

## Task 6: Making API Calls in a Profile Website with React

### Objective:

Learn to make API calls in a React application to fetch and display profile data. This task includes setting up Axios for basic API calls, handling responses and errors, and using advanced data fetching libraries like React Query and SWR for efficient data management.

### Pre-requisites:

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

### Concepts Covered:

- Setting Up Axios and Basic Requests
- Handling API Responses and Errors
- Implementing React Query for Data Fetching
- Exploring SWR for Efficient Data Fetching

### Concepts:

#### 1. Setting Up Axios and Basic Requests:

Install Axios and set up a basic API call to fetch profile data from a mock API or a public API endpoint.

```
npm install axios
```

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

const ProfileList = () => {
  const [profiles, setProfiles] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    axios.get('https://jsonplaceholder.typicode.com/users')
      .then(response => {
        setProfiles(response.data);
        setLoading(false);
      })
      .catch(error => {
        setError(error.message);
        setLoading(false);
      });
  }, []);
}
```

```

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

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

export default ProfileList;

```

## 2. Handling API Responses and Errors:

Display the fetched profile data in a component and implement error handling for the API call.

```

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

const ProfileList = () => {
  const [profiles, setProfiles] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    axios.get('https://jsonplaceholder.typicode.com/users')
      .then(response => {
        setProfiles(response.data);
        setLoading(false);
      })
      .catch(error => {
        setError(error.message);
        setLoading(false);
      });
  }, []);

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

  return (
    <div>
      <h2>Profile List</h2>
      <ul>
        {profiles.map(profile => (

```

```

        <li key={profile.id}>{profile.name} - {profile.email}</li>
      )})
    </ul>
  </div>
);
};

export default ProfileList;

```

### 3. Implementing React Query for Data Fetching:

Install and set up React Query. Use the `useQuery` hook to fetch and display profile data, and explore caching and background refetching features.

```
npm install react-query
```

```

// src/components/ProfileList.js
import React from 'react';
import { useQuery } from 'react-query';
import axios from 'axios';

const fetchProfiles = async () => {
  const { data } = await axios.get('https://
jsonplaceholder.typicode.com/users');
  return data;
};

const ProfileList = () => {
  const { data, error, isLoading } = useQuery('profiles', fetchProfiles);

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

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

export default ProfileList;

```

### 4. Exploring SWR for Efficient Data Fetching:

Install and use SWR for a similar data fetching task. Compare its handling of caching and revalidation strategy with React Query.

```
npm install swr
```

```
// src/components/ProfileList.js
import React from 'react';
import useSWR from 'swr';
import axios from 'axios';

const fetcher = url => axios.get(url).then(res => res.data);

const ProfileList = () => {
  const { data, error } = useSWR('https://jsonplaceholder.typicode.com/users', fetcher);

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

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

export default ProfileList;
```

## 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 Axios, React Query, and SWR:

```
npm install axios react-query swr
```

## Tasks:

### Part 1: Introduction to API Calls with Axios (30 minutes)

#### 1. Setting Up Axios and Basic Requests (15 minutes):

- Install Axios and set up a basic API call to fetch profile data from a mock API or a public API endpoint.
- Example:

```
npm install axios
```

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

const ProfileList = () => {
  const [profiles, setProfiles] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    axios.get('https://jsonplaceholder.typicode.com/users')
      .then(response => {
        setProfiles(response.data);
        setLoading(false);
      })
      .catch(error => {
        setError(error.message);
        setLoading(false);
      });
  }, []);

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

  return (
```

```

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

    export default ProfileList;

```

- Goal: Learn to perform basic HTTP GET requests using Axios to retrieve data.

## 2. Handling API Responses and Errors (15 minutes):

- Display the fetched profile data in a component and implement error handling for the API call.
- Example:

```

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

const ProfileList = () => {
  const [profiles, setProfiles] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    axios.get('https://jsonplaceholder.typicode.com/users')
      .then(response => {
        setProfiles(response.data);
        setLoading(false);
      })
      .catch(error => {
        setError(error.message);
        setLoading(false);
      });
  }, []);

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

  return (
    <div>
      <h

```

2>Profile List

```
{profiles.map(profile => (- {profile.name} - {profile.email}
)))}
);};
```

```
export default ProfileList;
````
```

- Goal: Understand how to manage API responses and handle potential errors during API calls.

## Part 2: Advanced Data Fetching with React Query and SWR (30 minutes)

### 1. Implementing React Query for Data Fetching (15 minutes):

- Install and set up React Query. Use the `useQuery` hook to fetch and display profile data, and explore caching and background refetching features.
- Example:

```
npm install react-query
```

```
// src/components/ProfileList.js
import React from 'react';
import { useQuery } from 'react-query';
import axios from 'axios';

const fetchProfiles = async () => {
  const { data } = await axios.get('https://
jsonplaceholder.typicode.com/users');
  return data;
};

const ProfileList = () => {
  const { data, error, isLoading } = useQuery('profiles', fetchProfiles);

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

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

```
export default ProfileList;
```

- Goal: Learn how React Query simplifies data fetching, caching, and synchronization with server state.

## 2. Exploring SWR for Efficient Data Fetching (15 minutes):

- Install and use SWR for a similar data fetching task. Compare its handling of caching and revalidation strategy with React Query.
- Example:

```
npm install swr
```

```
// src/components/ProfileList.js
import React from 'react';
import useSWR from 'swr';
import axios from 'axios';

const fetcher = url => axios.get(url).then(res => res.data);

const ProfileList = () => {
  const { data, error } = useSWR('https://jsonplaceholder.typicode.com/users', fetcher);

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

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

export default ProfileList;
```

- Goal: Understand the SWR approach for efficient and fast data fetching, and how it differs from React Query in managing server state.

### Instructions:

1. Write the required code in the appropriate files in the `src/components` directory.
2. Open the terminal and navigate to the project directory.









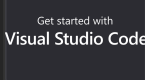



3. Run the React project using:

```
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 | 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>
-  **Overview | TanStack Query React Docs**  
TanStack Query (FKA React Query) is often described as the missing data-fetching library f...  
 <https://react-query.tanstack.com/overview>
-  **404: This page could not be found**  
<https://swr.vercel.app/getting-started>  
<https://swr.vercel.app/getting-started>
-  **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/>
-  **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>

## Videos:

**Axios**  
Crash Course

TraversyMedia.com

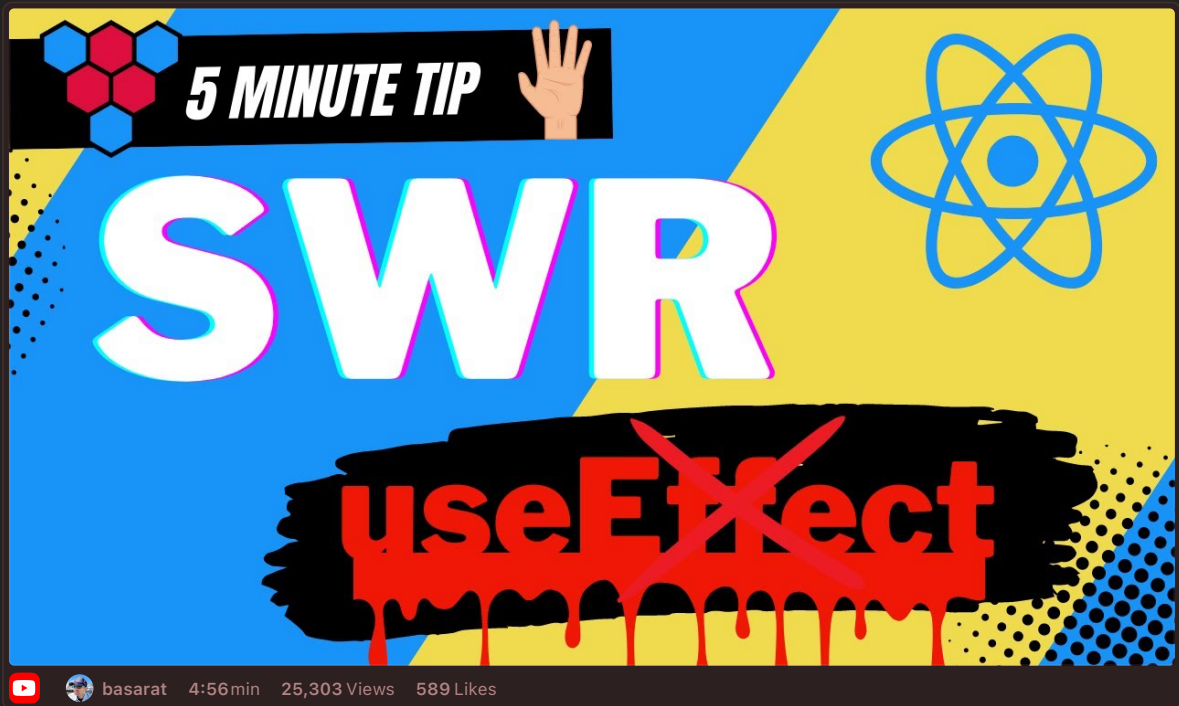
Traversy Media 42:20 min 365,009 Views 8,284 Likes

**REACT QUERY**

**INTRODUCTION**

01

Codevolution 5:26 min 226,901 Views 3,234 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` directory.

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/*
```

Commit your changes with a meaningful message:

```
git commit -m "Completed task 26"
```

## 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/*

# Navigate to the project directory
cd profile-website

# Run your code
npm start

# Add, commit, and push your changes
git add src/components/*
git commit -m "Completed task 6"
git push origin main

# Create a pull request on GitHub
```