**BAB V**

**OTENTIKASI DAN OTORISASI DALAM PERMINTAAN HTTP**

1. **Authentication**
   1. **SessionStorage**

Penggunaan *session storage* digunakan hanya dalam kondisi halaman aktif, ketika halaman sudah tidak digunakan maka session akan hilang. Jika anda menyimpan data dengan teknik ini maka data tersebut hanya bisa digunakan sekali atau jika membuka window tab baru maka anda perlu mengenerate ulang session tersebut.

1. Create session

Berikut ini adalah contoh cara menyimpan data dengan session storage.

Format penggunaan sessionStorage:

sessionStorage.setItem('name\_key', 'value string');

Contoh implementasi pada microfrontend, pada function handlerSubmit() tambahkan script session storage untuk menyimpan nilai balik dari microservices seperti script dibawah ini:

A picture containing text, font, line, screenshot

Description automatically generated

Dari script diatas dapat dilihat bahwa memiliki 3 buah session yang disimpan dengan nama *access\_token, refresh\_token* dan *id\_token* dimana masing-masing key menyimpan nilai balik dari microservices. Untuk mengecek apakah session yang anda buat tersimpan pada browser anda dapat mengeceknya dengan membuka Developer Console pada browser masuk kedalam tab Application dan pada kolom sebelah kiri cari bagian Session Storage.

A screenshot of a computer

Description automatically generated

1. Retrive session

Berikut ini adalah contoh cara memanggil session yang telah disimpan:

Format untuk memanggil session storage:

sessionStorage.getItem(‘key\_name’)

1. Remove session

* Remove key

Format:

sessionStorage.removeItem('key\_name');

* Remove all

Format:

sessionStorage.clear();

* 1. **LocalStorage**

Local storage sama halnya dengan session storage dari segi kegunaannya, yang berbeda ialah jika menggunakan teknik ini session atau data yang anda simpan akan tetap aktif selama belum dihapus key local storage. Dengan kata lain jika anda membuka di banyak tab window browser maka session akan terikat dengan domain web apps.

1. Create session

Format:

localStorage.setItem('key\_name', 'value string');

Contoh kasus:

A picture containing text, font, line, screenshot

Description automatically generated

Dari script diatas dapat dilihat bahwa memiliki 3 buah session yang disimpan dengan nama *access\_token, refresh\_token* dan *id\_token* dimana masing-masing key menyimpan nilai balik dari microservices. Untuk mengecek apakah session yang anda buat tersimpan pada browser anda dapat mengeceknya dengan membuka Developer Console pada browser masuk kedalam tab Application dan pada kolom sebelah kiri cari bagian Local Storage.

A screenshot of a computer

Description automatically generated

1. Retrive session

Format:

localStorage.getItem('key\_name')

1. Remove session

* Remove key

Format:

localStorage.removeItem('key\_name');

* Remove all

Format:

localStorage.clear();

* 1. **Session dengan third party**
* **React Cookies**

Contoh kali ini akan menyimpan sebuah session atau data dengan memanfaatkan library js cookies, library JS-Cookies merupakan penyimpanan yang ringan sehingga dapat digunakan untuk menyimpan atau mengambil data sebagai cookie pada browser. Berikut adalah tahapan dari penggunaan JS-Cookies:

1. Installasi JS-Cookeis dengan node

Untuk menggunakan library ini silakan anda tambahakan pada project react js dengan membuka command line dan masukan syntax dibawah ini:

npm install js-cookie

1. Menambahkan library JS-Cookies

Setelah menambahkan dependencies pada project react, tambahkan code dibawah ini kedalam script Form.js

import Cookies from 'js-cookie';

1. Menggunakan JS-Cookies
2. SET Cookies

Untuk menyimpan data kedalam cookies, pada function ini membutuhkan tiga parameter utama yaitu name, value, dan option object.

Cookies.set('name', 'value', { … });

Key name diperuntukan untuk penamaan dari cookies, sedangkan key value berisi nilai yang akan disimpan, dan parameter terakhir harus bernilai objek. Ini dapat anda manfaatkan untuk menyimpan data informasi lain dengan bentuk objek. Contoh:

Cookies.set('access\_token', response.access\_token, { expired: 120 });

1. GET Cookies

Jika ingin memanggil data yang disimpan dengan Cookies dapat menggunakan:

Cookies.get('name');

1. REMOVE Cookies

Sedangkan jika ingin menghapus cookies dapat menggunakan function:

Cookies.remove('name');

* **React Session**

Library JS-Session sebenarnya banyak bentuknya, atau banyak ditawarkan didalam package manager NPM. Salah satunya ialah Redux React Session API.

redux-react-session sangat berguna untuk mempertahankan session adalah aplikasi react menggunakan redux store. API ini menyediakan metode seperti *initSessionService*, *refreshFromLocalStorage, checkAuth* dan beberapa fitur lanjutan seperti Immutable JS.

1. Installasi JS Redux React Session

Silakan anda tambahkan library redux-react-session pada project react anda dengan syntax dibawah ini:

npm i redux-react-session

1. Menggunakan JS Redux React Session
2. Add Session Reduces

|  |
| --- |
| import { combineReducers } from 'redux';  import { sessionReducer as session } from 'redux-react-session/immutable';    const reducers = {  session  };  const reducer = combineReducers(reducers); |

1. Inisialisasi service session

|  |
| --- |
| import { createStore } from 'redux';  import { sessionService } from 'redux-react-session';    const store = createStore(reducer)    sessionService.initSessionService(store); |

Untuk lebih lengkapnya silakan anda kunjungi <https://www.npmjs.com/package/redux-react-session>

* **React Firebase**

Library JS yang satu ini merupakan media penyimpanan data yang telah disiapkan oleh google.com cara penggunaan library ini anda perlu membuat atau memiliki account google developer agar dapat menggunakan fitur Firebase. Data yang disimpkan kedalam firebase ini biasanya digunakan sebagai data analitik atau berbagi data antar platform. Cara kerja dari firebase ini sama halnya dengan media lainnya yang telah dijelaskan, jika media penyimpanan lainnya terikat dengan browser klien untuk storing data, namun penyimpanan firebase ini bersifat asynchronous dan tidak terikat dengan browser klien. Tetapi seluruh data yang tersimpan akan berkomunikasi dengan *cloudflare* milik Google.

1. Membuat console developer Firebase

Silakan anda membuat account developer google dengan mengakses link <https://firebase.google.com/>. Selanjutnya anda dapat membuat project firebase sesuai dengan instruksi yang telah diberikan pada site tersebut.

A picture containing text, screenshot, electric blue, azure

Description automatically generated

***Gambar 4.3.1.*** *Create Project Firebase*

1. Menambahkan library JS-Firebase

Setelah anda membuat project Firebase, selanjutnya anda perlu menginstall library JS-Firebase pada project anda dengan memasukan syntax berikut:

npm install -g firebase-tools

Jika sudah berhasil menginstallasi library tersebut silakan anda cek project react anda dan akan memiliki file bernama *firebase.json*

1. Menggunakan JS-Firebase
2. Inisialisasi Firebase config

Buatlah file bernama Firebase.js dan panggillah library firebase dan juga masukan firebase config anda yang telah didaftarkan pada Register App di firebase google web app. Berikut adalah contohnya:

|  |
| --- |
| import { initializeApp } from "firebase/app"  import { getFirestore } from "firebase/firestore"  const firebaseConfig = {  apiKey: "API\_KEY",  authDomain: "AUTH\_DOMAIN",  projectId: "PROJECT\_ID",  storageBucket: "STORAGE\_BUCKET",  messagingSenderId: "MESSAGING\_SENDER\_ID",  appId: "APP\_ID"  }  // Initialize Firebase and Firestore  const app = initializeApp(firebaseConfig)  const db = getFirestore(app)  export {db} |

1. SET Firebase

Untuk menyimpan sebuah data dengan menggunakan firebase library yang digunakan ialah firestore. Dimana library ini sudah satu package dengan firebase. Function yang dapat digunakan ialah addDoc dan collection.

AddDoc digunakan untuk membuat fungsi pengiriman data secara asynchronous, sedangkan untuk menyimpan name, value atau objek lainnya kedalam cloudflare dapat menggunakan function bernama collection.

addDoc(collection(db, 'name'), 'value')

Contohnya:

|  |
| --- |
| import {db} from './firebase'  import {collection, addDoc, Timestamp} from 'firebase/firestore'  …  const handlerSubmit = (e) => {    e.preventDefault();    …  await addDoc(collection(db, 'id\_token'), decodeIDToken)    …  } |

1. GET Firebase

Firebase v9 menampilkan fungsi baru yang disebut *query*, yang digunakan untuk menentukan dokumen mana yang ingin diambil dari kumpulan atau grup kumpulan.

|  |
| --- |
| import {useState, useEffect} from 'react'  import {collection, query, orderBy, onSnapshot} from "firebase/firestore"  import {db} from './firebase'  …  const q = query(collection(db, 'id\_token')  onSnapshot(q, (querySnapshot) => {  setTasks(querySnapshot.docs.map(doc => ({  id: doc.id,  data: doc.data()  })))  })  … |

Dari kode di atas, dalam menggunakan fungsi kueri, untuk menentukan bahwa ingin mengambil semua dokumen bernama id\_token. Lalu meneruskan kueri ke fungsi onSnapshot, yang mengembalikan querySnapshot dimana telah dipetakan dan disimpan dalam state.

Untuk pemakaian dari firebase itu sendiri anda dapat mengeceknya lebih lengkap pada situs https://firebase.google.com/docs/projects/learn-more

1. **Handling HTTP Requests**

HTTP Request merupakan suatu mekanisme pengiriman data dari client pada protokol HTTP yang merepresentasikan aksi yang akan dilakukan pada suatu sumberdaya (server) yang sudah diidentifikasikan sebelumnya. Pihak server yang menerima request akan menjawabnya melalui mekanisme HTTP Response. HTTP Request pada dasarnya mendefenisikan 9 metode aksi yang akan dilakukan pada suatu sumber daya yaitu:

1. Permintaan GET: Permintaan GET digunakan untuk mengambil data dari server. Ini adalah jenis permintaan yang paling umum digunakan dan sering digunakan untuk meminta halaman web, gambar, file CSS/JavaScript, dan sebagainya.
2. Permintaan POST: Permintaan POST digunakan untuk mengirim data ke server, seperti data formulir yang diisi oleh pengguna atau data JSON dari aplikasi JavaScript. Ini digunakan ketika pengguna melakukan tindakan seperti mengirim formulir atau membuat perubahan dalam aplikasi.
3. Permintaan PUT: Permintaan PUT digunakan untuk memperbarui atau membuat data baru di server. Ini umumnya digunakan dalam aplikasi yang mengimplementasikan API RESTful untuk memperbarui entitas yang ada.
4. Permintaan DELETE: Permintaan DELETE digunakan untuk menghapus data dari server. Seperti PUT, ini juga umumnya digunakan dalam API RESTful untuk menghapus entitas yang ada.
5. Permintaan OPTIONS: Permintaan OPTIONS digunakan untuk mengambil informasi tentang kemampuan komunikasi server. Ini sering digunakan dalam Cross-Origin Resource Sharing (CORS) untuk menentukan apakah permintaan lintas domain akan diizinkan.
6. Permintaan HEAD: Permintaan HEAD serupa dengan GET, tetapi server hanya harus mengembalikan header respons tanpa konten aktual. Ini berguna untuk memeriksa status permintaan tanpa mengunduh seluruh konten.
7. Permintaan PATCH: Permintaan PATCH digunakan untuk mengirim perubahan parsial ke sumber daya di server. Ini memungkinkan penggunaan sumber daya yang efisien saat hanya perlu memperbarui bagian kecil dari sumber daya.
8. Permintaan TRACE: Permintaan TRACE digunakan untuk menguji jalur komunikasi antara client dan server. Ini mengembalikan pesan yang dikirim oleh server, dan sering digunakan untuk debugging atau pemecahan masalah.
9. Permintaan CONNECT: Permintaan CONNECT digunakan untuk memulai koneksi ke server proxy, biasanya untuk mendirikan koneksi SSL/TLS di atasnya.

Salah satu penggunaanya adalah mekanisme pengiriman data melalui suatu HTML FORM pada suatu situs web. Setiap jenis permintaan memiliki kegunaan dan kasus penggunaan yang berbeda tergantung pada kebutuhan aplikasi atau situs web yang digunakan. HTTP Request dapat diklasifikasikan berdasarkan cara mereka diproses atau kebutuhan sinkronisasi antara permintaan dan respons. Berikut adalah penjelasan singkat mengenai jenis permintaan HTTP yang umum:

1. **Synchronous (Sinkronus):**

Permintaan HTTP sinkronus adalah permintaan di mana kode yang membuat permintaan menunggu sampai permintaan selesai dan respons diterima sebelum melanjutkan eksekusi kode berikutnya. Dalam lingkungan web, ini bisa mengakibatkan pembekuan atau kegagalan dalam responsifitas aplikasi karena UI akan terblokir selama proses permintaan berlangsung. Permintaan sinkronus biasanya dilakukan dengan menggunakan metode seperti XMLHttpRequest (XHR) dalam JavaScript atau modul http di Node.js tanpa menggunakan fitur async/await.

1. **Asynchronous (Asinkronus):**

Permintaan HTTP asinkronus adalah permintaan di mana kode yang membuat permintaan tidak menunggu respons dari server. Sebaliknya, permintaan tersebut dikirim secara asinkron dan kode dapat melanjutkan eksekusi tanpa harus menunggu respons. Dalam JavaScript modern, permintaan asinkronus sering dilakukan menggunakan metode seperti fetch() di browser atau dengan menggunakan library seperti Axios di browser atau Node.js. Fitur async/await juga dapat digunakan untuk menangani permintaan asinkronus dengan cara yang lebih mudah dibaca dan dipahami.

1. **Real-time:**

Permintaan HTTP real-time adalah permintaan di mana klien dan server berkomunikasi secara terus-menerus, biasanya dalam waktu nyata, tanpa harus menunggu permintaan baru dari klien. Ini menciptakan pengalaman interaktif di mana data diperbarui secara otomatis saat berubah, tanpa perlu refresh halaman. Teknologi yang sering digunakan untuk mencapai real-time HTTP termasuk WebSockets, Server-Sent Events (SSE), dan HTTP Long Polling. Ini digunakan dalam aplikasi seperti aplikasi perpesanan, streaming data, dan aplikasi kolaboratif.

Sinkronus cocok untuk tugas-tugas sederhana yang tidak memerlukan respons cepat dari server. Asinkronus umumnya lebih disukai untuk permintaan HTTP dalam aplikasi web modern karena meningkatkan responsifitas dan kinerja aplikasi. Sementara itu, permintaan HTTP real-time cocok untuk kasus penggunaan yang memerlukan komunikasi terus-menerus antara klien dan server dalam waktu nyata.

Dalam JavaScript modern memiliki beberapa bentuk metode permintaan HTTP seperti:

* + - 1. FETCH

Fetch merupakan fungsi bawaan dalam JavaScript yang digunakan untuk melakukan permintaan HTTP asinkronus ke server dan mengambil sumber daya. Ini adalah cara modern dan lebih kuat untuk mengambil data dari server daripada XMLHttpRequest (XHR), yang merupakan metode yang lebih tua.

Contoh:

|  |
| --- |
| const raw = "";  const requestOptions = {    method: "GET",    body: raw,    redirect: "follow"  };  fetch("http://localhost:3003/api/user/fetch-all", requestOptions)    .then((response) => response.text())    .then((result) => console.log(result))    .catch((error) => console.error(error)); |

* + - 1. AXIOS

Axios ialah sebuah library JavaScript yang digunakan untuk membuat permintaan HTTP dari browser atau dari Node.js. Ini menyediakan antarmuka yang bersih dan mudah digunakan untuk berkomunikasi dengan server melalui protokol HTTP atau HTTPS. Berikut adalah fitur utama yang dimiliki Axios:

1. Sederhana dan Mudah Digunakan: Axios dirancang dengan antarmuka yang intuitif, yang membuatnya mudah dipelajari dan digunakan oleh pengembang, baik yang baru maupun yang berpengalaman.
2. Berbasis Promise: Axios menggunakan Promise, yang memungkinkan Anda untuk menangani permintaan asinkron dengan mudah menggunakan metode .then() atau dengan menggunakan async/await di JavaScript modern.
3. Mendukung Browser dan Node.js: Axios dapat digunakan baik di sisi klien (browser) maupun di sisi server (Node.js). Ini memungkinkan pengembang untuk menggunakan Axios di berbagai lingkungan pengembangan.
4. Penanganan Otomatis dari Respons: Axios secara otomatis menguraikan respons berformat JSON dan mentransformasikannya menjadi objek JavaScript. Ini membuat penanganan respons menjadi lebih sederhana.
5. Interseptor: Axios mendukung fitur interseptor yang memungkinkan Anda menambahkan logika ke permintaan dan respons secara global. Ini sangat berguna untuk menambahkan header otomatis, menangani kesalahan, dan hal-hal lainnya.
6. Dukungan untuk Pembatasan Waktu: Anda dapat mengatur waktu maksimum yang diizinkan untuk permintaan menggunakan konfigurasi Axios, sehingga Anda dapat menghindari permintaan yang terlalu lama atau yang berlarut-larut.

Untuk menggunakan Axios, perlu menginstalnya terlebih dahulu dalam proyek, terutama jika menggunakan Node.js. Pemasangan ini dapat dilakukan melalui npm (Node Package Manager) atau Yarn:

|  |
| --- |
| npm install axios |

Cara penggunaan AXIOS:

|  |
| --- |
| const axios = require('axios');  let data = '';  let config = {    method: 'get',    maxBodyLength: Infinity,    url: 'http://localhost:3003/api/user/fetch-all',    headers: { },    data : data  };  axios.request(config)  .then((response) => {    console.log(JSON.stringify(response.data));  })  .catch((error) => {    console.log(error);  }); |

Axios adalah library yang populer dan sangat berguna untuk mengelola permintaan HTTP dalam aplikasi web. Dengan fitur-fitur yang dimilikinya, Axios membuatnya mudah untuk berkomunikasi dengan server dan mengelola respons dengan efisien.

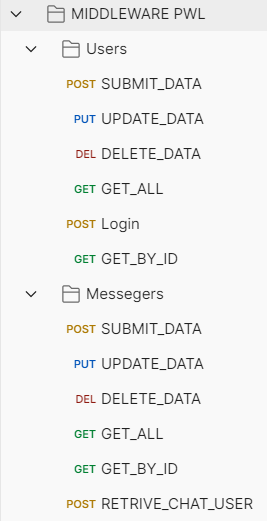
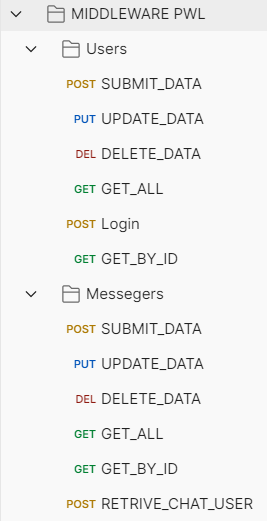
* + - 1. Request

Meskipun request mulai tidak sepopuler karena pengembangnya beralih ke alternatif seperti Axios, tetapi masih bisa digunakan. Ini adalah library HTTP client yang kuat untuk Node.js yang mendukung fitur seperti stream, form data, cookies, dan lain-lain.

|  |
| --- |
| var request = require('request');  var options = {    'method': 'GET',    'url': 'http://localhost:3003/api/user/fetch-all',    'headers': {    }  };  request(options, function (error, response) {    if (error) throw new Error(error);    console.log(response.body);  }); |

1. **Latihan: Membangun Aplikasi Messenger – Chapter 3**

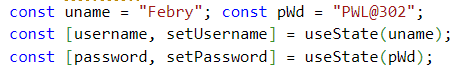
Pada module sebelumnya anda telah membuat middleware untuk backend dari aplikasi messenger, dimana memiliki bentuk HTTP Request sebagai berikut:



Disini kita akan membuat koneksi antar middleware dengan react framework menggunakan library AXIOS sebagai perantara jalur komunikasi.

1. Otentikasi Aplikasi

Pada halaman SIGN-IN, anda telah membuat bentuk layout dan data dummy pengguna, dimana sebelumnya berbentuk nilai konstanta seperti:



Sekarang kita perlu merubah otentikasi tersebut secara dinamik dengan cara mengkoneksikan kedalam middleware yang telah dibuat dengan cara menggunakan uri end-poin:

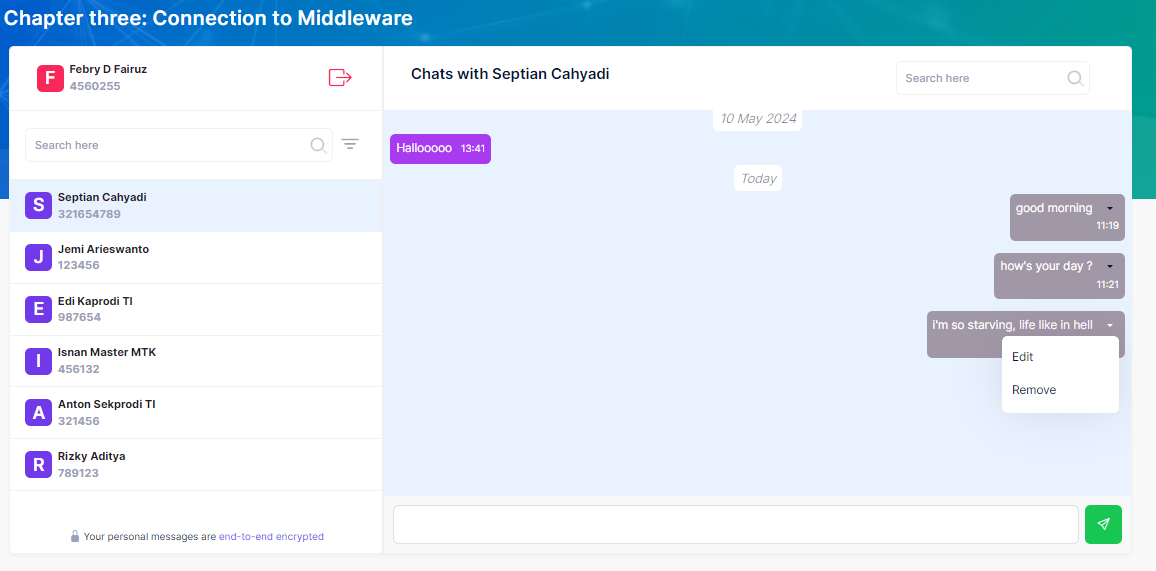
<http://localhost:3003/api/user/signin>

maka bentuk function HandlerSignIn perlu dilakukan perubahan seperti dibawah ini:

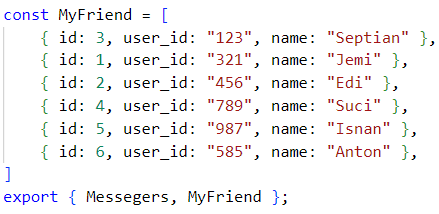
|  |
| --- |
| const middleware\_uri = process.env.REACT\_APP\_MIDDLEWARE\_URL;  const [signin, setSignIn] = useState({ loading: false, data: [], message: "" });  const HandlerSignIn = (e) => {    e.preventDefault();    setSignIn({ loading: true, data: [], message: "" });    if (username && password) {      POST\_SIGN\_IN({ username: username, password: password });     } else {         alert("Please fill up the field with correctly");     }  }  const POST\_SIGN\_IN = (param) => {    setSignin({ loading: true, data: [], message: "" });    let config = {      method: "post",      maxBodyLength: Infinity,      url: middleware\_uri + "/api/user/signin",      headers:{"Content-Type": "application/json"},      data: JSON.stringify(param),    };    axios      .request(config)      .then((response) => {        const result = response.data.data;        if (result) {          localStorage.setItem("user\_account", JSON.stringify(result));          setSignin({            loading: false,            data: response.data,            message: "Successfully sign in, please wait for a moment..",          });          setTimeout(() => {            window.location.replace("/");          }, 1000);        } else {          setSignin({            loading: false,            data: [],            message:              "Invalid Grant (Incorrect Credential. Please check username & password)",          });        }      })      .catch((error) => {        setSignin({ loading: false, data: [], message: error.message });      });  }; |

Code diatas memiliki sebuah fungsi bernama POST\_SIGN\_IN yang digunakan untuk mengirim permintaan HTTP POST untuk proses otentikasi pengguna (sign in). Berikut adalah penjelasan lebih rinci tentang code tersebut:

1. **const POST\_SIGN\_IN = (param, setSignin) => { ... }:** Ini adalah deklarasi fungsi yang disebut POST\_SIGN\_IN. Fungsi ini menerima dua parameter:
   1. **param**: Objek yang berisi data yang akan dikirimkan sebagai bagian dari permintaan HTTP POST. Biasanya berisi informasi kredensial pengguna, seperti nama pengguna dan kata sandi.
   2. **setSignin**: Fungsi yang digunakan untuk memperbarui status sign in dalam komponen React. Ini akan digunakan untuk mengatur status loading, data hasil permintaan, dan pesan kesalahan.
2. **setSignin({ loading: true, data: [], message: "" });**: Ini adalah pemanggilan fungsi setSignin yang mengatur status sign in menjadi sedang dalam proses, tanpa data, dan tanpa pesan.
3. **let config = { ... }**: Deklarasi objek config yang berisi konfigurasi untuk permintaan HTTP yang akan dikirim menggunakan Axios. Konfigurasi ini mencakup metode (method), URL, header, dan data yang akan dikirim dalam permintaan.
4. **axios.request(config)**: Menggunakan Axios untuk membuat permintaan HTTP sesuai dengan konfigurasi yang telah ditentukan sebelumnya dalam objek config.
5. **.then((response) => { ... })**: Menangani respons yang diterima dari server setelah permintaan HTTP selesai. Dalam blok .then, data respons diambil dari objek respons dan diperiksa.
6. **localStorage.setItem("user\_account", JSON.stringify(result));**: Jika proses otentikasi berhasil, data pengguna disimpan di dalam local storage browser dengan kunci "user\_account". Ini berguna untuk menyimpan informasi otentikasi dan mempertahankan sesi pengguna.
7. **setSignin({ ... })**: Fungsi setSignin dipanggil untuk memperbarui status sign in dalam komponen React. Terjadi dua kemungkinan pembaruan: jika otentikasi berhasil, status loading diatur menjadi false, data respons disetel, dan pengguna diarahkan kembali ke halaman utama setelah satu detik; jika otentikasi gagal, pesan kesalahan ditampilkan kepada pengguna.
8. **.catch((error) => { ... })**: Menangani kesalahan yang terjadi selama proses permintaan HTTP. Jika terjadi kesalahan, pesan kesalahan ditampilkan kepada pengguna dan status sign in diatur kembali menjadi tidak sedang dalam proses.
9. Request Target Data

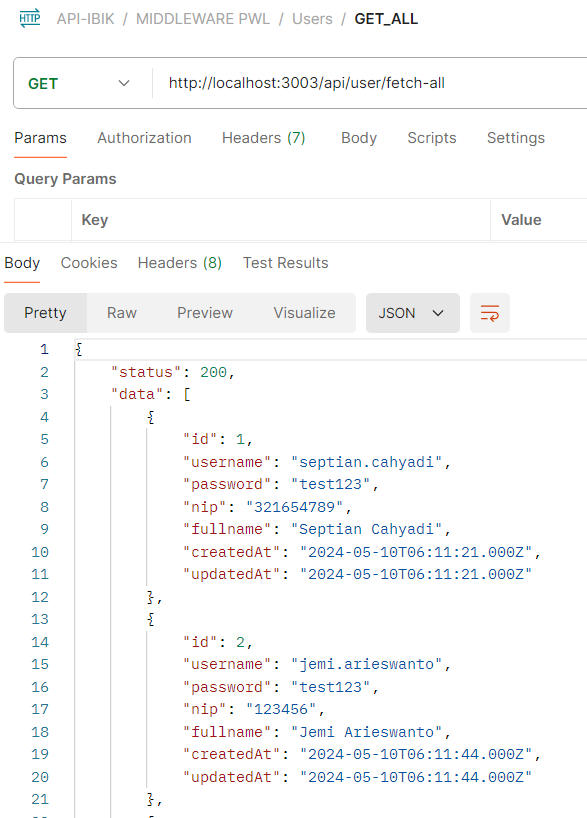


Jika melihat gambar diatas kita akan mengkoneksikan atau membuat data kontak dan chat menjadi lebih dinamik dengan menghubungkan antara middleware dengan react framework. Jika sebelumnya kita memiliki bentuk nilai konstanta untuk daftar pengguna seperti dibawah ini:



Bentuk nilai konstanta diatas kita akan ubah dengan cara memanggil end-poin pada middleware berdasarkan uri : <http://localhost:3003/api/user/fetch-all>

Jika dijalankan pada postman maka memiliki nilai output sebagai berikut:



Ubahlah file komponen ChapterThree seperti dibawah ini:

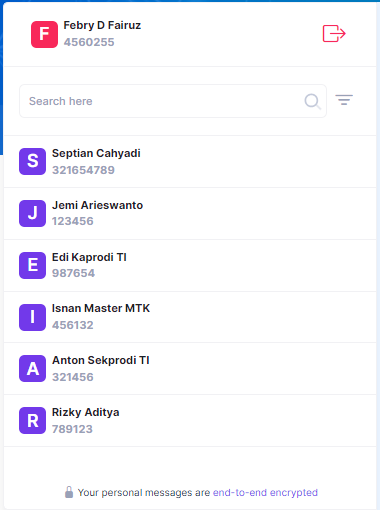
|  |
| --- |
| export function ChapterThree() {    const accountArr = localStorage.getItem("user\_account");    const myprofile = JSON.parse(accountArr);    const [selectedUser, setSelectedUser] = useState({});      const HandlerSelectedChat = (data) => {      setSelectedUser(data);    };    const [myFriends, setMyFriends] = useState({      loading: false,      data: [],      messages: "",    });    useEffect(() => {      FETCH\_CONTACT\_CHAT();    }, []);    const FETCH\_CONTACT\_CHAT = () =>{  …    }  return (      <div>        <h1 className="text-white mb-5">          Chapter three: Connection to Middleware        </h1>        <div className="px-5">          <div className="row">            <div className="col-2 col-lg-3 col-xxl-4 px-0">              {myFriends.loading ? (                <MySkeleton />              ) : myFriends.messages ? (                <AlertNotif color={"danger"} message={myFriends.messages} />              ) : Object.values(myFriends.data).length > 0 ? (                <ContactUI                  my\_account={myprofile}                  friends={myFriends.data}                  selectedUser={selectedUser}                  HandlerSelectedChat={HandlerSelectedChat}                />              ) : "" }            </div>            <div className="col-10 col-lg-9 col-xxl-8 px-0">              {(Object.keys(selectedUser).length > 0) ? (                <MessegersUI                  profile={myprofile}                  selectedUser={selectedUser}                />              ) : (                …              )}            </div >          </div >        </div >      </div >    ); |

1. **const [selectedUser, setSelectedUser] = useState({});**: Ini adalah pendefinisian state selectedUser menggunakan hook useState(). State ini akan digunakan untuk menyimpan informasi tentang pengguna yang dipilih oleh pengguna untuk memulai percakapan.
2. **const HandlerSelectedChat = (data) => { setSelectedUser(data); };**: Ini adalah fungsi yang digunakan untuk menangani peristiwa saat pengguna memilih pengguna lain untuk memulai percakapan. Fungsi ini memperbarui state selectedUser dengan data pengguna yang dipilih.
3. **const [myFriends, setMyFriends] = useState({ ... });**: Ini adalah pendefinisian state myFriends menggunakan hook useState(). State ini akan digunakan untuk menyimpan informasi tentang teman-teman pengguna yang diambil dari server.
4. **useEffect(() => { FETCH\_CONTACT\_CHAT(); }, []);**: Ini adalah efek samping (side effect) yang dijalankan setelah render pertama komponen. Efek ini bertujuan untuk menjalankan fungsi FETCH\_CONTACT\_CHAT() untuk mengambil daftar teman pengguna dari server.
5. **{myFriends.loading ? ( … ) : myFriends.messages ? ( … ) : Object.values(myFriends.data).length > 0 ? ( … ) : "" }**: Ini adalah ekspresi kondisional yang digunakan untuk menampilkan konten berdasarkan status loading, pesan kesalahan messages, dan data teman myFriends.data. Jika loading adalah true, maka akan ditampilkan komponen MySkeleton. Jika ada messages, maka akan ditampilkan AlertNotif dengan warna merah. Jika myFriends.data berisi data teman, maka akan ditampilkan ContactUI dengan properti yang sesuai.

Sedangkan bentuk AXIOS untuk merequest API middleware sebagai berikut:

|  |
| --- |
| const FETCH\_CONTACT\_CHAT = () => {    setContact({ loading: true, data: [], message: "" });    let config = {      method: "get",      maxBodyLength: Infinity,      url: middleware\_uri + "/api/user/fetch-all",      headers: headerApp,    };    axios      .request(config)      .then((response) => {        const result = response.data.data;        if (result) {          setContact({ loading: false, data: result, message: "" });        } else {          setContact({            loading: false,            data: [],            message:              "Unable to establish a connection to the server.  Please try again later.",          });        }      })      .catch((error) => {        setContact({ loading: false, data: [], message: error.message });      });  }; |

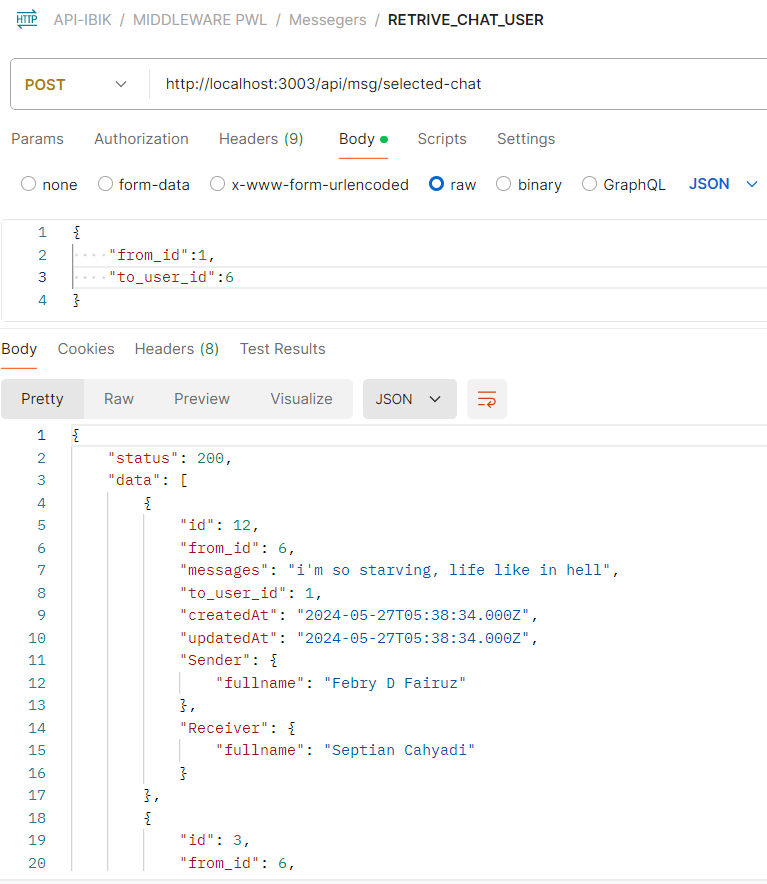
Dari bentuk code diatas maka akan menampilkan bentuk seperti dibawah ini, pastikan anda telah memiliki data pada table users:



Jika anda telah merubah pada bagian Contact UI, selanjutnya ialah merubah bentuk nilai konstanta daftar chat yang sebelumnya seperti ini:



Diubah menjadi bentuk REST API dengan end-poin <http://localhost:3003/api/msg/selected-chat> dan memiliki bentuk parameter dan hasil seperti dibawah ini:



Ubahlah file MessengerUI anda, yang dimana sebelumnya memanggil chat menggunakan nilai static menjadi bentuk dinamik dengan menggunakan middleware yang telah anda buat seperti dibawah ini:

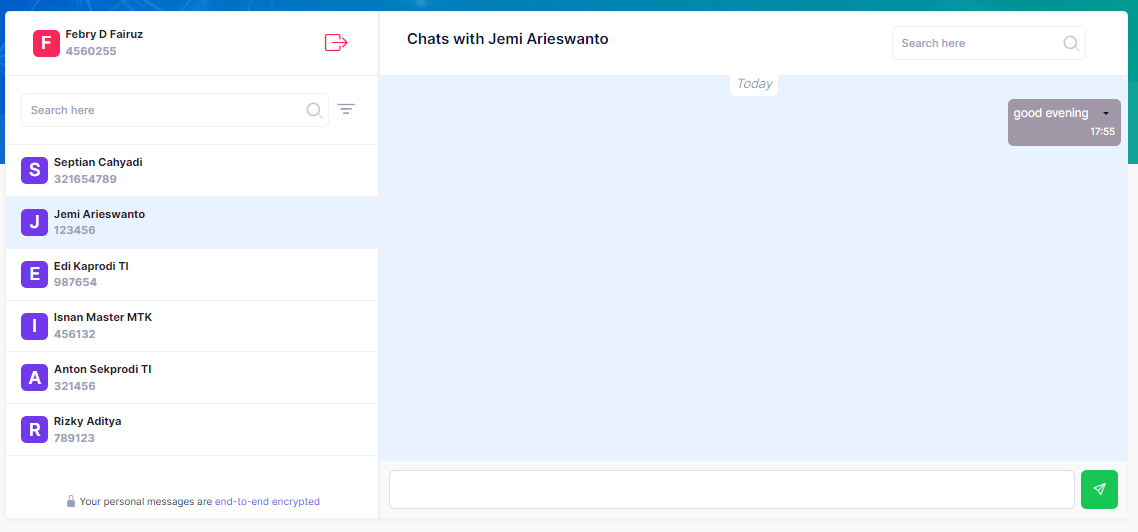
|  |
| --- |
| export function MessegersUI({ profile, selectedUser }) {    const [selectedChat, setSelectedChat] = useState({      loading: false,      data: [],      message: "",    });    const [myChat, setMyChat] = useState([]);    const ReloadData = (user\_id) => {      const param = { from\_id: profile.id, to\_user\_id: user\_id };      GET\_SELECTED\_CHAT(param, setSelectedChat, setMyChat);    }    const endOfMessagesRef = useRef(null);    const scrollToBottom = () => {      endOfMessagesRef.current?.scrollIntoView({ behavior: "smooth" });    };    const [search, setSearch] = useState([]);    const ResultMessageData = useMemo(() => {      let computedData = [];      if (Object.values(myChat).length > 0) {        computedData = myChat.map((msg) => {          return {            ...msg,            date\_fmt: moment(msg.createdAt).format("YYYY-MM-DD"),            isOutgoing: msg.from\_id === profile.id          }        });      }      computedData.sort((a, b) => (a.id > b.id ? 1 : -1));      if (search) {        computedData = computedData.filter(          listData => {            return Object.keys(listData).some(key =>              listData[key].toString().toLowerCase().includes(search)            );          }        );      }      return computedData;    }, [myChat, profile.id, search]);    useEffect(() => {      ReloadData(selectedUser.id)    }, [selectedUser.id]);    const [writeChat, setWriteChat] = useState("");    const [chatMsg, setChatMsg] = useState("");    const [sendChat, setSendChat] = useState({ loading: false, data: [], message: "" });    const HandlerSendChat = (e) => {      e.preventDefault();      if (writeChat.trim()) {        const msg = writeChat.toLowerCase();        const paramPost = {          from\_id: profile.id,          messages: msg,          to\_user\_id: selectedUser.id,        }        SEND\_CHAT(paramPost, setSendChat);        setWriteChat("");        setChatMsg("");        scrollToBottom();        setTimeout(() => {          ReloadData(selectedUser.id)        }, 500);      } else {        setChatMsg("Please fill up the field");      }    };    return (      <div className="card card-flush h-100 mb-5 mb-xl-10 rounded-0 rounded-end-1">        ...        <div className="card-body d-flex flex-column justify-content-between p-0 bg-light-primary h-100">          <div className="chat-message px-2 h-100" style={StylesMessager.chatBox}>            {Object.values(selectedChat.data).length > 0 ? (              <>                <ChatBodyWithGrouped data={ResultMessageData} profile={profile} />                <div ref={endOfMessagesRef} />                {sendChat.message ?  <AlertNotif color={"danger"} message={sendChat.message} /> : ""}              </>            ) : ""}          </div>          <div className="chat-send bg-light p-3">            <form              method="post"              autoComplete="off"              onSubmit={(e) => HandlerSendChat(e)}            >              <div className="d-flex justify-content-between align-items-center">                <input                  type="text"                  className="form-control me-2"                  autoFocus={true}                  value={writeChat}                  onChange={(e) => setWriteChat(e.target.value)}                />                <ButtonPrimary                  items={{                    title: "Send",                    btn\_class: "btn-icon btn-success",                    type: "submit",                  }}                >                  <i className="bi bi-send"></i>                </ButtonPrimary>              </div>              <span className="text-danger">{chatMsg}</span>            </form>          </div>        </div>      </div>    );  } |

1. **const [selectedChat, setSelectedChat] = useState({ ... });**: Ini adalah pendefinisian state selectedChat menggunakan hook useState(). State ini digunakan untuk menyimpan informasi tentang percakapan yang dipilih, seperti status loading, data pesan, dan pesan kesalahan.
2. **const [myChat, setMyChat] = useState([]);**: Ini adalah pendefinisian state myChat menggunakan hook useState(). State ini digunakan untuk menyimpan data pesan yang diterima atau dikirim oleh pengguna dalam percakapan.
3. **const ReloadData = (user\_id) => { ... }**: Ini adalah fungsi yang digunakan untuk memuat ulang data percakapan dengan teman yang dipilih. Fungsi ini membuat permintaan HTTP untuk mendapatkan data percakapan menggunakan fungsi GET\_SELECTED\_CHAT, dan kemudian memperbarui state selectedChat dan myChat.
4. **const endOfMessagesRef = useRef(null);**: Ini adalah penggunaan hook useRef() untuk membuat referensi ke elemen HTML. Referensi ini akan digunakan untuk menggulir halaman ke bagian bawah setelah data percakapan diperbarui.
5. **const scrollToBottom = () => { ... }**: Ini adalah fungsi yang digunakan untuk menggulir halaman ke bagian bawah setelah data percakapan diperbarui. Fungsi ini menggunakan referensi yang dibuat sebelumnya untuk menggulir halaman.
6. **const [search, setSearch] = useState([]);**: Ini adalah pendefinisian state search menggunakan hook useState(). State ini digunakan untuk menyimpan kata kunci pencarian yang dimasukkan oleh pengguna.
7. **const ResultMessageData = useMemo(() => { ... }, [myChat, profile.id, search]);**: Ini adalah penggunaan hook useMemo() untuk menghitung kembali data pesan setelah terjadi perubahan pada state myChat, profile.id, atau search. Ini membantu mengoptimalkan performa dengan menghindari penghitungan ulang data pesan setiap kali terjadi perubahan pada komponen.
8. **useEffect(() => { ReloadData(selectedUser.id) }, [selectedUser.id]);**: Ini adalah efek samping (side effect) yang dijalankan setiap kali terjadi perubahan pada selectedUser.id. Efek ini memanggil fungsi ReloadData() untuk memuat ulang data percakapan dengan teman yang dipilih.

Sedangkan untuk bentuk REST API dari GET\_SELECTED\_CHAT sebagai berikut:

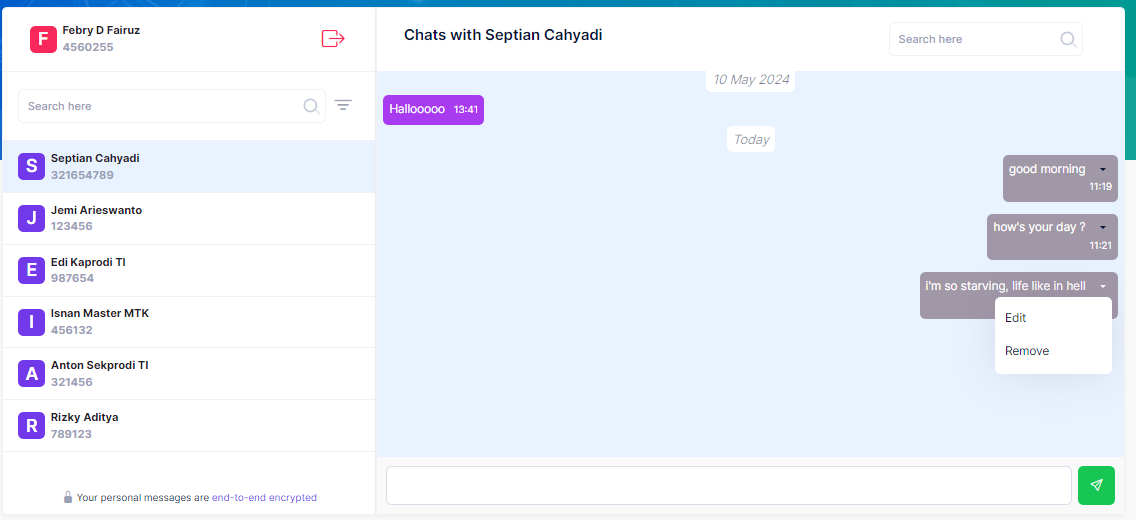
|  |
| --- |
| const GET\_SELECTED\_CHAT = (param) => {    setChat({ loading: true, data: [], message: "" });setMyChat("");    let config = {      method: "post",      maxBodyLength: Infinity,      url: middleware\_uri + "/api/msg/selected-chat",      headers: headerApp,      data: JSON.stringify(param),    };    axios      .request(config)      .then((response) => {        const result = response.data.data;        if (result) {          setMyChat(result);          setChat({ loading: false, data: result, message: "" });        } else {          setMyChat("");          setChat({            loading: false,            data: [],            message:              "Unable to establish a connection to the server. Please try again later.",          });        }      })      .catch((error) => {        setMyChat("");        setChat({ loading: false, data: [], message: error.message });      });  }; |

Jika anda telah mengupdate script tersebut maka memiliki bentuk output seperti dibawah ini, dimana anda telah membuat sebuah chat secara real-time:

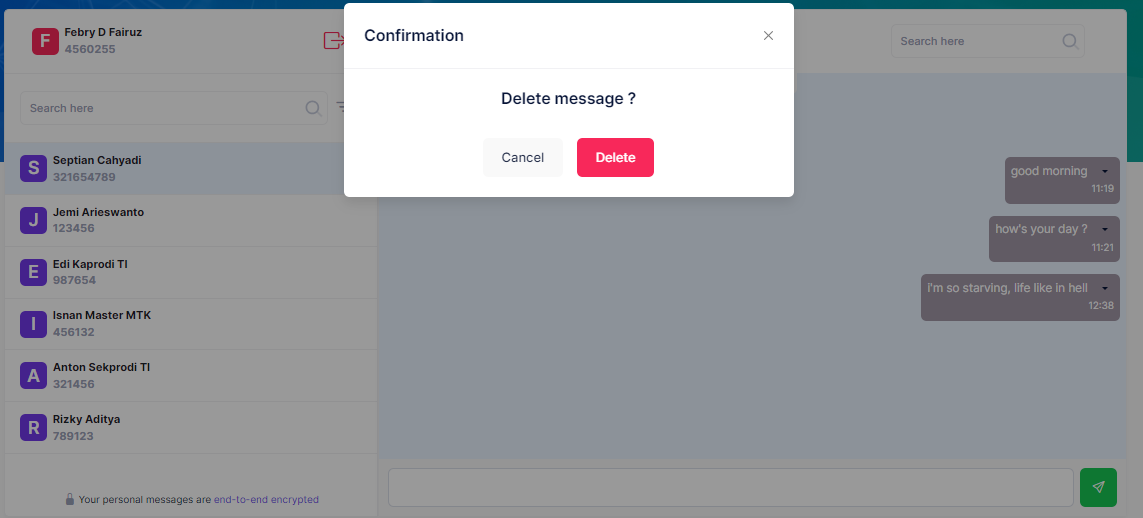


1. **Tugas Pelatihan**

Silakan anda lanjutkan soal Latihan diatas dengan menambahkan aksi seperti update dan remove data dengan middleware sebagai bentuk komunikasi HTTP Request PUT dan DELETE:



1. Jika memilih salah satu chat Sender (akun anda), maka chat tersebut bisa di edit.
2. Jika memilih salah satu chat Sender (akun anda), maka chat tersebut bisa di hapus namun berikan pop up consent seperti dibawah ini:



1. Buatlah aksi untuk sign-out, jika mengklik tombol atau mengakses <http://localhost:3004/sign-out> maka akan menghapus semua data pengguna pada browser

Pengumpulan tugas Latihan praktikum dikumpulkan kedalam GITHUB masing-masing mahasiswa berdasarkan repository yang telah dibuat PWL-TI-21-PA-NPM. File source code disimpan sesuai nama project-praktikum dan masukan kedalam repositori tersebut. Buatkanlah file dokumen dalam bentuk file pdf yang berisi Screen Capture dari hasil program yang telah dikerjakan. Simpan dalam file PDF tersebut kedalam project tersebut.

Tambahkan Collaborator management access pada repository ke @*IrvanRizkyAriansyah* dan *@FebryFairuz*