1. What is FLUX?

⇨ Flux is an architectural pattern used to manage data flow in applications, particularly in React. The main idea behind Flux is that data should flow in a single direction, which makes it easier to understand and debug your application.

In Flux, the data flow goes like this:

a) Actions: These are payloads of information that send data from your app to the dispatcher.
b) Dispatcher: This is a central hub that manages all the actions. It receives the actions and sends them to the appropriate store.
c) Stores: Stores hold the state (data) of your application. When they receive an action from the dispatcher, they update the state and emit a change event.
d) View: This is the React component that listens to changes in the store and re-renders when the data changes.

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

⇨ Redux is a state management library that centralizes an application's state in a single store, making state changes predictable through actions and reducers. It's often used with React to manage complex application state.
Key Ideas in Redux:

a) Single Source of Truth: The entire application's state is stored in a single JavaScript object.
b) State is Read-Only: The state can't be changed directly; instead, you dispatch actions that describe changes.
c) Changes with Pure Functions (Reducers): Reducers are functions that take the current state and an action, returning a new state.

Using Redux with React:

a) Create a Store: The store holds the application's state, created using createStore.
b) Define Actions: Actions are JavaScript objects that describe what changes in the state.
c) Create Reducers: Reducers define how the state changes in response to actions.

d) Connect Components: Use connect or the useSelector and useDispatch hooks from react-redux to link components to the Redux store, enabling them to access state and dispatch actions.

3. What is a reducer?

   ⇨ A reducer is a function that determines how the application's state changes based on an action. It’s a pure function that takes the current state and an action as inputs and returns a new state without altering the original one.

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

   - Context API is ideal for simpler, smaller applications where a few pieces of state need to be shared across components. It's built into React, lightweight, and great for passing down data without props.
   - Redux is better suited for larger applications with complex state management needs. It provides a structured approach with actions, reducers, and middlewares like redux-thunk for handling asynchronous actions.

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

   ⇨ Redux Thunk is a middleware for Redux that allows action creators to return functions instead of plain action objects. This is particularly useful for handling asynchronous tasks, like API calls, where you need to dispatch actions before, during, or after the operation.

   Why use Redux Thunk:

   a) Asynchronous Actions: Enables you to manage async operations, like fetching data, by dispatching actions at different stages.
   b) Conditional Logic: Allows for conditional dispatching of actions based on the current state or other logic.