React and React Native Basics

Single Page Application with React Router

9

PERHATIAN

Jangan lupa untuk isi form absensi. Bagi instruktur mohon mengisi form absensi yang telah diinfokan dan konfirmasi nomor urut peserta yang hadir.

Untuk student harap mengisi form absensi di sini sebelum kelas dimulai. Untuk kode peserta dapat ditanyakan kepada instruktur dan jangan lupa mencantumkan pertemuan ke 1

Single Page Application

Single Page Application

Intro

Dikutip dari MDN (https://developer.mozilla.org/en-US/docs/Glossary/SPA), definisi Single Page Application, yang lazim disebut SPA adalah :

SPA (Aplikasi satu halaman) adalah implementasi aplikasi web yang memuat hanya satu dokumen web, dan kemudian memperbarui konten isi dokumen tunggal tersebut melalui API JavaScript seperti XMLHttpRequest dan Fetch ketika konten yang berbeda akan ditampilkan.

Oleh karena itu, ini memungkinkan pengguna untuk menggunakan situs web tanpa memuat halaman baru dari server, yang dapat menghasilkan peningkatan kinerja dan pengalaman yang lebih dinamis, dengan beberapa kerugian tradeoff seperti SEO, lebih banyak upaya yang diperlukan untuk mempertahankan status, menerapkan navigasi, dan melakukan pemantauan kinerja yang berarti.

Single Page Application

Lifecycle : Traditional vs SPA, taken from https://morioh.com/p/60ea607c78e3

Secara teori, single page application merupakan aplikasi yang bekerja di dalam browser yang tidak membutuhkan reload page saat digunakan. Dengan kata lain, pengguna atau user tidak akan berpindah halaman dengan melakukan request kepada server setiap kali terjadi interaksi pada aplikasi. Yang membedakan SPA dengan non-SPA adalah single page application hanya akan melakukan load terhadap satu halaman dari server kemudian mekanisme routing yang biasanya di-handle oleh server kini dibebankan pada client. Akibatnya, website yang menggunakan SPA memiliki performa yang lebih cepat tanpa harus load halaman secara terus menerus.

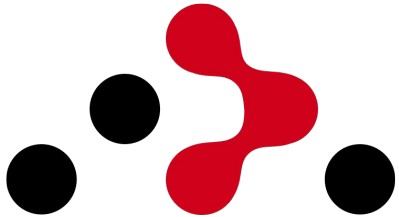
Sumber : https://socs.binus.ac.id/2018/12/06/teknologi-single-page-application-spa/

React Router Basic

React Router Basic

Intro + Instalasi

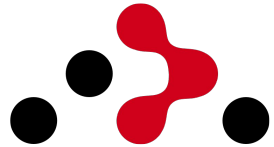
React Router adalah koleksi dari komponen yang mampu melakukan navigasi aplikasi secara deklaratif. Seperti pada aplikasi web pada umumnya, kita akan membutuhkan fasilitas router untuk melakukan routing. Routing ini bisa berupa tombol atau link, sesuai bagaimana temen-temen menerapkan stylingnya. Pada sesi ini, kita akan implementasikan fungsionalitas nya.



Instalasi
Bagian ini terbilang sangat mudah. Silakan buka terminal kamu dan eksekusi perintah berikut :

```
> npm install react-router-dom@6
```

React Router Basic



PRIMARY COMPONENTS

Ada 3 component utama dalam meng-implementasikan React Routers :

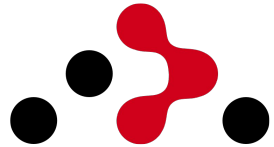
1. Router, yaitu <BrowserRouter>
2. Route, yaitu <Route> yang akan menjadi nested component dari <Routes>
3. Navigation, di antaranya adalah <Link>, <NavLink>, dan <Redirect>

Semua component-component itu dapat temen-temen pakai pada aplikasi berbasis web dengan melakukan import dari “react-router-dom”. Konsep yang akan dipakai di sini adalah dengan melihat Navigation component sebagai “perubah router”

BrowserRouter harus menjadi element paling atas untuk membungkus App **ATAU** content dari App

Mari kita ikuti step-by-step nya pada slide-slide selanjutnya

React Router Basic



IMPLEMENTASI

Kita awali dengan membuat sebuah React app baru dengan perintah create-react-app. Setelah itu, kita lanjutkan dengan instalasi react-router-dom. Nah, setelah itu, mari kita ikuti langkah-langkah berikut:

```
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import './index.css';
4  import App from './App';
5  import reportWebVitals from './reportWebVitals';
6  import { BrowserRouter } from 'react-router-dom';
7
8  ReactDOM.render(
9    <React.StrictMode>
10     <BrowserRouter>
11       <App />
12     </BrowserRouter>
13   </React.StrictMode>,
14   document.getElementById('root')
15 );
16
17 // If you want to start measuring performance in your app, pass a function
18 // to log results (for example: reportWebVitals(console.log))
19 // or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
20 reportWebVitals();
```

1. Implementasi Router

Router yang kita pakai kali ini adalah BrowserRouter. Router ini akan kita simpan sebagai pembungkus komponen App kita di file **index.js**, agar semua yang di dalam nya akan menjadi children dari Router dan dapat kita implementasikan fitur-fitur react router

React Router Basic

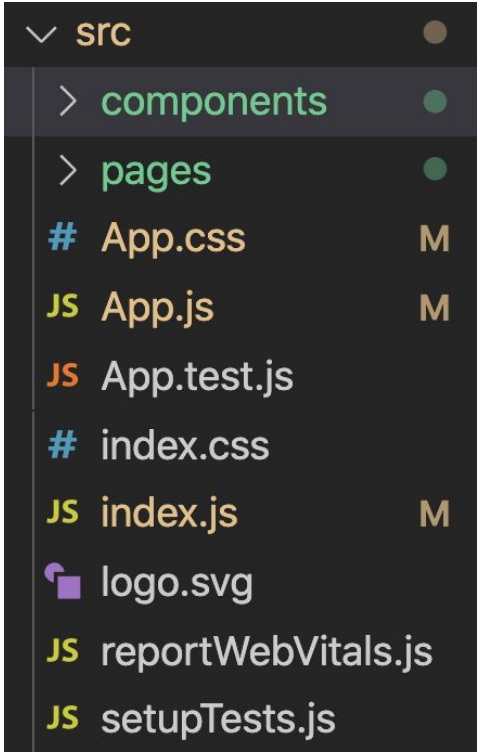
IMPLEMENTASI

2. Membuat folder Components dan Pages

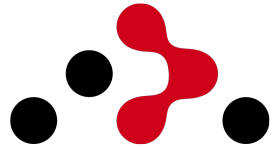
Buatlah 2 folder tersebut di dalam folder src/ sehingga struktur nya menjadi seperti gambar di samping.

Components akan berisi file-file yang berisi component yang umum dipakai

Pages akan berisi file-file yang akan bertindak sebagai layaknya sebuah “halaman”



React Router Basic



IMPLEMENTASI

3. Membuat 2 file dalam folder Pages

Buatlah 2 buah file : Home.js dan About.js di dalam folder src/Pages/

Isi dari 2 file ini bebas, yang penting bisa memberikan konten yang jelas untuk masing-masing halaman. Gambar di samping adalah contoh yang sederhana

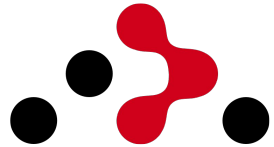
src > pages > JS Home.js > ...

```
1  export default function Home() {
2    return (
3      <div>
4        <h1>Home</h1>
5        <p>This is my Home page</p>
6      </div>
7    )
8  }
```

src > pages > JS About.js > ...

```
1  export default function About() {
2    return (
3      <div>
4        <h1>About Us</h1>
5        <p>This is my About Us page</p>
6      </div>
7    )
8  }
```

React Router Basic



IMPLEMENTASI

4. Implementasi Link

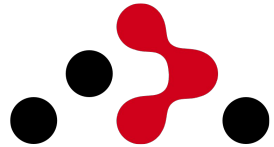
Buatlah 1 buah file : Navbar.js di dalam folder src/Components/

File ini akan berisi implementasi Link yang dibutuhkan untuk merubah route (route changer)

src > components > JS Navbar.js > ...

```
1  import { Link } from "react-router-dom";
2
3  export default function Navbar() {
4    return (
5      <nav>
6        <Link to="/">Home</Link> | <Link to="/about">About Us</Link>
7      </nav>
8    )
9  }
```

React Router Basic



IMPLEMENTASI

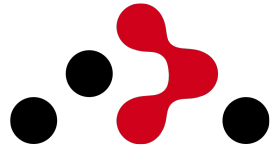
5. Update file App.css

Update file App.css untuk sedikit memberikan style agar tampilan pages nya nanti akan cukup baik walaupun belum kita maksimalkan secara estetis

src > # App.css > ...

```css
.container {
  width: 800px;
  margin: 0 auto;
}

nav {
  padding: 1rem 0;
  border-bottom: 1px solid black;
}
```

React Router Basic



IMPLEMENTASI

src > JS App.js > ...

```
1  import "./App.css";
2  import { Routes, Route } from "react-router-dom";
3  import Navbar from "./components/Navbar";
4  import Home from "./pages/Home";
5  import About from "./pages/About";
6
7  function App() {
8    return (
9      <div className="container">
10       <Navbar />
11       <Routes>
12         <Route path="/" element={<Home />} />
13         <Route path="/about" element={<About />} />
14       </Routes>
15     </div>
16   );
17 }
18
19 export default App;
```

6. Implementasi Routes dan Route

Update file src/App.js untuk implementasi :

- Routes dan Route
- Semua components dan pages yang sudah kita buat
- CSS

Di sini kita melakukan praktek pengiriman props ke dalam element Route via atribut element

React Router Basic

IMPLEMENTASI

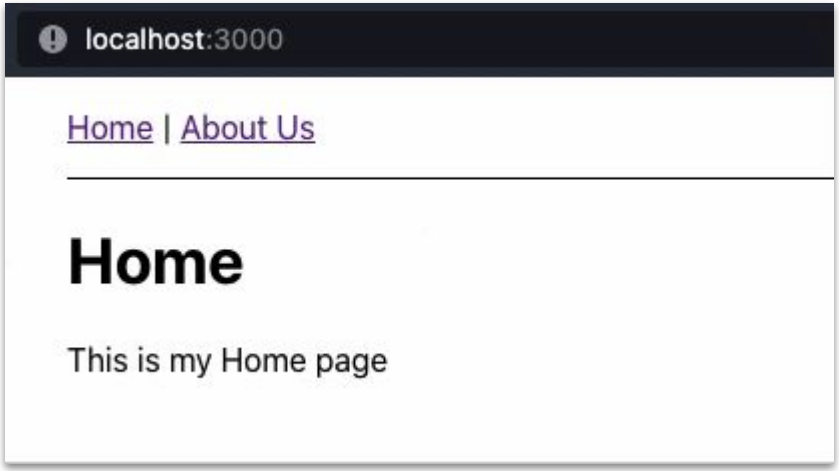
7. Jalankan dan buka di browser

Jalankan aplikasi dengan perintah

```
> npm start
```

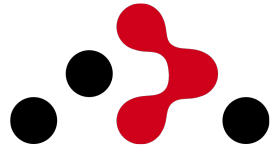
…dan akses di browser di URL http://localhost:3000

Expected result adalah seperti di samping. Silakan lihat Lampiran 1 untuk lebih jelasnya



Nested Route

Nested Route



INTRO + IMPLEMENTASI

Bagaimana jika kita mempunyai sebuah halaman, yang memiliki sub halaman ? Di sini kita bisa handle situasi seperti ini dengan menggunakan tekhnik **nesting**. Mari kita implementasikan tekhnik ini melalui sebuah contoh kasus dimana di dalam halaman About Us, kita mau tambahkan 2 buah sub halaman yaitu : About The Company dan About Me

1. Membuat 2 sub halaman di folder Pages (src/Pages/)

…dengan nama AboutCompany.js dan AboutMe.js, dengan content seperti di bawah ini :

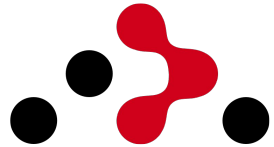
src > pages > JS AboutCompany.js > ...

```
export default function AboutCompany() {
  return (
    <div>
      <h2>About Company</h2>
      <p>
        This is an About Company page,
        which is a sub page of About Us page
      </p>
    </div>
  )
}
```

src > pages > JS AboutMe.js > ...

```
export default function AboutMe() {
  return (
    <div>
      <h2>About Me</h2>
      <p>
        This is an About Me page,
        which is a sub page of About Us page
      </p>
    </div>
  )
}
```

Nested Route



IMPLEMENTASI

2. Update file src/Pages/About.js

src > pages > JS About.js > ...

```
1   import { Link, Outlet } from "react-router-dom";
2
3   export default function About() {
4     return (
5       <div>
6         <h1>About Us</h1>
7         <p>This is my About Us page</p>
8         <Link to="about-company">About The Company</Link> |{" "}
9         <Link to="about-me">About Me</Link>
10        <Outlet />
11      </div>
12    )
13  }
```

3. Update file App.js

```
6   import AboutCompany from "./pages/AboutCompany";
7   import AboutMe from "./pages/AboutMe";
```

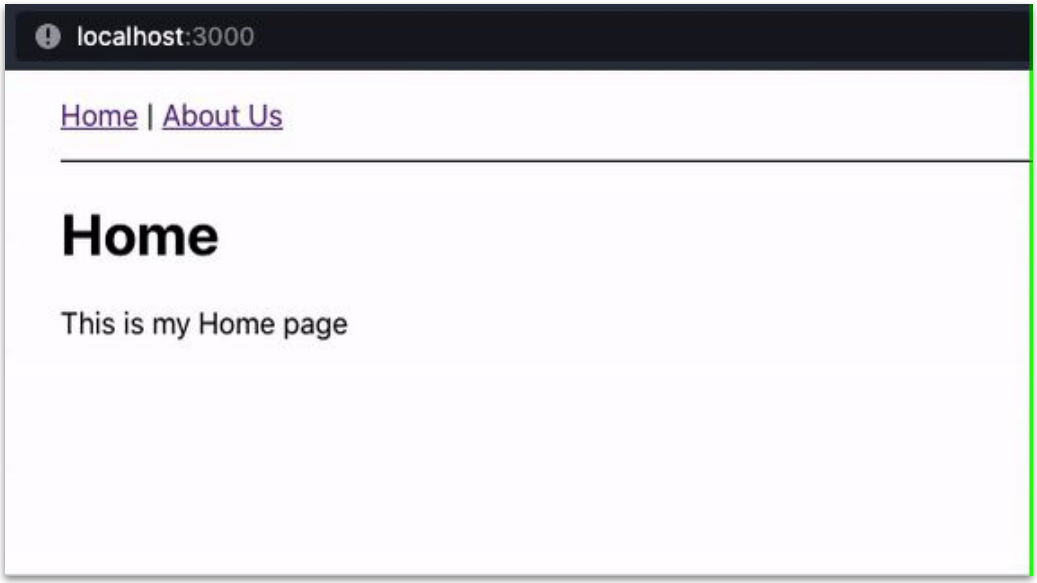
```
13          <Routes>
14            <Route path="/" element={<Home />} />
15            <Route path="/about" element={<About />}>
16              <Route path="about-company" element={<AboutCompany />} />
17              <Route path="about-me" element={<AboutMe />} />
18            </Route>
19          </Routes>
```

Nested Route

IMPLEMENTASI

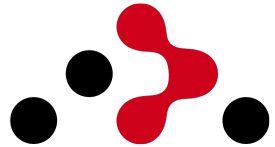
4. Re-run the app dan buka di browser

Untuk lebih jelasnya, silakan lihat di Lampiran 2



URL Parameters

URL Parameters



INTRO + IMPLEMENTASI

Seringkali kita membutuhkan alamat URL yang dinamis dengan menggunakan parameter. Kita bisa menggunakan format **:<nama params>** untuk melakukan hal ini dan membaca value dari **<nama params>** tersebut. Misalkan untuk halaman profil member dari halaman members

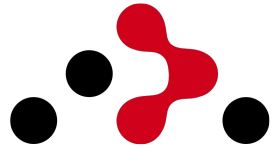
1. Membuat halaman Members di folder Pages (src/Pages/)

…dengan nama Members.js dan isi seperti berikut ini :

src > pages > JS Members.js > ...

```
1   import { Link, Outlet } from "react-router-dom";
2
3   export default function Members() {
4     return (
5       <div>
6         <h1>Members page</h1>
7         <p>Select a member to be shown</p>
8         <Link to="John Doe">John Doe</Link> |{" "}
9         <Link to="Doe Jane">Doe Jane</Link>
10        <Outlet />
11      </div>
12    )
13  }
```

URL Parameters



IMPLEMENTASI

Pada halaman detail member ini, kita menggunakan hooks yang bernama **useParams** untuk bisa membaca params yang akan kita setting pada Routes nantinya. Berikut adalah contoh implementasinya

2. Membuat halaman Member di folder Pages (src/Pages/)

…dengan nama Member.js dan isi seperti berikut ini :

src > pages > JS Member.js > ...

```
1   import { useParams } from "react-router-dom";
2
3   export default function Member() {
4     let params = useParams()
5
6     return (
7       <div>
8         <h3>{params.memberName}</h3>
9         <p>This is the detail page of {params.memberName}</p>
10      </div>
11    )
12  }
```

URL Parameters

IMPLEMENTASI

Pada App.js, kita menambahkan Route-Route baru seperti yang terlihat pada contoh. Perhatikan bahwa kita memberikan hanya **:memberName** pada path yang akan mengakses page Member

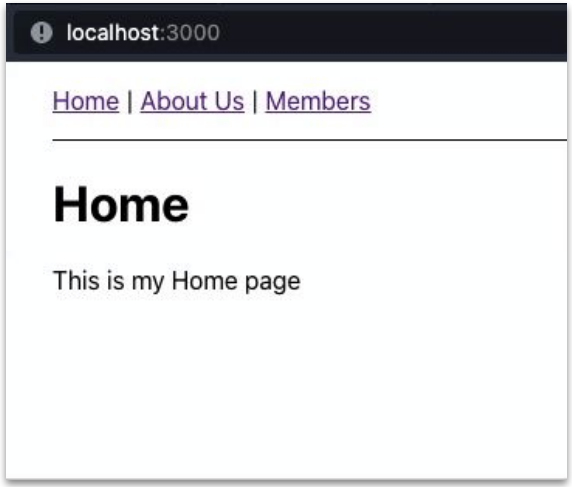
3. Update Routes di App.js

…dengan menambahkan beberapa Route baru :

```
21          <Route path="/members" element={<Members />}>
22            <Route path=":memberName" element={<Member />} />
23          </Route>
```

4. Buka di browser

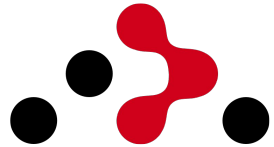
…dan hasilnya akan seperti pada contoh di samping. Silakan lihat Lampiran 3 untuk hasil jelasnya



Programmatic Navigation

Single Page Application with React Router - 9

Programmatic Navigation



INTRO + IMPLEMENTASI

src > JS items.js > ...

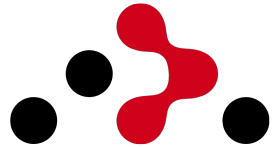
```
let items = [
  {
    id: 1,
    name: 'God of Blade',
    power: '1000'
  },
  {
    id: 2,
    name: 'Supreme Dagger',
    power: '1200'
  },
  {
    id: 3,
    name: 'Hollow Axes',
    power: '1300'
  },
]

export function getItems() {
  return items;
}
```

Bagaimana jika kita akan mengolah data yang dinamis ? Sebagai contoh, data hasil pemanggilan dari API ? Maka kita bisa memanfaatkan programmatic navigation untuk melakukan navigasi secara dinamis. Pada contoh kasus kali ini, kita akan menggunakan sebuah file yang berisi data array of object sebagai simulasi dari hasil return dari sebuah API call. Kemudian, kita akan menampilkan detail masing-masing item ke dalam 1 buah komponen saja secara programmatic

1. Membuat file data

…dengan nama items.js dan isi seperti di samping ini

Programmatic Navigation



src > pages > JS Items.js > ...

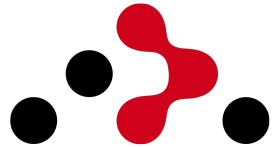
```
1   import { Outlet } from "react-router-dom";
2   import { getItems } from "../items";
3   import ItemRow from "../components/ItemRow";
4
5   export default function Items() {
6     const items = getItems()
7     return (
8       <div>
9         <h1>Items page</h1>
10        <p>Select an item to be shown</p>
11        <table>
12          <tr>
13            <th>Item</th>
14            <th>Action</th>
15          </tr>
16          {
17            items.map(item => (
18              <ItemRow key={item.id} item={item} />
19            ))
20          }
21        </table>
22
23        <Outlet />
24      </div>
25    )
26  }
```

IMPLEMENTASI

2. Membuat halaman Items di folder Pages

…dengan nama Items.js dan isi seperti di samping ini

Programmatic Navigation



IMPLEMENTASI

3. Membuat component ItemRow di folder Components

…dengan nama ItemRow.js dan isi seperti di bawah ini

Di sini kita menggunakan sebuah hooks yang bernama **useNavigate** untuk melakukan navigasi secara programmatic. Pada JSX nya kita lakukan seperti pada baris 10

src > components > JS ItemRow.js > ...

```
1   import { useNavigate } from "react-router";
2
3   export default function ItemRow({ item }) {
4     const navigate = useNavigate();
5
6     return (
7       <tr>
8         <td>{item.name}</td>
9         <td>
10          <button onClick={() => navigate(`${item.id}`)}>Show detail</button>
11        </td>
12      </tr>
13    );
14  }
```

Programmatic Navigation

IMPLEMENTASI

4. Membuat halaman Item di folder Pages

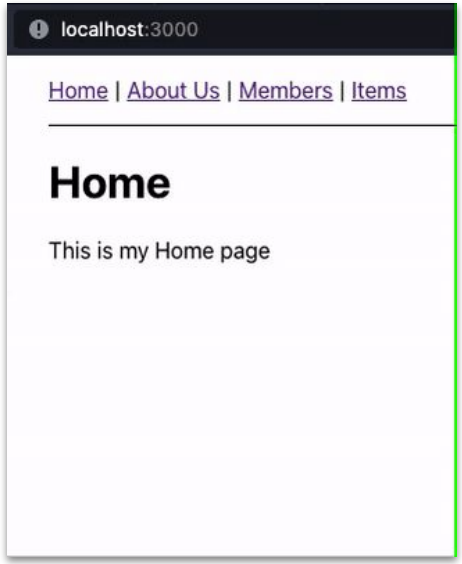
…dengan nama Item.js dan isi seperti di bawah ini

src > pages > JS Item.js > ...

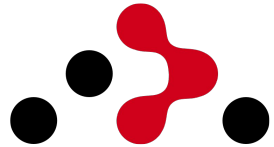
```
1   import { useParams } from "react-router-dom";
2   import { getItems } from "../items";
3
4   export default function Item() {
5     let params = useParams()
6     let items = getItems()
7     let item = items.find(
8       item => item.id === Number(params.itemId)
9     )
10
11    return (
12      <div>
13        <h3>{item.name}</h3>
14        <p>Power : {item.power}</p>
15      </div>
16    )
17  }
```

5. Buka di browser

…dan hasilnya akan seperti pada contoh di bawah. Silakan lihat Lampiran 4 untuk hasil jelasnya



Programmatic Navigation

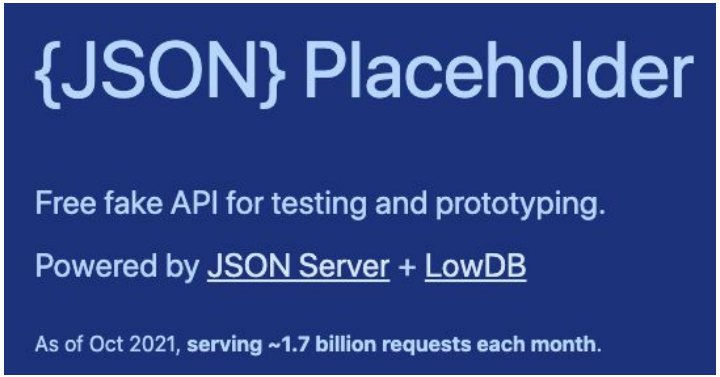


CHALLENGE

Bagaimana kalau kita step up ke tingkat yang lebih tinggi lagi ?

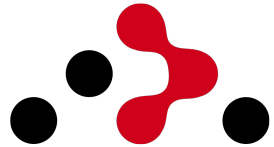
Caranya cukup “mudah”. Silakan praktekan Programmatic Navigation ini bersama dengan implementasi data fetching dari API yang sudah tersedia. Sebagai contoh, silakan ambil daftar Users dari https://jsonplaceholder.typicode.com/users kemudian siapkan tampilan untuk masing-masing User dengan Programmatic Navigation. Contoh salah satu API endpoint nya adalah https://jsonplaceholder.typicode.com/users/1 , dimana URL tersebut memanfaatkan ID dari setiap User untuk mengakses data detailnya.

Challenge ini hanya sebatas penambah jam terbang saja, tidak untuk dikumpulkan



Protected Route

Protected Route



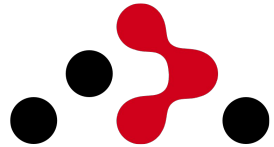
INTRO

Sudah menjadi kebutuhan standard dalam pengembangan aplikasi, bahwa akan ada fitur atau halaman yang hanya bisa diakses oleh user tertentu. Dan penentu akses ini adalah melalui login. Kita akan implementasi flow login sederhana, dimana fitur atau halaman Items hanya bisa dilihat oleh user yang sudah login.

Flow yang akan kita implementasi untuk pembelajaran kali ini cukup sederhana, yaitu jika kita akses halaman yang hendak kita proteksi dengan authentication, maka kita akan diarahkan ke halaman Login terlebih dahulu, JIKA kita belum login (authenticated). Dan setelah melakukan login, maka kita akan diarahkan ke halaman yang hendak kita akses sebelum login. Dalam studi kasus kali ini, kita akan proteksi halaman Items.

Mekanisme login yang akan kita gunakan hanyalah memanfaatkan localStorage untuk membuat item bernama **token**. Token ini akan menjadi penanda kita apakah kita sudah login atau belum

Protected Route



IMPLEMENTASI

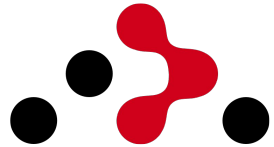
1. Membuat component ProtectedRoute di dalam folder src/Components

…dengan nama ProtectedRoute.js dan isi seperti di bawah ini. Kita menggunakan fitur **Navigate** untuk melakukan navigasi ke halaman Login. Kita menggunkana hooks yang bernama **useLocation** untuk bisa mendapatkan lokasi terakhir sebelum diarahkan ke Login, agar setelah Login, kita akan diarahkan ke halaman yang kita tuju sebelum Login. Lokasi terakhir dikirimkan dalam bentuk props ke dalam atribut state.

src > components > JS ProtectedRoute.js > ...

```
1   import { useLocation, Navigate } from "react-router-dom";
2
3   export default function ProtectedRoute({ children }) {
4     let location = useLocation();
5
6     if (!localStorage.getItem("token")) {
7       return <Navigate to="/login" state={{ from: location }} />;
8     }
9
10    return children;
11  }
```

Protected Route



IMPLEMENTASI

2. Membuat halaman Login di folder src/Pages

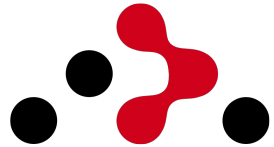
…dengan nama Login.js dan isi seperti di bawah ini. Di sini kita akan implementasi simulasi mekanisme login sederhana dengan langsung membuat item bernama **token** pada localStorage. Kemudian, kita akan redirect user ke halaman yang hendak di akses sebelum login dengan memanfaatkan nilai yang sudah di props dari ProtectedRoute

Untuk studi kasus ini, halaman Login hanya akan diakses dari halaman yang Protected. Tidak ada Link khusus untuk akses halaman Login

src > pages > JS Login.js > ...

```
1   import { useLocation, useNavigate } from "react-router-dom";
2
3   export default function Login() {
4     let navigate = useNavigate()
5     let location = useLocation()
6     let from = location.state?.from?.pathname || "/"
7
8     const handleLogin = () => {
9       localStorage.setItem("token", "login")
10      navigate(from, { replace: true })
11    }
12
13    return (
14      <div>
15        <h1>Login Page</h1>
16        <p>Klik the button to login</p>
17        <button
18          className="login-button"
19          onClick={() => handleLogin()}>
20            Login
21        </button>
22      </div>
23    )
24  }
```

Protected Route



IMPLEMENTASI

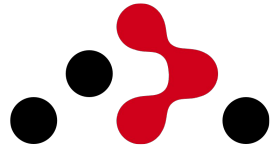
3. Update App.js

Import file-file yang baru dibuat, dan melakukan proteksi terhadap route Items, juga menambahkan route baru untuk Login

```
12  import ProtectedRoute from "./components/ProtectedRoute";
13  import Login from "./pages/Login";
```

```
25          <Route path="/members" element={<Members />}>
26            <Route path=":memberName" element={<Member />} />
27          </Route>
28          <Route
29            path="/items"
30            element={
31              <ProtectedRoute>
32                <Items />
33              </ProtectedRoute>
34            }>
35            <Route path=":itemId" element={<Item />} />
36          </Route>
37          <Route path="/login" element={<Login />} />
```

Protected Route



IMPLEMENTASI

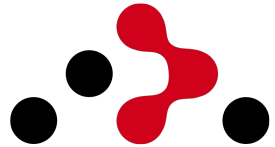
4. Update Navbar.js

Terutama untuk proses Logout. Tombol logout hanya muncul kalau sudah login. Jika user melakukan logout, maka pada akhirnya akan diarahkan ke halaman Home

src > components > JS Navbar.js > ...

```
1  import { Link, useNavigate } from "react-router-dom";
2
3  export default function Navbar() {
4    let navigate = useNavigate()
5
6    const handleLogout = () => {
7      localStorage.removeItem("token")
8      navigate("/")
9    }
10
11   return (
12     <nav>
13       <Link to="/">Home</Link> |{" "}
14       <Link to="/about">About Us</Link> |{" "}
15       <Link to="/members">Members</Link> |{" "}
16       <Link to="/items">Items</Link> |{" "}
17       { localStorage.getItem("token")
18         &&
19         <button
20           className="logout-button"
21           onClick={() => handleLogout()}>Logout</button>
22       }
23     </nav>
24   )
25 }
```

Protected Route



IMPLEMENTASI

5. Update App.css

Sedikit memberikan style pada tombol-tombol Login dan Logout.

Untuk kali ini, silakan berkreasi sesuai dengan selera

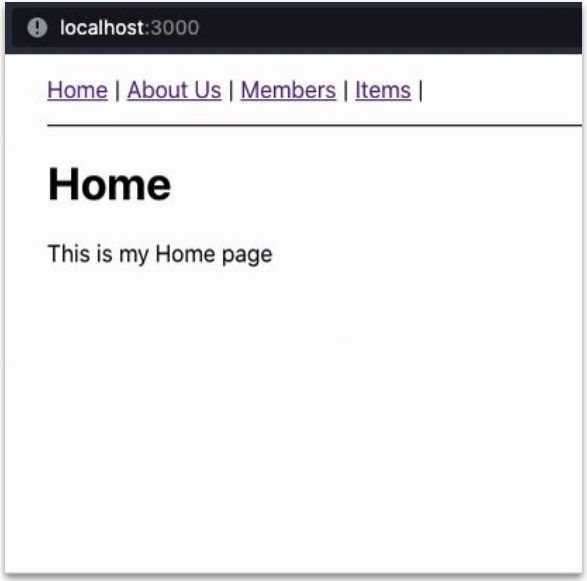
```
.login-button {
  background-color: #4CAF50;
  border: none;
  border-radius: 5px;
  color: white;
  padding: 10px 16px;
  text-align: center;
  text-decoration: none;
  display: inline-block;
  cursor: pointer;
  font-size: 14px;
}

.logout-button {
  background-color: #AF4C50;
  border: none;
  border-radius: 5px;
  color: white;
  padding: 8px 16px;
  text-align: center;
  text-decoration: none;
  display: inline-block;
  cursor: pointer;
}
```

Protected Route

IMPLEMENTASI

6. Buka di browser

…dengan hasil kurang lebih seperti ini. Silakan lihat Lampiran 5 untuk lebih jelasnya

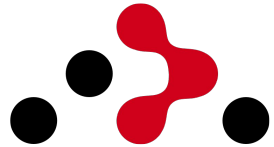


CHALLENGE

Sedikit challenge di akhir bab ini, silakan dicoba untuk menyiapkan Link khusus untuk Login. Kurang menantang ? Baik, coba dipikirkan bagaimana yang seharusnya terjadi jika kita sudah login sebelumnya, lalu masuk ke halaman Login ? **Dan seperti biasa, bukan task untuk dikumpulkan**

Redirect

Redirect



INTRO

Tidak seperti versi sebelumnya, di versi ini tidak ada function atau component khusus dengan nama “Redirect” untuk melakukan redirection. Di versi ini, kita lebih diarahkan kepada konsep “navigasi”, dimana jika kita perhatikan secara seksama, Redirect pun sebenarnya adalah navigasi yang biasanya dilakukan karena / setelah kondisi tertentu. Misalkan kita akan Redirect user setelah login. Ini berarti kita akan melakukan navigasi ke suatu route setelah login selesai.

Kita akan berkenalan dengan hook **useNavigate** untuk melakukan navigasi secara programmatic, dan component **Navigate** untuk melakukan navigasi dari JSX

Redirect

IMPLEMENTASI

Hook useNavigate

Kita perhatikan potongan kode pada bab sebelumnya. Di sini kita memanfaatkan hook useNavigate untuk melakukan navigasi secara programmatic. Dalam kasus ini, kita bisa sebut flow ini sebagai Redirect

src > pages > JS Login.js > ...

```
1   import { useLocation, useNavigate } from "react-router-dom";
2
3   export default function Login() {
4     let navigate = useNavigate()
5     let location = useLocation()
6     let from = location.state?.from?.pathname || "/"
7
8     const handleLogin = () => {
9       localStorage.setItem("token", "login")
10      navigate(from, { replace: true })
11    }
```

Redirect

IMPLEMENTASI

Component Navigate

Kita perhatikan potongan kode pada bab sebelumnya. Di sini kita memanfaatkan component Navigate untuk melakukan navigasi dari JSX. Dalam kasus ini, kita bisa sebut flow ini sebagai Redirect

src > components > JS ProtectedRoute.js > ...

```
1   import { useLocation, Navigate } from "react-router-dom";
2
3   export default function ProtectedRoute({ children }) {
4     let location = useLocation();
5
6     if (!localStorage.getItem("token")) {
7       return <Navigate to="/login" state={{ from: location }} />;
8     }
9
10    return children;
11  }
```

Thank You

PT Hacktivate Teknologi Indonesia
Gedung Aquarius Pondok Indah
Jalan Sultan Iskandar Muda No.7
Kebayoran Lama, Jakarta Selatan

www.hacktiv8.com