React Advanced (Hooks, Firebase, Redux)

Lists and Hooks

Q. Explain Life cycle in Class Component and functional component with Hooks

Life Cycle in Class Components

A class component has **three main phases** in its lifecycle:

1. Mounting (Component is created and shown on screen)

Happens when the component appears for the first time.

Methods:

- **constructor()** → component is starting
- **render()** → UI is drawn
- **componentDidMount()** → runs after UI appears (good for API calls)

2. Updating (Component gets updated when props/state change)

Happens when something changes inside the component.

Methods:

- **render()** → UI updates
- **componentDidUpdate()** → runs after update (good for responding to changes)

3. Unmounting (Component is removed from screen)

Method:

- **componentWillUnmount()** → cleanup work (like removing event listeners or stopping timers)

 Life Cycle in Functional Components (With Hooks)

Functional components do not have lifecycle methods. Instead, they use **useEffect**, which can act like all lifecycle phases.

useEffect Hook = componentDidMount + componentDidUpdate + componentWillUnmount

1. On Mount (runs once when component appears)

```
useEffect(() => {
  console.log("Component Mounted");
}, []);
```

✓ same as **componentDidMount**

2. On Update (runs when state/props change)

```
useEffect(() => {
  console.log("Component Updated");
}, [count]);
```

✓ same as **componentDidUpdate**

3. On Unmount (cleanup work)

```
useEffect(() => {
```

```
  return () => {
    console.log("Component Unmounted");
  };
}, []);
```

✓ same as **componentWillUnmount**

Firebase

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
import { getAuth } from "firebase/auth";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
```

```
// For Firebase JS SDK v7.20.0 and later, measurementId is optional
const firebaseConfig = {
  apiKey: "AIzaSyCoUH-qhb8Wlrjui6IZzdAIJvvrrGdPqjo",
  authDomain: "my-app-d6abf.firebaseapp.com",
  projectId: "my-app-d6abf",
  storageBucket: "my-app-d6abf.firebasestorage.app",
  messagingSenderId: "730414707667",
  appId: "1:730414707667:web:085510d1f184acba78b526",
  measurementId: "G-G5M6Z8QFYK"
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
export const auth= getAuth(app);
```

Redux

Q. What is Redux?

→ Redux is a **state management library** used mainly with **React** apps (but it can work with any JavaScript app).

→ Redux helps you **store and manage data in one central place**, so every component can access it easily without passing props everywhere.

➢ **Why We Use Redux**

- To manage **global state** (e.g., user info, cart items, theme).
- To avoid **prop drilling** (sending data from parent → child → grandchild).
- To make state changes **predictable**, easier to debug, and organized.

Q. What is Redux Thunk used for?

→ Redux Thunk allows you to write **functions inside actions** so you can do things like:

- Fetch API data
- Wait for a response
- Dispatch actions **after** the data arrives

Without Redux Thunk, actions can only return **plain objects** (synchronous).

➢ **Why We Use Redux Thunk**

Because sometimes you need to:

- Call an API
- Wait for loading
- Handle success or error
- Then update the store

Redux by itself **cannot handle async code**, so Thunk helps.

Q. What is Pure Component? When to use Pure Component over Component?

- **What is a Pure Component?**

A **Pure Component** is a React component that **does not re-render** if its **props or state are the same** as before.

It only re-renders **when something actually changes**.

So it helps your app run faster.

- **When to use Pure Component?**

Use **Pure Component** when:

- Your component gets **simple props** (numbers, strings, booleans).
- You want to **stop unnecessary re-renders**.
- Your data does **not change often**.

Q. What is the second argument that can optionally be passed tosetState and what is its purpose?

In **React class components**, setState can take **two arguments**:

- **1st argument:**

The **new state** (object or function).

- **2nd argument (optional):**

A **callback function**.

➢ **What is the second argument?**

It's a **callback function** that runs **after the state update is finished and the component has re-rendered**.

➢ **Why do we use it? (Purpose)**

Because setState is **asynchronous**, so if you want to do something **immediately after the state is updated**, you use the callback.

Q. CRUD Application using API (Redux)

```
import axios from 'axios';
import React, { useEffect, useState } from 'react'

const App = () => {

  const [data, alldata] = useState([]);
  const [title, settitle] = useState('');
  const [price, setprice] = useState('');
  const [description, setdescription] = useState('');
  const [category, setcategory] = useState('');
  const [image, setimage] = useState('');
  const [id, setid] = useState('');

  const [editmode, setdeditmode] = useState(false);
```

```
  // editmode true


  const editmodefun = () => {
    setdeditmode(true)
  }


  //datafetch


  const datafetch = async () => {


    try {
      const mydata = await
axios.get('https://68cc57c7716562cf50774e3d.mockapi.i
o/Product')
      alldata(mydata.data)
    } catch (error) {
      console.log("data not fetch")
```

```
  } finally {
    console.log("all ok")
  }
}

//datainsert

const datainsert = async () => {

  const mydata = {
    "title": title,
    "price": price,
    "description": description,
    "category": category,
    "image": image,
  }
```

```
    try {
      await
axios.post('https://68cc57c7716562cf50774e3d.mockapi.
io/Product', mydata)
      datafetch()
      formreset()
    } catch (error) {
      console.log("data not inserted")
    } finally {
      console.log("all ok")
    }


  }

  //datadelet

  const datadelet = async (id) => {
```

```
    try {
      await
axios.delete(`https://68cc57c7716562cf50774e3d.mockapi.io/Product/${id}`)

      datafetch()

    } catch (error) {
      console.log("data not deleted")
    } finally {
      console.log("all ok")
    }
  }

  //dataupdate
```

```
  const dataupdate = async () => {


    const mydata = {
      "title": title,
      "price": price,
      "description": description,
      "category": category,
      "image": image,
    }
    try {
      await
axios.put(`https://68cc57c7716562cf50774e3d.mockapi.i
o/Product/${id}`, mydata)
      datafetch()
      formreset()
    } catch (error) {
      console.log("data not updated")
```

```
  } finally {
    console.log("all ok")
  }
}

//formreset

const formreset = () => {

  settitle('');
  setprice('');
  setdescription('');
  setcategory('');
  setimage('');
  setid('');
}
```

```
useEffect(() => {
  datafetch()

}, []);

return (
  <div>
    <input
      type="text"
      placeholder="Title"
      value={title}
      onChange={(e) => settitle(e.target.value)}
    />
    <input
      type="text"
      placeholder="Price"
      value={price}
```

```
  onChange={(e) => setprice(e.target.value)}
/>
<input
  type="text"
  placeholder="Description"
  value={description}
  onChange={(e) => setdescription(e.target.value)}
/>
<input
  type="text"
  placeholder="Category"
  value={category}
  onChange={(e) => setcategory(e.target.value)}
/>
<input
  type="text"
  placeholder="Image URL"
```

```
        value={image}
        onChange={(e) => setimage(e.target.value)}
      />
      {editmode ? (<button onClick={dataupdate}>Update</button>): (<button onClick={datainsert}>ADD</button>)}

    <ul>
      {data.map(item=>(
        <li key={item.id}>
          {item.title} - {item.price}

          <button onClick={()=>datadelet(item.id)}>Delete</button>
          <button onClick={()=>{
            editmodefun();
            settitle(item.title);
            setprice(item.price);
```

```
            setdescription(item.description);
            setcategory(item.category);
            setid(item.id);
            setimage(item.image);
          }}>Edit</button>
        </li>
      ))}
    </ul>
    </div>
  )
}

export default App
```