Dokumentasi Login SSO untuk Front-end dan Back-end

Dokumen ini menyajikan panduan teknis lengkap tentang implementasi login Single Sign-On (SSO) dari perspektif front-end dan back-end. Ditujukan untuk dibagikan kepada pengguna lain, panduan ini mencakup langkah-langkah rinci, contoh kode, dan penjelasan komprehensif tentang proses login SSO.

Pengenalan Login SSO

Single Sign-On (SSO) adalah mekanisme otentikasi yang memungkinkan pengguna untuk mengakses beberapa aplikasi dengan satu set kredensial. Dalam konteks ini, kita akan membahas implementasi SSO menggunakan React untuk front-end dan sebuah backend API.

1 ## Keuntungan SSO

SSO meningkatkan keamanan dengan mengurangi jumlah kata sandi yang perlu diingat pengguna, serta menyederhanakan proses login untuk berbagai aplikasi.

2 ## Komponen Utama

Implementasi SSO melibatkan integrasi antara aplikasi klien (front-end), server otentikasi SSO, dan backend aplikasi.

3 ## Alur Kerja

Proses SSO dimulai dengan redirect ke server otentikasi, diikuti dengan verifikasi token, dan akhirnya akses ke aplikasi.

Implementasi Front-end: Komponen React

Implementasi front-end menggunakan React melibatkan pembuatan komponen AuthLogin yang menangani proses login. `AuthLogin` adalah komponen utama yang saya gunakan dalam menangani proses autentikasi aplikasi, termasuk integrasi dengan SSO (Single Sign-On). Berikut adalah penjelasan dari tiap langkah dalam proses login SSO yang ada di komponen ini:

1. Persiapan Alur Login SSO
   Komponen AuthLogin memuat fungsi untuk mengarahkan pengguna ke halaman otentikasi SSO eksternal. Saat pengguna memilih opsi login SSO, aplikasi akan membangun URL otentikasi yang mencakup parameter penting, seperti URL tujuan kembali (redirect URI). Hal ini memastikan bahwa setelah berhasil login, pengguna dikembalikan ke aplikasi kita.
2. Fungsi Redirect ke Halaman SSO Dalam proses redirect, aplikasi menyusun URL login yang menyertakan informasi seperti redirectUri (alamat URL aplikasi yang menerima hasil login) dan data pengguna sementara sebagai simulasi. Fungsi ini kemudian mengarahkan pengguna ke halaman login SSO eksternal untuk memulai proses autentikasi.
3. Parsing Parameter dari URL Setelah berhasil login di server SSO, pengguna akan di-redirect kembali ke aplikasi kita dengan menyertakan beberapa parameter dalam URL, seperti token autentikasi dan informasi pengguna. Di sini, aplikasi akan membaca parameter tersebut menggunakan fungsi parsing URL. Data yang diperoleh, seperti token dan informasi pengguna, akan disimpan agar dapat digunakan dalam sesi aplikasi.
4. Pengelolaan Data Hasil Login Setelah URL diparsing, aplikasi menyimpan token dan data pengguna ke dalam storage lokal atau session sesuai kebutuhan. Informasi ini kemudian digunakan untuk mengatur status login pengguna di aplikasi.

Alur Kerja SSO Login di Front-end

- Server SSO melakukan autentikasi dan mengembalikan pengguna ke aplikasi dengan token dan informasi pengguna.
- Parameter URL dari hasil redirect diparsing dan data login disimpan dalam aplikasi.
- Status login pengguna diatur menggunakan token untuk otentikasi di seluruh aplikasi.

Fungsi Utama dalam Komponen AuthLogin

1

apiFetch

Fungsi ini menangani permintaan API dengan menambahkan token otentikasi ke header jika tersedia. Jika respons adalah 401 (Unauthorized), token dihapus dan pengguna diarahkan ke halaman login.

2

handleLogin

Fungsi ini menangani proses login dengan mengirimkan kredensial ke endpoint API. Jika berhasil, token disimpan di localStorage dan pengguna diarahkan ke dashboard.

3

useEffect

useEffect digunakan untuk memeriksa parameter URL setelah komponen dimuat. Jika token SSO ditemukan, token tersebut disimpan di localStorage, dan halaman dimuat ulang untuk memperbarui status autentikasi pengguna, memastikan akses yang tepat setelah login SSO.

4

parseQuery dan parseQueryOptions

Fungsi-fungsi ini mengurai parameter URL untuk mengekstrak token dan informasi pengguna setelah redirect SSO.

5

redirectToSSO

Fungsi ini mengarahkan pengguna ke halaman login SSO dengan menyertakan URI redirect yang dienkode.

Implementasi useEffect untuk Penanganan Token SSO

Komponen menggunakan useEffect untuk menangani ekstraksi token dari parameter URL setelah redirect SSO:

```
useEffect(() => {
  const options = parseQuery(window.location.search);
  const tokenSSO = options?.[0]?.token;
  if (options) {
    localStorage.setItem('token', tokenSSO)
    const tokenSSOGet = localStorage.getItem('token');
    if (tokenSSOGet) {
      window.location.reload()
    }
  }
}, []);
```

Kode ini mengekstrak token dari URL, menyimpannya di localStorage, dan me-refresh halaman jika token ditemukan.

Fungsi parseQueryOptions

Deskripsi

Fungsi parseQueryOptions digunakan untuk mengurai parameter dari URL yang diterima aplikasi setelah pengguna berhasil login melalui SSO. Fungsi ini memastikan bahwa parameter seperti token dan user dapat diambil dan disimpan dalam objek yang lebih mudah diakses.

Struktur Kode

- **Parameter:** location
  location adalah parameter URL lengkap yang diterima saat pengguna di-redirect kembali ke aplikasi. Parameter ini berisi data dalam format query string (contoh: ?token=abc123&user=admin).
- **Objek** optionValues**:**
  Sebuah objek yang digunakan untuk menyimpan hasil parsing, dengan nilai awal berupa string kosong untuk kunci token dan user.

Langkah-langkah Utama

1. **Parsing URL:**
   Memanfaatkan library query-string, fungsi ini menguraikan query string dari location menjadi objek query yang lebih mudah dibaca dan diakses.
2. **Validasi Parameter:**
   Fungsi ini memeriksa apakah parameter yang diterima (option, token, user) adalah string. Jika valid, maka nilai tersebut akan disimpan di objek optionValues.
3. **Return Data Terstruktur:**
   Fungsi mengembalikan objek optionValues yang telah terisi dengan nilai token dan user untuk digunakan dalam aplikasi, seperti dalam autentikasi atau tampilan profil pengguna.

Contoh Penggunaan

Misalkan URL redirect yang diterima aplikasi adalah sebagai berikut:

```
http://localhost:3000/application/login?token=abc123&user=admin
```

Saat parseQueryOptions menerima location tersebut, hasil parsing akan mengembalikan objek seperti:

```
{
    token: 'abc123',
    user: 'admin'
}
```

Kode Fungsi

```
function parseQueryOptions(location) {
    const query = queryString.parse(location);

    const optionValues = {
        token: '',
        user: ''
    };

    if (typeof query.option === "string") {
        optionValues.option = query.option;
    }
    if (typeof query.token === "string") {
        optionValues.token = query.token;
    }
    if (typeof query.user === "string") {
        optionValues.user = query.user;
    }

    return optionValues;
}
```

Catatan Integrasi

Fungsi ini dapat langsung diintegrasikan ke komponen front-end yang menangani alur login SSO. Setelah data diambil, objek optionValues ini dapat digunakan untuk memverifikasi status login pengguna dan mengakses data profil secara langsung, sehingga memungkinkan aplikasi untuk menampilkan informasi pengguna yang telah login.

Fungsi redirectToSSO

Deskripsi

Fungsi `redirectToSSO` bertugas untuk mengarahkan pengguna dari aplikasi ke halaman autentikasi SSO eksternal. Dalam alur ini, fungsi membuat URL lengkap yang menyertakan parameter `redirectUri`, yang memastikan pengguna akan kembali ke halaman aplikasi setelah login berhasil.

Struktur Kode

- **Parameter** `redirectUri`**:**
  `redirectUri` adalah URL yang akan digunakan sebagai tujuan redirect setelah proses autentikasi SSO selesai. Dalam contoh ini, URL aplikasi yang digunakan adalah `http://localhost:5000/application/login`. Fungsi ini meng-encode URL agar sesuai dengan format URL.
- **Parameter** `userParamEncoded`**:**
  `userParamEncoded` adalah contoh data JSON yang telah di-encode, berisi informasi dasar pengguna (seperti ID dan username) yang bisa disimulasikan. Informasi ini ditambahkan untuk keperluan uji coba atau dalam konteks tertentu di mana data pengguna perlu diteruskan.
- **Mengalihkan Pengguna ke URL SSO:**
  `window.location.href` mengarahkan pengguna ke halaman autentikasi SSO dengan menambahkan `redirectUri` sebagai parameter di URL. Ini memastikan bahwa setelah login di halaman SSO, pengguna akan dikembalikan ke aplikasi dengan membawa token dan informasi pengguna.

Alur Kerja Fungsi

1. **Pembuatan URL Redirect:**
   Fungsi menyusun URL yang terdiri dari alamat SSO eksternal serta parameter `redirectUri`, yang menentukan ke mana pengguna akan diarahkan setelah autentikasi selesai.
2. **Pengarahan Pengguna ke URL SSO:**
   Dengan `window.location.href`, aplikasi mengalihkan pengguna ke halaman login SSO, membawa `redirectUri` sebagai bagian dari URL untuk memastikan bahwa aplikasi dapat menerima hasil autentikasi dengan benar.

Contoh Penggunaan

Misalkan URL SSO yang digunakan adalah:

```
https://3wzg6m6x-3000.asse.devtunnels.ms/?
redirect_uri=http%3A%2F%2Flocalhost%3A5000%2Fapplication%2Flogin
```

Saat fungsi `redirectToSSO` dijalankan, pengguna akan diarahkan ke URL ini, kemudian setelah login di halaman SSO, pengguna akan kembali ke URL aplikasi (`http://localhost:5000/application/login`) dengan membawa token dan data pengguna.

Kode Fungsi

```
// Redirect to SSO login
const redirectToSSO = () => {
  const redirectUri = encodeURIComponent('http://localhost:5000/application/login');
  const userParamEncoded = encodeURIComponent(JSON.stringify({ message: "User found", user: { id: 1,
username: "lawak" } }));
  window.location.href = `https://3wzg6m6x-3000.asse.devtunnels.ms/?redirect_uri=${redirectUri}`;
};
```

Catatan Integrasi

Fungsi ini dapat langsung digunakan di aplikasi dengan mengaktifkannya sebagai tindakan pada tombol login SSO. Penggunaan `redirectUri` yang terstruktur membantu aplikasi memastikan bahwa pengguna dapat kembali dengan aman setelah proses autentikasi SSO selesai.

Formulir Login dan Validasi pada AuthLogin

Deskripsi

Formulir login pada komponen AuthLogin mendukung dua metode autentikasi:

1. **Login Biasa:** Pengguna dapat masuk dengan memasukkan email dan password.
2. **Login dengan SSO:** Tombol terpisah untuk mengarahkan pengguna ke halaman login SSO eksternal.

Komponen ini menggunakan **Formik** untuk manajemen form dan **Yup** untuk validasi, mempermudah pengelolaan nilai input dan kesalahan dalam satu kerangka kerja.

Struktur Formulir Login Biasa

- **Email Field:**
  Menggunakan TextField untuk menangani input email. Validasi dilakukan dengan **Yup** untuk memastikan bahwa email memiliki format yang benar dan wajib diisi.
- **Password Field:**
  OutlinedInput digunakan untuk input password. Pengguna juga bisa mengklik ikon visibilitas untuk melihat atau menyembunyikan password. Validasi Yup memastikan bahwa password wajib diisi.
- **Validasi Form:**
  Validasi menggunakan Yup mencakup format email yang benar dan kebutuhan bahwa kedua field (email dan password) harus diisi sebelum form dapat dikirimkan.

Struktur Tombol Login dengan SSO

- **Tombol Login SSO:**
  Tombol tambahan ini memungkinkan pengguna untuk login dengan SSO. Ketika diklik, fungsi redirectToSSO akan dijalankan, mengarahkan pengguna ke halaman autentikasi SSO eksternal, tanpa memerlukan input email dan password.

Alur Kerja

1. **Login Biasa:**
   Pengguna mengisi email dan password, lalu mengklik tombol "Log In" untuk masuk. Formik akan memvalidasi input, menampilkan pesan kesalahan jika ada, dan menjalankan fungsi handleLogin jika semua validasi terpenuhi.
2. **Login dengan SSO:**
   Pengguna cukup mengklik tombol "Login with SSO", yang langsung mengarahkan ke halaman login SSO eksternal tanpa input tambahan.

Komponen Utama

- **Formik:** Mengelola status form dan menjalankan validasi dengan Yup.
- **Yup Validation:** Memastikan input sesuai, seperti email dalam format yang benar dan password yang tidak boleh kosong.
- **Button for SSO Login:** Menyediakan akses cepat ke halaman autentikasi SSO eksternal.

Kode Fungsi:

```
<Formik
  initialValues={{
    email: '',
    password: '',
    submit: null,
  }}
  validationSchema={Yup.object().shape({
    email: Yup.string().email('Invalid email').max(255).required('Email is required'),
    password: Yup.string().max(255).required('Password is required'),
  })}
  onSubmit={handleLogin}
>
  {({ errors, handleBlur, handleChange, handleSubmit, isSubmitting, touched, values }) => (
    <form noValidate onSubmit={handleSubmit}>
      <TextField
        error={Boolean(touched.email && errors.email)}
        fullWidth
        helperText={touched.email && errors.email}
        label="Email Address"
        margin="normal"
        name="email"
        onBlur={handleBlur}
        onChange={handleChange}
        type="email"
        value={values.email}
        variant="outlined"
      />

      <OutlinedInput
        id="outlined-adornment-password"
        type="password"
        value={values.password}
        name="password"
        onBlur={handleBlur}
        onChange={handleChange}
        label="Password"
        endAdornment={
          <InputAdornment position="end">
            <IconButton
              aria-label="toggle password visibility"
              onClick={handleClickShowPassword}
              edge="end"
            >
              {showPassword ? <Visibility /> : <VisibilityOff />}
            </IconButton>
          </InputAdornment>
        }
      />
      {errors.submit && (
        <Box mt={3}>
          <FormHelperText error>{errors.submit}</FormHelperText>
        </Box>
      )}

      <Box mt={2}>
        <Button
          color="primary"
          disabled={isSubmitting}
          fullWidth
          size="large"
          type="submit"
          variant="contained"
        >
          Log In
        </Button>
      </Box>

      <Box mt={2}>
        <Button
          fullWidth
          onClick={redirectToSSO}
          variant="contained"
        >
          Login with SSO
        </Button>
      </Box>
    </form>
  )}
</Formik>
```

Pemahaman Teknis tentang Login SSO di Front-end

Implementasi Single Sign-On (SSO) di aplikasi front-end memberikan banyak keuntungan:

1. **Pengalaman Pengguna**: SSO memungkinkan pengguna mengakses berbagai aplikasi dengan satu login, meningkatkan kenyamanan.
2. **Keamanan**: Mengurangi risiko kebocoran data dengan memungkinkan penggunaan autentikasi yang lebih kuat, seperti otentikasi multi-faktor.
3. **Pengelolaan Pengguna**: Mempermudah manajemen pengguna dengan mengelola akses di satu tempat.
4. **Integrasi dengan API**: Memudahkan aplikasi terhubung dengan backend yang mendukung SSO melalui token akses.
5. **Fleksibilitas**: Dapat diintegrasikan dengan berbagai penyedia identitas, memungkinkan pengguna login dengan akun yang sudah ada.
6. **Pengelolaan Token**: Pastikan token akses disimpan dengan aman untuk mencegah penyalahgunaan.
7. **Desain Responsif**: Antarmuka login harus mudah diakses dan jelas membedakan antara login biasa dan SSO.

Dengan memahami aspek-aspek ini, pengguna dan pengembang dapat memanfaatkan SSO untuk menciptakan sistem yang aman dan user-friendly.

Implementasi Back-end

Untuk implementasi back-end, beberapa langkah perlu dilakukan:

Endpoint Otentikasi

Buat endpoint untuk menangani permintaan login, misalnya '/api/auth/login'. Endpoint ini harus memverifikasi kredensial pengguna dan mengembalikan token JWT jika valid.

Verifikasi Token

Implementasikan middleware untuk memverifikasi token JWT pada setiap permintaan yang memerlukan otentikasi. Middleware ini harus memeriksa keberadaan dan validitas token di header Authorization.

Integrasi SSO

Buat endpoint untuk menangani callback SSO. Endpoint ini harus memverifikasi token yang diterima dari server SSO dan membuat sesi untuk pengguna jika valid.

Fungsi Utama dalam Komponen AuthLogin

1

Public Key

Fungsi ini bertanggung jawab untuk memuat public key dari file yang disimpan dalam folder `/config/keys/`. Public key ini digunakan untuk memverifikasi token SSO yang diterima dari frontend.

2

getUserDataFromSSO

Fungsi ini menangani permintaan untuk mendapatkan data pengguna dari SSO. Fungsi ini menerima `ssoToken` dari request body. Jika token tidak ada, fungsi akan mengembalikan respons error dengan status 400. Jika token ada, fungsi akan memverifikasi token menggunakan public key. Jika verifikasi berhasil, pengguna dicari di database `compro` berdasarkan username yang terdapat dalam token.

3

Penanganan Error

Fungsi ini juga menangani potensi kesalahan yang dapat terjadi selama proses verifikasi token atau pencarian pengguna di database. Jika token tidak valid atau jika pengguna tidak ditemukan, fungsi akan mengembalikan respons yang sesuai dengan status HTTP yang relevan (401 untuk token tidak valid dan 404 untuk pengguna tidak ditemukan).

Menambahkan Public Key

Fungsi ini bertanggung jawab untuk memuat public key dari file yang disimpan dalam folder `/config/keys/`. Public key ini digunakan untuk memverifikasi token SSO yang diterima dari frontend.

```
const publicKey = fs.readFileSync(path.join(__dirname, '../config/keys/public.key'), 'utf8');
```

Fungsi getUserDataFromSSO

Fungsi ini menangani permintaan untuk mendapatkan data pengguna dari SSO. Fungsi ini menerima `ssoToken` dari request body. Jika token tidak ada, fungsi akan mengembalikan respons error dengan status 400. Jika token ada, fungsi akan memverifikasi token menggunakan public key. Jika verifikasi berhasil, pengguna dicari di database `compro` berdasarkan username yang terdapat dalam token.

```
exports.getUserDataFromSSO = async (req, res) => {
    const { ssoToken } = req.body; // Ambil ssoToken dari request body

    if (!ssoToken) {
        return res.status(400).json({ error: 'SSO token is required' });
    }

    try {
        // Verifikasi token SSO
        const decoded = jwt.verify(ssoToken, publicKey, { algorithms: ['RS256'] });

        // Cek apakah username di SSO cocok dengan yang ada di database compro
        const ssoUser = await User.findOne({ where: { username: decoded.username } });

        if (ssoUser) {
            const { id, username } = ssoUser; // Ambil hanya field yang diperlukan
            return res.json({ message: 'User found', user: { id, username } });
        } else {
            return res.status(404).json({ error: 'User not found in compro database' });
        }
    } catch (err) {
        return res.status(401).json({ error: 'Invalid SSO token' });
    }
};
```

Penanganan Error

Fungsi ini juga menangani potensi kesalahan yang dapat terjadi selama proses verifikasi token atau pencarian pengguna di database. Jika token tidak valid atau jika pengguna tidak ditemukan, fungsi akan mengembalikan respons yang sesuai dengan status HTTP yang relevan (401 untuk token tidak valid dan 404 untuk pengguna tidak ditemukan).

Kesimpulan dan Langkah Selanjutnya

Implementasi login SSO melibatkan integrasi yang erat antara front-end dan back-end. Berikut adalah ringkasan langkah-langkah kunci dan saran untuk pengembangan lebih lanjut:

1

Uji Keamanan

Lakukan pengujian keamanan menyeluruh untuk memastikan integritas proses SSO dan perlindungan data pengguna.

2

Dokumentasi

Buat dokumentasi yang komprehensif untuk pengguna dan pengembang, termasuk panduan integrasi dan troubleshooting.

3

Pemantauan

Implementasikan sistem pemantauan untuk melacak kinerja dan keandalan proses SSO, serta mendeteksi potensi masalah keamanan.

4

Peningkatan Berkelanjutan

Terus perbarui dan tingkatkan implementasi SSO sesuai dengan standar keamanan terbaru dan kebutuhan pengguna.