1. **What is FLUX?**

Flux is a pattern that is responsible for managing the data flow in an application. It is commonly used to keep things organized.

**FLUX is comprised of the following:**

**Action:** A way to describe something that happened in the app, like "add an item" or "remove an item."

**Dispatcher:** The central part that takes actions and sends them to the right place. **Store:** Where the app's data lives. When an action happens, the store updates the data.

**View:** It is basically what the user sees. The view shows the data from the store and can trigger actions.

**Example:**

Let’s say we have a shop app, and we want to add an apple to the items list. Then it would work like this:

**Action**: "Add an apple to the list."

**Dispatcher**: Receives the action and sends it to the store.

**Store**: Adds the apple to the list of items.

**View**: Shows the updated list with the new apple

1. **What is Redux? How do you use it with React components?**

Redux is a tool that helps manage the state of an application in a clear and predictable way. It is like a single place where all the app's data lives. It's especially useful for larger apps where many components need to share and update data.

**Following are the parts of Redux:**

**Store**: The central storage of the app's state.

**Action**: An object that describes something that happened e.g., “ADD”.

**Reducer**: A function that takes the current state and an action, then returns a new state.

**Dispatch**: A method to send actions to the store to update the state.

Here is an example use of Redux in our component:

import React from 'react';

import { useSelector, useDispatch } from 'react-redux';

function App() {

const items = useSelector((state) => state.items); // it access current the state

const dispatch = useDispatch(); // it sends actions and triggers state change to the store

const addItem = () => {

dispatch({ type: 'ADD\_ITEM', payload: 'New Item' });

};

return (

<div>

<ul>

{items.map((item, index) => (

<li key={index}>{item}</li>

))}

</ul>

<button onClick={addItem}>Add Item</button>

</div>

);

}

export default App;

1. **What is a reducer?**

A reducer is a function in Redux that determines how the state of an application changes in response to an action. It takes two arguments: the current state and an action, and returns a new state based on that action.

Example:

Let’s say we have a simple counter app as follows:

const initialState = { count: 0 };

function counterReducer(state = initialState, action) {

switch (action.type) {

case 'INCREMENT':

return { count: state.count + 1 };

case 'DECREMENT':

return { count: state.count - 1 };

default:

return state;

}

}

Here if the action type is INCREMENT the reducer returns a new state with the count increased by 1.

If the action type is DECREMENT, the reducer returns a new state with the count decreased by 1.

Otherwise, the reducer returns the current state.

1. **How do you choose between ContextAPI and Redux for global state management?**

I would prefer ContextAPI when the project is small and there are a smaller number of components. Also, when I don’t need to implement advanced features.

I would use Redux when I am working on a large and complex project and there are many components that needs to access large and complex state. Also, when I will need to implement advanced features e.g. middleware for handling async operations.

1. **What is redux thunk and why do you want to use it?**

Redux Thunk is a middleware for Redux that allows us to write action creators that return a function instead of an action. This function can perform side effects, like making asynchronous API calls, and then dispatch actions based on the result.

**We use redux thunk because:**

* It helps in handling Async operations. Redux can only handle synchronous operations. Redux Thunk allows us to handle async operations like fetching data from an API before dispatching an action to update the state.
* Thunk allows us to control the flow. We can delay the dispatch of an action or dispatch multiple actions conditionally, giving us more control over when and how the state is updated.