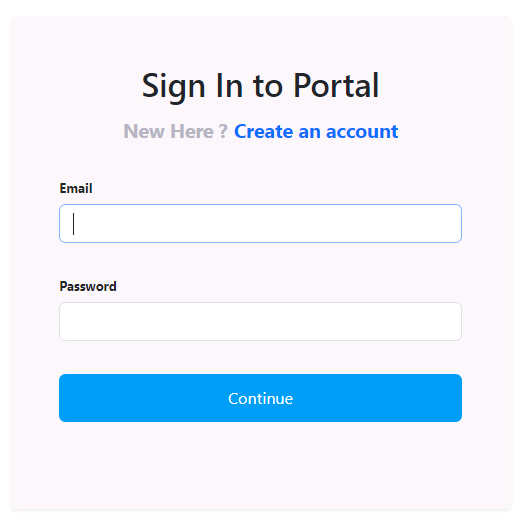
**BAB V**

**AUTHENTIFICATIONS**

1. **Form Sign In**

Pada bab ini akan menjelaskan bagaimana membuat sebuah session login untuk authentification pada sebuah website. Dari materi sebelumnya telah dibuat sebuah halaman MicroFrontend untuk Sign In user dengan isian email dan password, dimana halaman ini akan diakses melalui route bernama /signin.



***Gambar 5.1.*** *Halaman Sign In*

Dari gambar diatas memiliki bentuk component function sebagai berikut:

Form.js

A screenshot of a computer code

Description automatically generated with low confidence

Dari script diatas setelah memiliki bentuk JSX dari form Sign In, maka selanjutnya perlu memasukan HOOK untuk mengelola data transaksi sign-in.

1. **Menambahkan HOOK state**

Pada file Form.js tambahkan hook state library useState untuk menampung data email dan password sebagai berikut:

|  |
| --- |
| const [objAuth, setObjAuth] = useState({    email: "",    password: ""  }); |

Setelah memasang state hook selanjutnya akan menambahkan properties onChange pada form inputan untuk mengambil atau mengisi nilai kedalam state objAuth.

Input text pada email:

|  |
| --- |
| <input type="email" name="email" className='form-control form-control-lg' **defaultValue={objAuth.email}**  **onChange={(e) => setObjAuth({ ...objAuth, email: e.target.value })}**  /> |

Input text pada password:

|  |
| --- |
| <input type="password" name="password" className='form-control form-control-lg'  **defaultValue={objAuth.password}**  **onChange={(e) => setObjAuth({ ...objAuth, password: e.target.value })}**  /> |

1. **Membuat handler submit**

Selanjutnya menambahkan handler submit untuk form sign-in, agar data-data yang telah disimpan pada state objAuth dapat dikelola atau dilakukan proses transaksi, seperti mengirimkan state objAuth kedalam microservices.

Buatlah sebuah function ES6 bernama handlerSubmit, dimana function ini akan mengelola data-data yang akan dieksekusi.

|  |
| --- |
| const handlerSubmit = (e) => {    e.preventDefault();  } |

Setelah membuat function handlerSubmit, inisialisasi function tersebut kedalam element form dengan menggunakan properties onSubmit.

|  |
| --- |
| <form method="post" …  **onSubmit={(e) => handlerSubmit(e)}** >  …  </form> |

1. **Membuat validasi**

Disini kita akan menambahkan validasi sederhana untuk mengecek isian email dan password tidak boleh kosong. Jika salah satu data isian kosong maka akan menampilkan alert bertulisan “All field is filled with correctly”. Sedangkan jika salah satu isian kosong maka akan menampilkan alert “Please fill up the form with correctly”.

|  |
| --- |
| const handlerSubmit = (e) => {    e.preventDefault();    if (objAuth.email !== "" && objAuth.password !== "") {      alert("All field is filled with correctly");  } else {      alert("Please fill up the form with correctly");    }  } |

A picture containing text, screenshot, font, white

Description automatically generatedA screenshot of a computer

Description automatically generated with low confidence

***Gambar 5.1.2.*** *Contoh alert validasi pada form*

1. **Microservices Authentification**

Silakan jalankan file microservices pada module sebelumnya hingga berjalan dengan lancar, pada dokumentasi microservices sebelumnya telah memiliki *end-point* untuk sesi authentification, yaitu:

A picture containing text, screenshot, font, line

Description automatically generated

Untuk mengetes *end-point* tersebut silakan buka Postman dan isi sesuai dengan target data pada gambar diatas:

A screenshot of a computer

Description automatically generated with medium confidence

***Gambar 5.2*** *Test end-point authentification pada postman*

1. **Testing Web Services API Auth**

Setelah mengisi seluruh data target pada postmant, sesuai dengan gambar 5.2, ketika mengklik tombol Send maka akan menampilkan hasil nilai balik pada backend dengan kondisi:

1. Jika isian email dan password benar maka akan mengembalikan nilai berupa object yang memiliki tiga buah key, yaitu *access\_token, refresh\_token* dan *id\_token*.
2. Sedangkan jika isian salah atau data tidak berada dalam database maka akan menampilkan sebuah objek dengan isian key, *result, data*, dan *message*. Pada key message akan menampilkan pesan kesalahan seperti “Cannot invoke”, yang berarti data user tersebut tidak ditemukan atau tidak ada didalam sistem database.

A screenshot of a computer

Description automatically generated with medium confidence

***Gambar 5.2.1.*** *Testing web services api auth*

1. **Mengkoneksikan Micro-frontend dengan Microservices**

Setelah menguji web services untuk api authentification berhasil maka selanjutnya perlu mengkoneksikan micro-frontend dengan microservices agar web aplikasi yang kita buat memiliki hak pengguna. Dalam contoh kasus kali ini, mengkoneksikan antara microfrontend dengan microservices sama seperti pada bab sebelumnya, yaitu menggunakan library bernama AXIOS.

Untuk menyimpan hasil koneksi data dengan AXIOS diperlukan membuat sebuah state untuk menampung nilai balik tersebut. Oleh karenanya disini akan membuat sebuah state bernama postAuth:

|  |
| --- |
| const [postAuth, setPostAuth] = useState({    loading: false,    data: [],    message: ""  }); |

Pada state *postAuth* memiliki tiga buah key, key pertama ialah *loading*. Dimana key ini diperuntukan untuk memberitahukan bahwa sedang terjadi koneksi antara *microfrontend* dengan *microservices*. Key *data* akan menyimpan nilai balik dari microservices, sedangkan key *message* diperuntukan menyimpan nilai balik microservices jika memiliki pesan kesalahan.

Untuk menggunakan library AXIOS ini perlu menginisialisasi atau mengimport pada heading script agar dapat menggunakan library tersebut.

import axios from "axios";

Selanjutnya tambakan script axios dibawah ini kedalam function yang telah dibuat yaitu handlerSubmit() pada kondisi validasi seluruh element terpenuhi:

A screenshot of a computer program

Description automatically generated with low confidence

Dari script diatas jika kondisi user account terdeteksi pada database maka state postAuth untuk *key* *message* akan menyimpan value berupa tulisan “*Welcome [email user]*” dan pada *key data* akan menyimpan object token seperti pada gambar 5.2.1. Sedangkan jika data tidak ada pada database atau terjadi kesalahan pada salah satu isian maka key message pada state postAuth berisi “*Email or password is not match*”.

Untuk menampilkan informasi state postAuth, pada bagain JSX form sign in anda dapat menampilkannya dengan menggunakan expression:

{(postAuth.message) ? <p>{postAuth.message} </p> : ""}

A screenshot of a login form

Description automatically generated with medium confidenceA screenshot of a login form

Description automatically generated with medium confidence

***Gambar 5.2.2.*** *Tampilan koneksi microfrontend dan microservices*

1. **Session pada browser**

Bagian ini akan menjelasakan beberapa teknik penyimpanan session/data yang paling dasar atau paling mudah digunakan. Pada JavaScript memiliki dua buah teknik yang dapat digunakan untuk menyimpan sebuah data pada browser yang digunakan, yaitu ada *SessionStorage* dan *LocalStorage*.

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

***Gambar 5.3.1.a.*** *Tampilan session storage*

1. Retrive session

Berikut ini adalah contoh cara memanggil session yang telah disimpan:

Format untuk memanggil session storage:

sessionStorage.getItem(‘key\_name’)

Contoh implementasi, memanggil session storage yang telah disimpan yaitu id\_token:

|  |
| --- |
| var id\_token = sessionStorage.getItem(‘id\_token’);  return (  <div className=”bg-info p-3 rounded”>       <p className=”fw-bolder”>ID Token:</p>       <pre> {id\_token} </pre>  </div>  ) |

A blue rectangle with black text

Description automatically generated with low confidence

Dikarenakan hasil dari backend nilai baliknya berupa encrypted data jika ingin melakukan decoded terhadap nilai tersebut diperlukan installasi library bernama JWT Decode (<https://www.npmjs.com/package/jwt-decode>). Contoh kasus pada bagian ini akan menggunakan JWT Decode untuk melakukan decripted dari key id\_token:

A screen shot of a computer program

Description automatically generated with low confidence

A screenshot of a computer

Description automatically generated with medium confidence

***Gambar 5.3.1.b.*** *Tampilan memanggil session storage*

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')

Contoh kasus:

A screen shot of a computer

Description automatically generated with low confidence

1. Remove session

* Remove key

Format:

localStorage.removeItem('key\_name');

* Remove all

Format:

localStorage.clear();

1. **Session dengan third party (library)**
2. **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');

1. **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

1. **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. **Latihan Praktikum**
2. Pada project sebelumnya silakan anda pasang Authentification untuk form sign-in.
3. Berikan contoh penggunaan basik untuk penyimpanan LocalStorage dan SessionStorage pada masing-masing script sign-in.
4. Berikan pendapat kalian mana yang lebih baik digunakan untuk melakukan penyimpanan data menggunakan LocalStorage, SessionStorage, React Cookies, React Session atau React Firebase. Dan berikan alasannya.

Pengumpulan tugas Latihan praktikum dikumpulkan kedalam GITHUB masing-masing mahasiswa berdasarkan repository yang telah dibuat PWL-TI-20-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 anda ke *@FebryFairuz*

**BAB VI**

**WEB SOCKET API**

1. **Pendahuluan**

WebSocket adalah teknologi yang memungkinkan komunikasi dupleks penuh dua arah antara klien dan server melalui koneksi soket tunggal yang persisten. Tujuannya adalah untuk memberikan apa yang pada dasarnya merupakan lapisan komunikasi TCP yang sedekat mungkin dengan pengembang aplikasi web sambil menambahkan beberapa abstraksi untuk menghilangkan gesekan tertentu yang seharusnya ada mengenai cara kerja web. Koneksi WebSocket dimulai dengan *hand-shake* permintaan/respons HTTP; di luar *hand-shake*, WebSocket dan HTTP pada dasarnya berbeda.

A picture containing text, screenshot, diagram, parallel

Description automatically generated

***Gambar 1.*** *Ilustrasi cara kerja WebSocket vs HTTP*

Teknologi WebSocket memiliki dua buah tipe blok inti:

* Protokol WebSocket. Memungkinkan komunikasi antara klien dan server melalui web, dan mendukung transmisi data biner dan string.
* API WebSocket. Memungkinkan Anda melakukan tindakan yang diperlukan, seperti mengelola koneksi WebSocket, mengirim dan menerima pesan, dan mendengarkan peristiwa yang dipicu oleh server.

Teknologi WebSocket memiliki penerapan yang luas. Anda dapat menggunakannya untuk berbagai tujuan, seperti streaming data antara layanan backend, atau menghubungkan backend dengan frontend melalui koneksi *full-duplex* yang tahan lama. Singkatnya, WebSockets adalah pilihan yang sangat baik untuk merancang sistem yang digerakkan oleh *event* dan membangun aplikasi dan layanan *real-time* yang sangat penting agar data segera dikirim.

WebSocket memiliki dua kategori yang diterapkan untuk melakukan komunikasi antar client-server:

* **Realtime Update**, di mana komunikasi bersifat searah, dan server mengalirkan pembaruan latensi rendah (request per detik) ke klien. Contohnya seperti membuat, Alert, realtime dashboard, atau pelacakan lokasi.
* **Bidirectional communication**, dimana klien dan server dapat mengirim dan menerima pesan. Contohnya seperti chat, video call, dan ruang kelas virtual (dua yang terakhir biasanya melibatkan fitur seperti jajak pendapat, kuis, dan Tanya Jawab). WebSockets juga dapat digunakan untuk mendukung fungsionalitas kolaborasi tersinkronisasi multi-pengguna, seperti beberapa orang yang mengedit dokumen yang sama secara bersamaan.

1. **Protocol**

Protokol WebSocket memungkinkan komunikasi dua arah yang berkelanjutan, *full-duplex*, antara server web dan klien web melalui koneksi TCP yang mendasarinya.

Singkatnya, protokol WebSocket terdiri dari *hand-shake* pembuka (memutakhirkan koneksi dari HTTP ke WebSockets), diikuti dengan transfer data. Setelah klien dan server berhasil menegosiasikan *hand-shake* untuk pembuka, maka koneksi WebSocket akan bertindak sebagai saluran komunikasi *full-duplex* yang persisten di mana masing-masing pihak dapat, secara mandiri, mengirim data sesuka hati.

A picture containing text, screenshot, font, line

Description automatically generated

***Gambar 1.1.*** *Ilustrasi cara kerja protocol websocket*

Klien dan server mentransfer data secara bolak-balik dalam unit konseptual yang disebut sebagai pesan, seperti yang akan dijelaskan secara singkat, dapat terdiri dari satu atau lebih *framing*. Setelah koneksi WebSocket memenuhi tujuannya, baru dapat diakhiri melalui *hand-shake* penutupan.

1. **URI Schema dan syntax**

Protokol WebSocket mendefinisikan dua skema URI untuk lalu lintas antara server dan klien:

* *ws*, digunakan untuk koneksi yang tidak terenkripsi.
* *wss*, digunakan untuk koneksi terenkripsi yang aman melalui Transport Layer Security (TLS).

URI WebSocket lainnya mengikuti sintaks generik, mirip dengan HTTP. Terdiri dari beberapa komponen: host, port, jalur, dan query, seperti yang contoh pada gambar dibawah ini:

A picture containing text, font, line, white

Description automatically generated

Gambar 1.2. Schema URI websocket

Perlu diketahui:

* Komponen port bersifat opsional; defaultnya adalah port 80 untuk ws, dan port 443 untuk wss.
* Pengidentifikasi fragmen tidak diizinkan di URI WebSocket.
* Karakter hash (#) harus di-escape sebagai %23.

1. **WebSocket API**

Server WebSocket dapat ditulis dalam bahasa pemrograman sisi server apa pun yang mendukung Berkeley sockets30. Server mendengarkan koneksi WebSocket masuk menggunakan soket TCP standar. Setelah *hand-shake* terbuka, server harus dapat mengirim, menerima, dan memproses pesan WebSocket. Untuk memanfaatkan teknologi WebSocket di sisi server, diperlukan aplikasi backend.

Node.JS memiliki sebuah websocket yang dapat digunakan, runtime JavaScript berbasis event-asynchronous yang ringan dan efisien. Node.js pilihan yang sangat baik untuk membangun aplikasi web *real-time* yang dapat diskalakan dan memelihara ratusan koneksi WebSocket bersamaan.

1. **Setting up websocket**

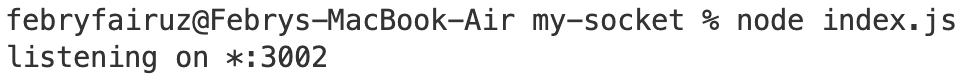
Buatlah sebuah folder baru bernama MY-SOCKET, lalu buatlah file bernama index.js. Bukalah folder tersebut dengan terminal dan masukan syntax untuk menambahkan library Express dan Socket.io:

|  |
| --- |
| npm install express  npm install socket.io |

Setelah berhasil menginstall kedua library tersebut buatlah script open connection untuk menjalan websocket pada file index.js:

|  |
| --- |
| const express = require('express');  const app = express();  const http = require('http');  const server = http.createServer(app);  app.get('/', (req, res) => {  res.send('<h1>Hello world</h1>');  });  server.listen(3002, () => {  console.log('listening on \*:3002');  }); |

Untuk menjalankan websocket, pada terminal project anda masukan syntax node index.js:



Jika ingin mengetes apakah websocket anda berjalan bukalah browser dan masukan location dan port yang sudah ditentukan:

A screenshot of a computer

Description automatically generated with low confidence

***Gambar 2.1.*** *Tes connection websocket*

1. **Websocket pada client-side**

Pada contoh kasus penggunaan websocket pada client-side akan membuat sebuah aplikasi chat sederhana secara real-time. Buatlah file bernama index.html pada project socket anda dan buatlah element sederhana untuk UI seperti dibawah ini:

A screenshot of a chat

Description automatically generated with medium confidence

***Gambar 2.2.*** *UI aplikasi chat*

Contoh script untuk bentuk ui seperti gambar diatas:

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title>Socket.IO chat</title>  <style>  body { margin: 0; padding-bottom: 3rem; font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, Helvetica, Arial, sans-serif; }  #form { background: rgba(0, 0, 0, 0.15); padding: 0.25rem; position: fixed; bottom: 0; left: 0; right: 0; display: flex; height: 3rem; box-sizing: border-box; backdrop-filter: blur(10px); }  #input { border: none; padding: 0 1rem; flex-grow: 1; border-radius: 2rem; margin: 0.25rem; }  #input:focus { outline: none; }  #form > button { background: #333; border: none; padding: 0 1rem; margin: 0.25rem; border-radius: 3px; outline: none; color: #fff; }  #messages { list-style-type: none; margin: 0; padding: 0; }  #messages > li { padding: 0.5rem 1rem; }  #messages > li:nth-child(odd) { background: #efefef; }  </style>  </head>  <body>  <ul id="messages"></ul>  <form id="form" action="">  <input id="input" autocomplete="off" /><button>Send</button>  </form>  </body>  </html> |

1. Integrating Socket.IO

Updatelah file index.js anda dan masukan syntax dibawah ini mengkoneksikan client:

|  |
| --- |
| app.get('/', (req, res) => {  res.sendFile(\_\_dirname + '/index.html');  }); |

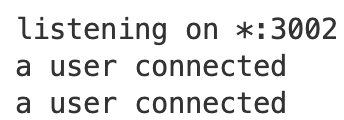
Tambahkan syntax dibawah ini untuk membuat hand-shake pembukaan:

|  |
| --- |
| io.on('connection', (socket) => {  console.log('a user connected');  }); |

Pada script client side diharuskan memanggil koneksi web socket dengan menambahkan code dibawah ini:

|  |
| --- |
| …  <script src="/socket.io/socket.io.js"></script>  <script>  var socket = io();  </script>  … |

Silakan anda lakukan mengecekan pada client side project anda, untuk mengecek hand-shake pembukaan berjalan atau tidak.



1. Emmit event

Proses utama dari Socket.IO adalah dapat mengirim dan menerima event apa pun yang diinginkan, dengan data apa pun yang diinginkan. Objek apa pun yang dapat dikodekan seperti yang akan dilakukan JSON, dan data biner juga didukung. Untuk mengirim dan menerima sebuah data pada socket, dapat menggunakan EMMIT.

Contoh, disini akan mengirimkan data berupa string chat, dimana data tersebut akan dikirimkan ketika mengklik tombol send.

|  |
| --- |
| <script>  var socket = io();  var messages = document.getElementById('messages');  var form = document.getElementById('form');  var input = document.getElementById('input');  form.addEventListener('submit', function(e) {  e.preventDefault();  if (input.value) {  **socket.emit('chat message', input.value);**  input.value = '';  }  });  </script> |

Dari script diatas emit digunakan untuk mengirim value dari isian input:text, dan key nya bernama chat message.

Format: socket.emit(key, value);

socket.emit('chat message', input.value);

Dan pada index.js, script socket, diharuskan membuat sebuah code untuk menerima key yang telah dikirimkan pada hand-shake pembukaan:

|  |
| --- |
| io.on('connection', (socket) => {  console.log('a user connected');  socket.on('chat message', msg => {  console.log('message: ' + msg);  });  }); |

Jika dijalankan maka akan menampilkan output sebagai berikut:

A picture containing text, font, screenshot

Description automatically generated

1. Broadcasting

Setelah mencoba pengiriman request kepada websocet dengan menggunakan emit, selanjutnya ialah menerima balik hasil dari request yang dikirimkannya. Teknik ini dinamakan Broadcasting method, anda dapat memanfaat syntax io.emit() untuk menerima nilai balik dari request.

Pada file socket tambahkan io.emit pada blok socket ‘chat message’:

|  |
| --- |
| io.on('connection', (socket) => {  console.log('a user connected');  socket.on('chat message', msg => {  console.log('message: ' + msg);  **io.emit('the chat message', msg);**  });  }); |

Dari script diatas dapat dilihat, disini akan mengirimkan kembali nilai value msg dengan key bernama ‘the chat message’.

Format: io.emit(key, value);

io.emit('the chat message', msg);

Pada file client-side anda perlu melakukan hand-shake terhadap key yang dikirimkan dengan menambahkan script seperti berikut:

|  |
| --- |
| <script>  ...  socket.on('the chat message', function(msg) {  alert(msg);  });  ...  </script> |

A screenshot of a computer

Description automatically generated with medium confidence

Selanjutnya tinggal anda membuat append terhadap value yang dikirimkan dengan menggunakan javascript native:

|  |
| --- |
| <script>  ...  socket.on('the chat message', function(msg) {  **var item = document.createElement('li');**  **item.textContent = msg;**  **messages.appendChild(item);**  **window.scrollTo(0, document.body.scrollHeight);**  });  ...  </script> |

Jika dites dengan multiple browser maka konsep Real-Time communication berjalan dengan baik seperti dibawah ini:

A screenshot of a computer

Description automatically generated with medium confidence

1. **Latihan**
2. Buatlah penggunaan websocket seperti contoh kasus diatas dengan menambahkan inputan user account.
3. Dapatkah anda membuat replica dari soal nomor 1 kedalam bentuk React JS.

Pengumpulan tugas Latihan praktikum dikumpulkan kedalam GITHUB masing-masing mahasiswa berdasarkan repository yang telah dibuat PWL-TI-20-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 anda ke *@FebryFairuz*