**BAB II**

**REACT COMPONENT**

1. **Konsep Komponen**

Komponen adalah unit dasar dalam pembangunan antarmuka pengguna (UI) yang dapat diulang dan dapat dipecah menjadi bagian-bagian yang lebih kecil. Mereka membantu memisahkan logika UI menjadi potongan-potongan yang lebih mudah dikelola.

Pada bab sebelumnya anda telah mengenal beberapa komponen utama yang dimiliki oleh React, yaitu:

1. **Class Components**

Komponen ini ditulis sebagai kelas JavaScript dan dapat memiliki state, lifecycle methods, dan method lainnya. Contoh:

|  |
| --- |
| import React, { Component } from 'react'  export default class MyRCC extends Component {    render() {      return (        <div>          <h1>Contoh class component</h1>        </div>      )    }  } |

1. **Functional Components**

Merupakan komponen yang ditulis sebagai fungsi JavaScript. Pada awalnya, *functional components* hanya dapat digunakan untuk komponen sederhana yang tidak memiliki *state* atau *lifecycle methods*. Namun, dengan diperkenalkannya *React Hooks*, *functional components* sekarang dapat memiliki *state* dan mengatur *lifecycle events*. Contoh:

|  |
| --- |
| export default function MyRFC(){  return (        <div>          <h1>Contoh functional component</h1>        </div>    )  } |

Atau dapat juga ditulis dengan bentuk functional ES6 (ECMAScript 6):

|  |
| --- |
| const MyRFC = ({ children, items, actions }) => {    return (        <div>          <h1>Contoh functional component</h1>        </div>    )  }  export default MyRFC; |

1. **JSX (Javascript XML)**

JSX adalah ekstensi sintaksis yang memungkinkan penulisan markup HTML di dalam JavaScript. JSX mempermudah pembuatan dan pemeliharaan struktur UI.

1. **Lifecycle Methods**

Lifecycle Methods memungkinkan komponen menanggapi perubahan siklus hidupnya, setiap komponen React memiliki tiga tahapan berbeda:

* **Mounting -** Mounting merepresentasikan rendering komponen React di node DOM yang diberikan.
* **Updating** - Memperbarui mewakili rendering ulang komponen React di node DOM yang diberikan selama perubahan / pembaruan status.
* **Unmounting** - Unmounting menunjukkan penghapusan komponen React.

1. **Import dan Export Komponen**

Importing dan Exporting Components pada React digunakan untuk mengelola modularitas dalam pengembangan aplikasi. Ini memungkinkan kita untuk memecah kode ke dalam bagian-bagian yang lebih kecil dan mandiri yang dapat digunakan atau diimpor di berbagai bagian aplikasi.

Kegunaan Importing Components:

* **Reusable Code:**

Dengan mengimpor komponen, Anda dapat menggunakan dan me-reuse komponen tersebut di berbagai bagian aplikasi Anda. Ini memungkinkan penggunaan kembali logika dan antarmuka pengguna yang sama di beberapa tempat.

* **Modular Development:**

Pembagian komponen ke dalam file-file terpisah memungkinkan pengembangan modular. Setiap komponen dapat dikembangkan dan diuji secara terpisah, memudahkan perbaikan bug dan pemeliharaan kode.

* **Collaboration:**

Dalam tim pengembangan, import-export mempermudah kolaborasi. Setiap anggota tim dapat fokus pada pengembangan komponen atau bagian tertentu tanpa harus khawatir dengan detail implementasi di bagian lainnya.

* **Readability:**

Kode yang dibagi menjadi komponen-komponen terpisah meningkatkan keterbacaan. File yang lebih kecil dan lebih terfokus membuat kode lebih mudah dimengerti dan dikelola.

Kegunaan Exporting Components:

* **Public Interface:**

Dengan mengekspor komponen tertentu, Anda mendefinisikan "interface publik" yang dapat digunakan oleh komponen lain. Ini membantu dalam menjaga privasi atau abstraksi internal komponen.

* **Namespace:**

Export memberikan ruang nama (namespace) untuk komponen atau fungsi tertentu. Dengan menyediakan interface yang jelas, penggunaan kode dapat lebih mudah dimengerti.

* **Code Splitting:**

Dengan membagi kode ke dalam modul-modul yang dapat diimpor, kita dapat mengimplementasikan teknik "*code splitting*" di mana hanya bagian-bagian tertentu dari aplikasi yang benar-benar dibutuhkan oleh pengguna yang diunduh, meningkatkan efisiensi dan performa aplikasi.

Dengan cara ini, kita memisahkan fungsionalitas dan membangun aplikasi React dengan cara yang lebih terstruktur dan modular.

Contoh penggunaan importing dan exporting component:

|  |
| --- |
| MyName.jsx |
| const MyName =() => {    return (        <h1>Nama saya adalah Febry Damatraseta Fairuz</h1>    )  }  export default MyRFC; |

|  |
| --- |
| MyProfile.jsx |
| import React, { Component } from 'react'  import MyName from './MyName'  export default class MyProfile extends Component {    render() {      return (        <div>          <h1>Biodata:</h1>  <MyName />        </div>      )    }  } |



1. **Konversi HTML Markup kedalam JSX**

Dalam React, markup atau tampilan antarmuka pengguna (UI) dibangun menggunakan JSX (JavaScript XML). JSX adalah ekstensi sintaksis yang memungkinkan penulisan markup HTML di dalam JavaScript. Jika melihat gambar dibawah ini:

A white background with black text

Description automatically generated

|  |  |
| --- | --- |
| bentuk dari markup HTML: | bentuk dari markup JSX: |
| <h1>Biodata:</h1>  <h1>Nama saya adalah Febry Damatraseta Fairuz</h1> | const MyBio = () => {    return (  <h1>Biodata:</h1>  <h1>Nama saya adalah Febry Damatraseta Fairuz</h1>    )  } |

Jika dilihat dari code pada bentuk HTML, untuk menampilkan markup element Heading dapat berjalan dengan baik. Namun ketika memasukan markup HTMLtersebut kedalam bentuk JSX, maka akan terjadi error seperti dibawah ini:

|  |
| --- |
| Adjacent JSX elements must be wrapped in an enclosing tag. Did you want a JSX fragment <>...</>? |

Hal ini dikarenakan JSX lebih ketat dan memiliki beberapa aturan yang lebih banyak daripada HTML. Jika Anda membaca pesan kesalahan di atas, pesan tersebut akan memberitahukan untuk memperbaiki markup, atau dapat menambahkan JSX *fragment* serperti dibawah ini:

|  |  |
| --- | --- |
| JSX Fragment | Single root element |
| const MyBio = () => {    return (  <>  <h1>Biodata:</h1>  <h1>Nama saya adalah Febry Damatraseta Fairuz</h1>  </>    )  } | const MyBio = () => {    return (  <div>  <h1>Biodata:</h1>  <h1>Nama saya adalah Febry Damatraseta Fairuz</h1>  </div>    )  } |

Jika tidak ingin menggunakan JSX fragment <> .... </> dapat juga menggunakan single root element block HTML seperti <div> ... </div>, <section>...</section>, <main>...</main> dal element block HTML lainnya.

Tidak hanya bentuk markup HTML saja yang memiliki penyesuaian jika diimplementasikan ke dalam JSX. Markup CSS pun perlu dilakukan penyesuaian, Ketika menggunakan CSS dalam JSX di React, gaya ditulis sebagai objek JavaScript bukan sebagai string CSS. Ini mengharuskan untuk mengikuti aturan penulisan properti dalam JavaScript, bukan aturan penulisan CSS. Dalam CSS, kita menggunakan kebab-case untuk nama properti yang terdiri dari dua kata atau lebih. Namun, dalam JavaScript, kebab-case tidak dapat digunakan karena tanda minus (-) akan diinterpretasikan sebagai operator pengurangan. Sebagai gantinya, kita menggunakan camelCase untuk menamai properti gaya dalam JSX. Berikut ini adalah contoh penerapan property CSS kedalam JSX:

|  |
| --- |
| CSS markup |
| .text-danger{ color:"red", font-size:"11px" } |
| JSX CSS markup |
| const myStyle = {  color: 'red',  fontSize:"11px"  }  const MyBio = () => {    return (  <div>  <h1 style={ myStyle }>Biodata:</h1>  <h1 className="text-danger">Nama saya adalah Febry Damatraseta Fairuz</h1>  </div>    )  } |

1. **Penggunaan *props* pada komponen**

Komponen-komponen React menggunakan props untuk berkomunikasi satu sama lain. Setiap komponen induk dapat memberikan beberapa informasi kepada komponen turunannya dengan memberikan props kepada mereka. Props mungkin mengingatkan Anda pada atribut HTML, tetapi Anda dapat mengoperkan nilai JavaScript apa pun melalui props, termasuk objek, array, dan fungsi.

Dalam React, props (properties) digunakan untuk mengirimkan data dari komponen induk ke komponen anak. Ada beberapa cara untuk melakukan parsing atau mengirimkan props antarkomponen. Cara-cara ini memungkinkan fleksibilitas dalam pengiriman data dan membantu dalam pembangunan aplikasi yang modular dan mudah dipelihara. Berikut adalah beberapa metode umum untuk parsing props:

1. **Parsing Props Secara Langsung**

Bisa langsung mengirimkan props ke komponen anak dengan menentukan atribut pada tag komponen anak:

|  |
| --- |
| const MyBio= (props) => {  return (  <h1>Nama saya: {props.name} </h1>  )  }  const MyProfile = () => {  return(  <div>  <h1>Biodata:</h1>  <MyBio name="Febry D Fairuz" />  </div>  )  } |

1. **Menggunakan Spread Operator**

Jika Anda memiliki objek props dan ingin meneruskannya secara utuh, Anda dapat menggunakan spread operator {...} untuk mengirimkan seluruh objek props ke komponen anak:

|  |
| --- |
| const myData = {  name: 'Febry D F',  age: 90  }  const MyBio= (props) => {  return (  <>  <h1>Nama saya: {props.name} </h1>  <h2>Usia: {props.age}</h2>  </>  )  }  const MyProfile = () => {  return(  <div>  <h1>Biodata:</h1>  <MyBio {...myData} />  </div>  )  } |

1. **Destructuring Props**

Destructuring memungkinkan Anda untuk 'membongkar' objek props di dalam parameter fungsi komponen, sehingga Anda bisa langsung menggunakan properti individual tanpa perlu menyebut props. setiap kali:

|  |
| --- |
| const MyBio= ({name, age}) => {  return (  <>  <h1>Nama saya: {name} </h1>  <h2>Usia: {age}</h2>  </>  )  }  const MyProfile = () => {  return(  <div>  <h1>Biodata:</h1>  <MyBio name="Febry D F" age= "80"/>  </div>  )  } |

1. **Props Children**

*props.children* adalah properti khusus yang memungkinkan komponen untuk me-render konten yang diberikan di antara tag pembuka dan penutup komponen tersebut. Ini sangat berguna untuk kasus seperti komponen wrapper atau higher-order components:

|  |
| --- |
| const MyProfile = () => {  return(  <div>  <h1>Biodata:</h1>  <MyBio name="Febry D F" age= "80"/>  <ClickMe>  Klik Saya!  </ClickMe>  </div>  )  }  const ClickMe = ({children}) =>{  return (  <button type='button'>  {children}  </button>  )  } |

1. **Mengirimkan Fungsi sebagai Props**

Props tidak hanya terbatas pada data statis atau dinamis tetapi juga bisa berupa fungsi yang memungkinkan komunikasi balik dari komponen anak ke induk:

|  |
| --- |
| const HandlerHitme = () =>{  alert("Anda telah mengetuk saya");  }  const MyProfile = () => {  return(  <div>  <h1>Biodata:</h1>  <MyBio name="Febry D F" age= "80"/>  <ClickMe onClickBtn={HandlerHitme}>  Klik Saya!  </ClickMe>  </div>  )  }  const ClickMe = ({children, onClickBtn}) =>{  return (  <button type='button' onClick={()=>onClickBtn()}>  {children}  </button>  )  } |

Metode-metode ini menunjukkan berbagai cara untuk memanipulasi dan mengirimkan data antarkomponen dalam aplikasi React, masing-masing dengan kegunaan spesifik tergantung pada kasus penggunaan Anda. Menggunakan cara yang tepat pada tempat yang tepat akan membuat kode Anda lebih bersih, lebih terorganisir, dan lebih mudah untuk dipelihara.

1. **Latihan: Membangun Aplikasi Messenger - Chapter 1**
2. Membuat Project React

Buatlah project dengan nama `pemograman-web-lanjut-npm` dengan menggunakan react *toolchain* *create*:

|  |
| --- |
| npx create-react-app pemograman-web-lanjut-[npm-anda]  cd pemograman-web-lanjut-[npm-anda] |

1. Menambahkan library npm

Tambahkan library dibawah ini untuk menunjang pembuatan aplikasi messeger kedalam project react anda:

|  |
| --- |
| npm i react-router-dom bootstrap bootstrap-icons moment |

* library bootstrap : sebagai CSS Framework untuk mendesain layout UI
* library bootstrap-icons : sebagai CSS Framework untuk penggunaan icon
* library moment : diperuntukan toolkit JS menampilkan, manipulasi, konversi atau parsing tanggal dan waktu.
* library react-router-dom : diperuntukan menangani routing yang berfungsi sebagai navigasi antar halaman atau view dalam aplikasi

1. Membangun komponen
2. Struktur Folder (*Feature-based Structure*)

A screenshot of a computer

Description automatically generatedBuatlah folder *apps/*, *components/*, *media/*, *modules/* dan file *.env* pada project react yang telah anda buat.

Folder *apps* : diperuntukan untuk menyimpan component configurasi dalam bentuk helper, constanta ataupun routing.

Folder *components*: berisi component react yang diperuntukan bagi layouting UI pada aplikasi seperti, header, navigations, buttons, dan lain sebagainya.

Folder *media*: jika anda memilik style-sheet sendiri atau custome file css ataupun dalam bentuk image, video ataupun audia anda dapat menyimpannya di folder ini.

Folder *modules*: berisi komponen module atau paket, berguna untuk mengorganisir kode sebagai serangkaian paket atau modul yang dapat dikembangkan dan dikelola secara independen..

Sedangkan file *.env*: digunakan untuk menyimpan konfigurasi lingkungan yang sensitif dan spesifik untuk aplikasi, seperti kredensial database, secret keys, dan variabel lingkungan lain yang tidak seharusnya disimpan secara langsung dalam kode sumber karena alasan keamanan.

1. Customize file package.json

* Jika anda ingin mengubah port project server react anda update syntax berikut ini:

|  |
| --- |
| **OS Windows** |
| "scripts": {      "start": "set PORT=3004 && react-scripts start",      "build": "set PORT=3004 && react-scripts build",      "test": "react-scripts test",      "eject": "react-scripts eject"    }, |
| **OS UNIX (Mac, Linux)** |
| "scripts": {      "start": "export PORT=3004 && react-scripts start",      "build": "export PORT=3004 && react-scripts build",      "test": "react-scripts test",      "eject": "react-scripts eject"    }, |

* Menambahkan CSS untuk mengatur tampilan dan format UI secara global, pada contoh disini jika anda telah melakukan tahap II., yaitu menambahkan library bootstrap pada project anda maka bukalah file index.js anda dan tambahkan code dibawah ini:

|  |
| --- |
| import 'bootstrap/dist/css/bootstrap.min.css';  import 'bootstrap-icons/font/bootstrap-icons.css' |

Jika anda memiliki file CSS external lainnya anda dapat memanggil file tersebut dengan cara:

|  |
| --- |
| import './media/css/styles.css' |

Maka bentuk utuh dari file index.js anda akan sebagai berikut:

|  |
| --- |
| import React from 'react';  import ReactDOM from 'react-dom/client';  import reportWebVitals from './reportWebVitals';  import 'bootstrap/dist/css/bootstrap.min.css';  import 'bootstrap-icons/font/bootstrap-icons.css'  import './media/css/styles.css'  const root = ReactDOM.createRoot(document.getElementById('root'));  root.render(    <React.StrictMode>      <App />    </React.StrictMode>  ); |

1. Membuat komponen Layout

Dalam membangun sebuah website hal yang perlu diperhatikan ialah pembuatan layout dari aplikasi yang akan dibuat, hal ini merujuk pada konsep desain atau skema yang telah dibuat. Layout merupakan elemen dasar dari sebuah desain karena salah satu kunci dari keberhasilan sebuah desain adalah layout atau tata letak elemen desainnya. Dalam contoh kasus pembangunan sebuah aplikasi messager untuk menentukan tata letak elemen-elemen desain pada suatu bidang dalam media tertentu. Berikut ini adalah bentuk skema komponen destructuring fragment untuk menyusun atau menentukan tata letak media, modular, ataupun paket yang akan dibuat:

|  |
| --- |
| <LayoutUI>  <Headers> … </Headers>  <Section> … </Section>  <Footers> … </Footers>  </LayoutUI> |

Berikut ini ialah tahapan membangun sebuah layout pada aplikasi react dengan memanfaatkan teknik *parsing props* model *children*:

Buatlah file *React Component Function* bernama LayoutInit.js, Headers.js dan Footers.js pada folder components, dan masukan script sebagai berikut:

|  |
| --- |
| *Headers.js* |
| import React from 'react'  export default function Headers() {      return (          <div className="">              <h1 className="">IBI Kesatuan</h1>              <ul className="">                  <li className="">                      Home                  </li>                  <li className="">                      <span className=""></span>                  </li>                  <li className="">                      PWL                  </li>              </ul>          </div>      )  } |
| *Footers.js* |
| import React from 'react'  export default function Footers() {      return (          <div className="">              <div className="">                  <div className="">                      <span className="">2024 &copy;</span>                      <span className="">IBI Kesatuan Bogor</span>                  </div>              </div>          </div>      )  } |
| *LayoutInit.js* |
| import Footers from "./Footers";  import Headers from "./Headers";  const LayoutInit = ({ children }) => {    return (      <div className="">        <div className="">          <div className="">              <Headers />            <section className="">              <div className="">                {children}              </div>            </section>            <Footers />          </div>        </div>      </div>    );  };  export default LayoutInit; |

Pada folder *modules*, buatlah folder paket pertama yaitu chapter-1. Didalam folder tersebut buatlah file *React Component Class* bernama *ChapterOne.js* dan buat code seperti dibawah ini:

|  |
| --- |
| import React, { Component } from 'react'  export default class ChapterOne extends Component {    render() {      return (        <div>          <h1 className="text-center ">  Chapter One: The Beginning  </h1>        </div>      )    }  } |

Setelah anda berhasil membuat Layout website anda, bukalah file App.js dan tulislah code dibawah ini untuk memasang Layout yang telah anda buat kedalam DOM react router:

|  |
| --- |
| import React from "react";  import { BrowserRouter } from "react-router-dom";  import BaseRoute from "./apps/BaseRoute";  import LayoutInit from "./components/LayoutInit";  function App({ basename }) {    return (      <BrowserRouter>        <LayoutInit>          <BaseRoute />        </LayoutInit>      </BrowserRouter>    );  }  export default App; |

BrowserRouter: komponen di library react-router-dom yang digunakan untuk menangani routing di aplikasi web yang dibuat dengan React. Routing di sini berarti proses menentukan apa yang ditampilkan kepada pengguna berdasarkan URL di browser, yang sangat penting dalam pembuatan Single Page Applications (SPA).

Sumber: <https://v5.reactrouter.com/web/api/BrowserRouter>

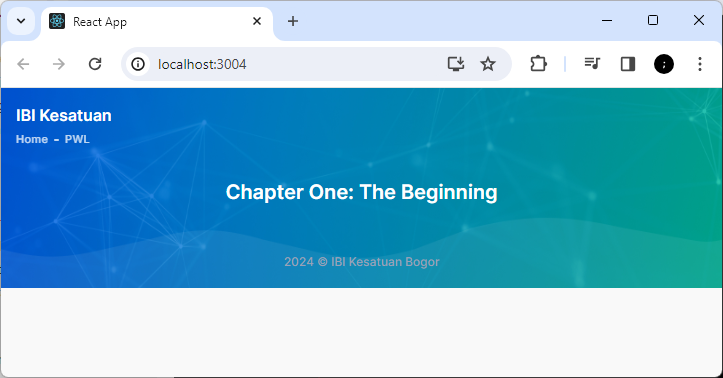
LayoutInit: merupakan import komponen dari file yang telah kita buat sebelumnya, ini menandakan bahwa seluruh page atau modular menggunakan konsep layout yang sama. Untuk memanggil komponen ini anda harus mengimport kan file tersebut terlebih dahulu.

BaseRoute: daftar komponen fundamental yang digunakan untuk menangani routing dalam aplikasi React. Buatlah file *React Functional Component* bernama BaseRoute.js di folder *apps*, untuk membuat daftar navigasi atau route apa saja yang akan dibuat:

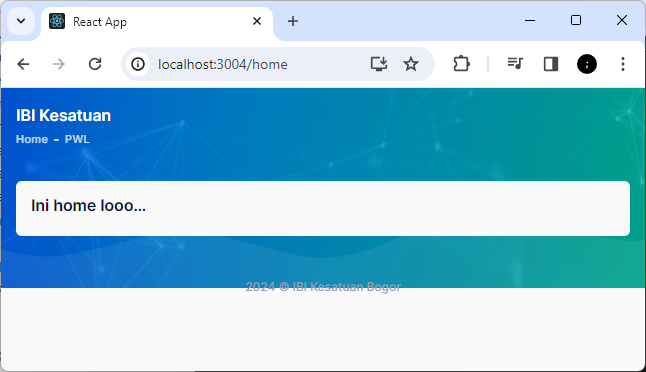
|  |
| --- |
| import React from "react";  import { Routes, Route } from "react-router-dom";  export default function BaseRoute() {    return (      <React.Suspense>        <Routes>          <Route index element={<ChapterOne />} />          <Route path="home" element={<Home />} />        </Routes>      </React.Suspense>    );  }  const Home = () => {    return <h3>Ini home looo…</h3>;  }; |

Tahap terakhir ialah menjalankan project react anda dengan baik didalam terminal anda seperti gambar dibawah ini dengan syntax `npm start`.

Ketika web aplikasi sudah berjalan maka akan melempar atau mengambil route dengan property *index*:

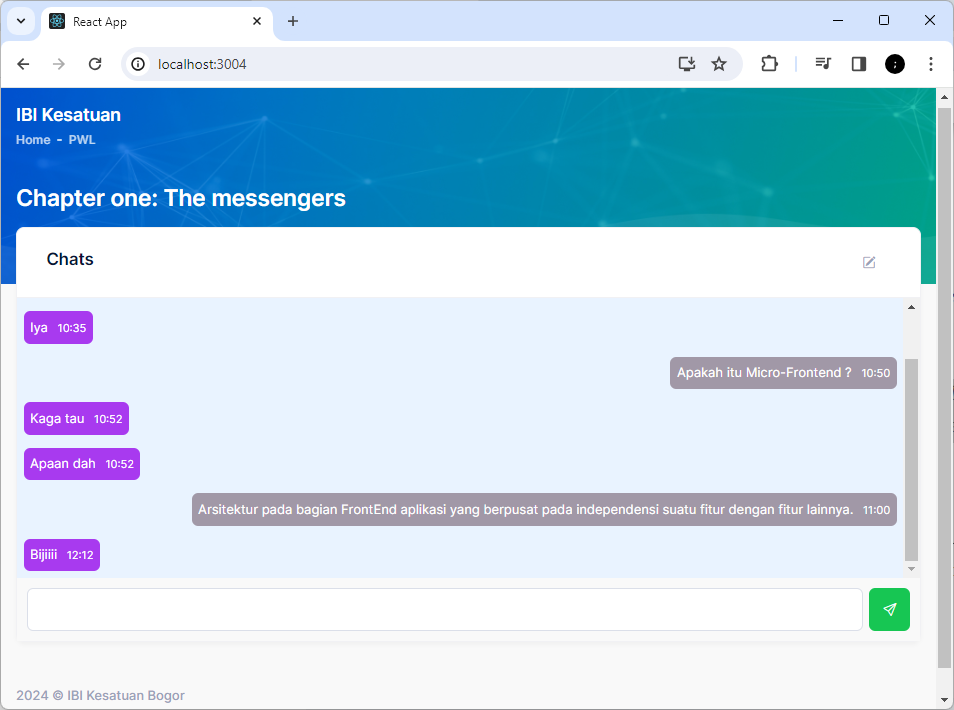


Sedangkan jika kita memasukan route path home, maka akan mengeksekusi komponen Home:



Note:  
Tampilan pada gambar diatas memiliki customize CSS, sehingga kemungkinan tampilan yang anda dapatkan tidak serupa namun hirarki dari fragment markup tetap sama. Oleh karenanya untuk tampilan desain CSS di bebaskan.

1. Membangun komponen modular Chat Messeger



Pada gambar diatas kita akan membangun sebuah chat messeger dengan react project dan memanfaatkan semua fitur yang disediakan. Disini kita akan membangun sebuah modular atau paket komponen pada section chapter-one. Oleh karenanya persiapkan markup html seperti gambar diatas.

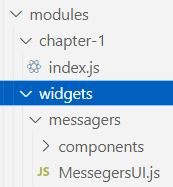
Pada RCC sebelumnya *ChapterOne*, silakan anda perbaharui scriptnya dengan memasukan atau meng-import RFC MessegerUI:

|  |
| --- |
| *ChapterOne.js* |
| import React, { Component } from 'react'  export default class ChapterOne extends Component {    render() {      return (        <div>          <h1 className='text-white mb-5'>  Chapter one: The messengers  </h1>        <MessegersUI />        </div>      )    }  } |

Komponen *MessagersUI* bisa anda ikuti tahapan dibawah ini:

1. **Modular Chat Messager**

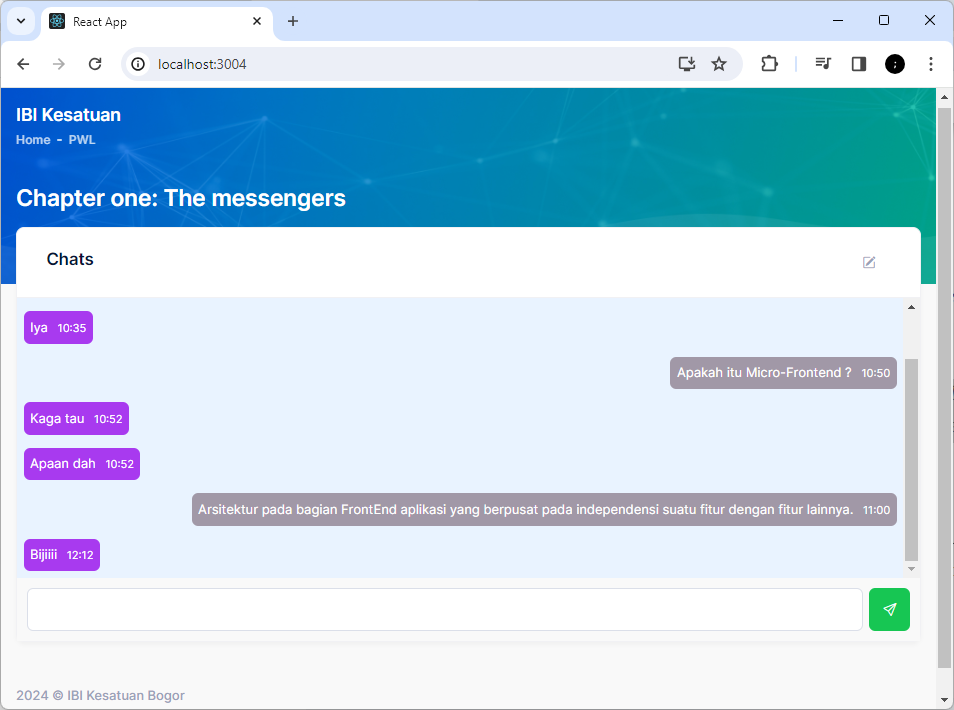
Untuk membangun sebuah modular biasanya akan memanfaatkan komponen RFC. Disini akan membuat sebuah folder bernama *widgets* didalam folder *modules*, didalamnya terdapat beberapa file sebagai berikut:



Folder paket *messagers* berfungsi untuk menampung atau menyimpan seluruh file komponen react yang dibutuhkan. Dimana paket tersebut memiliki sebuah RFC bernama MessagersUI.js yang merupakan susunan layout tata letak yang akan ditentukan, sedangkan folder components berisi file komponen yang diperlukan seperti bentuk komponen button, chat bubble dan lain-lain.

1. **Membangun Layout Messeger UI**

Jika melihat gambar dibawah ini maka sudah ditentukan bentuk layout dari paket atau widget chat messager memiliki tata letak sebagai berikut:



Maka dapat digambarkan bentuk hirarki dari Messeger UI sebagai berikut:

|  |
| --- |
| <Card>  <Card.header>  <Card.title> Chats </Card.title>  <Button> <icon-pencil /> </Button >  </Card.header>  <Card.body>  <Chat.messagers>  <Chat.messages>  <Chat.list-message>…</Chat.list-message>  </Chat.messages>  <Chat.form>  <input type=”text” />  <Button> <icon-send /> </Button >  </Chat.form>  </Chat.messagers>  </Card.body>  </Card> |

Setelah menentukan hirarki layout dari messagers, pada file MessegersUI.js perlu disesuaikan bentuk hirarki tersbut kedalam markup JSX. Disini akan memberikan contoh penggunaan Card yang dimiliki bootstrap (<https://getbootstrap.com/docs/5.3/components/card/#example>):

|  |
| --- |
| *MessegersUI.js* |
| import React from "react";  import { ButtonPrimary, ButtonSecondary } from "./components/ButtonUI";  export default function MessegersUI() {    return (      <div className="card">        <div className="card-header">          <h3 className="card-title align-items-start flex-column">            <span className="fw-bold mb-2 text-gray-900">Chats</span>          </h3>          <div className="card-toolbar">            <ButtonSecondary items={{              title: "Create new chat",              btn\_class: "btn-icon btn-clear",            }} >              <i className="bi bi-pencil-square"></i>            </ButtonSecondary>          </div>        </div>        <div className="card-body p-0">          <div            className="chat-message px-2 bg-light-primary"            style={StylesMessager.chatBox}          >  Kosongkan dahulu            ...          </div>          <div className="chat-send bg-light p-3">            <form method="post" autoComplete="off">              <div className="d-flex  justify-content-between  align-items-center">                <input type="text"  className="form-control me-2"  autoFocus={true} />                <ButtonPrimary items={{                  title: "Send",                  btn\_class: "btn-icon btn-success",                  type: "submit"                }}                >                  <i className="bi bi-send"></i>                </ButtonPrimary>              </div>            </form>          </div>        </div>      </div>    );  } |

Setelah mengatur tata letak, disini akan memberikan contoh penggunaan CSS Inline pada markup JSX. Jika anda lihat pada script dibawah terdapat kotak hijau yang berada pada markup JSX:



Style css inline pada property div diatas, mengubah markup style css inline menjadi bentuk object pada javascript. Oleh karenanya anda perlu membuat sebuah variable bernama StylesMessager dan key bernama chatBox:

|  |
| --- |
| const StylesMessager = {    chatBox: {      minHeight: "200px",      maxHeight: "45vh",      overflowY: "auto",    }  }; |

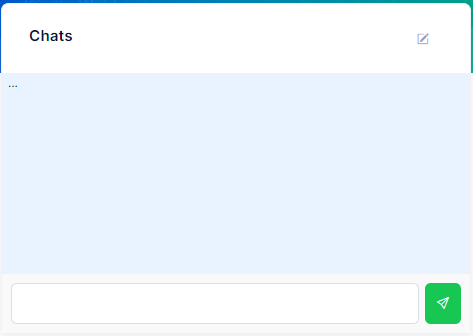
Jika anda melihat script RFC MessegerUI memiliki bentuk fragment JSX bernama ButtonPrimary dan ButtonSecondary:

|  |
| --- |
|  |
|  |

Fragment tersebut perlu anda buat dengan memanfaatkan RFC ES6 passing props, pada folder *components* buatlah file bernama ButtonUI.js

|  |
| --- |
| *ButtonUI.js* |
| const ButtonPrimary = ({ children, items, actions }) => {    return (      <button className={"btn btn-success " + items.btn\_class}  type="submit"  title={items.title}  onClick={actions}>        { children }      </button>    )  }  const ButtonSecondary = ({ children, items, actions }) => {    return (      <button className={"btn btn-clear " + items.btn\_class}  type="button"  title={items.title}  onClick={actions}>        { children }      </button>    )  }  export { ButtonPrimary, ButtonSecondary } |

Komponen ButtonUI yang terdiri dari RFC ES6 *ButtonPrimary* dan *ButtonSecondary* akan dijadikan sebagi bentuk komponen UI dalam bentuk button. Maka tampilan sementara akan seperti berikut:

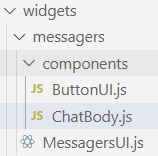


1. Penerapkan React Hook useState

Pada file MessegerUI.js kita tambahkan array data sebagai ilustrasi bentuk chat sebagai berikut:

|  |
| --- |
| *MessegerUI.js* |
| export default function MessagersUI() {  const chatArr = [      { id: 1,  message: "Hi",  from: "Febry",  date: "2024-02-22 10:30:00"  },      { id: 2,  message: "Iya",  from: "Isnan",  date: "2024-02-22 10:35:00"  },      { id: 3,  message: "Apakah itu Micro-Frontend ?",  from: "Febry",  date: "2024-02-22 10:50:00"  },      { id: 4,  message: "Kaga tau",  from: "Isnan",  date: "2024-02-22 10:52:00"  },      { id: 5,  message: "Apaan dah",  from: "Isnan",  date: "2024-02-22 10:52:00"  },      { id: 6,  message: "Arsitektur pada bagian FrontEnd aplikasi yang berpusat pada independensi suatu fitur dengan fitur lainnya.",  from: "Febry",  date: "2024-02-22 11:00:00"  },      { id: 7,  message: "Bijiiii",  from: "Isnan",  date: "2024-02-22 12:12:00"  },    ]    const [myChat, setMyChat] = useState(chatArr);  ……  } |

useState: Digunakan untuk menambahkan state ke dalam functional components. Memungkinkan komponen untuk memiliki state internal yang dapat diupdate.

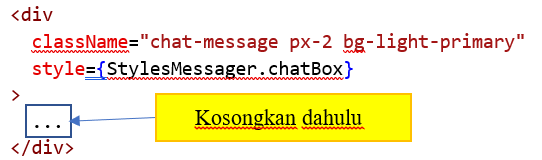
Dari code diatas data array objek ilustrasi percakapan akan disimpan kedalam state bernama setMyChat. Untuk mengakses atau membaca data array tersebut dapat menggunakan myChat. Untuk merender disini akan dibuatkan sebuah komponen RFC bernama ChatBody.js didalam folder components, komponen ini akan berisi daftar atau list dari array objek yang akan ditampilkan:

|  |
| --- |
| *ChatBody.js* |
| import moment from "moment";  import React from "react";  export default function ChatBody({ data }) {    const listdata = data;    return (      <div className="chat-items">        {listdata.map((v, index) => (          <div className="chat-item"  style={styleChatItems.chatBubleItems}  key={index}>            <div className="chat text-white rounded my-2 p-2"  style={ styleChatItems.chatBubleSender }>              <span className="me-3">{v.message}</span>              <span className="chat-date"  style={{ fontSize: "11px" }}>  {moment(v.date).format("HH:mm")}  </span>            </div>          </div>        ))}      </div>    );  } |

Untuk CSS style buble chat dapat menambahkan code seperti ini:

|  |
| --- |
| const styleChatItems = {    chatBubleItems: {      display: "flex",      flexDirection: "column",    },    chatBubleSender: {      textAlign: "right",      backgroundColor: "#a198a7",      alignSelf: "flex-end"    },    chatBubleReceiver: {      backgroundColor: "#a83aef",      alignSelf: "flex-start"    },  }; |

Selanjutnya anda perlu merubah code



Yang tadinya masih kosong dengan diinisialisasi menggunakan tanda (…), perlu anda ganti dengan cara memanggil fragment ChatBody dan perlu mengirimkan parameters myChat seperti:

|  |
| --- |
| <div className="chat-message px-2 bg-light-primary"      style={StylesMessager.chatBox} >     <ChatBody data={myChat} />  </div> |

Maka tampilannya akan seperti gambar dibawah ini:



Jika dilihat list daftar chat tersebut belum bisa membedakan mana yang atas nama akun sendiri atau orang lain. Oleh karenanya diperlukan logika operation untuk melakukan atau menyempurnakan desain yang telah ditentukan.

Bentuk operator logika sederhana yang bisa digunakan ialah dengan memanfaatkan kondisi IF…ELSE.. pada chat item. Untuk menentukan jika objek dari chat item atas nama Febry maka akan dianggap sebagai SENDER jika tidak akan diaanggap sebagai RECEIVER.

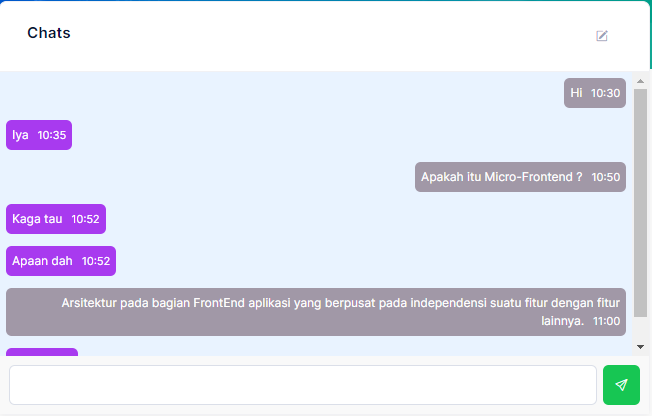
Pada script RFC ChatBody kita perlu menambahkan inisialisasi akun SENDER:

|  |
| --- |
| export default function ChatBody({ data }) {    const itsme = "Febry";  …………  } |

Ubahlah code chat-item sebagai berikut:

|  |
| --- |
| *Sebelum:* |
| <div className="chat text-white rounded my-2 p-2"  style={styleChatItems.chatBubleSender}>     <span className="me-3">{v.message}</span>     <span className="chat-date"  style={{ fontSize: "11px" }}>  {moment(v.date).format("HH:mm")}  </span>  </div> |
| *Sesudah:* |
| <div className="chat text-white rounded my-2 p-2"  style={  ((v.from === itsme) ?  Menambahkan SHORT IF  styleChatItems.chatBubleSender : styleChatItems.chatBubleReceiver  )}>     <span className="me-3">{v.message}</span>     <span className="chat-date"  style={{ fontSize: "11px" }}>  {moment(v.date).format("HH:mm")}  </span>  </div> |

Jika dijalankan maka akan menampilkan bentuk output sementara seperti gambar dibawah ini:



1. Penggunaan React Hook useRef

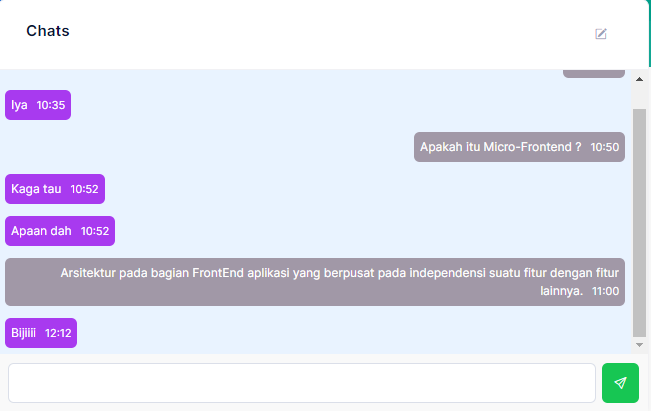
Jika anda lihat pada gambar output sementara diatas, posisi fokus chat selalu berada di atas atau menunjuk pada chat pertama. Jika ingin membuat posisi chat berada di scroll paling bawah atau menampilkan chat terbaru maka bisa memanfaatkan sebuah react hook bernama useRef.

*useRef* adalah sebuah Hook dalam React yang digunakan untuk mendapatkan referensi langsung ke elemen DOM atau instance komponen dalam functional components. Ini sering digunakan untuk mengakses elemen DOM secara langsung tanpa menggunakan state atau props, memungkinkan operasi seperti fokus pada input, pembacaan atau penulisan nilai, dan manipulasi DOM lainnya. *useRef* juga berguna untuk menyimpan nilai yang persisten selama umur (lifecycle) komponen tanpa menyebabkan komponen tersebut re-render ketika nilai tersebut berubah.

Kembali ke script RFC MessegersUI anda, tambahkan useRef seperti berikut ini:

|  |
| --- |
| *MessegersUI.js* |
| export default function MessegersUI() {  ……  const [myChat, setMyChat] = useState(chatArr);  const endOfMessagesRef = useRef(null);  const scrollToBottom = () => {      endOfMessagesRef.current?.scrollIntoView({  behavior: "smooth"  });  };  useEffect(() => {    scrollToBottom();  }, [myChat]);  return (  ……  <div            className="chat-message px-2 bg-light-primary"            style={StylesMessager.chatBox}          >            <ChatBody data={myChat} />            <div ref={endOfMessagesRef} />          </div>  ……  )  ……  } |

Setelah anda save dan periksa kembali hasil pekerjaan anda, apakah posisi chat sudah focus menampilkan data paling baru atau berada pada scrool paling bawah.



1. Menambahkan data chat baru dengan menggunakan action JSX

* Membuat react hook useState untuk menampung value chat dari textbox

Pada RFC MessegersUI tambahkan useState writeChat untuk menyimpan nilai baru yang berasal dari textbox:

|  |
| --- |
| import moment from "moment"; //tambahkan import moment  export default function MessegersUI() {  ……  const [writeChat, setWriteChat] = useState("");  ……  } |

Dan pada markup JSX textbox tambahkan inisialisasi value dan event onChange:

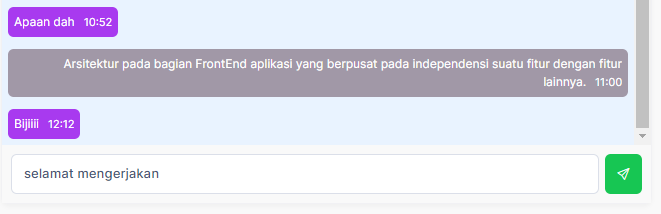
|  |
| --- |
| Sebelum: |
| <input type="text"  className="form-control me-2"  autoFocus={true}  /> |
| Sesudah: |
| <input type="text"  className="form-control me-2"  autoFocus={true}  value={writeChat}  onChange={(e) => setWriteChat(e.target.value)} /> |

* Menambahkan event handler submit pada chat form

Pada markup form JSX tambahkan event onSubmit *HandlerSendChat* dan buatlah fungsi ES6 untuk menangkap atau mengeksekusi perintah submit form:

|  |
| --- |
| …  const HandlerSendChat = (e) => {      e.preventDefault();      const objChat = {  id: 99,  message: writeChat,  from: "Febry",  date: moment().format('YYYY-MMM-DD HH:mm')  }      setMyChat([...myChat, objChat]);      setWriteChat("");  }  Contoh spread operator  … |
| <form method="post"  autoComplete="off"  onSubmit={(e) => HandlerSendChat(e)} >  …  </form> |

Script diatas merupakan syxtax untuk memberikan perintah ketika submit form akan menambahkan data object baru contoh:



Ketika kita menulis sebuah kalimat chat “selamat mengerjakan” setiap value atau character yang di ketikan akan disimpan kedalam useState, disini dinamakan setWriteChat(). Kalimat tersebut akan tersimpan pada variable bernama writeChat.

Namun ketika melakukan submit atau klik tombol SEND akan mengeksekusi pembuatan object baru dengan format isian:

const obj = {id:99, message “selamat mengerjakan”, from:”Febry}, date:”2024-02-25 16:16:00”}

dimana isian tersebut merujuk pada code

const objChat = { id: 99, message: writeChat, from: "Febry", date: moment().format('YYYY-MMM-DD HH:mm') }

function moment() merupakan library yang digunakan untuk mendapatkan waktu terhadap timezone saat ini.

Setelah membuat objek baru maka objek tersebut perlu ditambahkan kedalam baris array myChat. Untuk menambahkan data baru pada array object dapat menggunakan spread operator (…variable) seperti halnya code yang telah anda buat:

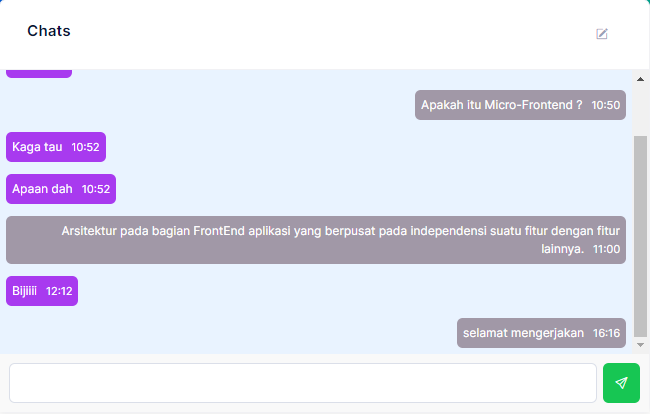
setMyChat([...myChat, objChat]);

Menambahkan baris objek baru kedalam array objek milik variable myChat

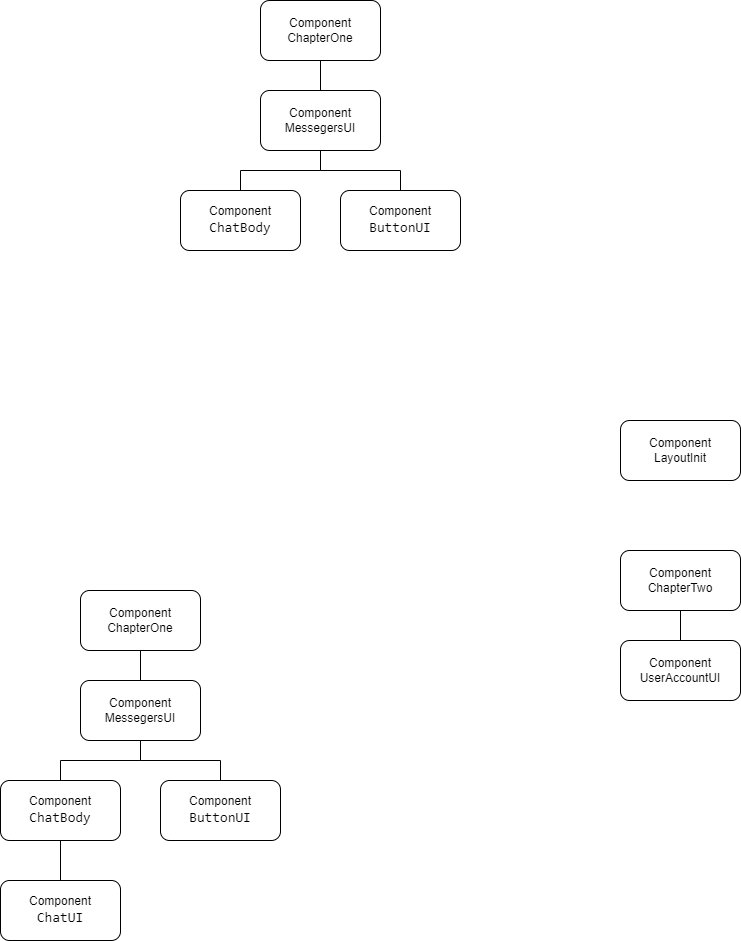
setWriteChat("");

Mengubah isian variable writeChat menjadi kosong

1. Final result

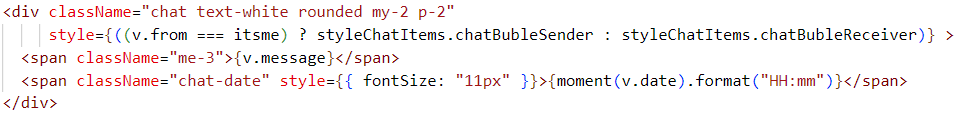


1. Ilustrasi UI sebagai Pohon Komponen Micro-frontend



1. **Tugas Pelatihan**
2. Dari latihan pembuatan chat messegers, ubahlah script pembuatan chat bubble kedalam bentuk RFC ES6 untuk menentukan mana chat SENDER dan chat RECEIVER. Sehingga dapat menjadi clean code (tidak ada selection di dalam komponen).

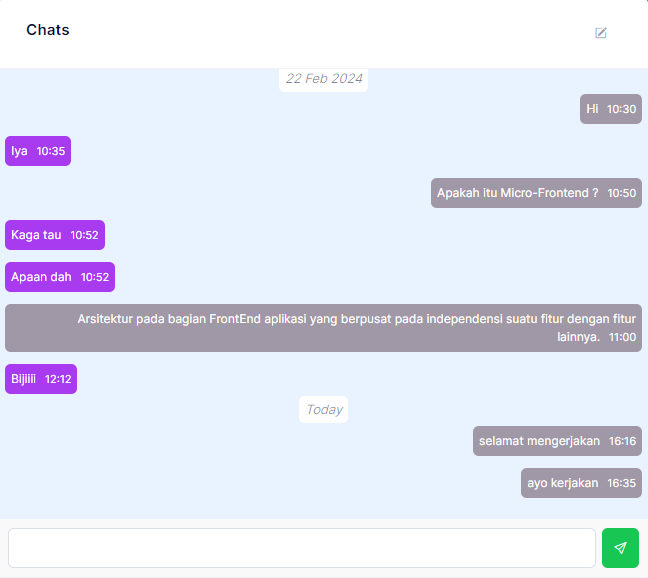
Sebelum:



Sesudah:

|  |
| --- |
| {listdata.map((v, index) => (  <div key={index}>  <ChatBubleItem data={v}>  <span className="me-3">{v.message}</span>  <span className="chat-date"  style={{ fontSize: "11px" }}>  {moment(v.date).format("HH:mm")}  </span>  </ChatBubleItem>  </div>  ))} |

1. Tampilkan tanggal history chat dan urutkan berdasarkan tanggal release chat tersebut seperti gambar dibawah ini:



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 anda ke @*IrvanRizkyAriansyah* dan *@FebryFairuz*