Code**: Errors thrown in asynchronous code (e.g., setTimeout, requestAnimationFrame) are not caught by error boundaries.
  3. **Server-Side Rendering**: Error boundaries do not catch errors during server-side rendering.

1. How do you implement error boundaries and what are predefined for that?
2. What is encapsulations in React?
3. What is polymorphism in React?
4. What are forms in React?
5. What is the lazy loading in React?
6. What is skeleton loader?
7. What is tree shaking in React? How it will happen?

* **Tree shaking** is an optimization technique used to eliminate dead code from the final JavaScript bundle. In React applications, it helps reduce the bundle size by removing unused components, functions, and other code. Tree shaking works best with ES6 modules and requires proper configuration of the bundler (e.g., Webpack) to enable static analysis and dead code elimination. By using tree shaking, developers can create more efficient and performant applications. This is particularly useful when using large libraries or utility functions where only a subset of the functionality is needed.
* **How Tree Shaking Works**:
  1. **Static Analysis:** Tree shaking relies on static analysis of the code to determine which parts of the code are actually used and which can be safely removed. This is typically done by analyzing the import and export statements in ES6 modules.
  2. **Eliminating Dead Code:** During the bundling process, the bundler identifies and removes unused code based on the static analysis. This results in a smaller and more efficient bundle.
* **Enable Tree Shaking:**
  1. **Use ES6 Modules**: Ensure that your code and any third-party libraries you use are written using ES6 module syntax (import and export). Tree shaking works best with ES6 modules because they allow for static analysis.
  2. **Configure Webpack**: Ensure that your Webpack configuration is set up to support tree shaking. This typically involves setting the mode to production and ensuring that the optimization settings are correctly configured.
  3. **Minification**: Minification tools like Terser (used by Webpack in production mode) further optimize the bundle by removing dead code and performing other optimizations.

1. What is Client side rendering and Server side rendering (CSR & SSR)?
2. What is used for SSR?
3. How to optimize SEO in CSR?
4. How to avoid re-rendering?
5. What is Lazy loading (Suspense and fallback)?
6. What is code splitting?
7. What is code coverage? What tools we use?
8. What is React Router?
9. How do you use private protector routes in react with router? Is this a part of react route?
10. How do you implemented authentication in React?
11. How to perform Automatic redirect after login?
12. How do you validate user is logged in OR valid user or not in React?
13. If login to dashboard, will we redirect to login screen if page is refreshed?
14. Draw diagram for login & save account channel functionality (how structure your component).
15. Explain the entire flow for the login and authentication in React? Explain scenarios in brief?
16. What are the roles of JWT tokens?
17. What happen when token gets expired and how user maintain the login? So that user having a seamless experience?
18. What is Web Worker?
19. Shallow copy vs Deep copy?
20. What is Flux architecture in React?
21. What are debugging tools in React?
22. How to use web-flow in React?
23. What are the Various vision or techniques to optimize react code or performance?
24. What are some good engineering practices that we can apply while creating a page in the React?
25. How you will identify which component is taking more time to load?
26. Differences between old react (before 16.8) and new react version (with hook)?
27. On performance basis how we decide this component will be best?
28. What is the process of CI/CD to deploy the code?
29. When you want to migrate the images from local to stage than prod sever how do you configure the any Web-Pack file?
30. We have 1 parent component and inside 1 child called A and inside A we have another child B. Variables are passed from Parent to A and B. In parent we are changing values used in child B but not used in child A, what is behavior, will A re-render and B re-render or both will re-render. vice versa.

**REDUX & API**

1. What is Redux and where & why do we use in React?
2. Explain the core components of react-redux?
   * There are four fundamental concepts of Redux in react which decide how the data will flow through components:
3. **Redux Store**: It is an object that holds the application state
4. **Action Creators**: These are unctions that return actions (objects)
5. **Actions**: Actions are simple objects which conventionally have two properties- type and payload
6. **Reducers**: Reducers are pure functions that update the state of the application in response to actions
7. What are core Principals/Designs or Concepts on which Redux is built up on?
8. What is the execution Process of Redux?
9. Explain the workflow or flow of data through Redux?
10. What is Redux toolkit and the middleware and how it works?
11. What are the use case of Redux uses?
12. What is Context API in React?
13. Difference b/w Context API and Redux?
14. Difference b/w useContext and Redux Toolkit?
15. How store works?
16. Where do wrap the store?
17. What is despatcher in Redux?
18. What is fetch and Axios?
19. How do you call API in React?
20. What will be your best approach to fetch the data from API in react? Give an example.
21. What are middle wares?
22. What is Thunk & Saga?
    * Thunk and Saga are both middleware libraries used in Redux for handling side effects such as asynchronous actions.
      1. **Redux Thunk** is a middleware that allows you to write action creators that return a function instead of an action. This function can then perform asynchronous operations (like API calls) and dispatch actions based on the results.
      2. **Redux-Saga** is a middleware that uses generator functions to handle side effects. It allows more complex and powerful asynchronous flows by using yield to pause and resume execution.
23. Is it possible to implement Redux Thunk or saga with the functional approach?
24. What is yield?
25. Explain one-way data binding in Redux.
26. What are hooks in a React Redux App?
    * useSelector
    * useDispatch
27. What is connect?
    * The connect() function connects a React component to a Redux store.
28. What are connect parameters?
29. What is combineReducers? What it does?

**JEST**

1. How we do the unit testing on the code?
2. What is Unit Testing?
3. For Unit Testing in react what we use and elaborate?
4. What is Jest? What are Jest tools?
5. Can you explain what mock functions are? How are they created?
6. What do you understand about snapshot testing in context with Jest?
7. How to check the element is present of the DOM using jest?
8. Jest tool – unit test cases (If function having third party dependency how you write unit test cases for this. Will you use mock/spi).

EXERCISE

Given an api, and fetch title image and rating of the product?

Increment and Decrement of Text size on click in React?

Create Stop Watch in React

Create todo list

1. What is the process that you will follow if you assign a new task and you don’t have any knowledge regarding that?