

## Task 3: Mastering React Hooks for a Profile Website

### Objective:

Explore and master React hooks for state management and side effects, create custom hooks, and utilize common hooks in a user profile website. This task aims to enhance your understanding of React hooks and their practical applications in managing state and side effects.

### Pre-requisites:

- Basic understanding of HTML, CSS, and JavaScript
- Familiarity with a code editor like Visual Studio Code
- Basic knowledge of React

### Concepts Covered:

- `useState` for State Management
- `useEffect` for Side Effects
- Custom Hooks
- Common Hooks ( `useContext` and `useReducer` )

### Concepts:

#### 1. Using `useState` for State Management:

Implement `useState` to track the number of profile views.

```
// src/components/Profile.js
import React, { useState } from 'react';

const Profile = ({ user }) => {
  const [views, setViews] = useState(0);

  const incrementViews = () => setViews(views + 1);

  return (
    <div>
      <h1>{user.name}</h1>
      <p>{user.occupation}</p>
      <button onClick={incrementViews}>View Profile</button>
      <p>Profile Views: {views}</p>
    </div>
  );
};

export default Profile;
```

#### 2. Utilizing `useEffect` for Side Effects:

Use `useEffect` in the `ProfileList` component to fetch profile data from a mock API or local JSON file on component mount.

```
// src/components/ProfileList.js
import React, { useEffect, useState } from 'react';
import axios from 'axios';

const ProfileList = () => {
  const [profiles, setProfiles] = useState([]);

  useEffect(() => {
    axios.get('/path/to/profiles.json')
      .then(response => {
        setProfiles(response.data);
      })
      .catch(error => {
        console.error('Error fetching profiles:', error);
      });
  }, []);

  return (
    <div>
      <h2>Profile List</h2>
      <ul>
        {profiles.map(profile => (
          <li key={profile.id}>{profile.name} - {profile.occupation}</li>
        ))}
      </ul>
    </div>
  );
};

export default ProfileList;
```

### 3. Creating Custom Hooks:

Develop a custom hook `useFetch` that encapsulates API calls and can be reused across components.

```
// src/hooks/useFetch.js
import { useState, useEffect } from 'react';
import axios from 'axios';

const useFetch = (url) => {
  const [data, setData] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    axios.get(url)
      .then(response => {
        setData(response.data);
      });
  }, [url]);
};
```

```

        setLoading(false);
    })
    .catch(error => {
        setError(error);
        setLoading(false);
    });
}, [url]);

return { data, loading, error };
};

export default useFetch;

// Usage in ProfileList.js
import React from 'react';
import useFetch from '../hooks/useFetch';

const ProfileList = () => {
    const { data: profiles, loading, error } = useFetch('/path/to/
profiles.json');

    if (loading) return <p>Loading...</p>;
    if (error) return <p>Error loading profiles: {error.message}</p>;

    return (
        <div>
            <h2>Profile List</h2>
            <ul>
                {profiles.map(profile => (
                    <li key={profile.id}>{profile.name} - {profile.occupation}
</li>
                ))}
            </ul>
        </div>
    );
};

export default ProfileList;

```

#### 4. Exploring Common Hooks ( useContext and useReducer ):

Implement useContext and useReducer in a UserContext to manage and distribute user state (like user info and authentication status) throughout the app.

```

// src/contexts/UserContext.js
import React, { createContext, useReducer } from 'react';

const UserContext = createContext();

const initialState = {
  user: null,
  isAuthenticated: false
};

const userReducer = (state, action) => {
  switch (action.type) {
    case 'LOGIN':
      return {
        ...state,
        user: action.payload,
        isAuthenticated: true
      };
    case 'LOGOUT':
      return {
        ...state,
        user: null,
        isAuthenticated: false
      };
    default:
      return state;
  }
};

const UserProvider = ({ children }) => {
  const [state, dispatch] = useReducer(userReducer, initialState);

  return (
    <UserContext.Provider value={{ state, dispatch }}>
      {children}
    </UserContext.Provider>
  );
};

export { UserContext, UserProvider };

// Usage in a component
import React, { useContext } from 'react';
import { UserContext } from '../contexts/UserContext';

const UserProfile = () => {
  const { state, dispatch } = useContext(UserContext);

  const login = () => {
    const user = { name: 'John Doe', occupation: 'Software Engineer' };
    dispatch({ type: 'LOGIN', payload: user });
  };
};

```

```

    };

    const logout = () => {
      dispatch({ type: 'LOGOUT' });
    };

    return (
      <div>
        {state.isAuthenticated ? (
          <div>
            <h1>Welcome, {state.user.name}</h1>
            <p>{state.user.occupation}</p>
            <button onClick={logout}>Logout</button>
          </div>
        ) : (
          <button onClick={login}>Login</button>
        )}
      </div>
    );
  };
};

export default UserProfile;

```

### Setup:

#### 1. Install Visual Studio Code (VS Code):

Download and install VS Code from [Visual Studio Code](#).

#### 2. React Developer Tools:

Install the React Developer Tools browser extension for [Chrome](#) or [Firefox](#).

#### 3. Git:

Install Git for version control. You can download it from [Git](#).

#### 4. Node.js and npm:

Ensure Node.js and npm are installed on your machine. You can download and install them from [Node.js](#).

#### 5. Setting Up the React Project:

- Open your terminal.
- Navigate to the directory where you want to create your project.
- Run the following command to create a new React project:

```
npx create-react-app profile-website
```

#### 6. Installing Additional Dependencies:

- Navigate to the project directory:

```
cd profile-website
```

- Install any additional dependencies:

```
npm install axios
```

## Tasks:

### Part 1: Exploring `useState` and `useEffect` (30 minutes)

#### 1. Using `useState` for State Management (15 minutes):

- In a component like `Profile`, implement `useState` to track the number of profile views.
- Example:

```
// src/components/Profile.js
import React, { useState } from 'react';

const Profile = ({ user }) => {
  const [views, setViews] = useState(0);

  const incrementViews = () => setViews(views + 1);

  return (
    <div>
      <h1>{user.name}</h1>
      <p>{user.occupation}</p>
      <button onClick={incrementViews}>View Profile</button>
      <p>Profile Views: {views}</p>
    </div>
  );
};

export default Profile;
```

- Goal: Learn to manage local state in a component, crucial for features like tracking profile views.
- #### 2. Utilizing `useEffect` for Side Effects (15 minutes):
- Use `useEffect` in the `ProfileList` component to fetch profile data from a mock API or local JSON file on component mount.
  - Example:

```
// src/components/ProfileList.js
import React, { useEffect, useState } from 'react';
import axios from 'axios';

const ProfileList = () => {
  const [profiles, setProfiles] = useState([]);

  useEffect(() => {
    axios.get('/path/to/profiles.json')
      .then(response => {
        setProfiles(response.data);
      })
      .catch(error => {
        console.error('Error fetching profiles:', error);
      });
  }, []);

  return (
    <div>
      <h2>Profile List</h2>
      <ul>
        {profiles.map(profile => (
          <li key={profile.id}>{profile.name} -
            {profile.occupation}</li>
        ))}
      </ul>
    </div>

  );
};

export default ProfileList;
```

- Goal: Understand how to handle side effects, such as data fetching, in functional components.

## Part 2: Custom Hooks and Common Hooks (30 minutes)

### 1. Creating Custom Hooks (15 minutes):

- Develop a custom hook `useFetch` that encapsulates API calls and can be reused across components.
- Example:

```
// src/hooks/useFetch.js
import { useState, useEffect } from 'react';
import axios from 'axios';

const useFetch = (url) => {
  const [data, setData] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    axios.get(url)
      .then(response => {
        setData(response.data);
        setLoading(false);
      })
      .catch(error => {
        setError(error);
        setLoading(false);
      });
  }, [url]);

  return { data, loading, error };
};

export default useFetch;

// Usage in ProfileList.js
import React from 'react';
import useFetch from '../hooks/useFetch';

const ProfileList = () => {
  const { data: profiles, loading, error } = useFetch('/path/to/profiles.json');

  if (loading) return <p>Loading...</p>;
  if (error) return <p>Error loading profiles: {error.message}</p>;

  return (
    <div>
      <h2>Profile List</h2>
      <ul>
        {profiles.map(profile => (
          <li key={profile.id}>{profile.name} - {profile.occupation}</li>
        ))}
      </ul>
    </div>
  );
};

export default ProfileList;
```



- Goal: Learn to abstract logic into custom hooks for better code reuse and simplicity.

## 2. Exploring Common Hooks ( useContext and useReducer ) (15 minutes):

- Implement useContext and useReducer in a UserContext to manage and distribute user state (like user info and authentication status) throughout the app.
- Example:

```
// src/contexts/UserContext.js
import React, { createContext, useReducer } from 'react';

const UserContext = createContext();

const initialState = {
  user: null,
  isAuthenticated: false
};

const userReducer = (state, action) => {
  switch (action.type) {
    case 'LOGIN':
      return {
        ...state,
        user: action.payload,
        isAuthenticated: true
      };
    case 'LOGOUT':
      return {
        ...state,
        user: null,
        isAuthenticated: false
      };
    default:
      return state;
  }
};

const UserProvider = ({ children }) => {
  const [state, dispatch] = useReducer(userReducer, initialState);

  return (
    <UserContext.Provider value={{ state, dispatch }}>
      {children}
    </UserContext.Provider>
  );
};

export { UserContext, UserProvider };

// Usage in a component
import React, { useContext } from 'react';
import { UserContext } from '../contexts/UserContext';
```

```

const UserProfile = () => {
  const { state, dispatch } = useContext(UserContext);

  const login = () => {
    const user = { name: 'John Doe', occupation: 'Software Engineer' };
    dispatch({ type: 'LOGIN', payload: user });
  };

  const logout = () => {
    dispatch({ type: 'LOGOUT' });
  };

  return (
    <div>
      {state.isAuthenticated ? (
        <div>
          <h1>Welcome, {state.user.name}</h1>
          <p>{state.user.occupation}</p>
          <button onClick={logout}>Logout</button>
        </div>
      ) : (
        <button onClick={login}>Login</button>
      )}
    </div>
  );
};

export default UserProfile;

```

- Goal: Gain familiarity with common hooks provided by React for advanced state management and context handling.










### Instructions:

1. Write the required code in the appropriate files in the `src/components` and `src/hooks` directories.
2. Open the terminal and navigate to the project directory.
3. Run the React project using:

```
npm start
```

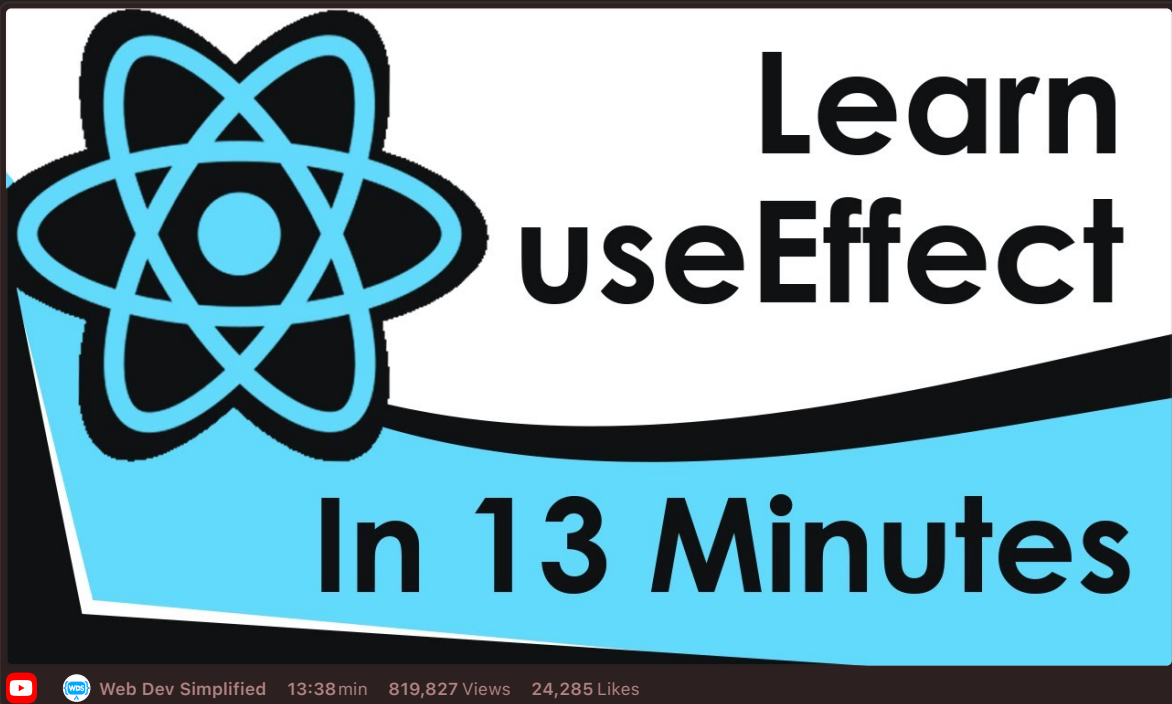
4. Open the project in a web browser to ensure the code displays correctly.
5. Use the browser's developer tools and React Developer Tools to debug and inspect the elements.

### Resources:

-  **Getting Started – React**  
A JavaScript library for building user interfaces  
 <https://reactjs.org/docs/getting-started.html>
-  **Create React App**  
Set up a modern web app by running one command.  
 <https://create-react-app.dev/>
-  **New React Developer Tools – React Blog**  
This blog site has been archived. Go to [react.dev/blog](https://react.dev/blog) to see the recent posts. A month a...  
 <https://reactjs.org/blog/2015/09/02/new-react-developer-tools.html>
-  **Documentation for Visual Studio Code**  
Find out how to set-up and get the most from Visual Studio Code. Optimized for building and d...  
 <https://code.visualstudio.com/docs>
-  **Getting Started | Axios Docs**  
Axios is a promise-based HTTP Client for node.js and the browser. It is isomorphic (= it can run in the...  
<https://axios-http.com/docs/intro>

## Videos:

- 



```
export function useUserContext() {  
  const user = useContext(Dashboard)  
  
  if (user === undefined) {  
    throw new Error('useUserContext')  
  }  
}
```

# useContext

## Simply Explained

Beginner

Cosden Solutions 15:46 min 137,875 Views 6,032 Likes

### GitHub Instructions:

#### 1. Open in Visual Studio Code:

After clicking on the "Open in Visual Studio Code" button from the GitHub Classroom confirmation page, VSCode will open the repository directly. If prompted, select "Open" or "Allow" to open the repository in VSCode.

#### 2. Open the Terminal in VSCode:

In VSCode, open a terminal by selecting Terminal > New Terminal from the top menu.

#### 3. Complete the Task:

In VSCode, write your solution in the appropriate files in the `src/components` and `src/hooks` directories.

#### 4. Run and Test Your Code:

Open your terminal, navigate to your project directory, and run:

```
npm start
```

#### 5. Commit Your Changes:

In the VSCode terminal, add your changes to git:

```
git add src/components/* src/hooks/*
```

Commit your changes with a meaningful message:

```
git commit -m "Completed task 23"
```

## 6. Push Your Changes to Your Repository:

Push your changes to your forked repository:

```
git push origin main
```

## 7. Create a Pull Request:

Go to your repository on GitHub.

Click on the "Pull Requests" tab.

Click the "New Pull Request" button.

Ensure the base repository is the original template repository and the base branch is `main`.

Ensure the head repository is your forked repository and the compare branch is `main`.

Click "Create Pull Request".

Add a title and description for your pull request and submit it.

## Summary of Commands:

```
# Open in Visual Studio Code

# Open terminal in VSCode

# Complete the task by editing src/components/* and src/hooks/*

# Navigate to the project directory
cd profile-website

# Run your code
npm start

# Add, commit, and push your changes
git add src/components/* src/hooks/*
git commit -m "Completed task 3"
git push origin main

# Create a pull request on GitHub
```