|  |  |  |
| --- | --- | --- |
| JSX | | |
| 1 | JSX – adding classes |  |
| 2 | JSX – adding inline styles |  |
| 3 | JSX – styled components |  |
| 4 | JSX – Using dynamic values/variables/javascript inside JSX |  |
| 5 | JSX – general Rules |  |
| 6 | JSX - React Fragments |  |
| 7 | JSX – props.children |  |
|  |  |  |
| BASIC REACT | | |
| 1 | Class & Functional Components |  |
| 2 | State & Props |  |
| 3 | Constructor Function |  |
| 4 | Controlled components |  |
| 5 | Lifecycle methods |  |
| 6 | Default & Named exports |  |
| 7 | reactDOM.render() |  |
| 8 | Synthatic Events |  |
|  |  |  |
| COMPONENT LIFECYCLE & LIFECYCLE METHODS | | |
| 1 | Constructor |  |
| 2 | ComponentWillMount |  |
| 3 | Render |  |
| 4 | componentDidMount |  |
| 5 | componentDidUpdate |  |
| 6 | ShouldComponentUpdate |  |
| 7 | ComponentWillUpdate |  |
| 8 | ComponentWillUnmount |  |
| 9 | ComponentWillReciveProps |  |
| 10 | Reconcilation |  |
|  |  |  |
| HOOKS | | |
| 1 | useState |  |
| 2 | useEffect |  |
| 3 | useRef |  |
| 4 | useCallback |  |
| 5 | useMemo |  |
| 6 | useContext |  |
| 7 | useReducer |  |
| 8 | useLayoutEffect |  |
|  |  |  |
| ADVANCE REACT | | |
| 1 | Pure Component |  |
| 2 | Context |  |
| 3 | Refs |  |
| 4 | Callback refs |  |
| 5 | Memo | Done |
| 6 | HOC |  |
| 7 | Render Props |  |
| 8 | Portals |  |
| 9 | React.lazy |  |
| 10 | suspense |  |
| 11 | Code Splitting |  |
| 12 | Error Boundaries |  |
| 13 | Profiler |  |
| 14 | Observables |  |
| 15 | Type checking |  |
|  |  |  |
|  |  |  |
|  |  |  |
| ROUTER | | |
| 1 | BrowserRouter |  |
| 2 | Switch |  |
| 3 | Route |  |
| 4 | NavLink, Link |  |
| 5 | \*withRouter |  |
| 6 | \*match |  |
| 7 | \*matchPath |  |
| 8 | \*history |  |
| 9 | \*location |  |
| 10 | Redirect |  |
| 11 | HashRouter |  |
| 12 | MemoryRouter |  |
| 13 | Static Router |  |
| 14 | useParams() |  |
| 15 | useLocation() |  |
| 16 | useHistory() |  |
| 17 | useRouteMatch() |  |
| 18 | Protect Routes |  |
|  |  |  |
| AXIOS | | |
| 1 | Get request |  |
| 2 | Post Request |  |
| 3 | Request with config |  |
| 4 | Create config |  |
| 5 | Multiple concurrent requests |  |
| 6 | Config defaults |  |
| 7 | Interceptor |  |
| 8 | Catch error |  |
| 9 | Cancel request |  |
|  |  |  |
| REDUX, REACT-REDUX | | |
| 1 | What is Redux |  |
| 2 | Why Redux |  |
| 3 | How Redux works |  |
| 4 | Three Principles |  |
| 5 | Actions |  |
| 6 | Action Creator |  |
| 7 | Reducer |  |
| 8 | Root Reducer |  |
| 9 | Store |  |
| 10 | Provider |  |
| 11 | Connect |  |
| 12 | Map State to Props |  |
| 13 | Map dispatch to Props |  |
| 14 | Using Middleware |  |
| 15 | Thunk Middleware |  |
| 16 | Reduxdevtool Middleware |  |
| 17 | Implementing Undo History |  |
| 18 | Performance |  |
| 19 | Usage with react-router |  |
| 20 | Usage with Typescript |  |
| 21 | Redux sub-apps |  |
| 22 | Using Immutable.js |  |
| 23 | Code splitting |  |
| 24 | Normalize State |  |
| 25 | Selectors |  |
| 26 | Reselect |  |
| 27 | Redux Persist |  |
| 28 | Container Pattern |  |
| 29 | Recompose |  |
|  |  |  |
| JEST | | |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| REACT REDUX RECIPES | | |
| 1 | Using Error Boundaries |  |
| 2 | Using dev and prod environment variables |  |
| 3 | Using Loader on different pages/components |  |
| 4 | Multiple API calls for a single component |  |
| 5 | Single API call for multiple components |  |
| 6 | Creating and using Protected Routes |  |
| 7 | Server Side Rendering |  |
| 8 | Using Lodash |  |
| 9 | Observables |  |
| 10 | Redux - mergeProps (as third param for connect) |  |
| 11 | Local storage for persisting state |  |
| 12 | Reducers per field vs per container |  |
| 13 | Using Container pattern |  |
| 14 | Pass object instead of mapDispatchToProps to second param of connect - Dan talks about this in his egghead videos |  |
| 15 | Fetch Data from API and display on UI |  |
| 16 | Post Form Data to server |  |
| 17 | FETCH data through local json file using axios |  |
| 18 | Creating state (inside constructor or outside constructor) |  |
| 19 | Creating methods (in class and functional, using arrow and normal func) |  |
| 20 | Binding events in JSX |  |
| 21 | passing arguments of event handlers |  |
| 22 | Passing props to children component |  |
| 23 | Using props in functional components |  |
| 24 | using state in class components |  |
| 25 | make setState Asynchronous |  |
| 26 | Conditional rendering using && operator |  |
| 27 | Conditional rendering using ternary operator |  |
| 28 | Preventing component from rendering |  |
| 29 | Rendering Lists |  |
| 30 | Handling Form element change |  |
| 31 | Handing form submission |  |
| 32 | Using refs in DOM |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| INTERVIEW QUESTIONS | | |
| 1 | Import Json file data in component |  |
|  |  |  |
| 2 | use of .map() function |  |
|  |  |  |
| 3 | use of .map() function inside another .map() |  |
|  |  |  |
| 4 | What’s difference in useCallback and useMemo hooks |  |
|  | These both are used for performance reason. useCallback cache complete function while useMemo cache result of function |  |
| 5 | Why should we not use React.memo all the time |  |
|  | every abstraction (and performance optimization) comes at a cost Specifically the cost for useCallback and useMemo are that you make the code more complex for your co-workers, you could make a mistake in the dependencies array, and you're potentially making performance worse by invoking the built-in hooks and preventing dependencies and memoized values from being garbage collected. Those are all fine costs to incur if you get the performance benefits necessary, but **it's best to measure first.** |  |
| 7 |  |  |
| 8 |  |  |
| 9 |  |  |
| 10 |  |  |
| 11 |  |  |
| 12 |  |  |
| 13 |  |  |
| 14 |  |  |
| 15 |  |  |
| 16 |  |  |
| 17 |  |  |
| 18 |  |  |
| 19 |  |  |
| 20 |  |  |
| 21 |  |  |
| 22 |  |  |
| 23 |  |  |
|  |  |  |
| NOTES | | |
| 1 | inside .map there should only be one html parent only as in JSX otherwise wrap all elements inside <> </> |  |
| 2 | By Default useEffect runs both after first render and after every update |  |
| 3 | Unlike componentDidMount and ComponentDidUpdate, effects scheduled with useEffect don’t block the browser from updating the screen |  |
| 4 | Function passed to useEffect is different on every render. Every time we re-render , we schedule a different effect, replacing the previous one |  |
| 5 | Most of the effects are asynchronous. In case they do there is separate useLayoutEffect hook identical to useEffect |  |
| 6 | React performs cleanup when the component unmounts. However effects run on every render and not just once. This is why react also cleans up effect from the previous render before running the render |  |
| 7 | For cleanup any effect while using useEffect hook, we return a function within useEffect that includes cleanup login inside it. |  |
| 8 |  |  |
| 9 |  |  |
| 10 |  |  |

object literal

constructor

this

window object

Date( )

Math( )

inbuilt constructors

Regular expression

Prototype

Prototypal inheritance

Object.create()

classes

sub classes

static methods

Synchrounous and Asynchrounous