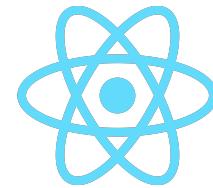




# Pemrograman Sisi Cleint

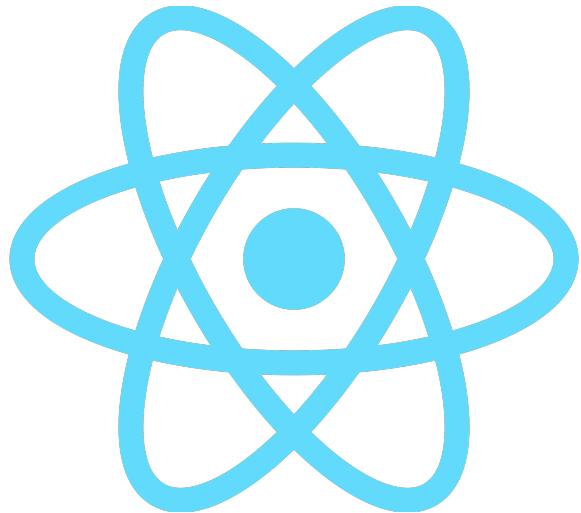
## ReactJS Fetch API

Muhammad Syaifur Rohman, S.Kom, M.CS





# Definisi



FYI

Dalam pengembangan aplikasi web modern, penggunaan API (Application Programming Interface) sangat umum untuk berinteraksi dengan server dan mengambil atau mengirim data. Menggunakan API memungkinkan aplikasi React untuk berkomunikasi dengan server dan memperbarui UI berdasarkan data yang dinamis.

# Persiapan

## Metode Fetch API

Fetch API adalah cara standar dan modern untuk melakukan permintaan HTTP di JavaScript. Ini mendukung promise, sehingga membuat penanganan permintaan asinkron menjadi lebih mudah dan bersih.

## API

Buat komponen yang akan mengambil dan menampilkan data dari API. Misalnya, kita akan mengambil daftar pengguna dari API

<https://jsonplaceholder.typicode.com/users>

# Melakukan GET

**useState:** Hook yang digunakan untuk menyimpan state users dan loading.

**useEffect:** Hook yang digunakan untuk menjalankan efek samping, seperti permintaan fetch. Efek ini akan dijalankan sekali setelah komponen dimuat.

**fetch:** Fungsi yang digunakan untuk mengambil data dari API. Hasilnya kemudian diubah menjadi JSON dan disimpan dalam state users.

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

function UserList() {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetch('https://jsonplaceholder.typicode.com/users')
      .then(response => response.json())
      .then(data => {
        setUsers(data);
        setLoading(false);
      })
      .catch(error => {
        console.error('Error fetching data:', error);
        setLoading(false);
      });
  }, []);

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

  return
    <div>
      <h1>Daftar Pengguna</h1>
      <ul>
        {users.map(user => (
          <li key={user.id}>{user.name}</li>
        ))}
      </ul>
    </div>
}

export default UserList;
```

GET

TRY...  
→

# Melakukan POST

**useState:** Hook yang digunakan untuk menyimpan state name dan email.

**handleSubmit:** Fungsi yang menangani pengiriman form. Fungsi ini mencegah perilaku default form, kemudian mengirim data ke server menggunakan fetch dengan metode POST.

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

function AddUser() {
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');

  const handleSubmit = (event) => {
    event.preventDefault();
    fetch('https://jsonplaceholder.typicode.com/users', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        name: name,
        email: email
      })
    })
    .then(response => response.json())
    .then(data => {
      console.log('User added:', data);
      // Reset form
      setName('');
      setEmail('');
    })
    .catch(error => {
      console.error('Error adding user:', error);
    });
  };

  return (
    <div>
      <h1>Tambahkan Pengguna</h1>
      <form onSubmit={handleSubmit}>
        <input
          type="text"
          placeholder="Nama"
          value={name}
          onChange={(e) => setName(e.target.value)}
        />
        <input
          type="email"
          placeholder="Email"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
        />
        <button type="submit">Tambahkan</button>
      </form>
    </div>
  );
}

export default AddUser;
```

POST





# GET dan POST

Kita dapat menggabungkan kedua operasi ini dalam satu komponen aplikasi. Berikut adalah contoh komponen yang menampilkan daftar pengguna dan memungkinkan penambahan pengguna baru.

**UserApp:** Komponen utama yang mengelola state users, loading, name, dan email. Komponen ini menampilkan daftar pengguna dan form untuk menambahkan pengguna baru.

**useEffect:** Mengambil data pengguna dari API ketika komponen pertama kali dimuat.

**handleSubmit:** Mengirim data pengguna baru ke API dan menambahkan pengguna tersebut ke dalam state users setelah berhasil ditambahkan.

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

function UserApp() {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(true);
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');

  useEffect(() => {
    fetch('https://jsonplaceholder.typicode.com/users')
      .then(response => response.json())
      .then(data => {
        setUsers(data);
        setLoading(false);
      })
      .catch(error => {
        console.error('Error fetching data:', error);
        setLoading(false);
      });
  }, []);

  const handleSubmit = (event) => {
    event.preventDefault();
    fetch('https://jsonplaceholder.typicode.com/users', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        name: name,
        email: email
      })
    })
      .then(response => response.json())
      .then(data => {
        setUsers([...users, data]);
        setName('');
        setEmail('');
      })
      .catch(error => {
        console.error('Error adding user:', error);
      });
  };

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

```
return (
  <div>
    <h1>Daftar Pengguna</h1>
    <ul>
      {users.map(user => (
        <li key={user.id}>{user.name}</li>
      ))}
    </ul>
    <h1>Tambahkan Pengguna</h1>
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        placeholder="Nama"
        value={name}
        onChange={(e) => setName(e.target.value)} />
      <input
        type="email"
        placeholder="Email"
        value={email}
        onChange={(e) => setEmail(e.target.value)} />
      <button type="submit">Tambahkan</button>
    </form>
  </div>
);

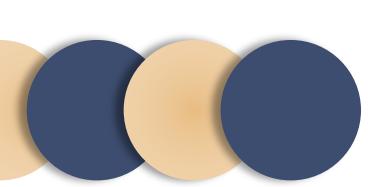
export default UserApp;
```

TRY...

## Daftar Pengguna

- Leanne Graham
- Ervin Howell
- Clementine Bauch
- Patricia Lebsack
- Chelsey Dietrich
- Mrs. Dennis Schulist
- Kurtis Weissnat
- Nicholas Runolfsdottir V
- Glenna Reichert
- Clementina DuBuque
- Monkey D Luffy

## Tambahkan Pengguna



# Best Practice

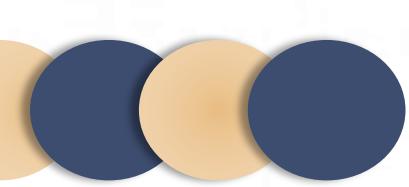
**Error Handling:** Selalu tangani kesalahan dalam permintaan API untuk memberikan feedback yang baik kepada pengguna dan mencegah crash aplikasi.

**Loading States:** Gunakan state untuk mengelola status loading sehingga pengguna mengetahui bahwa data sedang diambil atau dikirim.

**Reusable Functions:** Pertimbangkan untuk membuat fungsi API reusable untuk memudahkan pemeliharaan dan konsistensi dalam menangani permintaan.

**Optimisasi Performa:** Pertimbangkan caching data atau menggunakan state management library seperti Redux atau Context API untuk mengelola state global jika aplikasi menjadi kompleks.

**Keamanan:** Pastikan untuk menangani data sensitif dengan hati-hati dan menghindari kebocoran informasi.



# Referensi

1. Eran Krinsbruner, **A Frontend Web Developer's Guide to Testing**, 2022
2. Adam Boduch , Roy Derks , Mikhail Sakhniuk, **React and React Native - Fourth Edition**, 2022
3. Maya Shavin , Raymond Camden, **Frontend Development Projects with Vue.js 3 - Second Edition**, 2023
4. Waweru Mwaura, **End-to-End Web Testing with Cypress**, 2021
5. <https://roadmap.sh/frontend>
6. <https://jsonplaceholder.typicode.com/users>



# Thanks

Muhammad Syaifur Rohman, S.Kom, M.CS

