**BÁO CÁO 8: TỐI ƯU DEMO REDUX VÀ XÂY DỰNG APP SỬ DỤNG REDUX SAGA VIẾT BẰNG TYPESCRIPT**

1. **Tối ưu demo Redux**
   1. **State trong reducers:**

* Trong contactReducer.js:

const contactReducer = (state = initialState, action) => {

  switch(action.type) {

    case types.FETCH\_CONTACT:

      const contacts = action.contacts;

      return { ...state, contacts };

    case types.ADD\_CONTACT:

      state = action.contacts;

      return state;

    case types.DELETE\_CONTACT:

      state = action.contact;

      return state;

    case types.EDIT\_CONTACT:

      state = action.contact;

      return state;

    default:

      return state;

  }

};

* Với action ADD\_CONTACT, việc trả về một state thì sẽ mất những state trước đó, trong khi đó chỉ cần thêm contact vừa được add và state cũ là được. Tương tự như các action khác. Vì thế sửa lại contactReducer như sau:

const contactReducer = (state = initialState, action) => {

  switch (action.type) {

    case types.FETCH\_CONTACT:

      const contacts = action.contacts;

      return { ...state, contacts };

    case types.ADD\_CONTACT:

      state.contacts.push(action.contact);

      return { ...state };

    case types.DELETE\_CONTACT:

      let index = state.contacts.length;

      while (index--) {

        if (state.contacts[index] === action.contact) {

          state.contacts.splice(index, 1);

          break;

        }

      }

      const delContact = action.contact;

      return { ...state, delContact };

    case types.EDIT\_CONTACT:

      for(let i = 0; i < state.contacts.length; i++) {

        if(state.contacts[i].id === action.contact.id) {

          state.contacts[i] = action.contact;

          break;

        }

      }

      const contactEdit = action.contact;

      return { ...state, contactEdit };

    default:

      return state;

  }

};

* 1. **Form**
* Trong AddModal.js:

const ho\_va\_ten = document.getElementById("nest-messages\_ho\_va\_ten").value;

    const sdt = document.getElementById("nest-messages\_sdt").value;

    const gender = document.getElementById("nest-messages\_gender").value;

    const age = document.getElementById("nest-messages\_age").value;

* Không nên sử dụng document.getElementById vì nó ảnh hưởng đến DOM thật mà trong Ant Design có hỗ trợ lấy value trong form nên em sẽ sửa lại như sau:

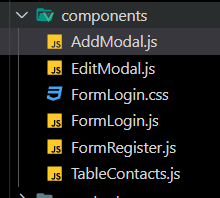
const ho\_va\_ten =  form.getFieldValue('ho\_va\_ten');

const sdt = form.getFieldValue('sdt');

const gender = form.getFieldValue('gender');

const age = form.getFieldValue('age');

* 1. **Cấu trúc thư mục**



* Ta có thể thấy AddModal, EditModal và TableContacts là những component cùng cấp với nhau chính vì thế em sẽ cấu trúc lại như sau:



* 1. **API**
* Trong mỗi action gọi api, việc viết lại đường dẫn sẽ mất rất nhiều thời gian nên em sẽ tạo sẵn một constant để chứa đường dẫn api để mỗi lần sử dụng sẽ tiện hơn
* urlApi.js:

export const contactData =

  "https://60b0f8b91f26610017fff943.mockapi.io/api/v1/contact\_data/";

export const userData =

  "https://60b0f8b91f26610017fff943.mockapi.io/api/v1/users/";

* Khi đó em chỉ cần khai báo mỗi lần gọi api như sau :

axios

.get(contactData)

.then((res) => {

   dispatch(fetchContact(res.data));

   })

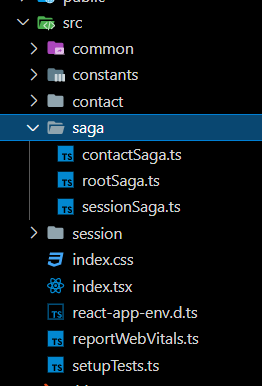
   .catch((err) => {

    console.log(err);

   });

1. **Xây dựng app sử dụng Redux Saga viết bằng TypeScript** 
   1. **Ý tưởng**

* Em sẽ chuyển app redux lần trước được viết bằng JavaScript thành TypeScript đồng thời sẽ quản lý các action trong redux bằng redux saga
  1. **Cách thực hiện**
* Đầu tiên, em sẽ thiết lập Redux Saga vào project react với folder tên là saga như sau :

****

* Thư mục này sẽ chứa các logic xử lý side effect trong redux cũng như thiết lập saga :
* Bên trong ***rootSata.ts :***

import { all } from "redux-saga/effects";

import { sessionSaga } from "./sessionSaga";

import { contactSaga } from "./contactSaga";

export default function\* rootSaga() {

  yield all([sessionSaga(), contactSaga()]);

}

* Lệnh yield all() sẽ gọi tất cả các file xử lý logic mỗi khi nghe được một action nào đó được gọi.
* Em sẽ sử dụng generator function để xử lý các logic trong project. Với mỗi hàm, em sẽ sử dụng lệnh yield. Ví dụ như sau:
* Bên trong ***sessionSaga.ts :***

function\* handleLogin(action: any) {

  const user = action.user;

  yield axios.get(userData).then((response) => {

    for (const userInt of response.data) {

      if (user.username === userInt.username) {

        bcrypt.compare(user.password, userInt.password).then((valid: any) => {

          if (valid) {

            const token = userInt.token;

            window.localStorage.setItem("jwtToken", token);

          }

        });

        break;

      }

    }

  });

  yield delay(500);

  yield put({

    type: types.SET\_CURRENT\_USER,

    auth: {

      username: user.username,

      password: user.password,

      token: window.localStorage.getItem("jwtToken"),

    },

  });

}

function\* handleRegister(action: any) {

  const user = action.user;

  const salt = bcrypt.genSaltSync(10);

  const hassPassword = bcrypt.hashSync(user.password, salt);

  const token = jwt.sign(

    {

      username: user.username,

      password: hassPassword,

    },

    "SECRET\_TOKEN"

  );

  const data = {

    username: user.username,

    password: hassPassword,

    token: token,

  };

  yield axios

    .post(userData, data)

    .then((response) => {

      console.log(response);

    })

    .catch((err) => {

      console.log(err);

    });

}

export function\* sessionSaga() {

  yield takeEvery(types.LOGIN, handleLogin);

  yield takeEvery(types.REGISTER, handleRegister);

}

* Tất cả các logic gọi Api đều được xử lý mỗi khi action tương ứng được thực hiện thông qua lệnh takeEvery. Ví dụ : Khi người dùng bấm vào nút đăng nhập, lập tức saga sẽ lắng nghe được và xử lý nhờ vào hàm handleLogin.
* Tương tự như ***sessionSaga***, ***contactSaga*** cũng xử lý logic khi nghe được action.
* Như vậy bên trong file ***sessionActions.ts*** và ***contactActions.ts*** chỉ còn lại những action trả về type và dữ liệu chứ không còn hàm xử lý logic nữa.
  1. **Vấn đề của redux thunk**
* Redux thunk cho phép sử dụng **mapDispatchToProps** rồi gọi các action thông qua props. Điều này làm code nở ra rất nhiều nếu có nhiều action.
* Chính vì thế em sẽ sử dụng **useDispatch** để gọi action và code sẽ trở nên gọn hơn.