22) What are the downsides of Redux compared to Flux?

Instead of saying downsides we can say that there are few compromises of using Redux over Flux. Those are as follows:

i. You will need to learn to avoid mutations: Flux is un-opinionated about mutating data, but Redux doesn't like mutations and many packages complementary to Redux assume you never mutate the state. You can enforce this with dev-only packages like redux-immutable state-invariant, Immutable.js, or instructing your team to write non-mutating code.

i. You're going to have to carefully pick your packages: While Flux explicitly doesn't try to solve problems such as undo/redo, persistence, or forms, Redux has extension points such as middleware and store enhancers, and it has spawned a rich ecosystem.

ii. There is no nice Flow integration yet: Flux currently lets you do very impressive static type checks which Redux doesn't support yet.

23) What is the difference between mapStateToProps() and mapDispatchToProps()? mapStateToProps() is a utility which helps your component get updated state (which is updated by some other components):

mapDispatchToProps() is a utility which will help your component to fire an action event (dispatching action which may cause change of application state):

24) Can I dispatch an action in reducer?

Dispatching an action within a reducer is an anti-pattern. Your reducer should be without side effects, simply digesting the action payload and returning a new state object. Adding listeners and dispatching actions within the reducer can lead to chained actions and other side effects.

25) How to reset state in Redux?

You need to write a root reducer in your application which delegate handling the action to the reducer generated by combineReducers().

26) What is the difference between React context and React Redux? You can use Context in your application directly and is going to be great for passing down data to deeply nested components which what it was designed for.

Whereas Redux is much more powerful and provides a large number of features that the Context API doesn't provide. Also, React Redux uses context internally but it doesn't expose this fact in the public API.

27) Why are Redux state functions called reducers?

Reducers always return the accumulation of the state (based on all previous and current actions). Therefore, they act as a reducer of state. Each time a Redux reducer is called, the state and action are passed as parameters. This state is then reduced (or accumulated) based on the action, and then the next state is returned. You could reduce a collection of actions and an initial state (of the store) on which to perform these actions to get the resulting final state.

28) How would you manage user authentication state in a Redux application?

Store authentication status and user data in Redux state; dispatch actions for login/logout; use middleware like Redux Thunk for asynchronous authentication processes.

29) What techniques can you employ to optimize performance in a Redux application dealing with large datasets?

Implement memoization for selectors, use selective rendering to minimize unnecessary re-renders, and consider pagination or virtualization for large data sets to improve performance.

30) How can Redux facilitate communication between unrelated components in a React application?

Redux allows components to access shared state and dispatch actions, enabling communication without direct parent-child relationships.

31) Explain the purpose of middleware in Redux and provide an example of its application. Middleware intercepts dispatched actions, enabling additional functionality like logging, async operations, or routing; Redux Thunk is commonly used for async action creators.

32) What approaches would you take to test Redux code, including reducers and action creators?

Write unit tests for reducers to ensure correct state transitions and action creators, using tools like Jest and Enzyme for comprehensive testing coverage.

34) Why is it recommended to maintain immutable state in Redux, and how can it be achieved?

Immutable state ensures predictable state updates and simplifies debugging; libraries like Immutable.js or using spread/rest operators help maintain immutability.

35) How would you persist Redux state across browser sessions for enhanced user experience?

Utilize middleware like Redux Persist to save state in local storage or cookies, allowing seamless restoration of state upon page reloads.

36) Is Thunk is a sync call or Async call?

Thunk in Redux is typically used to handle asynchronous logic, making it an asynchronous call. Thunks are functions that return another function, which receives the Redux store's dispatch method as an argument, allowing for delayed dispatch of actions. This delayed dispatch is commonly used for asynchronous operations such as API calls, making Thunk a fundamental tool for managing asynchronous behavior in Redux.

37) What are the common methods for making API calls in React applications?

Common methods include using the built-in fetch API or libraries like Axios or React Query to perform HTTP requests asynchronously and handle responses within React components.

38) How do you handle loading states and error handling when making API calls in React?

Utilize state variables to manage loading indicators while waiting for API responses, and implement error handling mechanisms such as try-catch blocks or conditional rendering based on the API response status.

39) Explain the concept of useEffect hook and its role in API calls in React.

The useEffect hook is used for handling side effects in functional components, such as API calls. By specifying dependencies, it ensures API calls are made only when necessary, preventing unnecessary re-renders.

40) How do you structure API calls to handle authentication and authorization in React applications?

Incorporate authentication tokens or cookies in API requests' headers to authenticate users, and implement authorization logic on the server side to restrict access to certain endpoints based on user roles or permissions.

41) What are the benefits of using libraries like Axios over the native fetch API for making API calls in React?

Axios offers advantages such as simpler syntax, automatic JSON parsing, request cancellation, and interceptors for global request/response handling, enhancing the developer experience and improving code readability and maintainability.

42) You're tasked with implementing pagination for a list of items fetched from an API in a React application. How would you structure the API calls to fetch paginated data, and how would you update the UI to display each page of results?

Implement pagination parameters in the API endpoint URL or request body to specify the page number and page size. In the UI, maintain state variables to track the current page, fetch data accordingly, and update the displayed results when users navigate between pages.

43) You need to display real-time updates from a WebSocket-based API in a React application. How would you integrate WebSocket communication to fetch and display real-time data, ensuring consistency and performance?

Utilize libraries like Socket.io or native WebSocket API to establish a persistent connection with the server, listen for incoming data updates, and update the React component's state or Redux store accordingly to reflect real-time changes in the UI

44)Your React component needs to display different UI components based on the response from an API call (e.g., success or error response). How would you conditionally render components based on API response data?

Use state variables to track the API call status (loading, success, or error), and conditionally render UI components using JSX expressions or conditional rendering techniques like ternary operators or logical && operators based on the current API call status.

45) Explain how you would implement optimistic updates in a React application when making API calls, ensuring a smooth user experience while waiting for the server's response.

Optimistic updates involve updating the UI optimistically (before receiving confirmation from the server) based on user actions, while simultaneously sending the corresponding API request. If the request fails, rollback the UI changes to maintain consistency between client and server data.

46) You need to make multiple API calls in sequence, where the result of one call determines the parameters for the next call (e.g., fetching a user's profile and then their posts). How would you manage chained API calls efficiently in a React application?

Use async/await syntax or promise chaining to ensure sequential execution of API calls, where the subsequent call depends on the result of the previous call. Handle errors and loading states appropriately to provide feedback to users during each step of the chained API calls process.

47) does browser understands jsx?

No, browsers do not inherently understand JSX. JSX is a syntax extension for JavaScript that React uses to describe how user interfaces should look. Before a React application is deployed to a browser, JSX code is transformed into regular JavaScript code using a tool like Babel.

48) In React How do you achieve the Two Way Binding ,because Default its a 1 Way Binding?

In React, achieving two-way binding, where changes in the UI update the state and vice versa, typically involves a combination of state management and event handling.

49) How to make AJAX request in Redux?

You can use redux-thunk middleware which allows you to define async actions.

50) Why you get "Router may have only one child element" warning? You have to wrap your Route's in a <Switch> block because <Switch> is unique in that it renders a route exclusively.