REACT-REDUX-MIDDLEWARE

Home.js M
logger.js U

client > src > store > middlewares > logger.js > logger > <function>

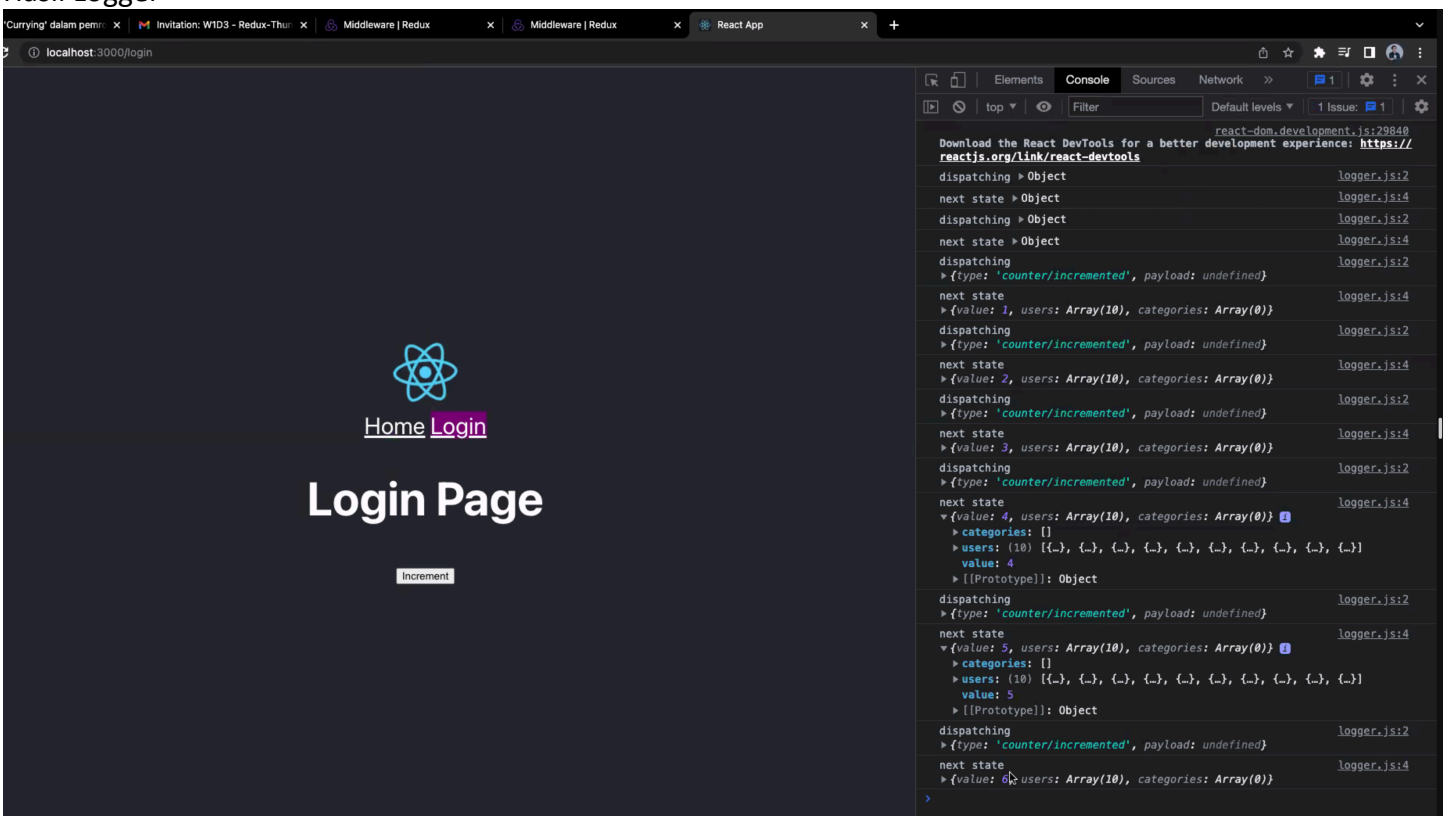
```
const logger = (store) => (next) => (action) => {
    console.log("dispatching", action);
    let result = next(action);
    console.log("next state", store.getState());
    return result;
};

export default logger;
```

index.js

client > src > store > index.js > store

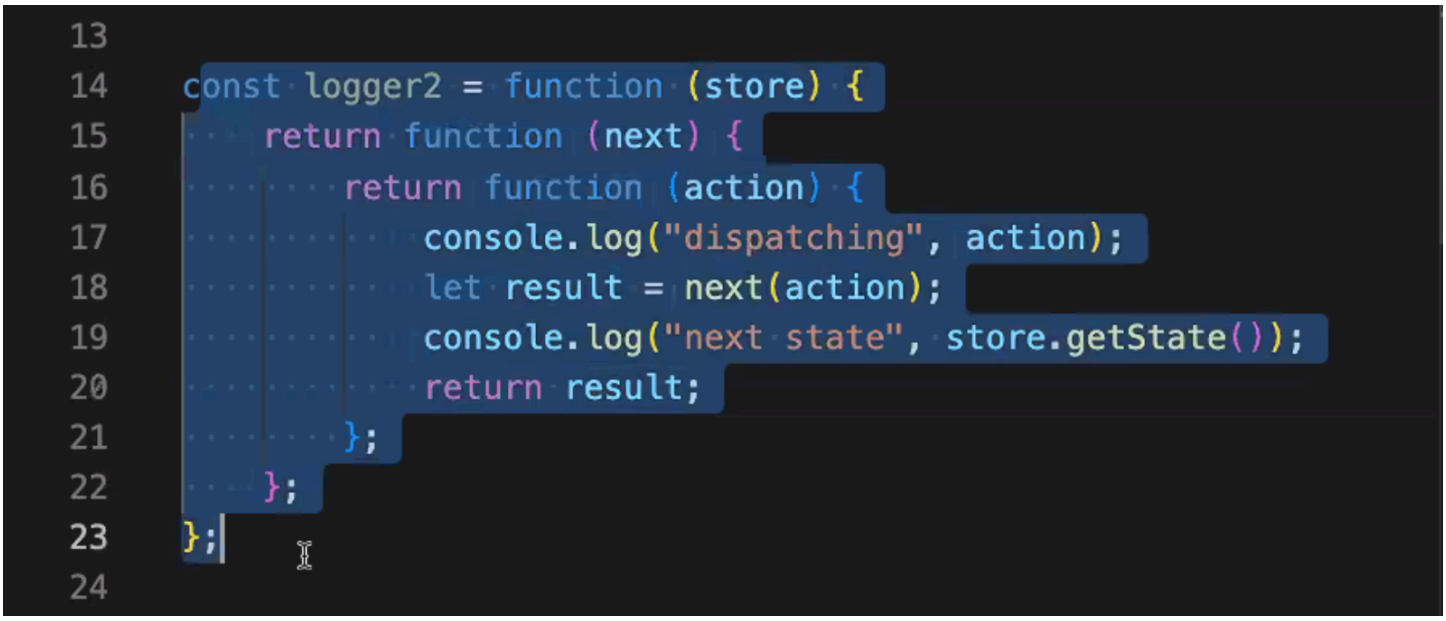
```
import { legacy_createStore as createStore, applyMiddleware } fr
import rootReducers from "./reducers/rootReducers";
import logger from "./middlewares/logger"

let store = createStore(rootReducers, applyMiddleware(logger));

export default store;
```

Hasil Logger



Higher order function

- Parameter nya adalah function
- Function yang mengembalikan sebuah function



```
13
14  const logger2 = function (store) {
15      return function (next) {
16          return function (action) {
17              console.log("dispatching", action);
18              let result = next(action);
19              console.log("next state", store.getState());
20              return result;
21          };
22      };
23  };
24
```

Middleware redux itu ***Currying*** **- logger(store)(next)(action)**

```
console.log(res); // 20
```

Hemm,, dua cara tadi hasilnya sama saja, jadi apa keuntungannya menggunakan teknik *currying* ????

Baik,, kita akan cari tau apa keuntungan menggunakan teknik ini.

Katakanlah kita ingin membuat fungsi untuk menghitung diskon pada harga jual sebuah barang.

```
function hitungDiskon(harga, diskon) {
    return harga * diskon
}
```

Dengan menggunakan fungsi **hitungDiskon** di atas jika kita ingin menghitung diskon 10% untuk banyak barang maka kita akan menuliskannya seperti ini.

```
function hitungDiskon(harga, diskon) {
    return harga * diskon
}

const diskon1 = hitungDiskon(1500,0.10); // 150

const diskon2 = hitungDiskon(2000,0.10); // 200

const diskon3 = hitungDiskon(5000,0.10); // 500


console.log(diskon1);
console.log(diskon2);
console.log(diskon3);
```

Sekilas tidak ada yang salah dengan code di atas, dan memang tidak ada yang salah. Hanya saja ada banyak pengulangan parameter untuk diskonnya.

Nah code di atas coba kita sederhanakan menggunakan teknik *currying*.

```
function hitungDiskon(diskon) {
    return (harga) => {
      return diskon * harga;
    }
}

const diskon10Persen = hitungDiskon(0.10)
```

Sekilas tidak ada yang salah dengan code di atas, dan memang tidak ada yang salah. Hanya saja ada banyak pengulangan parameter untuk diskonnya.

Nah code di atas coba kita sederhanakan menggunakan teknik *currying*.

```
function hitungDiskon(diskon) {
    return (harga) => {
      return diskon * harga;
    }
}

const diskon10Persen = hitungDiskon(0.10)

const diskon1 = diskon10Persen(1500); // 150

const diskon2 = diskon10Persen(2000); // 200

const diskon3 = diskon10Persen(5000); // 500


console.log(diskon1);
console.log(diskon2);
console.log(diskon3);
```

Hasilnya sama, hanya saja sekarang tidak ada pengulangan untuk parameter diskonnya. Hal ini menghindari kesalahan jika kita mengulangi penulisan parameter untuk diskonnya. Dengan currying untuk fungsi **hitungDiskon** di atas, kita juga bisa menggunakan fungsi **hitungDiskon** untuk diskon lain. Misalnya untuk diskon 20%

Nah code di atas coba kita sederhanakan menggunakan teknik *currying*.

```
function hitungDiskon(diskon) {
    return (harga) => {
      return diskon * harga;
    }
}

const diskon10Persen = hitungDiskon(0.10)

const diskon1 = diskon10Persen(1500); // 150

const diskon2 = diskon10Persen(2000); // 200

const diskon3 = diskon10Persen(5000); // 500


console.log(diskon1);
console.log(diskon2);
console.log(diskon3);
```

Hasilnya sama, hanya saja sekarang tidak ada pengulangan untuk parameter diskonnya. Hal ini menghindari kesalahan jika kita mengulangi penulisan parameter untuk diskonnya. Dengan currying untuk fungsi **hitungDiskon** di atas, kita juga bisa menggunakan fungsi **hitungDiskon** untuk diskon lain. Misalnya untuk diskon 20%

```
function hitungDiskon(diskon) {
    return (harga) => {
      return diskon * harga;
    }
}

const diskon20Persen = hitungDiskon(0.20)
```

Memahami konsep dan teknik di atas sangat penting ketika kalian ingin mempelajari bahasa pemrograman fungsional murni seperti Elixir, Erlang, Clojure, dan Haskell.

REDUX THUNK

Why ? redux biasa hanya synchronous update, Thunk bisa melakukan interaksi async

package:

```
npm install redux-thunk

yarn add redux-thunk
```

The thunk middleware is the default export.

▶ **More Details: Importing the thunk middleware**

Then, to enable Redux Thunk, use `applyMiddleware()`:

```
import { createStore, applyMiddleware } from 'redux'
import thunk from 'redux-thunk'
import rootReducer from './reducers/index'

const store = createStore(rootReducer, applyMiddleware(thunk))
```

Injecting a Custom Argument

Since 2.1.0, Redux Thunk supports injecting a custom argument into the thunk middleware. This is typically useful for cases like using an API service layer that could be swapped out for a mock service in tests.

For Redux Toolkit, the `getDefaultMiddleware` callback inside of `configureStore` lets you pass in a custom `extraArgument`:

```
import { configureStore } from '@reduxjs/toolkit'
import rootReducer from './reducer'
import { myCustomApiService } from './api'

const store = configureStore({
  reducer: rootReducer,
  middleware: getDefaultMiddleware =>
    getDefaultMiddleware({
      thunk: {
        extraArgument: myCustomApiService
      }
    })
})
```

Home.js M | logger.js U | index.js M

client > src > store > action > index.js > fetchUser

```
        type: COUNTER_INCREMENT,
        payload,
    };
};

export const fetchSuccess = function (payload) {
    return {
        type: USERS_FETCH_SUCCESS,
        payload,
    };
};

export const fetchUser = function () {
    return function (dispatch) {
        fetch("http://localhost:3001/users")
            .then((res) => {
                if (!res.ok) throw new Error("error ngab");
                return res.json();
            })
            .then((data) => {
                dispatch(fetchSuccess(data));
            })
            .catch(console.log);
    };
};
```

index.js M

client > src > store > index.js > ...

```
import { legacy_createStore as createStore, applyMiddleware } fr
import rootReducers from "./reducers/rootReducers";
import logger from "./middlewares/logger";
import thunk from "redux-thunk";

let store = createStore(rootReducers, applyMiddleware(logger, thu

export default store;
```

logger.js U | index.js M

client > src > store > action > index.js > fetchSuccess

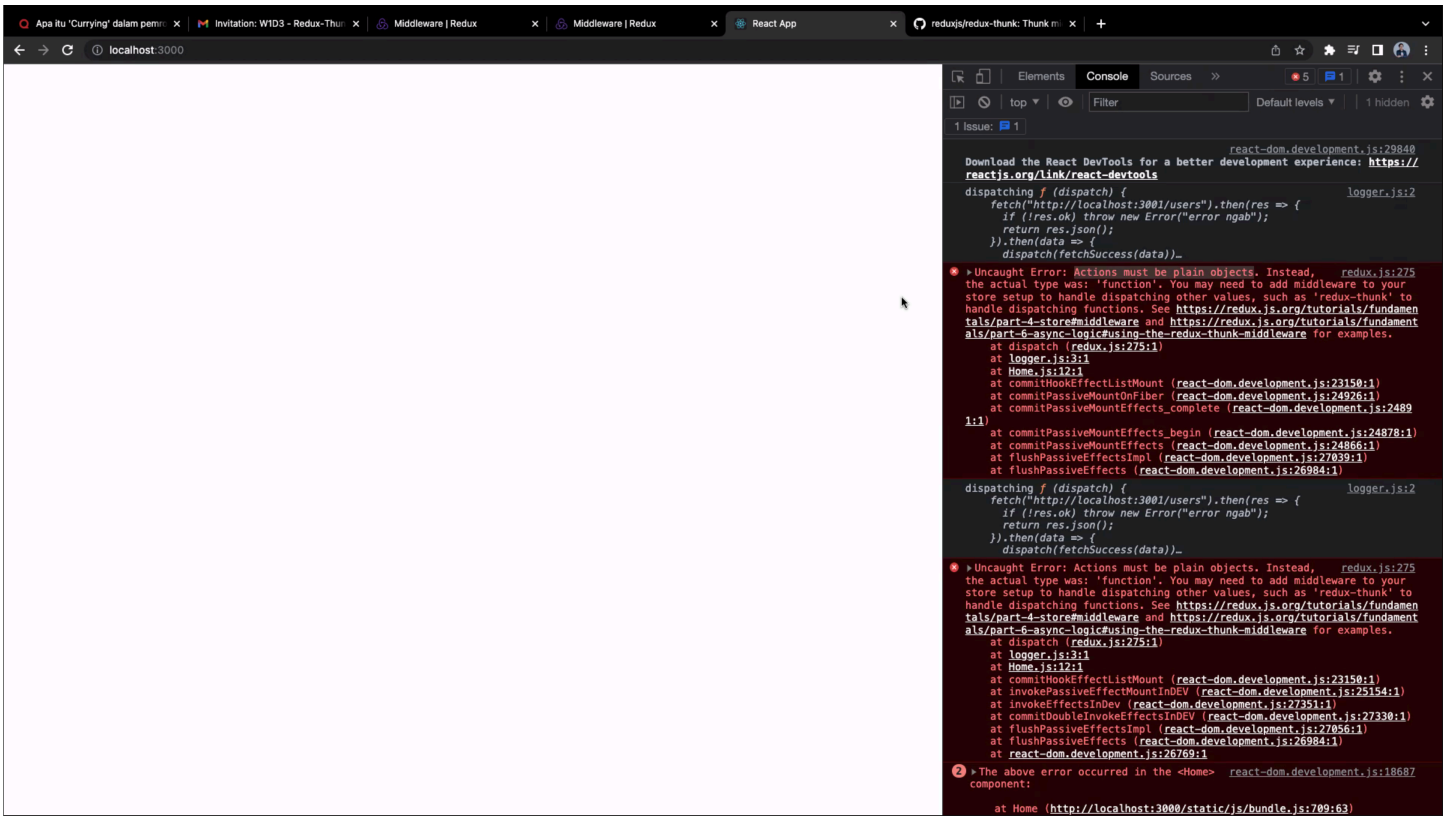
```
        type: COUNTER_INCREMENT,
        payload,
    };
};

export const fetchSuccess = function (payload) {
    return {
        type: USERS_FETCH_SUCCESS,
        payload,
    };
};

export const fetchUser = function () {
    return function (dispatch) {
        fetch("http://localhost:3001/users")
            .then((res) => {
                if (!res.ok) throw new Error("error ngab");
                return res.json();
            })
            .then((data) => {
                dispatch(fetchSuccess(data));
            })
            .catch(console.log);
    };
};
```

index.js M | Home.js M

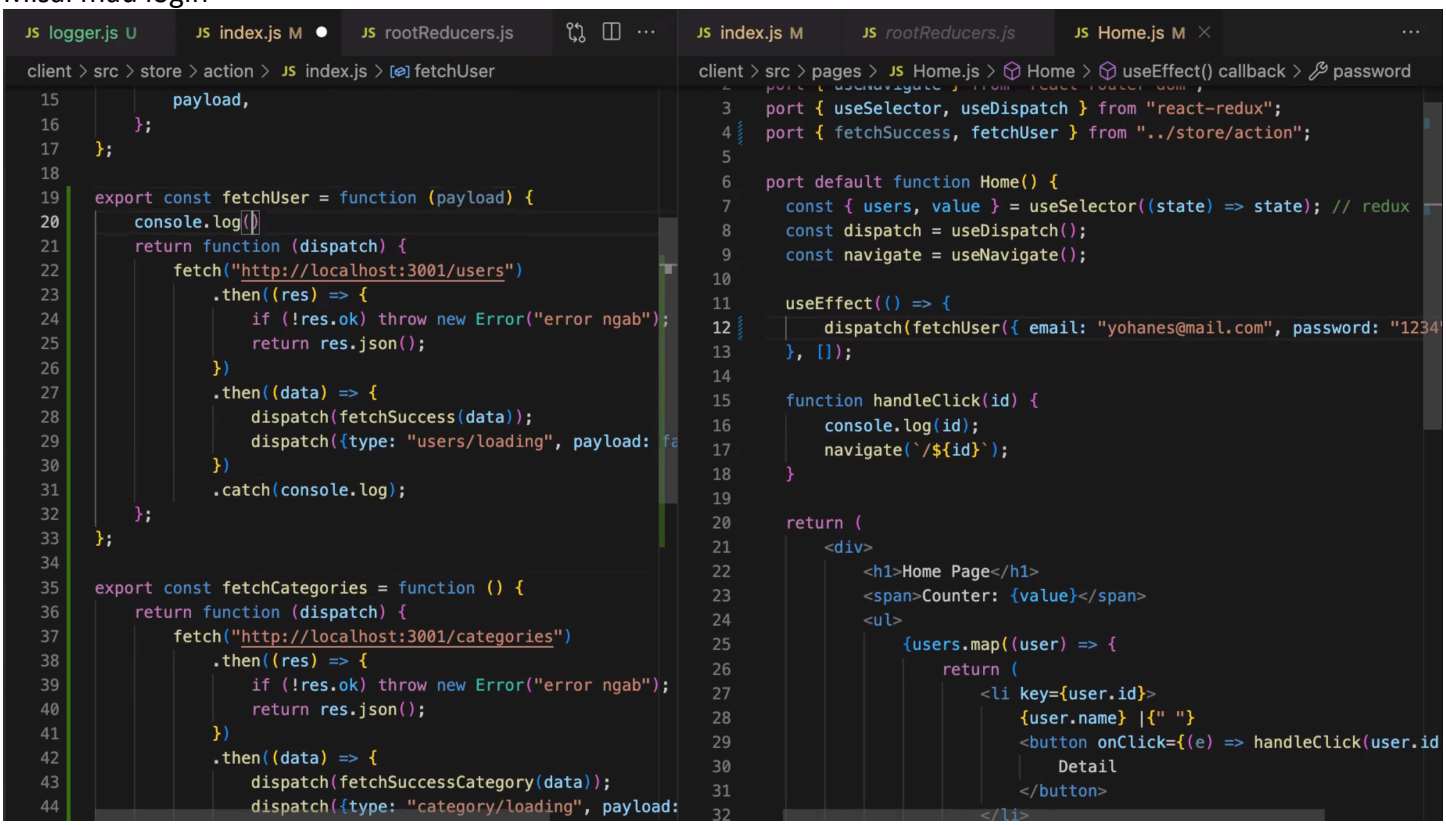
client > src > pages > Home.js > Home > useEffect() callback

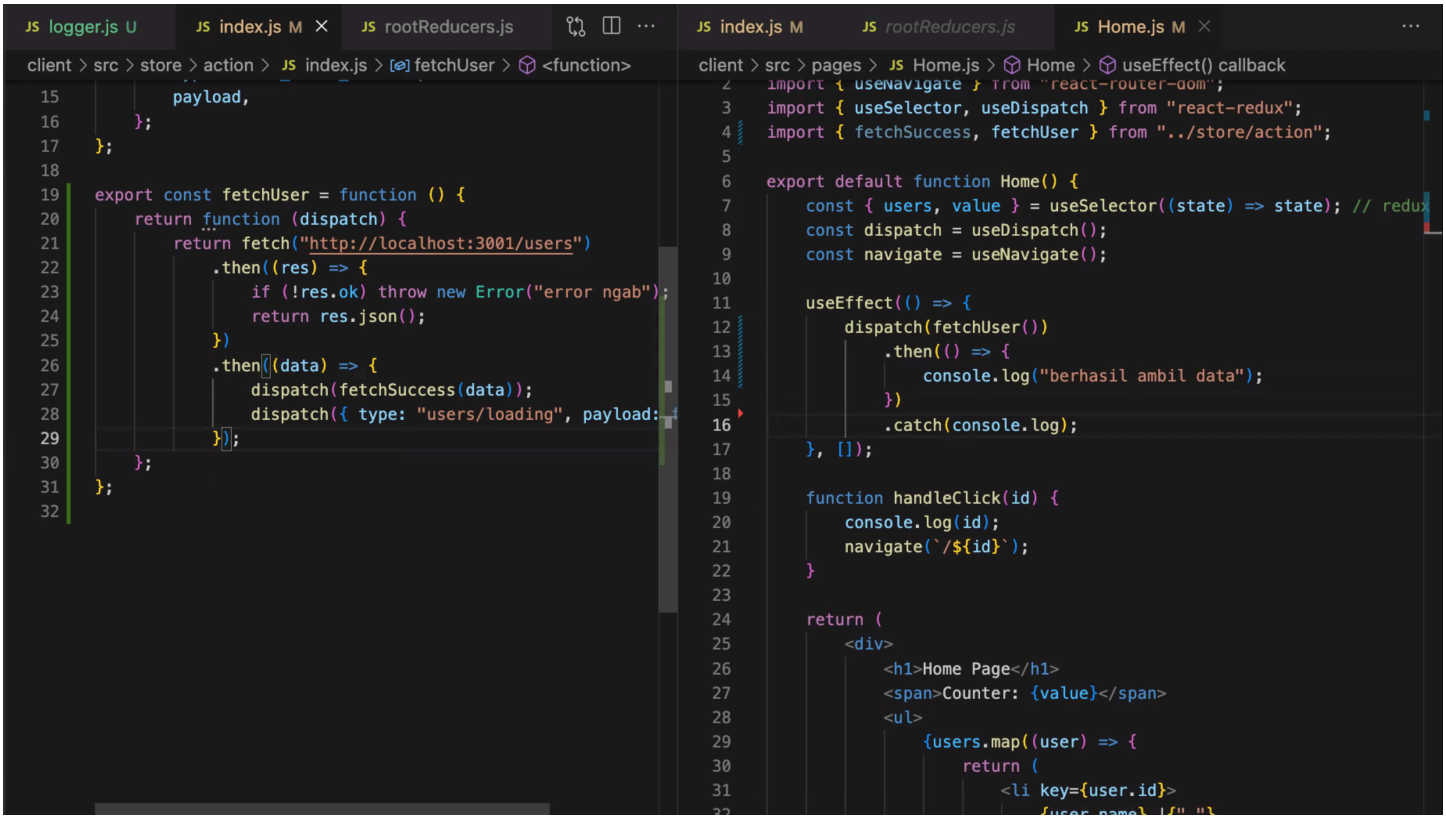
```
import { fetchSuccess, fetchUser } from "../store/action";

export default function Home() {
    const { users, value } = useSelector((state) => state); // re
    const dispatch = useDispatch();
    const navigate = useNavigate();

    useEffect(() => {
        dispatch(fetchUser());
    }, []);

    function handleClick(id) {
        console.log(id);
        navigate(`/${id}`);
    }

    return (
        <div>
            <h1>Home Page</h1>
            <span>Counter: {value}</span>
            <ul>
                {users.map((user) => {
                    return (
                        <li key={user.id}>
                            {user.name} |{" "}
                            <button onClick={(e) => handleClick(u
                                Detail
                            </button>
                        </li>
                    );
```

TANPA THUNK

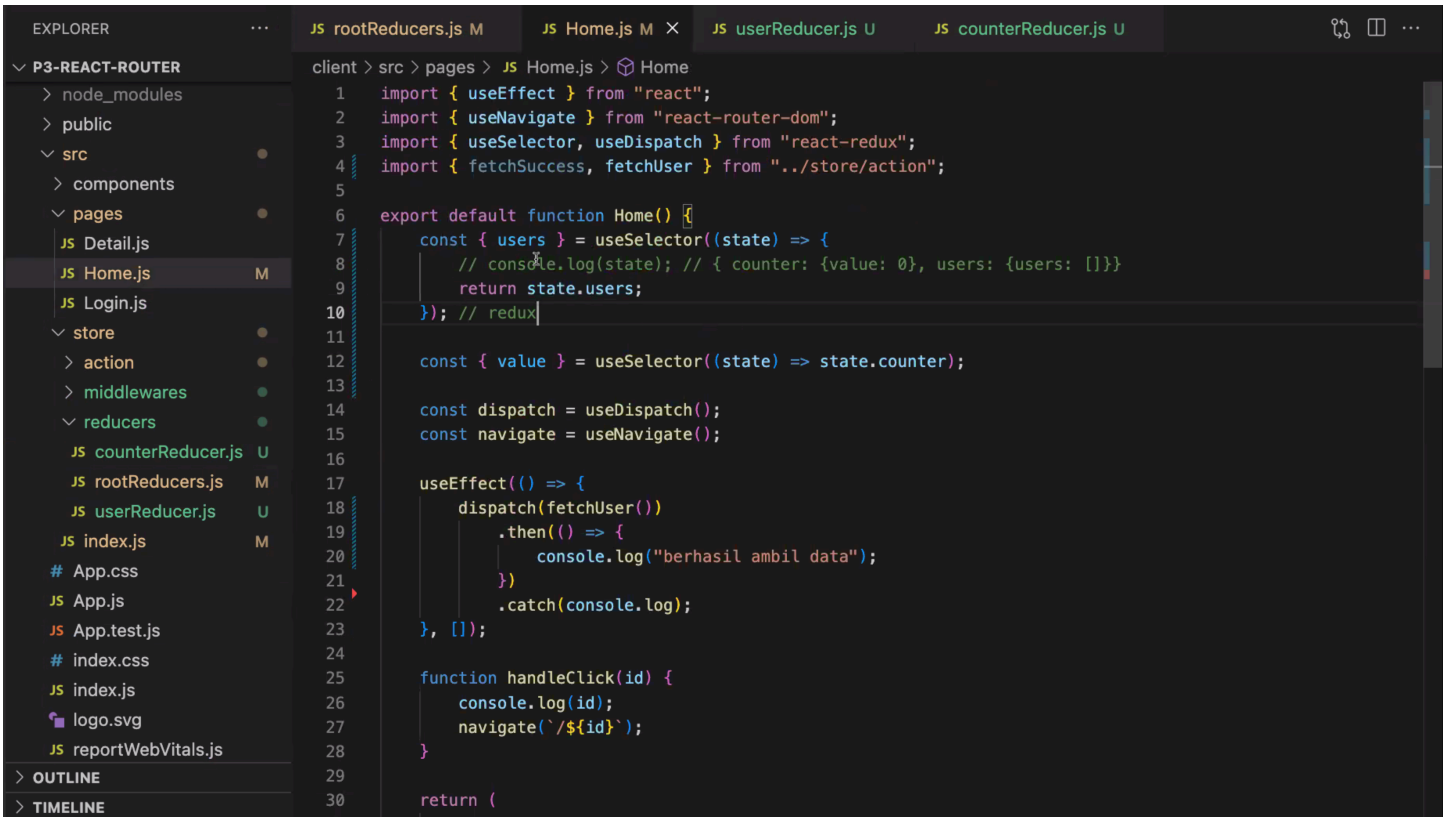


Misal mau login



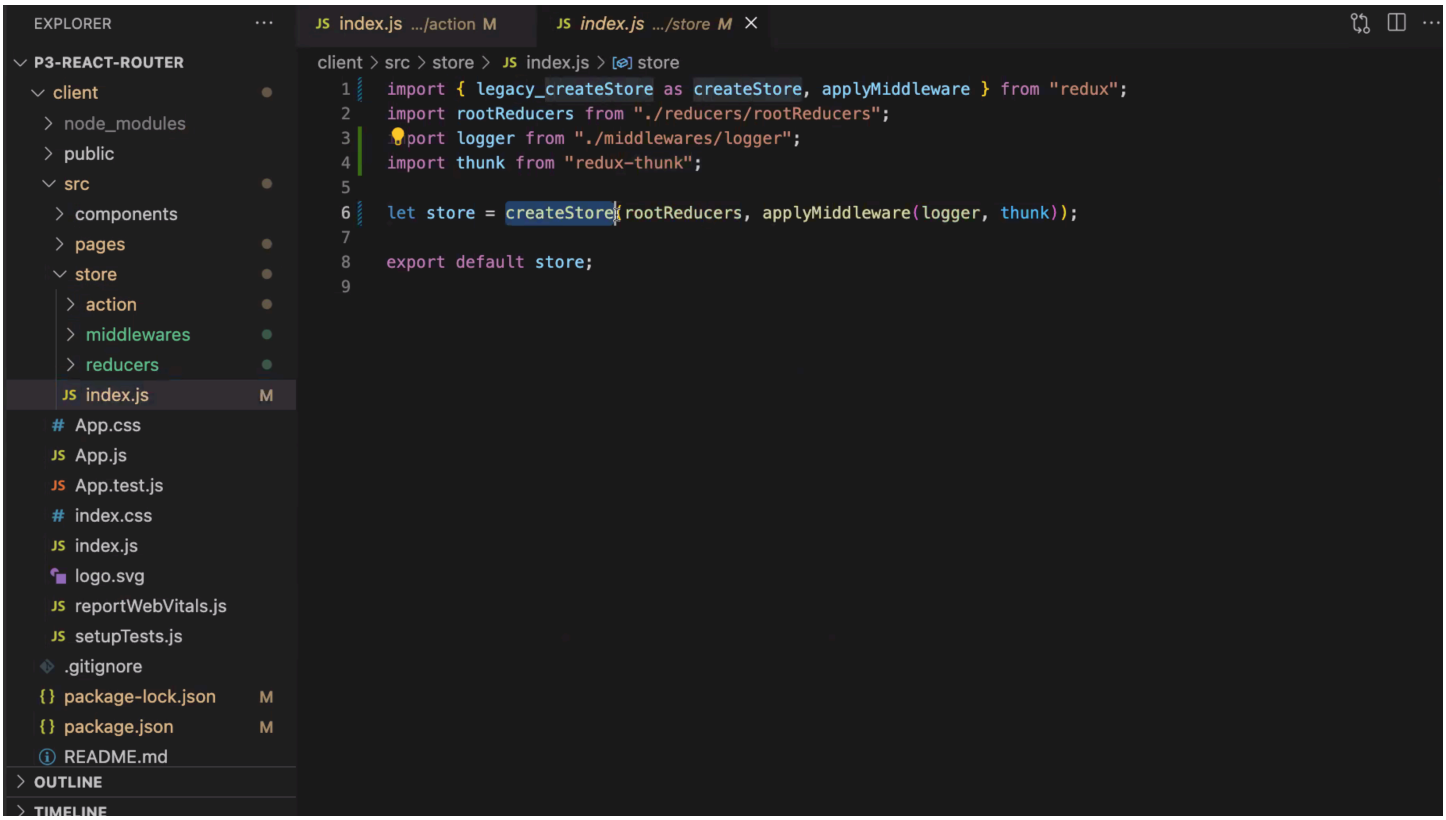


GABISA PAKE HOOKS KARENA BUKAN REACT COMPONENTS, USE (HELPER)

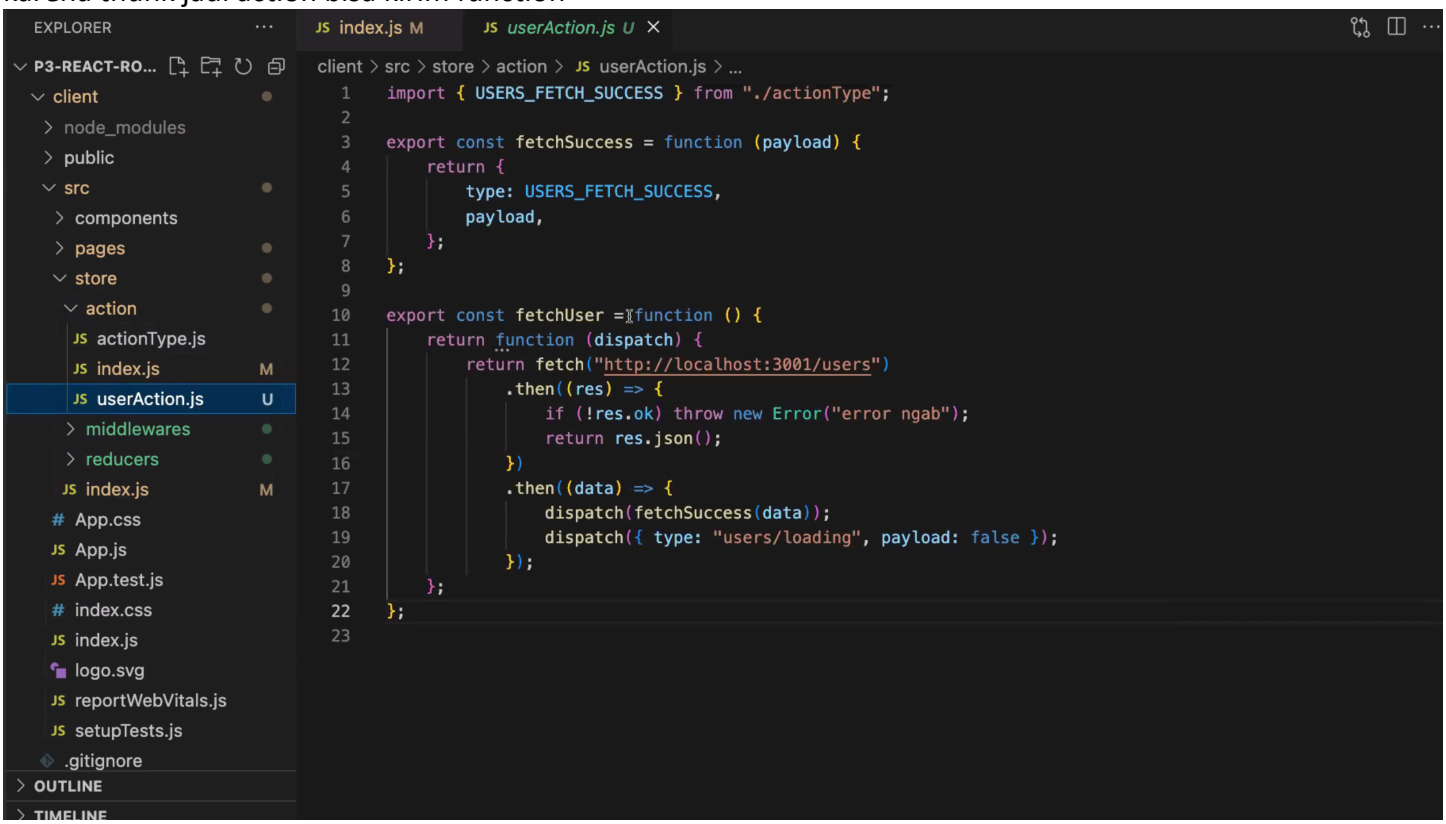


===== REVIEW =====

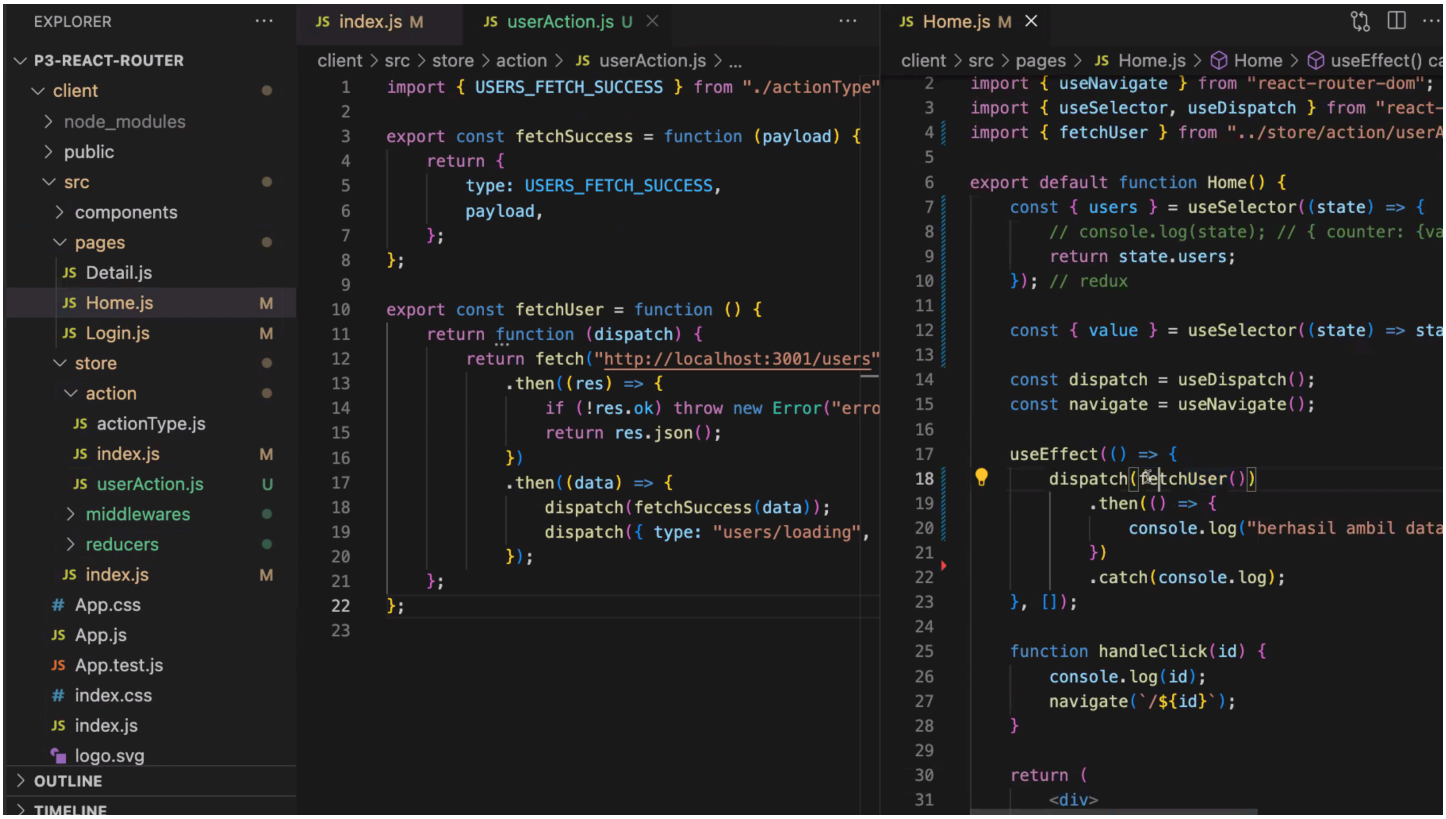
Install redux



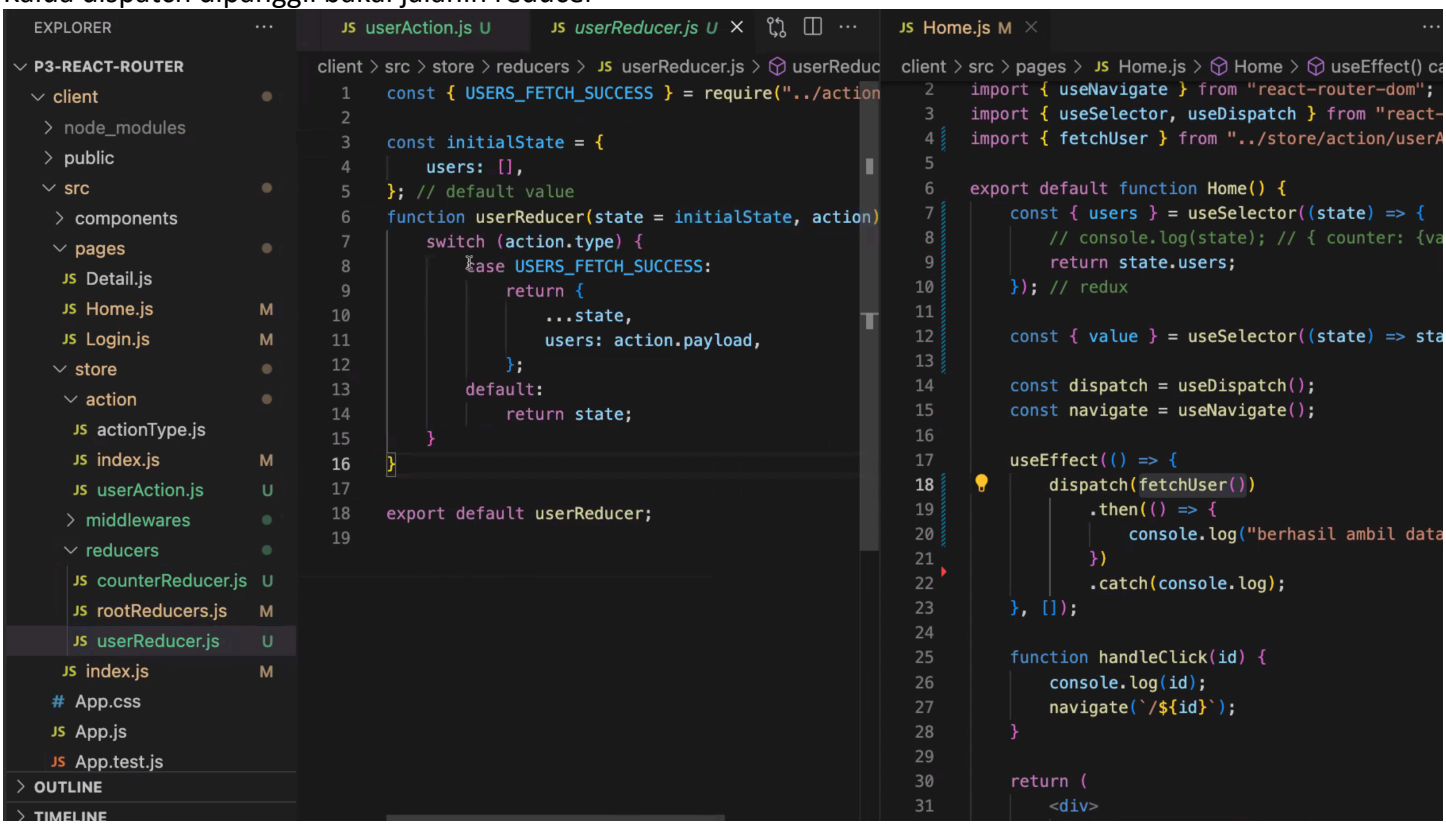
Ada fitur middleware di pasang di createStore, applymiddleware logger dan thunk
karena thunk jadi action bisa kirim function



Di home ada redux thunk, middleware dispatch didalemnya fetchuser untuk fetch async ke server



Kalua dispatch dipanggil bakal jalanin reducer



State user jadi reaktif

index.js M | userAction.js U | userReducer.js U

client > src > store > reducers > userReducer.js > userReducer

```
const { USERS_FETCH_SUCCESS } = require("../action/actionType");

const initialState = {
    users: [],
}; // default value
function userReducer(state = initialState, action) {
    switch (action.type) {
        case USERS_FETCH_SUCCESS:
            return {
                ...state,
                users: action.payload,
            };
        default:
            return state;
    }
}

export default userReducer;

```

Home.js M

client > src > pages > Home.js > Home

```
import { fetchUser } from "../store/action/userAction";

export default function Home() {
    const { users } = useSelector((state) => {
        // console.log(state); // { counter: {value: 0}, users
        return state.users;
    }); // redux

    const { value } = useSelector((state) => state.counter);

    const dispatch = useDispatch();
    const navigate = useNavigate();

    useEffect(() => {
        dispatch(fetchUser())
            .then(() => {
                console.log("berhasil ambil data");
            })
            .catch(console.log);
    }, []);

    function handleClick(id) {
        console.log(id);
        navigate(`/${id}`);
    }

    return (
        <div>
            <h1>Home Page</h1>
```

MATERI KE-2 ITU COMBINED REDUCER JADI MODULAR

index.js M | userAction.js U | userReducer.js U

client > src > store > reducers > userReducer.js > userReducer

```
const { USERS_FETCH_SUCCESS } = require("../action/actionType");

const initialState = {
    users: [],
}; // default value
function userReducer(state = initialState, action) {
    switch (action.type) {
        case USERS_FETCH_SUCCESS:
            return {
                ...state,
                users: action.payload,
            };
        default:
            return state;
    }
}

export default userReducer;

```

rootReducers.js M

client > src > store > reducers > rootReducers.js > ...

```
import { combineReducers } from "redux";

port userReducer from "./userReducer";
import counterReducer from "./counterReducer";

const rootReducers = combineReducers({
    users: userReducer,
    counter: counterReducer,
});

export default rootReducers;

```