# DIGA4015A:
# Interactive Media 4A | React Exercises Solutions

## Note:

This is **one** method of completing the following exercises, not the only method. How you choose to answer will depend on how you interpret the question, but the logic should remain roughly the same.

*I have left out all the App.js code needed to render these components.

---

### Exercise Question 1: To Do List

Create a To-do List app that allows users to add items to a list and mark them as completed. Use the **useState** hook to manage the state of the to-do list.

Task:

- **Input field** for adding new to-dos
- **Button to submit** the new to-do
- **Display a list** of to-dos
- Each to-do should have a **button or checkbox to mark it as completed**
- When marked completed, the to-do item should **visually indicate its status** (e.g. strike-though text)

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

export default function Todo() {
  const [todoList, setTodoList] = useState([]);
  const [newTodo, setNewTodo] = useState("");

  function addNewTodo(event) {
    //This will handle adding a new Todo to the list on form submit.
    event.preventDefault(); //This, again, prevents the automatic refresh of the page on a form submit.
    if (!newTodo) return;

    //Building out the todo structure. No need to create a seperate variable for this if you want.
    const newItemToAdd = {
```

```
    id: Date.now(),
    text: newTodo,
    completed: false,
  };

  //Changes the Todo list. And make the input field blank after submission.
  setTodoList([...todoList, newItemToAdd]);
  setNewTodo("");
}

function toggleCompleted(id) {
  //Search for the id of the todo, and toggles the boolean.
  setTodoList(
    todoList.map((todo) =>
      todo.id === id ? { ...todo, completed: !todo.completed } : todo
    )
  );
}

return (
  <>
    <h1>To Do List</h1>
    <form onSubmit={addNewTodo}>
      <input
        type="text"
        placeholder="Enter new todo item"
        value={newTodo}
        onChange={(e) => setNewTodo(e.target.value)}
      />
      <button>Add</button>
    </form>
    <ul>
      {todoList.map((todo) => (
        <li
          key={todo.id}
          style={{ textDecoration: todo.completed ? "line-through" : "none" }}
        >
          {todo.text}
          <input type="checkbox" onChange={() => toggleCompleted(todo.id)} />
        </li>
      ))}
    </ul>
  </>
 );
}
```

### Exercise Question 2: Fetching Data on Component Mount and Clean-up

Implement a component that fetches data from an API when the component mounts, and provides a clean-up mechanism to cancel the fetch if the component unmounts before the data is loaded.

Task:

- Create a functional component **UserProfile.jsx**
- Use the **useEffect** hook to make an API call to **get the user's data** (you can use *https://jsonplaceholder.typicode.com/users/1* as a mock API).
- Imagine the user closes the tab/component—ensure that the component unmounts **before** the data is fetched, the fetch is cancelled to prevent memory leaks.
- Display the fetched user data, or a loading indicator if the data has not been loaded yet.

Hints:

- JavaScript has an **in-built method** called `AbortController` for cases such as these.
- Remember to try to handle potential errors in fetching data.

```
import { useState, useEffect } from "react";
const URL = "https://jsonplaceholder.typicode.com/users/1"; //API URL

function UserProfile() {
  const [loading, setLoading] = useState(null);
  const [err, setErr] = useState(null);
const [userData, setUserData] = useState(null);

useEffect(() => {
  //ssetup a new instance of AbortController().
  const controller = new AbortController();
  const signal = controller.signal;

  //fetch from API
  async function fetchUserData() {
    try {
      setLoading(true); //Set loading state to true, we later output certain content depending on this.
      const response = await fetch(URL, { signal });
      //General error checking
      if (!response.ok) {
        throw new Error(`HTTP Error. Status: ${response.status}`);
      }
      const data = await response.json();
      setUserData(data);
      setLoading(false); //Content will change again after loading has completed because we are changing state.
    } catch (error) {
      //General error handling.
      if (error.name === "AbortError") {
        console.log("Fetch Aborted");
      } else {
        //Using state to capture possible error messages. Not absolutely necessary.
        setErr(error.message);
        setLoading(false);
      }
    }
  }

  fetchUserData();

  //Before component unmounts, clean up function
  return () => {
    controller.abort();
  };
}, []); //Empty dependancy list, meaning it will only run once, on mount.

//Render loading content if loading.
```

```
  if (loading) {
    return <h1>Loading ...</h1>;
  }

  //Render error if error.
  if (err) {
    return <h1>Error: {err}</h1>;
  }

  //Render user profile data if all successful.
  return (
    <>
      <h1>User Profile</h1>
      {userData && (
        <section>
          <h1>{userData.name}</h1>
          <p>Email: {userData.email}</p>
          <p>Phone: {userData.phone}</p>
        </section>
      )}
    </>
  );
}

export default UserProfile;
```

## Exercise Question 3: Detect and Display User's Online/Offline Status

Create an application (i.e. make a component) to detect and display when a **user goes offline and comes back online** (e.g. like MSTeams, Slack, Google Meet, etc.).
The correct solution to this exercise will show "🟢 **Online"** when you are connected to the internet, and show **"🔴 Offline"** when you are disconnected from the internet; so ensure you enable/disable your Wi-Fi to test if your code works.

Task:

- Create a component **NetworkStatus.jsx**.
- Use the **useEffect** hook to detect when the user **goes online or offline** *(see hints below)*.
- Listen for the `online` and `offline` events from the `window` object.

- Update the UI to display **"🟢 Online"** when the user is connected and **"🔴 Offline"** when disconnected *(you can copy and paste these icons for use in your HTML).*
- Ensure that event listeners are **removed when the component unmounts** to prevent memory leaks.

Hints:

- JavaScript provides the `navigator.onLine` property to check if the user is online or not.
- Use `window.addEventListener("online", callback)` and `window.addEventListener("offline", callback)`.
- Clean up event listeners inside the **cleanup function** of `useEffect.`

```
//No comments——this is the easiest exercise of the lot, despite sounding the most difficult.

import React, { useState, useEffect } from "react";

function OnlineStatus() {
  const [isOnline, setIsOnline] = useState(navigator.onLine);

  useEffect(() => {
    const handleOnline = () => setIsOnline(true);
    const handleOffline = () => setIsOnline(false);

    window.addEventListener("online", handleOnline);
    window.addEventListener("offline", handleOffline);

    return () => {
      window.removeEventListener("online", handleOnline);
      window.removeEventListener("offline", handleOffline);
    };
  }, []);

  return (
    <div>
      <h2>Network Status</h2>
      <p>{isOnline ? "🟢 Online" : "🔴 Offline"}</p>
    </div>
  );
}
```

```
export default OnlineStatus;
```