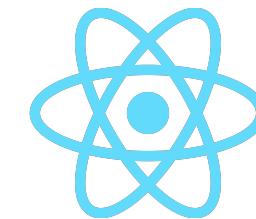




# Pemrograman Sisi Cleint

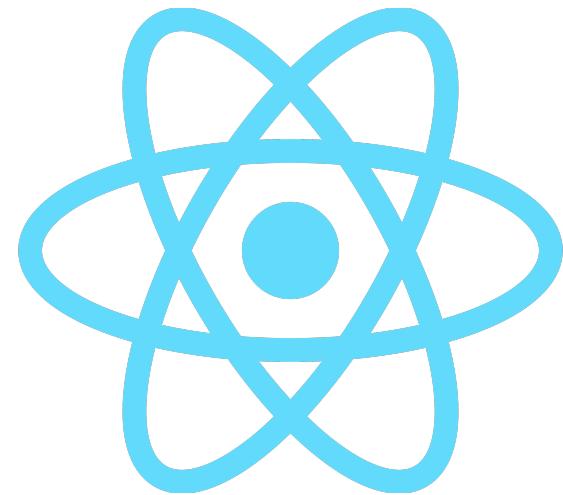
## ReactJS Context API

Muhammad Syaifur Rohman, S.Kom, M.CS





# Definisi



FYI

Context API adalah fitur bawaan di React yang menyediakan cara untuk berbagi data melalui seluruh komponen aplikasi tanpa harus meneruskan props secara manual di setiap level. Dengan Context API, Anda dapat mengelola data global yang perlu diakses oleh banyak komponen, seperti tema, bahasa, atau informasi pengguna.

# Konsep

## Penggunaan

Kata "API" di Context API mengacu pada "Application Programming Interface" yang dalam konteks ini adalah sekumpulan fungsi dan mekanisme yang disediakan oleh React untuk mengelola dan berbagi state secara efisien. Ini bukan API seperti REST API atau Web API yang digunakan untuk komunikasi server-klien, melainkan API internal React untuk manajemen state.

## Why

Context API sangat berguna ketika Anda memiliki data atau state yang perlu diakses oleh banyak komponen di berbagai level dalam pohon komponen. Ini menghindari masalah yang dikenal sebagai "prop drilling," di mana data harus diteruskan melalui banyak level komponen, membuat kode sulit untuk dikelola dan dipelihara.

# Konsep

## Konsep Dasar

**Context:** Tempat untuk menyimpan data yang ingin dibagikan.

**Provider:** Komponen yang membungkus bagian dari aplikasi Anda dan menyediakan context.

**Consumer:** Komponen atau hook yang mengakses data dari context.

## Steps

### Langkah-langkah Menggunakan Context API

1. Membuat Context
2. Membungkus Aplikasi dengan Provider
3. Mengakses Context dengan Consumer atau Hook

# Membuat Context

```
// context/ThemeContext.js
import React, { createContext, useState } from 'react';

export const ThemeContext = createContext();

export const ThemeProvider = ({ children }) => {
  const [theme, setTheme] = useState('light');

  const toggleTheme = () => {
    setTheme((prevTheme) => (prevTheme === 'light' ? 'dark' : 'light'));
  };

  return (
    <ThemeContext.Provider value={{ theme, toggleTheme }}>
      {children}
    </ThemeContext.Provider>
  );
};
```

Langkah pertama adalah membuat context menggunakan fungsi createContext.

**createContext:** Membuat context baru bernama ThemeContext.

**ThemeProvider:** Komponen yang membungkus bagian aplikasi dan menyediakan state theme serta fungsi toggleTheme kepada komponen anak melalui context.

# Membungkus Aplikasi dengan Provider

```
// App.js
import React from 'react';
import { ThemeProvider } from './context/ThemeContext';
import ThemedComponent from './ThemedComponent';

function App() {
  return (
    <ThemeProvider>
      <ThemedComponent />
    </ThemeProvider>
  );
}

export default App;
```

Anda perlu membungkus bagian dari aplikasi Anda dengan `ThemeProvider` untuk menyediakan context ke komponen anak.

**ThemeProvider:** Membungkus komponen `ThemedComponent` dan menyediakan context ke komponen tersebut.

# Mengakses Context dengan Consumer atau Hook

```
// ThemedComponent.js
import React, { useContext } from 'react';
import { ThemeContext } from './context/ThemeContext';

function ThemedComponent() {
  const { theme, toggleTheme } = useContext(ThemeContext);

  return (
    <div style={{ background: theme === 'light' ? '#fff' : '#333', color: theme === 'light' ? '#000' : 'fff' }}>
      <p>Current theme: {theme}</p>
      <button onClick={toggleTheme}>Toggle Theme</button>
    </div>
  );
}

export default ThemedComponent;
```

Anda dapat mengakses data dari context menggunakan hook useContext atau komponen Context.Consumer.

**useContext:** Hook yang digunakan untuk mengakses context. Mengambil nilai theme dan fungsi toggleTheme dari ThemeContext.

# Menggunakan Komponen Context.Consumer

```
// ThemedComponent.js
import React from 'react';
import { ThemeContext } from './context/ThemeContext';

function ThemedComponent() {
  return (
    <ThemeContext.Consumer>
      {({ theme, toggleTheme }) => (
        <div style={{ background: theme === 'light' ? '#fff' : '#333', color: theme === 'light' ? '#000' : 'fff' }}>
          <p>Current theme: {theme}</p>
          <button onClick={toggleTheme}>Toggle Theme</button>
        </div>
      )}
    </ThemeContext.Consumer>
  );
}

export default ThemedComponent;
```

**ThemeContext.Consumer:** Komponen yang digunakan untuk mengakses context tanpa menggunakan hook. Menyediakan fungsi render prop yang menerima context value sebagai argumen.

# Hasil



The image shows a large grid of binary digits (0s and 1s) on a light gray background. The grid is composed of many smaller grids of binary digits, creating a repeating pattern across the entire area. This visual effect is similar to a digital watermark or a binary matrix.



# Studi Kasus

## User Authentication

Context API juga sangat berguna untuk mengelola user authentication, di mana informasi pengguna perlu diakses oleh banyak komponen di seluruh aplikasi.



# Step 1 – User Context

**AuthProvider:** Menyediakan state user dan fungsi login dan logout kepada komponen anak melalui context.

```
// context/AuthContext.js
import React, { createContext, useState } from 'react';

export const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);

  const login = (userData) => {
    setUser(userData);
  };

  const logout = () => {
    setUser(null);
  };

  return (
    <AuthContext.Provider value={{ user, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
};
```

# Step 2 – Provider

Membungkus Aplikasi dengan Provider

```
// App.js
import React from 'react';
import { AuthProvider } from './context/AuthContext';
import Login from './Login';
import UserProfile from './UserProfile';

function App() {
  return (
    <AuthProvider>
      <Login />
      <UserProfile />
    </AuthProvider>
  );
}

export default App;
```

# Step 3 – Login Component

```
// Login.js
import React, { useContext, useState } from 'react';
import { AuthContext } from './context/AuthContext';

function Login() {
  const { login } = useContext(AuthContext);
  const [username, setUsername] = useState('');

  const handleSubmit = (e) => {
    e.preventDefault();
    login({ name: username });
  };

  return (
    <form onSubmit={handleSubmit}>
      <input type="text" value={username} onChange={(e) => setUsername(e.target.value)} />
      <button type="submit">Login</button>
    </form>
  );
}

export default Login;
```

Menggunakan Context di Komponen

# Step 4 – UserProfile Component

**Login:** Komponen untuk login. Mengambil fungsi login dari AuthContext.

**UserProfile:** Komponen untuk menampilkan profil pengguna. Mengambil state user dan fungsi logout dari AuthContext.

```
// UserProfile.js
import React, { useContext } from 'react';
import { AuthContext } from './context/AuthContext';

function UserProfile() {
  const { user, logout } = useContext(AuthContext);

  if (!user) {
    return <p>Please log in.</p>;
  }

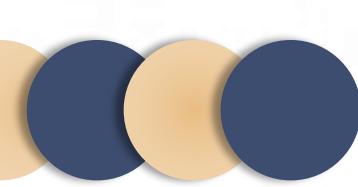
  return (
    <div>
      <p>Welcome, {user.name}</p>
      <button onClick={logout}>Logout</button>
    </div>
  );
}

export default UserProfile;
```

# Hasil



The image shows a large grid of binary digits (0s and 1s) on a light gray background. The grid is composed of many smaller grids of binary digits, creating a repeating pattern across the entire area. This visual effect is similar to a digital watermark or a binary matrix.



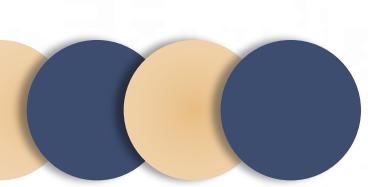
# Best Practice

**Provider Wrap:** Selalu bungkus bagian pohon komponen yang membutuhkan akses ke state context dengan Provider.

**Konsumen:** Gunakan useContext hook untuk mengakses context dengan cara yang lebih bersih dan efisien.

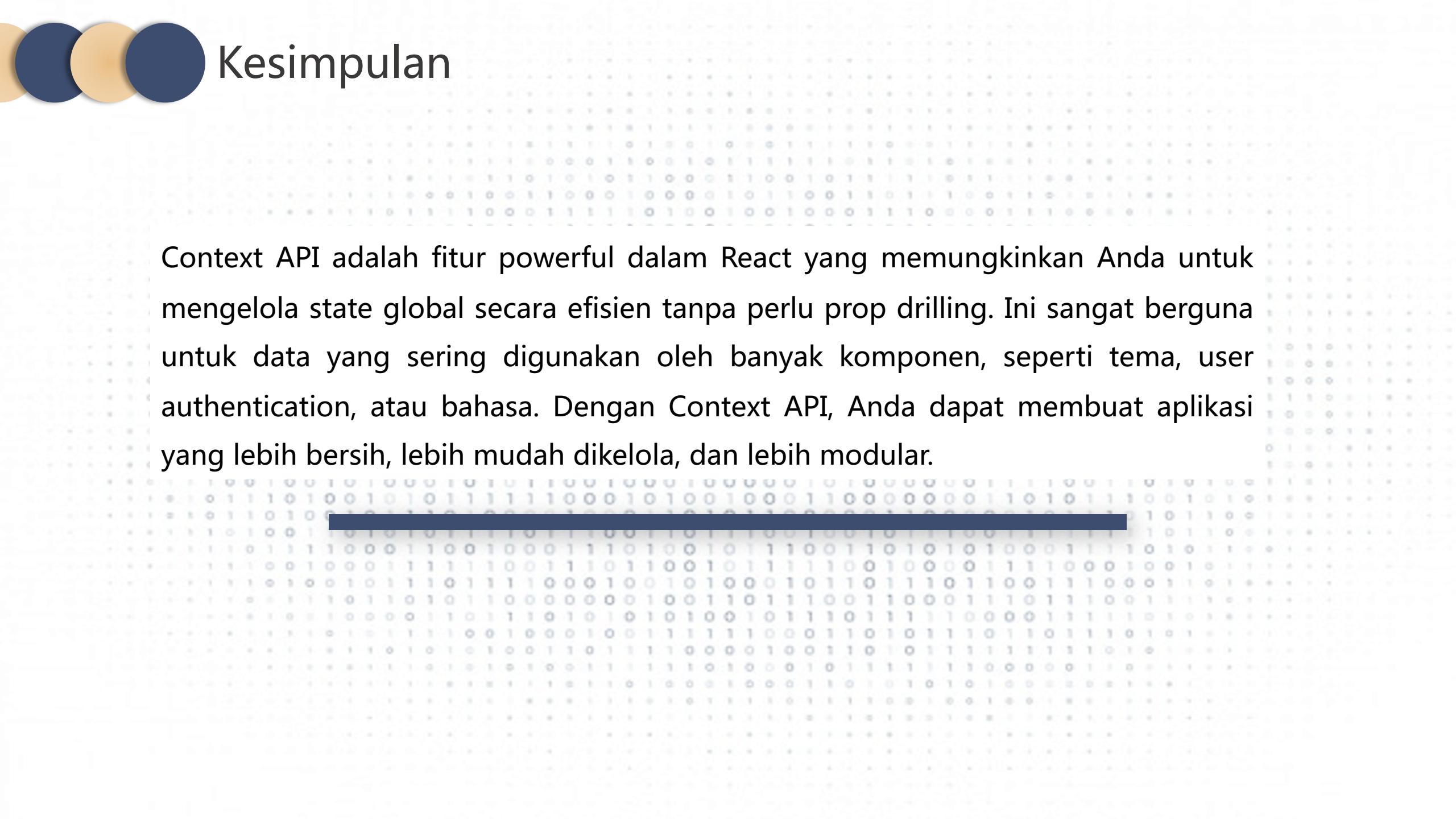
**Memoization:** Gunakan React.memo atau useMemo untuk menghindari rendering ulang yang tidak perlu.

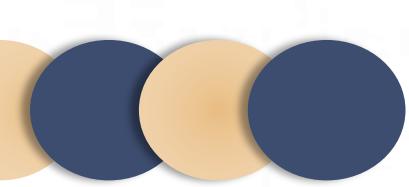
**Organisasi Kode:** Pisahkan context, provider, dan konsumen ke dalam modul yang terorganisir untuk meningkatkan keterbacaan dan pemeliharaan kode.



# Kesimpulan

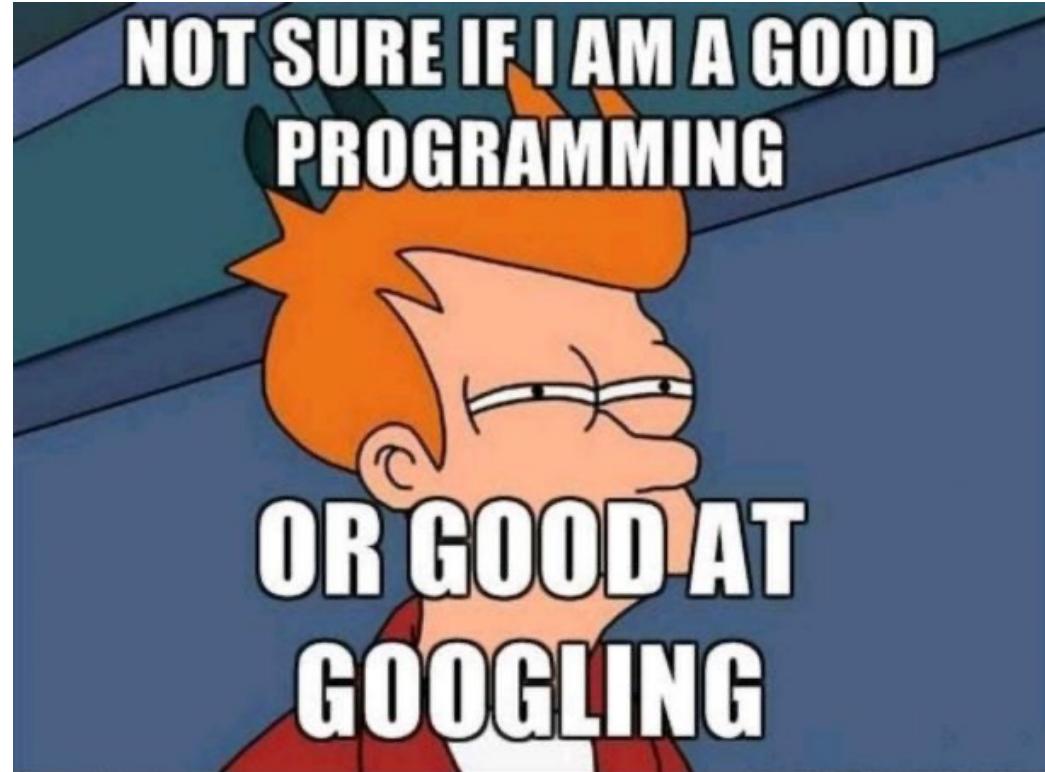
Context API adalah fitur powerful dalam React yang memungkinkan Anda untuk mengelola state global secara efisien tanpa perlu prop drilling. Ini sangat berguna untuk data yang sering digunakan oleh banyak komponen, seperti tema, user authentication, atau bahasa. Dengan Context API, Anda dapat membuat aplikasi yang lebih bersih, lebih mudah dikelola, dan lebih modular.





# 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>
7. <https://redux.js.org/tutorials/fundamentals/part-2-concepts-data-flow>



# Thanks

Muhammad Syaifur Rohman, S.Kom, M.CS

