React-js

Question 1: What is React.js? How is it different from other JavaScript frameworks and libraries?

- React.js is a JavaScript library developed by Facebook for building user interfaces (UIs), especially for single-page applications where the UI changes dynamically.
- React focuses on the view layer of an application (the "V" in MVC), meaning it is mainly concerned with rendering the UI efficiently.

  Differences from other frameworks/libraries:
- **Library vs Framework**: React is a library, not a full framework like Angular. This means React is focused on UI, and you often need other libraries (like React Router or Redux) for full application functionality.
- **Declarative**: React allows developers to describe what the UI should look like for a given state, rather than how to update it step by step.
- **Component-based**: React uses reusable components to build UIs, unlike jQuery which manipulates the DOM directly.
- **Virtual DOM**: React uses a virtual DOM to optimize rendering and make UI updates faster.

Question 2: Explain the core principles of React such as the virtual DOM and component-based architecture.

1.Virtual DOM

- Instead of updating the real DOM directly (which is slow), React maintains a virtual representation of the DOM in memory.
- When the state changes, React diffs the virtual DOM with the previous version and only updates the parts of the real DOM that changed.
- Benefit: Fast, efficient updates.

2. Component-based Architecture:

- React apps are built using components, which are self-contained, reusable pieces of UI.
- Components can be functional (stateless or using hooks) or class-based (with state and lifecycle methods).
- **Benefit:** Easier code maintenance, reusability, and separation of concerns.

3.Unidirectional Data Flow:

- Data flows from parent to child components via props.
- **Benefit:** Predictable state management and easier debugging.

4.JSX (JavaScript XML):

- JSX allows you to write HTML-like code in JavaScript, making UI code readable and easier to write.

Question 3: What are the advantages of using React.js in web development?

1. **High Performance:** Thanks to the virtual DOM, updates are fast and efficient.
2. **Reusable Components:** Components can be reused across the app, reducing code duplication.
3. **Easy to Learn:** React focuses only on the view layer, making it simpler than full frameworks.
4. **Strong Community Support**: React has a huge community, lots of libraries, and frequent updates.
5. **SEO Friendly:** React can be rendered on the server-side (with Next.js), improving SEO.
6. **Rich Ecosystem**: Integrates easily with tools for routing (React Router), state management (Redux, Recoil), and testing.
7. **Cross-platform Development:** With React Native, you can build mobile apps using React knowledge.

2.JSX (JavaScript XML)

Question 1: What is JSX in React.js? Why is it used?

- JSX (JavaScript XML) is a syntax extension for JavaScript that allows you to write HTML-like code inside JavaScript.
- React uses JSX to define the UI structure in a more readable and declarative way.
  Why it is used:
- Makes the code more readable and maintainable because HTML structure and JavaScript logic can coexist.
- Allows React to convert JSX into JavaScript function calls (React.createElement) that build the virtual DOM efficiently.
- Provides a clear visual representation of the UI in the code.

Question 2: How is JSX different from regular JavaScript? Can you write JavaScript inside JSX?

Differences from regular JavaScript:

- JSX looks like HTML but is not a string or HTML—it is compiled into JavaScript objects that React uses to create the virtual DOM.
- HTML attributes use camelCase in JSX (e.g., className instead of class, onClick instead of onclick).
- JSX requires closing tags even for self-closing elements (e.g., <img />, <input />).

Using JavaScript inside JSX:

- Yes, you can write JavaScript expressions inside curly braces {} in JSX.

Ex.

```
const name = "Sahil";

<h1>Hello,  {name}!</h1

<p>{2 + 3}</p>
```

Question 3: Discuss the importance of using curly braces {} in JSX expressions.

- Curly braces allow you to embed JavaScript expressions inside JSX.
- Without {}, JSX treats content as plain text, not dynamic data.
- Curly braces make JSX dynamic and interactive, letting the UI respond to data changes.
  Ex.

```
const age = 20;

<p>My age is {age}</p

<p>Next year, I will be {age + 1}</p>
```

3.Components (fundamental & class Components)

Question 1: What are components in React? Explain the difference between functional components and class components.

Components in React:

- Components are reusable, self-contained pieces of UI in React.
- They allow you to split the UI into independent parts that can manage their own logic and rendering.
- Every React app is built by combining multiple components.

- **Types of Components:**

  Functional component:-

- A JavaScript function that returns JSX.
- Can use hooks (useState, useEffect) for state.
- Use hooks like useEffect.
- Simpler, easier to read.
- Render no required.

  Class Component:-

- A **JavaScript class** extending React.Component.
- Uses this.state to manage state.
- Uses built in lifecycle methods (componentDidmount , ComponentDidupdate) etc.
- More verbose , older style.
- Render required to return JSX.

Ex.

Functional Component:

```
function Greeting(props) {
  return <h1>Hello, {props.name}!</h1>;
 }
```

Class Component:

```
class Greeting extends React.Component {
  render() {
return <h1>Hello, {this.props.name}!</h1>;
  }

}
```

Question 2: How do you pass data to a component using props?

- Props (properties) are used to pass data from a parent component to a child component.
- Props are read-only inside the child component.

Ex.

```
Function child(props) {
   Return <p>welcome, {props.username} ! </p>;
}
```

```
Function parent() {
  Return <child username="sahil" />;
}
```

Question 3: What is the role of render() in class components?

- In class components, render() is a mandatory method that returns the JSX to display in the UI.
- React calls render() whenever the state or props change, updating the virtual DOM.
- Without render(), the class component cannot display any content.

Ex.

```
class Welcome extends React.Component {

  render() {

    return <h1>Hello, {this.props.name}!</h1>;

  }

}
```

4. Props and State

Question 1: What are props in React.js? How are props different from state?

Props:

- Props are used to pass data from a parent component to a child component.
- They are read-only; a child component cannot modify them.
- Props make components dynamic and reusable.

Difference between props and state:

Props:

- Data passed from parent to child
- Mutability - read only
- Source - passed by parent
- Usage - used to make component resuable
- Ex. <child name="xyz">

State:

- Data managed inside the component
- Mutability – mutable

- Source - Defined inside the component
- Usage - Used to manage dynamic behavior of a component
- Ex. this.state = { count : 0 }

Question 2: Explain the concept of state in React and how it is used to manage component data.

State:

- State is a JavaScript object that holds dynamic data for a component.
- When the state changes, React re-renders the component to reflect the new data.
- State allows components to remember information and react to user interactions**.**

Ex.

```
Import { usestate } from "react";



Function Counter() {

   Const [count , setcount ] = usestate(0);

Retuen (

  <div>

   <p>count: {count}</p>

   <button onclick={() => setcount(count + 1)}>Increment</button>

</div>

);

}
```

Question 3: Why is this.setState() used in class components, and how does it work?

- In class components, you cannot modify state directly using this.state.count = 1.
- Instead, you use this.setState() to update state.

How it works:

- this.setState() merges the new state with the existing state.
- It triggers a re-render of the component, so the UI reflects the updated state.

Ex.

```
class Counter extends React.Component {
  constructor() {
    super();
    this.state = { count: 0 };
  }
  increment = () => {
    this.setState({ count: this.state.count + 1 });
  }
  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={this.increment}>Increment</button>
      </div>
    );
  }
}
```

5. Handling Events in React

Question 1: How are events handled in React compared to vanilla JavaScript? Explain the concept of synthetic events.

Event handling in React:

- When you write an event handler in React (for example, onClick={handleClick}), React does not bind this directly to the DOM node in the same way as vanilla JavaScript.
- React attaches a single event listener at the root of the application.
- When an event occurs, React captures it, processes it, and then calls your component's event handler.

What are Synthetic Events?

1.Cross-browser consistency
Synthetic events normalize browser differences, so properties like event.target, event.type, and event.preventDefault() behave the same in all browsers.

2. Performance optimization
Because React uses event delegation and one listener at the root, it reduces the number of event listeners in the DOM and improves performance.

3. Same interface as native events
Synthetic events expose the same methods and properties as native DOM events (such as stopPropagation() and preventDefault()), so they feel familiar to JavaScript developers.

Question 2: What are some common event handlers in React.js? Provide examples of onClick, onChange, and onSubmit.

1. onClick – Triggered when an element is clicked.

```
function Button() {

const handleClick = () => alert("Button clicked!");

return <button onClick={handleClick}>Click Me</button>;

 }
```

2. onChange – Triggered when the value of an input changes.

```
function InputField() {

 const handleChange = (event) => console.log(event.target.value);

 return <input type="text" onChange={handleChange} />;

 }
```

3. onSubmit – Triggered when a form is submitted.

```
function Form() {

 const handleSubmit = (event) => { event.preventDefault();

alert("Form submitted!");

};

 return (
 <form onSubmit={handleSubmit}>

 <button type="submit">Submit

</button>

 </form>

); }
```

Question 3: Why do you need to bind event handlers in class components?

- In class components, this does not automatically refer to the component instance inside methods.
- Without binding, this in event handlers will be undefined or point to the wrong object.
- Binding ensures that this correctly refers to the component instance, so you can access this.state or this.props.

Ex.

```
class Button extends React.Component {

  handleClick() {

    console.log(this.state); // 'this' is undefined

  }

  render() {

    return <button onClick={this.handleClick}>Click Me</button>;

  }

}
```

Alternative: Use **arrow functions**, which automatically bind this.

```
handleClick = () => {

 console.log(this.state.count);

 }
```

6. Conditional Rendering

Question 1: What is conditional rendering in React? How can you conditionally render elements in a React component?

- Conditional Rendering means displaying different UI elements or components based on certain conditions (like state or props).
- In React, instead of manipulating the DOM manually, you use JavaScript logic inside JSX to decide what should be rendered.

Example:

```
function Greeting({ isLoggedIn }) {
  if (isLoggedIn) {
    return <h1>Welcome Back!</h1>;
  } else {
    return <h1>Please sign in.</h1>;
  }
}
```

Question 2: Explain how if-else, ternary operators, and && (logical AND) are used in JSX for conditional rendering.

a) if-else statement

- Can be used outside JSX to decide what to render.

```
function Greeting({ isLoggedIn }) {

let message; if (isLoggedIn) {

message = <h1>Welcome Back!</h1>;

 }

 else {

message = <h1>Please sign in.</h1>;

}

 return <div>{message}</div>;

 }
```

b) Ternary operator (condition ? true : false**)**

- Useful inside JSX for inline conditional rendering.

```
function Greeting({ isLoggedIn }) {

return ( <div> {isLoggedIn ? <h1>Welcome Back!</h1> :
```

<h1>Please sign in.</h1>

}

</div>);

}

c) Logical AND (&&) operator

- Renders an element only if the condition is true.
- If the condition is false, React renders nothing.

```
function Notification({ hasMessages }) {

return (

 <div> {hasMessages && <p>You have new messages!</p>

}

 </div>

);

 }
```

7.Lists and Keys

Question 1: How do you render a list of items in React? Why is it important to use keys when rendering lists?

- In React, you can render a list of elements by using the JavaScript map() function to iterate over an array and return JSX elements for each item.

Example:

```
function FruitsList() {

 const fruits = ["Apple", "Banana", "Mango"];

 return (

 <ul>

{fruits.map((fruit, index) => (

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

 ))}
 </ul>

);

 }
```

Question 2: What are keys in React, and what happens if you do not provide a unique key?

Keys:

- Keys are unique identifiers assigned to each list element.
- They should be stable, predictable, and unique among siblings.
- Example: key={fruit.id} or key={index} (index is less preferred if the list can change).

Consequences of not providing unique keys:

1. React may re-use wrong elements, leading to incorrect rendering or UI bugs.
2. Updates to the list can be less efficient, because React cannot track elements correctly.

Example without keys (not recommended):

```
<ul>

  <li>Apple</li>

  <li>Banana</li>

  <li>Mango</li>

</ul>
```

8. Forms in React

Question 1: How do you handle forms in React? Explain the concept of controlled components.

1. How do you handle forms in React? Concept of controlled components

Handling forms in React:

- React encourages using controlled components, where form input values are controlled by React state.
- When a user types in an input, the state updates, and the input value comes from the state.

Controlled Component Example:

```
import { useState } from "react";
```

```
function NameForm() {

const [name, setName] = useState("");

const handleChange = (event) => setName(event.target.value);

const handleSubmit = (event) => {

event.preventDefault();

alert(`Submitted name: ${name}`); };

return (

<form onSubmit={handleSubmit}>

<input type="text" value={name} onChange={handleChange} />

<button type="submit">Submit</button>

</form>

);

}
```

Question 2: What is the difference between controlled and uncontrolled components in React?

Controlled Components

A controlled component is a form element whose value is fully controlled by React state.

You store the input value in a state variable, and every change updates that state. The input's displayed value always comes from React.

Because React controls the data:

- You can easily validate input
- You can conditionally enable/disable fields
- You have one single source of truth (React state)

However, this also means:

- You need more code (state + change handlers)
- Every keystroke triggers a state update

Conceptually:
User types → onChange fires → React state updates → UI re-renders

Uncontrolled Components

An uncontrolled component lets the DOM manage the form data instead of React state.

You don't store the value in React state. Instead, you use a ref to read the value directly from the DOM when needed (for example, on form submit).

Because the DOM controls the data:

- Less code is required
- It feels closer to traditional HTML forms

But:

- Validation and dynamic behavior are harder
- React doesn't automatically know the current value

Conceptually:
User types → DOM updates itself → value accessed via ref when needed

9. Lifecycle Methods (Class Components)

Question 1: What are lifecycle methods in React class components? Describe the phases of a component's lifecycle.

Lifecycle Methods:

- Lifecycle methods are **special methods in React class components** that are called at **different stages of a component's life** (creation, updating, removal).
- They allow developers to **perform actions at specific times**, such as fetching data, updating the DOM, or cleaning up resources.

Phases of a component's lifecycle:

1. **Mounting** – When the component is created and inserted into the DOM.
   - Methods:
     - constructor() – Initialize state and bind methods.
     - static getDerivedStateFromProps() – Update state based on props.
     - render() – Returns the JSX to render.
     - componentDidMount() – Called after the component is added to the DOM.
2. **Updating** – When the component re-renders due to state or props changes.
   - Methods:
     - static getDerivedStateFromProps() – Update state from props.
     - shouldComponentUpdate() – Decide whether to re-render.
     - render() – Render updated JSX.
     - getSnapshotBeforeUpdate() – Capture some info before DOM updates.
3. **Unmounting** – When the component is removed from the DOM.

- Method:
    - componentWillUnmount() – Cleanup tasks like removing timers or event listeners.

Question 2: Explain the purpose of componentDidMount(), componentDidUpdate(), and componentWillUnmount().

a) componentDidMount()

- Called once after the component is mounted.
- Common uses:
    - Fetching data from APIs
    - Setting up subscriptions or timers
    - Manipulating the DOM

```
componentDidMount() {

console.log("Component has mounted!");

 }
```

b) componentDidUpdate(prevProps, prevState)

- Called after the component updates due to state or props changes.
- Useful for:
    - Responding to prop or state changes
    - Making API calls when data changes
    - Updating the DOM in response to changes

```
componentDidUpdate(prevProps, prevState) {

if (this.state.count !== prevState.count) {

 console.log("Count has changed!");

 }

 }
```

c) componentWillUnmount()

- Called just before the component is removed from the DOM.
- Useful for:
    - Cleaning up timers, subscriptions, or event listeners
    - Preventing memory leaks

```
componentWillUnmount() {

console.log("Component is about to be removed."); }
```

10.Hooks (useState, useEffect, useReducer, useMemo, useRef, useCallback)

Question 1: What are React hooks? How do useState() and useEffect() hooks work in functional components?

React Hooks:

- Hooks are special functions that let you use state and other React features in functional components, which previously were only available in class components.
- They make functional components more powerful and easier to manage.

useState()

- Allows you to add state to functional components.
- Returns a state variable and a function to update it.

useEffect()

- Lets you perform side effects (like data fetching, subscriptions, or DOM manipulation) in functional components.
- It replaces lifecycle methods: componentDidMount, componentDidUpdate, and componentWillUnmount.

Question 2: What problems did hooks solve in React development? Why are hooks.

Hooks solved several problems in React:

- Eliminated the need for class components for state and lifecycle logic.
- Made code more reusable through custom hooks.
- Simplified stateful logic sharing between components.
- Improved readability and maintainability by reducing boilerplate code.

Hooks are important because they let developers write modern, cleaner, and reusable code without switching to classes.

Question 3: What is useReducer ? How we use in react app?

- useReducer is a hook for managing complex state logic (like in Redux) using reducer functions.
- Useful when state depends on previous state or has multiple sub-values.

Ex.

```
import { useReducer } from "react";

 const initialState = { count: 0 };

 function reducer(state, action) {

 switch (action.type) {
```

```
 case "increment":

return { count: state.count + 1 };

case "decrement":

 return { count: state.count - 1 };

default: return state;

}
```

Question 4: What is the purpose of useCallback & useMemo Hooks?

- useCallback: Memorizes a function so that it is not recreated on every render, useful for passing stable functions to child components.
- useMemo: Memorizes a computed value so that expensive calculations are not repeated unnecessarily.

```
import { useMemo, useCallback } from "react";

 const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a, b]);

 const memoizedFunction = useCallback(() => doSomething(a), [a]);
```

Question 5: What's the Difference between the useCallback & useMemo Hooks?

useCallback

useCallback is used to memoize a function.

When a component re-renders, functions are normally re-created. useCallback returns the same function reference between renders as long as its dependencies don't change.

Why this matters:

- Prevents unnecessary re-renders of child components that receive functions as props
- Useful with React.memo

useMemo

useMemo is used **to memoize the result of a computation.**

If a calculation is expensive, useMemo ensures it only re-runs when its dependencies change, instead of on every render.

Why this matters:

- Improves performance for heavy calculations
- Avoids unnecessary recomputation

Conceptually:
Memoizes the returned value

Question 6 : What is useRef ? How to work in react app?

- useRef creates a mutable reference that persists across renders without causing re-renders.
- **Common uses:**
    - Access DOM elements directly
    - Store previous values
    - Keep mutable variables that don't trigger render

```
import { useRef } from "react";

function InputFocus() {

const inputRef = useRef();

 const focusInput = () => inputRef.current.focus();

return (

 <div>

 <input ref={inputRef} type="text" />

 <button onClick={focusInput}>Focus Input</button>

 </div>

 );

 }
```

11. Routing in React (React Router)

Question 1: What is React Router? How does it handle routing in single-page applications?

React Router:

- React Router is a library for handling routing in React applications.
- It allows you to navigate between different components/pages without reloading the entire browser page, making it ideal for single-page applications (SPAs).

How it works in SPAs:

- React Router updates the UI based on the URL using the History API.
- Instead of loading a new HTML page, React Router renders different components dynamically, giving the feel of multiple pages.

Example:

```
import { BrowserRouter as Router, Routes, Route } from "react-router-dom";

 import Home from "./Home";

 import About from "./About";

 function App() {

return (

 <Router>

<Routes>

<Route path="/" element={<Home />} />

 <Route path="/about" element={<About />} />

 </Routes> </Router>

 );

 }
```

Question 2: Explain the difference between BrowserRouter, Route, Link, and Switch components in React Router.

BrowserRouter

BrowserRouter is the router container that uses the browser's HTML5 History API to keep the UI in sync with the URL.

- It must wrap your entire application (or at least the part where routing is needed)
- It listens to URL changes and decides which component to render

Conceptually:
It provides routing context to the whole app

Without BrowserRouter, routing will not work.

Route

Route is used to map a URL path to a React component.

- When the browser URL matches the path, the corresponding component is rendered
- It defines *what should appear* for a specific URL

Conceptually:
URL → Component mapping

Example idea:
If the path is /login, show the Login component.

Link

Link is used for navigation between routes.

- It works like an <a> tag but does not reload the page
- It updates the URL using React Router instead of making a full HTTP request

Conceptually:
Changes URL without refreshing the page

This is what enables single-page application behavior.

Switch (React Router v5)

Switch is used to render only the first matching route.

- It checks routes one by one
- Stops after finding the first match
- Prevents multiple components from rendering for similar paths

Conceptually:
First match wins

12. React – JSON-server and Firebase Real Time Database

Question 1: What do you mean by RESTful web services?

Key ideas:

- Data is treated as resources (users, products, posts, etc.)
- Each resource is identified by a URL
- Standard HTTP methods are used:
    - GET → fetch data
    - POST → add new data
    - PUT / PATCH → update data
    - DELETE → remove data
- Data is usually exchanged in JSON format

Question 2: What is JSON-Server? How do we use it in React?

JSON-Server is a fake REST API tool that allows developers to create a backend using a simple JSON file, without writing server-side code.

Main points:

- It acts like a real REST API
- Supports GET, POST, PUT, PATCH, and DELETE
- Very useful for frontend development and testing

How it is used in React:

- Create a db.json file with data
- Run JSON-Server to expose REST APIs
- React app sends HTTP requests to these APIs
- Helps test React features like CRUD operations

In simple words:
JSON-Server is a quick backend for React apps during development.

Question 3: How do you fetch data from a JSON-Server API in React? Explain the role of fetch() or axios().

In React, data from a JSON-Server API is fetched using HTTP requests inside lifecycle logic (usually useEffect).

Role of fetch()

- Built-in JavaScript function
- Sends HTTP requests to APIs
- Returns a Promise
- Requires manual JSON parsing

Role of axios()

- Third-party library
- Easier syntax than fetch
- Automatically converts response to JSON
- Better error handling

Both are used to:

- Fetch data (GET)
- Send data (POST)
- Update data (PUT/PATCH)
- Delete data (DELETE)

In simple words:
fetch() and axios() act as bridges between React and the backend API.

Question 4: What is Firebase? What features does Firebase offer?

Firebase is a Backend-as-a-Service (BaaS) platform provided by Google that helps developers build applications without managing servers.

Main features of Firebase:

- **Realtime Database** – real-time data synchronization
- **Firestore** – scalable NoSQL cloud database
- **Authentication** – login with email, Google, Facebook, etc.
- **Hosting** – deploy web apps easily
- **Cloud Storage** – store images, videos, files
- **Cloud Functions** – serverless backend logic
- **Analytics** – track app usage

In simple words:
Firebase provides backend services so developers can focus on frontend development.

Question 5: Discuss the importance of handling errors and loading states when working with APIs in React.

Handling loading and error states is very important when working with APIs in React.

Loading state importance:

- Shows users that data is being fetched
- Prevents blank screens
- Improves user experience

Example:

- Show a spinner or "Loading…" message

Error handling importance:

- Handles network failures
- Prevents app crashes
- Displays meaningful error messages

Example:

- "Failed to load data. Please try again."

13. Context API

Question 1: What is the Context API in React? How is it used to manage global state across multiple components?

- The Context API in React is a feature used to share data globally across multiple components without passing props manually at every level (avoids prop drilling).
- Normally, data in React flows from parent to child using props.
- When many components at different levels need the same data (like user info, theme, language, or authentication status), passing props becomes difficult and messy.

- The Context API solves this problem.

How it manages global state:

- A Context is created to store shared data
- A Provider supplies this data to the component tree
- Any component inside the Provider can access the data directly

In simple words:
Context API allows multiple components to access the same state from one central place.

Question 2: Explain how createContext() and useContext() are used in React for sharing state.

createContext()

createContext() is used to create a Context object.

- It defines the data that needs to be shared
- It returns a Context with two components:
  - Provider → supplies the data
  - Consumer → receives the data (now mostly replaced by useContext)

The Provider wraps the components that need access to shared state.

useContext()

useContext() is a React Hook used to consume context data inside a functional component.

- It allows direct access to shared data
- Eliminates the need for Consumer components
- Makes code cleaner and easier to read

Any component inside the Provider can read and update the shared state using useContext().

Flow in simple terms

1. Create a context using createContext()
2. Wrap components with the Context Provider
3. Pass shared state through Provider
4. Access shared state using useContext()

14. State Management (Redux, Redux-Toolkit or Recoil)

Question 1: What is Redux, and why is it used in React applications?

Explain the core concepts of actions, reducers, and the store.

Redux is a state management library used in React applications to manage global application state in a predictable and centralized way.

In large React applications, passing state through props becomes complex and hard to maintain. Redux solves this by keeping all shared state in one central place, making state changes predictable and easier to debug.

Why Redux is used in React:

- Centralized state management
- Predictable state updates
- Easier debugging with time-travel and logs
- Works well for large and complex applications

Core Concepts of Redux

Actions

Actions are plain JavaScript objects that describe what happened in the application.

- They have a type property
- They may include data (payload)

In simple words:
Actions tell Redux what to do.

Reducers

Reducers are pure functions that decide how the state should change.

- They take the current state and an action
- They return a new updated state
- They never modify the existing state directly

In simple words:
Reducers update the state based on actions.

Store

The store is the central container that holds the entire application state.

- It stores the state

- It allows access to the state
- It dispatches actions to reducers

In simple words:
The store is the single source of truth.

Redux Flow (in words)

UI → dispatch action → reducer updates state → store saves state → UI re-renders

Question 2: How does Recoil simplify state management in React compared to Redux?

Recoil is a React-specific state management library designed to be simpler and more intuitive than Redux.

How Recoil simplifies state management:

1. **Less Boilerplate**
   Recoil does not require actions, reducers, or dispatch functions.
   You directly read and update state.
2. **Atomic State**
   State is divided into small pieces called **atoms**, instead of one big store.
3. **Derived State with Selectors**
   Computed state is handled using **selectors**, without complex reducer logic.
4. **React-like Syntax**
   Recoil uses hooks, making it feel natural to React developers.
5. **Better for Small to Medium Apps**
   Easier to learn and quicker to implement than Redux.

Redux vs Recoil (conceptual difference)

- Redux uses **one global store** with strict rules
- Recoil uses **multiple atoms** that components subscribe to directly
- Redux is powerful but verbose
- Recoil is simpler and more flexible.