

BAB III

HOOKS dan ROUTER DOM

1. Materi Pembelajaran

A. Hooks

Hooks diperkenalkan di React 16.8 untuk memungkinkan penggunaan state dan fitur-fitur React lainnya dalam komponen fungsi tanpa perlu menulis kelas. Hooks menawarkan solusi yang lebih sederhana dan lebih intuitif untuk menggunakan berbagai fitur React dalam komponen yang bersifat fungsi. Berikut adalah beberapa fitur utama dari hooks:

1) **useState:**

Hook menambahkan *state* ke dalam komponen fungsi. Sebelumnya, *state* hanya bisa digunakan dalam komponen kelas. *useState* memberikan pasangan nilai: *state* saat ini dan fungsi yang memperbarui *state* tersebut.

```
import React, { useState } from "react";

const [username, setUsername] = useState();
const [password, setPassword] = useState();
```

2) **useEffect:**

Mirip dengan *lifecycle methods componentDidMount*, *componentDidUpdate*, dan *componentWillUnmount* dalam komponen kelas, *useEffect* memungkinkan melakukan efek samping dalam komponen fungsi. Efek samping ini bisa berupa data *fetching*, *subscriptions*, atau secara manual mengubah DOM dari React.

```
import React, { useEffect } from "react";

useEffect(() => {
  setUsername("Febry");
}, [selectedChat]);
```

3) **useContext:**

Hook ini memungkinkan untuk "menggunakan" dan berlangganan ke React context tanpa perlu menggunakan komponen *wrapper* atau *Consumer*. Ini memudahkan akses ke data global seperti tema, preferensi bahasa, atau data autentikasi.

```
import { createContext, useContext } from 'react';

const ThemeContext = createContext(null);

export default function MyApp() {
  return (
    <ThemeContext.Provider value="dark">
      <Form />
    </ThemeContext.Provider>
  )
}
```

```
function Form() {
  return (
    <Panel title="Welcome">
      <Button>Sign up</Button>
      <Button>Log in</Button>
    </Panel>
  );
}

function Panel({ title, children }) {
  const theme = useContext(ThemeContext);
  const className = 'panel-' + theme;
  return (
    <section className={className}>
      <h1>{title}</h1>
      {children}
    </section>
  )
}

function Button({ children }) {
  const theme = useContext(ThemeContext);
  const className = 'button-' + theme;
  return (
    <button className={className}>
      {children}
    </button>
  );
}
```

Welcome

Sign up

Log in

4) **useReducer:**

useReducer adalah alternatif untuk *useState* yang lebih cocok untuk state dengan logika yang kompleks atau ketika state berikutnya bergantung pada state sebelumnya. *useReducer* juga memudahkan pengujian unit dari *reducers* karena mereka biasanya pure functions.

```
import { useReducer } from 'react';

function reducer(state, action) {
  if (action.type === 'incremented_age') {
    return {
      age: state.age + 1
    };
  }
  throw Error('Unknown action.');
```

```
export default function Counter() {
  const [state, dispatch] = useReducer(reducer, { age: 42 });

  return (
    <>
      <button onClick={() => {
        dispatch({ type: 'incremented_age' })
      }}>
        Click me to add your age
      </button>
      <p>Hello! You are {state.age}</p>
    </>
  );
}
```

Click me to add your age

Hello! You are 44.

5) useCallback:

Hook ini digunakan untuk menghafal *callback*. Ini berguna ketika *callback* diteruskan ke komponen yang dioptimasi dengan *React.memo*, karena membantu mencegah rendering yang tidak perlu dengan memastikan bahwa fungsi tidak dibuat ulang di setiap render.

```
import { useCallback } from 'react';

export default function ProductPage({ productId, referrer, theme }) {
  const handleSubmit = useCallback((orderDetails) => {
    post('/product/' + productId + '/buy', {
      referrer,
      orderDetails,
    });
  }, [productId, referrer]);
}
```

6) useMemo:

Mirip dengan *useCallback*, *useMemo* digunakan untuk menghafal nilai. Ini berguna untuk operasi perhitungan yang kompleks. Dengan *useMemo*, React akan menghafal hasil perhitungan tersebut dan hanya mengulanginya jika salah satu dependensinya berubah.

```
import { useMemo } from 'react';
import { filterTodos } from './utils.js'

export default function TodoList({ todos, theme, tab }) {
  const visibleTodos = useMemo(
    () => filterTodos(todos, tab),
    [todos, tab]
  );
  return (
    <div className={theme}>
```

`<p>Note: filterTodos is artificially slowed down!</p>`

``

`{visibleTodos.map(todo => (`

`<li key={todo.id}>`

`{todo.completed ?`

`<s>{todo.text}</s> :`

`todo.text`

`}`

``

`))}`

``

`</div>`

`);`

`}`

All Active Completed

☐ Dark mode

Note: `filterTodos` is artificially slowed down!

- Todo 1
- Todo 2
- Todo 3
- Todo 4
- ~~Todo 5~~
- ~~Todo 6~~
- ~~Todo 7~~
- ~~Todo 8~~
- Todo 9

7) useRef:

`useRef` mengembalikan objek ref yang dapat digunakan untuk menyimpan nilai yang bertahan selama masa hidup penuh dari komponen, seperti referensi ke elemen DOM atau nilai yang tidak ingin diulangi pada setiap render.

`import { useRef } from 'react';`

`export default function Counter() {`

`let ref = useRef(0);`

`function handleClick() {`

`ref.current = ref.current + 1;`

`alert('You clicked ' + ref.current + ' times!');`

`}`

`return (`

`<button onClick={handleClick}>`

`Click me!`

`</button>`

`);`

`}`

Click me!

...e at 786946de.sandpack-bundler-4bw.pages.dev says

You clicked 4 times!

OK

8) useImperativeHandle:

Hook ini digunakan bersama dengan `forwardRef` untuk memodifikasi instance yang diekspos kepada komponen induk ketika menggunakan ref. Ini adalah cara untuk mengekspos fungsi spesifik ke komponen induk sambil menyembunyikan detail implementasi internal.

`import { forwardRef, useImperativeHandle } from 'react';`

`const MyInput = forwardRef(function MyInput(props, ref) {`

```
useImperativeHandle(ref, () => {
  return {
    // ... your methods ...
  };
}, []);
// ...
```

9) **useLayoutEffect:**

Hampir identik dengan *useEffect*, tetapi dipicu secara sinkron setelah semua perubahan DOM terjadi. Hook ini berguna untuk membaca layout dari DOM dan melakukan perubahan sinkron (misalnya, mengukur ukuran elemen).

```
import { useRef, useLayoutEffect, useState } from 'react';
import { createPortal } from 'react-dom';
import TooltipContainer from './TooltipContainer.js';

export default function Tooltip({ children, targetRect }) {
  const ref = useRef(null);
  const [tooltipHeight, setTooltipHeight] = useState(0);

  useLayoutEffect(() => {
    const { height } = ref.current.getBoundingClientRect();
    setTooltipHeight(height);
    console.log('Measured tooltip height: ' + height);
  }, []);

  let tooltipX = 0;
  let tooltipY = 0;
  if (targetRect !== null) {
    tooltipX = targetRect.left;
    tooltipY = targetRect.top - tooltipHeight;
    if (tooltipY < 0) {
      // It doesn't fit above, so place below.
      tooltipY = targetRect.bottom;
    }
  }

  return createPortal(
    <TooltipContainer x={tooltipX} y={tooltipY} contentRef={ref}>
      {children}
    </TooltipContainer>,
    document.body
  );
}
```

Hover over me (tooltip above)

This tooltip fits above the button

Hover over me (tooltip below)

Hover over me (tooltip below)

10) useDebugValue:

Hook ini bisa digunakan untuk menampilkan label untuk *custom hooks* dalam React DevTools, memudahkan debugging.

```
import { useDebugValue } from 'react';

function useOnlineStatus() {
  // ...
  useDebugValue(isOnline ? 'Online' : 'Offline');
  // ...
}
```

Hooks meningkatkan kemampuan dan keluwesan komponen fungsi dengan memberikan akses ke fitur-fitur yang dulunya hanya terbatas pada komponen kelas. Dengan menggunakan hooks, pengembang dapat menulis komponen yang lebih bersih dan lebih modular dengan lebih mudah.

B. Router Dom

React Router DOM adalah pustaka yang digunakan untuk menangani routing dalam aplikasi web berbasis React. Ini tidaklah menjadi bagian dari fitur resmi React JS, tetapi merupakan pustaka eksternal yang dibuat khusus untuk menangani navigasi dan routing dalam aplikasi React. Namun, karena sangat umum digunakan dalam pengembangan aplikasi web React, sering kali dianggap sebagai bagian dari ekosistem React.

```
npm install react-router-dom
```

Untuk menggunakan router pada website anda, pada file App.js anda perlu menambahkan BrowserRouter sebagai berikut:

```
import React from "react";
import { BrowserRouter } from "react-router-dom";

function App() {
  return (
    <BrowserRouter>
      {/* your app goes here */}
    </BrowserRouter>
  );
}

export default App;
```

- 1) **Routing:** React Router DOM memungkinkan menentukan rute-rute aplikasi, yang memungkinkan navigasi antara berbagai komponen atau tampilan dalam aplikasi berbasis React. Ini memungkinkan untuk membuat aplikasi dengan navigasi multi-halaman seperti yang sering kita lihat dalam pengembangan web.

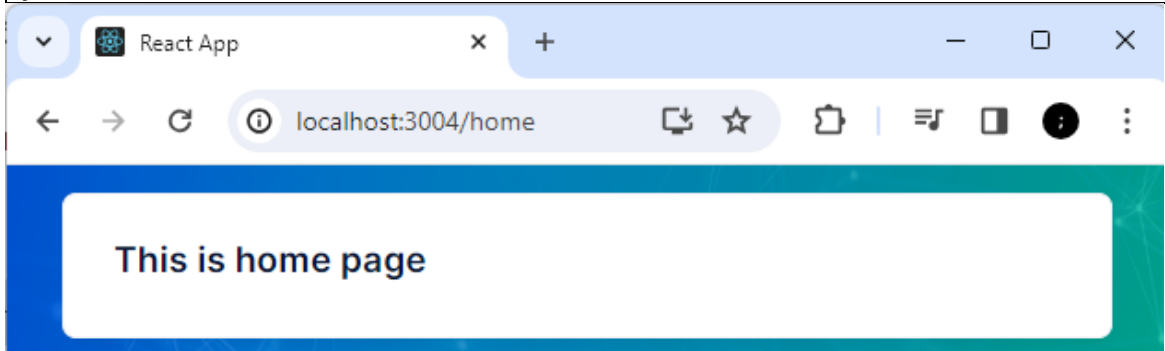
```
import React from "react";
```

```
import { BrowserRouter, Routes, Route } from "react-router-dom";

function App() {
  return (
    <BrowserRouter>
      <React.Suspense>
        <Routes>
          <Route path="home" element={<Home />} />
        </Routes>
      </React.Suspense>
    </BrowserRouter>
  );
}

export default App;

const Home = () =>{
  return <h1>This is home page</h1>
}
```



- 2) **Nested Routing:** dapat menentukan rute-rute bersarang di dalam rute lainnya, sehingga memungkinkan pembuatan tata letak dan navigasi yang kompleks. Ini berguna ketika memiliki hierarki tampilan dalam aplikasi, di mana satu komponen dapat memiliki rute-rute sendiri.

```
import { BrowserRouter, Routes, Route, Outlet } from "react-router-dom";

function App() {
  return (
    <BrowserRouter>
      <React.Suspense>
        <Routes>
          <Route path="home" element={<Home />} />
          <Route path='profile' element={<Profile />} >
            <Route path="identity" element={<Identity />} />
            <Route path="config" element={<Config />} />
          </Route>
        </Routes>
      </React.Suspense>
    </BrowserRouter>
  );
}
```

```

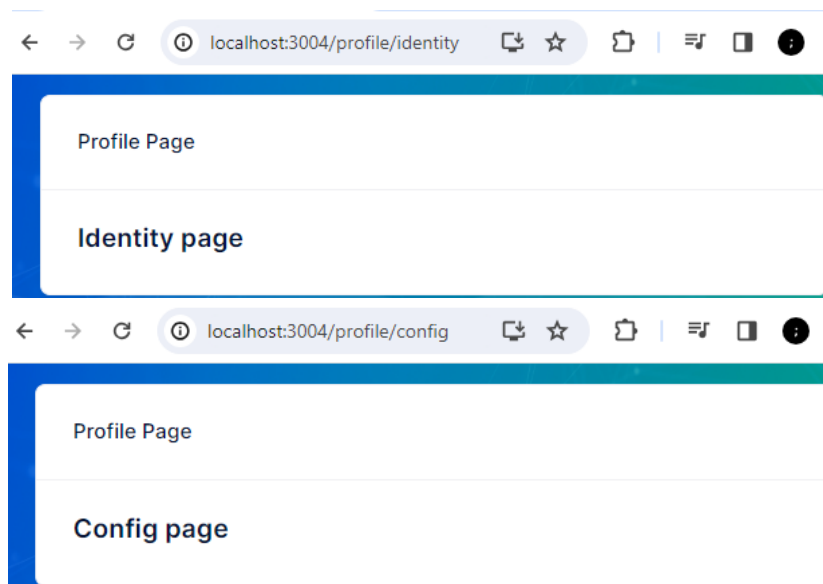
    );
  }

  const Profile = () => {
    return (
      <div className="card">
        <div className="card-header">
          <h3 className="card-title">
            Profile Page
          </h3>
        </div>
        <div className="card-body">
          <Outlet />
        </div>
      </div>
    )
  }

  const Identity = () =>{
    return <h1>Identity page</h1>
  }

  const Config = () =>{
    return <h1>Config page</h1>
  }

```



- 3) **Link dan NavLink Components:** React Router DOM menyediakan komponen Link dan NavLink yang memungkinkan navigasi antara rute-rute tanpa perlu memuat ulang halaman. Ini membuat navigasi dalam aplikasi React terasa lebih responsif dan halus.

Link Components:

```
import { Link } from "react-router-dom";
```

```

const Identity = () =>{
  return (
    <div>
      <h1>Identity page</h1>
      <Link to={"home"}>Go To Home Page</Link>
    </div>
  )
}

```

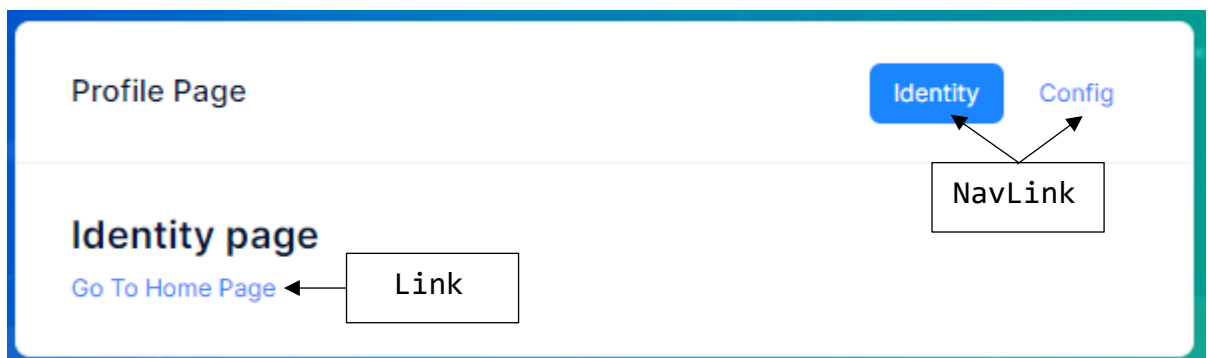


```
)
}
```

NavLink Components:

```
import { NavLink } from "react-router-dom";
```

```
const Profile = () => {
  return (
    <div className="card my-5 mx-10">
      <div className="card-header">
        <h3 className="card-title">Profile Page</h3>
        <div className="card-toolbar">
          <ul className="nav nav-pills">
            <li className="nav-item">
              <NavLink className="nav-link" to={"identity"} >
                Identity
              </NavLink>
            </li>
            <li className="nav-item">
              <NavLink className="nav-link" to={"config"} >
                Config
              </NavLink>
            </li>
          </ul>
        </div>
      </div>
      <div className="card-body">
        <Outlet />
      </div>
    </div>
  )
}
```

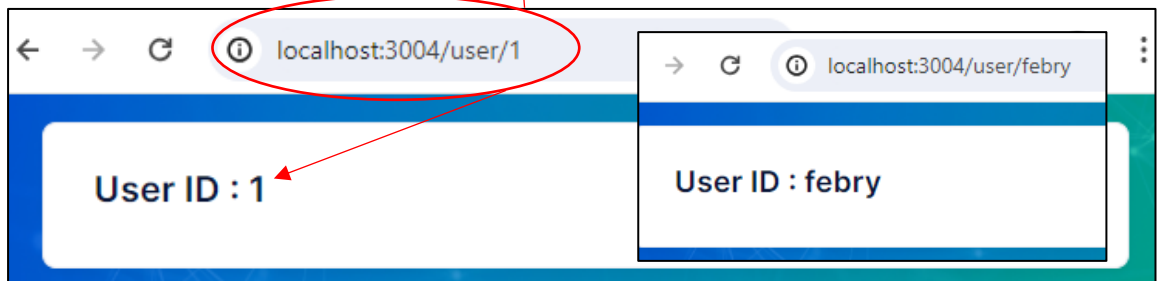


- 4) **Router Component:** Dengan menggunakan komponen Route, dapat menentukan komponen-komponen mana yang harus ditampilkan ketika URL sesuai dengan pola rute yang diberikan. Ini memungkinkan untuk mengaitkan komponen-komponen dengan rute-rute tertentu dalam aplikasi .

- 5) **Parameter dan Query Parsing:** React Router DOM memungkinkan menangkap parameter dan query string dari URL, sehingga memungkinkan membuat rute-rute dinamis yang merespons perubahan pada URL. Menggunakan library bernama useParams:

```
import { useParams } from "react-router-dom";
```

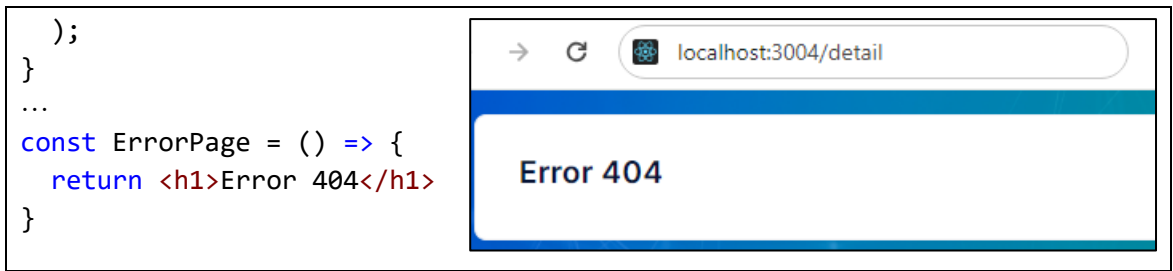
```
function App() {
  return (
    <BrowserRouter>
      <React.Suspense>
        <Routes>
          ...
          <Route path="user/:user_id" element={<User />} />
        </Routes>
      </React.Suspense>
    </BrowserRouter>
  );
}
...
const User = () =>{
  let params = useParams();
  return <h1>User ID: {params.user_id}</h1>
}
```



- 6) **Navigate Components:** Komponen Navigate memungkinkan untuk mengarahkan pengguna ke rute lain secara dinamis yang ditampilkan pada satu waktu:

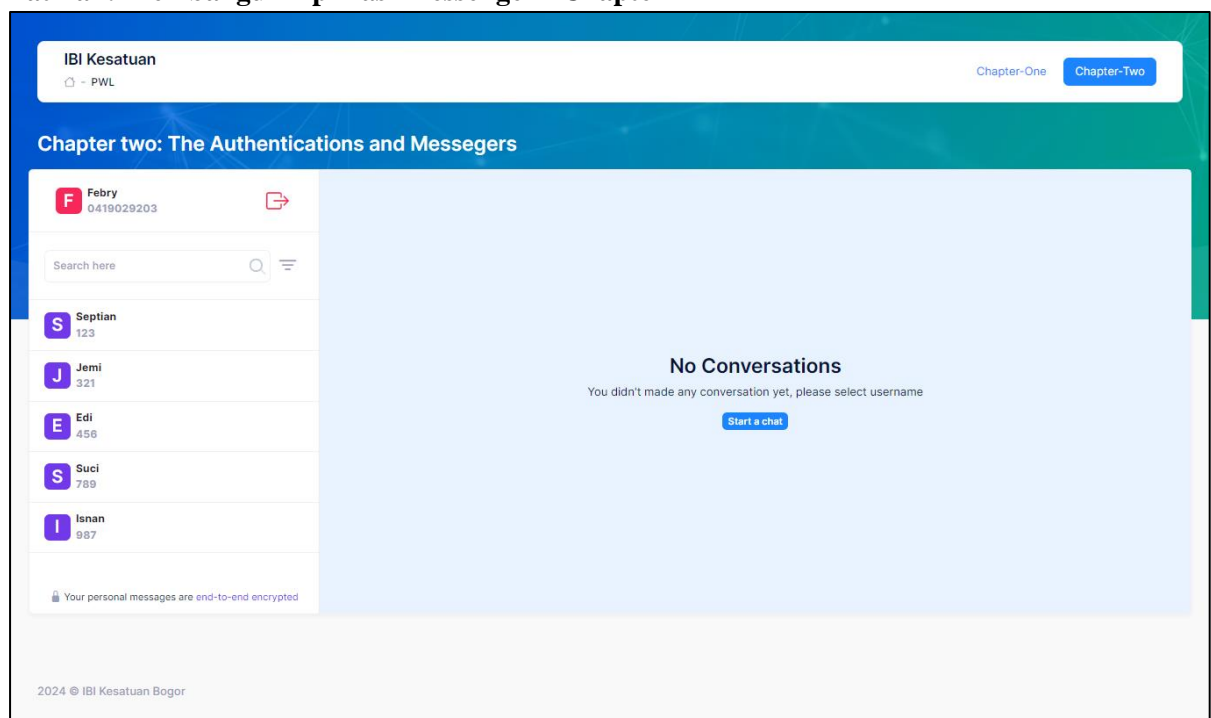
```
import { Navigate } from "react-router-dom";

function App() {
  return (
    <BrowserRouter>
      <React.Suspense>
        <Routes>
          ...
          <Route path="detail" element={<Navigate to="/error" />} />
          <Route path="error" element={<ErrorPage />} />
        </Routes>
      </React.Suspense>
    </BrowserRouter>
  );
}
```

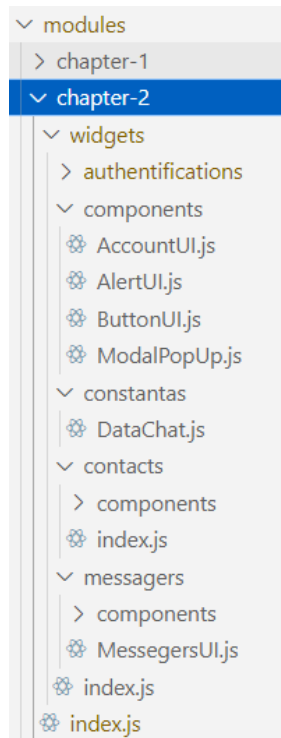


Secara keseluruhan, React Router DOM merupakan salah satu alat yang paling penting dalam pengembangan aplikasi web React karena menyediakan cara yang kuat dan terstruktur untuk menangani navigasi dan routing.

2. Latihan: Membangun Aplikasi Messenger - Chapter 2



Pada pembahasan sebelumnya telah dibuat skema rancangan awal bentuk Layout dari sebuah MessengersUI, dimulai melakukan retrieve chat, hingga menambah atau membuat chat baru. Pada bagian ke dua ini akan membahas bagaimana mengimplementasikan sebuah bentuk chat berdasarkan kontak pengguna. Pada bagian akan mengilustrasikan sebuah struktur folder project dari react framework sebagai berikut:



Karna kita akan membuat sebuah module baru bernama chapter-2, react function component dibuat bernama index.js yang sejajar dengan folder `chapter-2/widgets`. Pada component tersebut disini kita perlu memetakan layout dari masing-masing widgets. Karna kita akan memiliki dua buah widgets terpisah yaitu widget kontak user dan widget messengers, maka pada RFC ChapterTwo (index.js) memiliki bentuk code sebagai berikut:

index.js

```
import React from "react";

export function ChapterTwo() {

  return (
    <div id="chapter-2">
      <h1 className="text-white mb-5">
        Chapter two: The Authentications and Messengers
      </h1>
      <div className="px-3">
        <div className="row">
          <div className="col-2 col-lg-3 col-xxl-4 px-0">
            (Daftar kontak disini)
          </div>
          <div className="col-10 col-lg-9 col-xxl-8 px-0">
            (Messeging disini)
          </div>
        </div>
      </div>
    </div>
  );
}
```

Chapter two: The Authentications and Messengers

Daftar kontak disini

Messeging disini

- I. Membuat data dummy untuk daftar kontak dan daftar chat
Pada folder constantas buatlah sebuah file bernama **DataChat.js** dimana berisi sebuah array objek yang berisi daftar kontak dan chat:

```
const Chat_1 = [
  { id: 1, from_id: "0419029203", messages: "P", date: "2024-02-22 10:30:00", to_user_id: "123" },
  { id: 2, from_id: "123", messages: "Ke pasar beli mangga", date: "2024-02-22 10:30:00", to_user_id: "0419029203" },
  { id: 3, from_id: "123", messages: "Tapi lupa untuk dibawa", date: "2024-02-22 10:30:00", to_user_id: "0419029203" },
  { id: 4, from_id: "0419029203", messages: "Eeeaaaa", date: "2024-02-22 10:45:00", to_user_id: "123" },
  { id: 5, from_id: "123", messages: "Sedang semangat kejar cinta", date: "2024-02-22 10:53:00", to_user_id: "0419029203" },
  { id: 6, from_id: "123", messages: "Tapi dia dengan orang yang buat luka", date: "2024-02-22 10:53:00", to_user_id: "0419029203" }
]

const Chat_2 = [
  { id: 1, from_id: "321", messages: "Nanya dong", date: "2024-02-22 20:10:00", to_user_id: "0419029203" },
  { id: 2, from_id: "0419029203", messages: "Kenapa?", date: "2024-02-22 20:15:00", to_user_id: "321" },
  { id: 3, from_id: "321", messages: "Singkong yang difermentasi itu namanya apa ?", date: "2024-02-22 20:15:00", to_user_id: "0419029203" },
  { id: 4, from_id: "321", messages: "Cape ya..", date: "2024-02-22 20:10:00", to_user_id: "0419029203" },
  { id: 5, from_id: "0419029203", messages: "Steak yang kematangan itu disebut apa ya ?", date: "2024-02-22 20:10:00", to_user_id: "321" },
  { id: 6, from_id: "0419029203", messages: "We're done ya..", date: "2024-02-22 20:10:00", to_user_id: "321" }
]

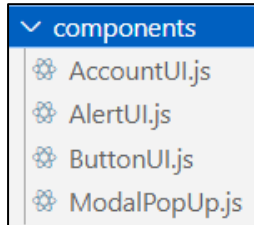
const Messengers = [{ chat_id: 1, user_id: "123", messages: Chat_1 },
{ chat_id: 2, user_id: "321", messages: Chat_2 }];

const MyFriend = [
  { id: 3, user_id: "123", name: "Septian" },
  { id: 1, user_id: "321", name: "Jemi" },
  { id: 2, user_id: "456", name: "Edi" },
  { id: 4, user_id: "789", name: "Suci" },
  { id: 5, user_id: "987", name: "Isnan" },
  { id: 6, user_id: "585", name: "Anton" }
]

export { Messengers, MyFriend };
```

II. Membangun Atomic Desain UI komponen

Atomic design adalah pendekatan dalam merancang sistem desain UI yang diorganisir berdasarkan tingkat kompleksitas komponen, mulai dari bagian-bagian yang paling kecil hingga menjadi elemen yang lebih besar dan kompleks. Pendekatan ini diperkenalkan oleh Brad Frost. Atomic design sering digunakan untuk merancang dan mengembangkan komponen UI.



Pada contoh kasus disini kita akan membangun bentuk atomic desain ui secara sederhana dan tidak kompleks. Komponen atomic yang perlu disiapkan ialah AccountUI, AlertUI, ButtonUI, dan ModalPopUp, penggunaan komponen tersebut nantinya kita akan gunakan dalam membangun sebuah rancangan aplikasi messeging ini, jika anda ingin menambahkan komponen bentuk ui baru anda dapat membuatnya di folder components.

Atomic AccountUI

```
const AccountLong = ({ data, color }) => {
  return (
    <div className="my-profile d-flex align-items-center">
      <div className="d-flex flex-stack">
        <div className="symbol symbol-30px">
          <div className={`symbol-label fs-2 fw-bold bg-${color} text-inverse-${color}`}>
            {data.name ? data.name.charAt(0) : "-"}
          </div>
        </div>
      </div>
      <div className="text-dark fw-bolder fs-7 ms-2 text-right">
        <span className="d-block">{data.name}</span>
        <span className="text-muted">{data.user_id ? data.user_id :
data.id}</span>
      </div>
    </div>
  );
};

export { AccountLong };
```

Atomic AlertUI

```
import React, { useState } from "react";

const states = {
  setState: null,
  changeState(data) {
    if (!this.setState) return;
    this.setState((prevData) => {
      return {
        ...prevData,
        ...data,
      };
    });
  }
};
```

```

    },
  };
  const AlertNotif = ({ color, message, icon }) => {
    const [data, setData] = useState({
      tipe: (color) ? color : "danger",
      message: (message) ? message : "message",
      icon: (icon) ? icon : "bi-exclamation-triangle",
    });

    states.setState = setData;

    return (
      <div
        className={
          "notice d-flex bg-light-" +
          data.tipe +
          " rounded border-" +
          data.tipe +
          " border border-dashed mb-9 p-6 "
        }
        id="alert-info"
      >
        <span className="me-4">
          <i className={"fs-1 text-" + data.tipe + " bi " +
data.icon}></i>
        </span>
        <div className="d-flex flex-stack flex-grow-1">
          <div className="fw-bold">
            <div className="fs-6 text-gray-
700">{data.message}</div>
          </div>
        </div>
      </div>
    );
  }

  export {AlertNotif}

```

Atomic ButtonUI

```

import { OverlayTrigger, Tooltip } from "react-bootstrap"

const ButtonIcon = ({ children, color, title, position="top", onAction })
=> {
  return (
    <OverlayTrigger delay={{ show: 250, hide: 400 }}
      placement={position} overlay={renderTooltip({ title: title })}>

```

```
        <button className={"btn btn-icon " + color} type="button"
onClick={onAction}>
            {children}
        </button>
    </OverlayTrigger>
)
}

const renderTooltip = (props) => {
    return (
        <Tooltip id="button-tooltip" {...props}>
            {props.title}
        </Tooltip>
    );
};

const ButtonPrimary = ({ children, items, actions }) => {
    return (
        <button className={"btn " + items.btn_class} type={items.type ?
items.type : "button"} 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>
    )
}

const ButtonSearch = ({ children, setSearch }) => {
    return (
        <div className="w-100 d-flex align-items-center mb-3 mb-lg-0 bg-white
border rounded">
            <input type="text" className='form-control form-control-sm form-
control-flush' placeholder='Search here' onChange={(e) =>
setSearch(e.target.value)} />
            {children}
        </div>
    )
}

export { ButtonPrimary, ButtonSecondary, ButtonSearch, ButtonIcon }
```


Atomic ModalPopUp

```
import React, { useState } from "react";
import { Modal } from "react-bootstrap";

const states = {
  setState: null,
  changeState(data) {
    if (!this.setState) return;
    this.setState((prevData) => {
      return {
        ...prevData,
        ...data,
      };
    });
  },
};

const handleClose = () => {
  states.changeState({
    open: false,
  });
};

const ModalPopUp = () => {
  const [data, setData] = useState({
    open: false,
    header: "ini header default",
    message: "ini message default",
    size: "md",
    footer: "",
    onClose: handleClose,
  });

  states.setState = setData;

  return (
    <Modal show={data.open} onHide={data.onClose} size={data.size}>
      <Modal.Header closeButton>
        <h3 className="modal-title">{data.header}</h3>
      </Modal.Header>
      <Modal.Body>{data.message}</Modal.Body>
      {(data.footer) ? (
        <Modal.Footer>
          <button onClick={data.onClose} className="btn btn-secondary px-10" >No</button>
          {data.footer}
        </Modal.Footer>
      ) : ''}
    </Modal>
  );
};
```

```

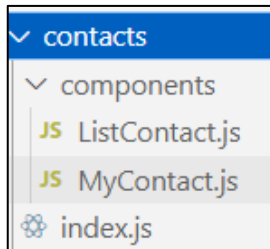
    </Modal>
  );
};

const openModal = ({ open=true, message, header, size, footer, onClose =
() => {} }) => {
  states.changeState({
    message,
    header,
    size,
    open,
    footer,
    onClose: () => {
      onClose();
      handleClose();
    },
  });
};

export default ModalPopUp;
export { openModal };

```

III. Developing widget kontak



Pada folder contacts kita akan membuat sebuah RFC untuk membangun layout dari widget kontak dengan nama file index.js (ContactUI). Sedangkan folder components memiliki dua buah file bernama RFC ListContact dan RFC MyContact. Kedua file tersebut merupakan komponen atomic desain ui yang diperuntukan dalam pembangunan sebuah widget kontak yang berpusat pada ContactUI. Berikut adalah implementasi dari code untuk ContactUI:

A. Membangun contact UI sebagai pondasi layout:

index.js

```

import React, { useMemo, useState } from "react";
import { AccountLong } from "../components/AccountUI";
import { ButtonIcon, ButtonSearch } from "../components/ButtonUI";
import { Link } from "react-router-dom";

export function ContactUI({ my_account, friends }) {
  const my_friends = friends;

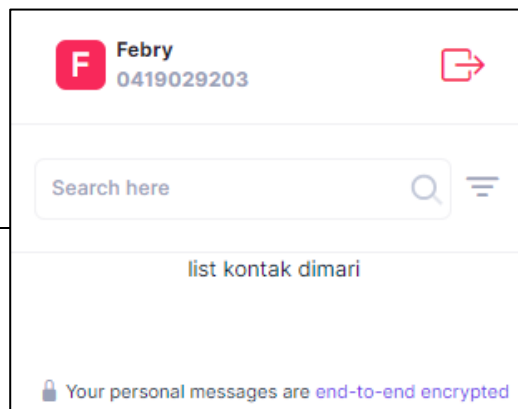
  return (
    <div className="card card-flush h-100 mb-5 mb-xl-10 rounded-0
rounded-start-1">
      <div className="card-header">
        <div className="card-title">
          <AccountLong data={my_account} color={"danger"} />

```

```

        </div>
        <div className="card-toolbar">
            <Link to={"/sign-out"} className="btn btn-icon btn-sm"
title="Sign out" >
                <i className="bi bi-box-arrow-right text-danger fw-bold fs-
2x"></i>
            </Link>
        </div>
    </div>
    <div className="card-body d-flex flex-column justify-content-
between p-0">
        <div className="my-contact border-top">
            <div className="filters p-5 border-bottom d-flex align-items-
center">
                <ButtonSearch >
                    <span className="svg-icon svg-icon-1 svg-icon-gray-400 me-
1">
                        <i class="bi bi-search"></i>
                    </span>
                </ButtonSearch>
                <ButtonIcon >
                    <i className={"bi fw-bold fs-2x bi-filter"}></i>
                </ButtonIcon>
            </div>
            <div className="friends">
                <p className="text-center">list kontak dimari</p>
            </div>
        </div>
        <p className="fs-8 text-center mb-0 py-3">
            <i className="bi bi-lock-fill"></i>
            <span className="ms-1">Your personal messages are <span
className="text-info">end-to-end encrypted</span></span>
        </p>
    </div>
</div>
);
}

```



B. Menambahkan HOOK untuk merender data dummy daftar kontak

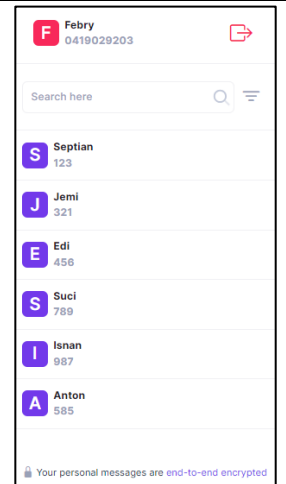
Untuk merender data kontak disini akan mengimplementasikan bentuk dari useMemo, pada script sebelumnya silakan anda tambahkan code dibawah ini:

```
export function ContactUI({ my_account, friends }) {
```

```
...
const ResultData = useMemo(() => {
  let computedData = my_friends;
  return computedData;
}, [my_friends])
...
}
```

Pada JSX anda dapat merender variable ResultData dengan menggunakan MAP:

```
...
<div className="friends">
  {/* <p className="text-center">list kontak
  dimari</p> */}
  {Object.values(ResultData).length > 0 ? (
    ResultData.map((v, index) => (
      <div className="friend-item " key={index}>
        <AccountLong data={v} color={"info"} />
      </div>
    ))
  ) : "No Data Found!"}
</div>
...
```



C. Membuat sebuah search kontak

Untuk membuat sebuah search kontak seperti pada gambar yang telah dibuat, disini kita akan memanfaatkan sebuah function bernama includes(), function tersebut akan mencari seluruh data pada bentuk array objek (filter()) dalam bentuk string.

Buatlah variable search yang disimpan dengan menggunakan useState:

```
const [search, setSearch] = useState([]);
```

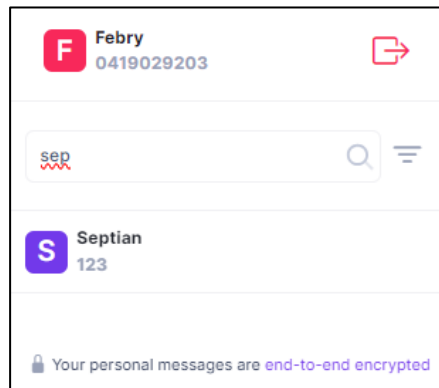
Pada ResultData kita akan menambahkan logika algoritma untuk melakukan pencarian data:

```
const ResultData = useMemo(() => {
  let computedData = my_friends;

  if (search) {
    computedData = computedData.filter(
      listData => {
        return Object.keys(listData).some(key =>
          listData[key].toString().toLowerCase().includes(search)
        );
      }
    );
  }

  return computedData;
}, [my_friends, search]);
```

Menambahkan logika search data dengan includes, dan mengembalikan nilai search secara asynronus



D. Membuat filter sorting kontak secara ascending atau descending

Pertama kita perlu membuat sebuah variable dengan state bernama sorting:

```
const [sorting, setSorting] = useState({ field: "", order: "" });
```

Pada variable sorting memiliki value dalam bentuk objek, isinya ialah field dan order. Key field diperuntukan menyimpan nama objek yang akan di urutkan, dan key order akan berisi asc atau desc.

Kembali pada variable ResultData kita akan menambahkan logika penggunaan localeCompare() untuk mengurutkan nilai array:

```
const ResultData = useMemo(() => {
  let computedData = my_friends;

  if (search) {
    computedData = computedData.filter(
      listData => {
        return Object.keys(listData).some(key =>
          listData[key].toString().toLowerCase().includes(search)
        );
      }
    );
  }

  if (sorting.field) {
    const reversed = sorting.order === "asc" ? 1 : -1;
    computedData = computedData.sort(
      (a, b) =>
        reversed * a[sorting.field].localeCompare(b[sorting.field])
    );
  }

  return computedData;
}, [my_friends, search, sorting]);
```

Menambahkan logika untuk mengurutkan data array secara asynconus

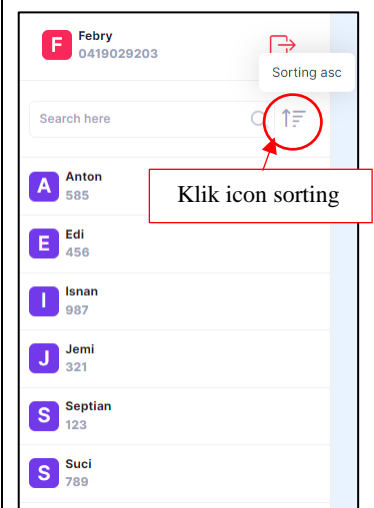
Setelah menambahkan logika sorting pada useMemo kita perlu menambahkan action pada tombol filter di JSX:

```
<ButtonIcon
  title={`Sorting ${sorting.order}`}
  title={}`Sorting ${sorting.order}``
```

```
onAction={() => toggleSorting('name')}}>
  <i className={"bi fw-bold fs-2x bi-filter"}></i>
</ButtonIcon>
```

Pada code diatas kita menambahkan sebuah property onAction bernama toggleSorting('name'), parameter 'name' dikirim pada function tersebut karena kita akan mensorting berdasarkan key 'name' pada data array object MyFriend. Oleh karenanya perlu membuat function tersebut seperti berikut:

```
const toggleSorting = (field) => {
  if (sorting.field === field) {
    setSorting({
      ...sorting,
      order: sorting.order === 'asc' ?
        'desc' : 'asc'
    });
  } else {
    setSorting({
      field: field,
      order: 'asc'
    });
  }
};
```

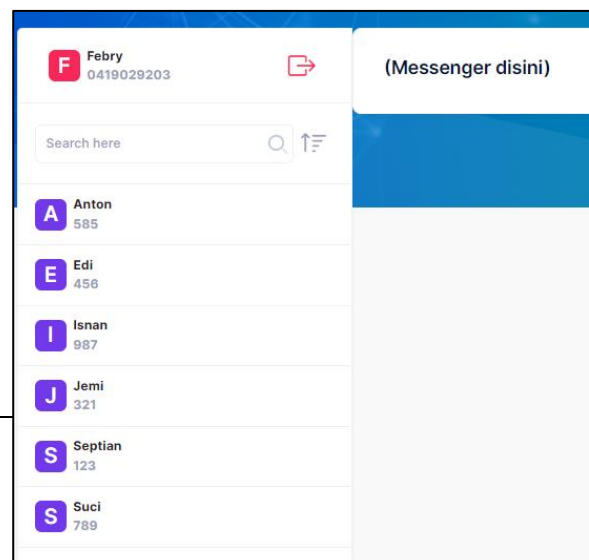


E. Memperbaharui komponen ChapterTwo untuk memanggil atomic ContactUI

Pada file index.js atau komponen ChapterTwo tambahkan code berikut ini untuk mengirimkan data dummy MyFriend dan MyProfile kedalam ContactUI:

```
import { MyFriend } from "../widgets/constantas/DataChat";
export function ChapterTwo() {
  const myprofile = {id: "0419029203", name:"Febry"};

  return (
    ...
    <div className="col-2 col-lg-3 col-xxl-4 px-0">
      /* (Daftar kontak disini) */
      {myprofile ? (
        <ContactUI
          my_account={myprofile}
          friends={MyFriend}
        />
      ) : ""}
    </div>
    ...
  );
}
```



IV. Developing widget Messengers

Setelah membuat widget contact, selanjutnya kita akan modifikasi tampilan atau komponen MessengersUI pada materi sebelumnya.

A. Memasang kondisi jika belum memilih kontak pengguna

Pada code MessengerUI kita perlu membuat sebuah komponen yang menampilkan informasi bahwa belum ada chat yang terpilih, disini akan membuat komponen bernama EmptyChat:

```
const EmptyChat = () => {  
  return (  
    <div>  
      <div className="info text-center">  
        <h1>No Conversations</h1>  
        <p>You didn't made any conversation yet,  
        please select username</p>  
        <span className="badge badge-primary">Start a chat</span>  
      </div>  
    </div>  
  );  
};
```

Selanjutnya kita perlu mengubah nilai variable myChat sehingga komunikasi antara layer dapat berjalan secara asynchronous ataupun realtime dengan memanfaatkan useMemo dan useEffect:

```
export function MessengersUI({ profile, selectedChat, selectedUser }) {  
  const [myChat, setMyChat] = useState([]);  
  const endOfMessagesRef = useRef(null);  
  ...  
  const ResultMessageData = useMemo(() => {  
    let computedData = myChat.map((msg) => ({  
      ...msg,  
      date_fmt: moment(msg.date).format("YYYY-MM-DD"),  
      isOutgoing: msg.from_id === profile.id  
    }));  
    if (search) {  
      computedData = computedData.filter(  
        listData => {  
          return Object.keys(listData).some(key =>  
            listData[key].toString().toLowerCase().includes(search)  
          );  
        }  
      );  
    }  
    return computedData;  
  }, [myChat, profile.id]);  
  
  useEffect(() => {  
    setMyChat(selectedChat)
```

```

    scrollToBottom();
  }, []);

  ...

```

Setelah merubah alur komunikasi pada myChat selanjutnya kita perlu memasang sebuah kondisi jika belum ada chat terpilih maka menampilkan komponen EmptyChat pada JSX:

```

{ResultMessageData.length > 0 ? (
  <>
    <div className="chat-message px-2 h-100"
      style={StylesMessenger.chatBox}>
      <ChatBodyWithGrouped data={ResultMessageData} profile={profile} />
      <div ref={endOfMessagesRef} />
    </div>
    <div className="chat-send bg-light p-3">
      <form
        method="post"
        autoComplete="off"
        onSubmit={(e) => HandlerSendChat(e)} >
        ...
      </form>
    </div>
  </>
) : (
  <EmptyChat />
)}

```

B. Memodifikasi data dummy chat

Pada komponen ChapterTwo update-lah kolom layer sebelah kanan yang sebelumnya menampilkan tulisan “Messeger disini” menjadi komponen MessengersUI beserta pengiriman data dummy chatnya:

```

import React, { useState } from "react";
import { ContactUI, MessengersUI } from "../widgets";
import { Messengers, MyFriend } from "../widgets/constantas/DataChat";

export function ChapterTwo() {
  const myprofile = {id: "0419029203", name:"Febry"};

  const [selectedUser, setSelectedUser] = useState(0);
  const [selectedChat, setSelectedChat] = useState([]);

  return (
    ...
    <div className="col-2 col-lg-3 col-xxl-4 px-0">
      {myprofile ? (
        <ContactUI

```



```

        my_account={myprofile}
        friends={MyFriend}
        selectedUser={selectedUser}
      />
    ) : ""}
  </div>
  <div className="col-10 col-lg-9 col-xxl-8 px-0">
    {myprofile ? (
      <MessagersUI
        profile={myprofile}
        selectedUser={selectedUser}
        selectedChat={selectedChat}
      />
    ) : ""}
  </div>
  ...
);
}

```

V. Passing parameter chat terpilih

Setelah memperbaharui skema dari MessagersUI kita perlu mengirimkan chat yang terkirim atau terpilih dan menginisialisasinya kedalam parameter MessagersUI dan ContactUI:

```

export function MessagersUI({ profile, selectedChat, selectedUser }) {
  ...
  useEffect(() => {
    setMyChat(selectedChat)
    scrollToBottom();
  }, [selectedChat]);
  ...
}

```

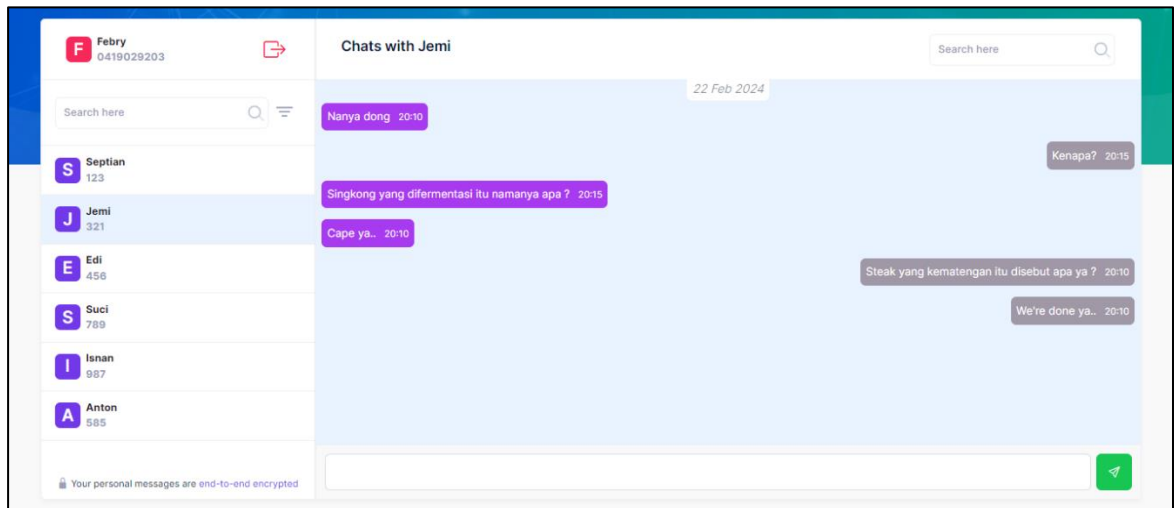
```

export function ContactUI({ my_account, friends, selectedUser,
  HandlerSelectedChat }) {
  ...
  return(
    ...
    {Object.values(ResultData).length > 0 ? (
      ResultData.map((v, index) => (
        <div className={"friend-item"} key={index}
          onClick={() => HandlerSelectedChat(v)}>
          <AccountLong data={v} color={"info"} />
        </div>
      ))
    ) : "No Data Found!"}
    ...
  )
}

```

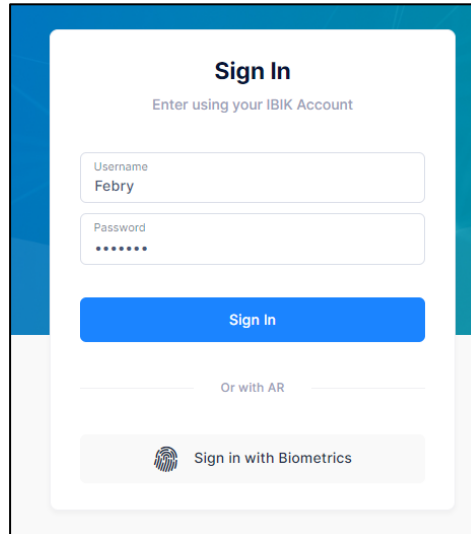
Pada komponen ChapterTwo kita perlu membuat sebuah action untuk HandlerSelectedChat sebagai berikut:

```
const HandlerSelectedChat = (data) => {  
  setSelectedUser(data);  
  const the_msg = [...Messegers]  
  const findChatByUserID = the_msg.find(item => item.user_id ===  
data.user_id)  
  if (findChatByUserID) {  
    setSelectedChat(findChatByUserID.messages);  
  } else {  
    setSelectedChat([]);  
  }  
}
```

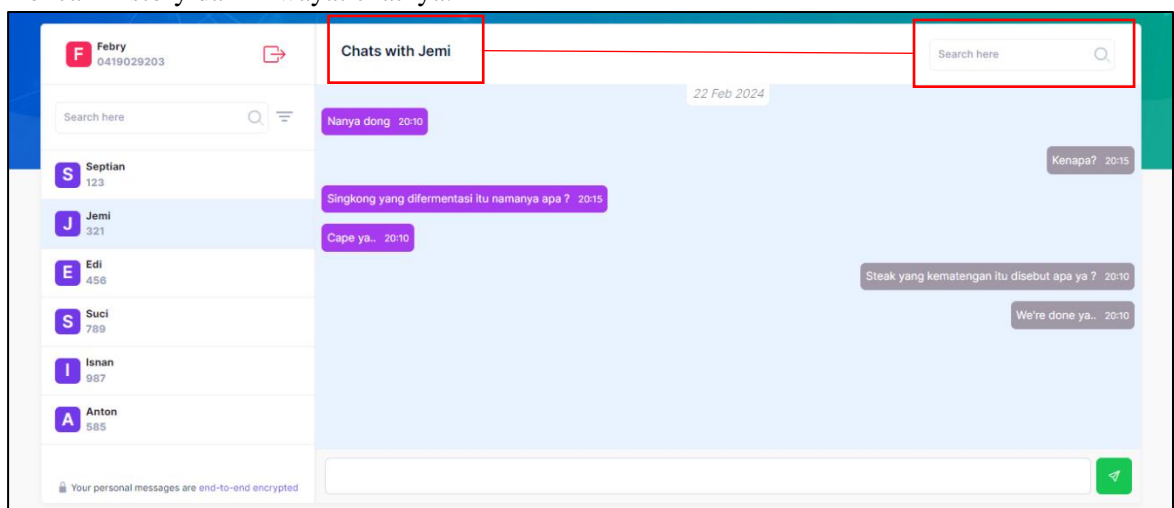


3. Tugas Pelatihan

1. Buatlah sebuah halaman baru untuk authentications berisi username dan password



2. Manfaatkan komponen react router dom untuk membuat sebuah alur hirarky navigasi sebagai berikut:
 - a. Jika memasukan address /sign-in maka akan menampilkan halaman Authentications
 - b. Jika memasukan address /sign-out maka akan menampilkan halaman Authentications
 - c. Jika memasukan address /chapter-1 maka akan menampilkan halaman Chapter One
 - d. Jika memasukan address /chapter-2 maka akan menampilkan halaman Chapter Two
 - e. Jika tidak memasukan sebuah address (hanya base url) menampilkan halaman Chapter One
 - f. Jika memasukan address /home maka akan menampilkan halaman Chapter Two
 - g. Jika memasukan address yang tidak ada pada daftar route maka menampilkan halaman Error 404
3. Dalam contoh kasus messaging apps diatas, dapatkan anda buat dan lengkapi kondisi untuk menampilkan visual hirarky terhadap chat yang terpilih, dan buatlah sebuah fitur search untuk mencari history dari Riwayat chatnya:



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 repository tersebut. Buatlah file dokumen dalam bentuk file pdf



I B I K

Matakuliah/Code

Dosen

Kelas

: Lab. Pemrograman Web Lanjut / TIFA3V0

: Febri Damatraseta Fairuz, S.T., M.Kom

: TI-21-PA dan TI-21-KA

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*