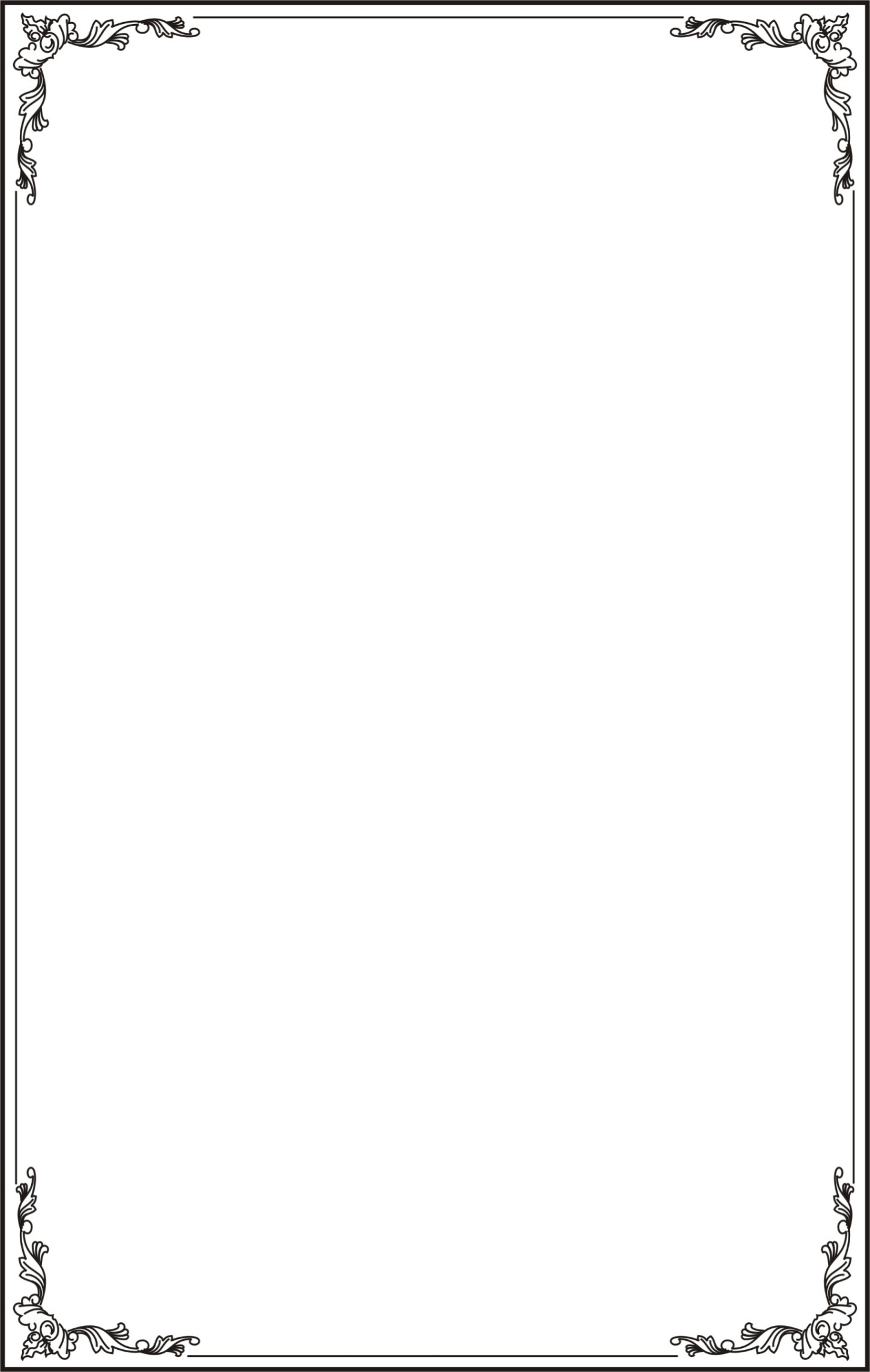
**CÔNG TY CỔ PHẦN**

**VIỄN THÔNG FPT**

****

**BÀI THU HOẠCH**

**ĐÀO TẠO TÂN BINH R&D TUẦN 3 & 4**

|  |  |
| --- | --- |
| Người hướng dẫn: | **MR. TRƯƠNG TẤN SANG** |
| Người thực hiện: | **TRẦN KHÁNH GIA UY** |
| Đơn vị công tác: | **Trung tâm giám sát và đảm bảo dịch vụ (SCC)** |
| Vị trí: | **Open Source Web Software Developer** |
| Bộ phận: | **SCC** |

***Thành phố Hồ Chí Minh, tháng 05 năm 2021***

# MỤC LỤC

[MỤC LỤC 2](#_Toc75714844)

[CHƯƠNG 1: TỐI ƯU DEMO REDUX VÀ XÂY DỰNG APP SỬ DỤNG REDUX SAGA VIẾT BẰNG TYPESCRIPT 3](#_Toc75714845)

[1.1 Tối ưu demo Redux và chuyển từ function-first thành feature-first 3](#_Toc75714846)

[1.2. Xây dựng app sử dụng Redux Saga viết bằng TypeScript 6](#_Toc75714847)

[CHƯƠNG 2: TÌM HIỂU VỀ WEBPACK VÀ JEST 9](#_Toc75714848)

[2.1 Webpack 9](#_Toc75714849)

[2.2 Jest 12](#_Toc75714850)

# CHƯƠNG 1: TỐI ƯU DEMO REDUX VÀ XÂY DỰNG APP SỬ DỤNG REDUX SAGA VIẾT BẰNG TYPESCRIPT

## Tối ưu demo Redux và chuyển từ function-first thành feature-first

* + 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:

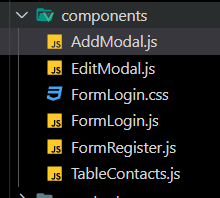
const ho\_va\_ten =  form.getFieldValue('ho\_va\_ten');

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));

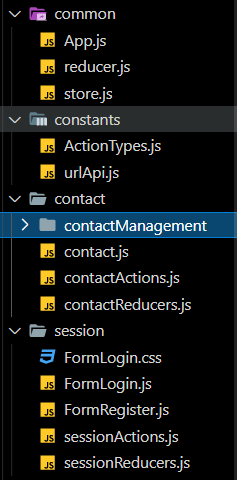
   })

   .catch((err) => {

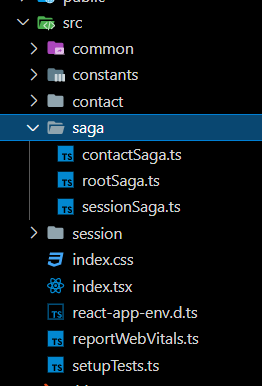
    console.log(err);

   });

* + 1. **Chuyển từ function-first thành feature-first**
* Tùy theo chức năng mà em sẽ chia nó thành một thư mục riêng (ví dụ như chức năng đăng kí đăng nhập em sẽ để thành folder *session*).
* Khi đó em tách cái file cũ trong folder action và reducer và bỏ vào các feature tương ứng
* Như vậy, với feature-first, em sẽ cấu trúc lại thư mục như sau



* 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);

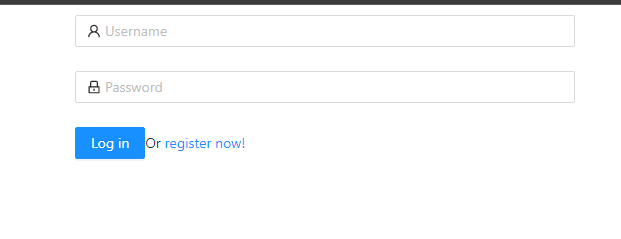
  yield takeEvery(types.REGISTER, handleRegister);

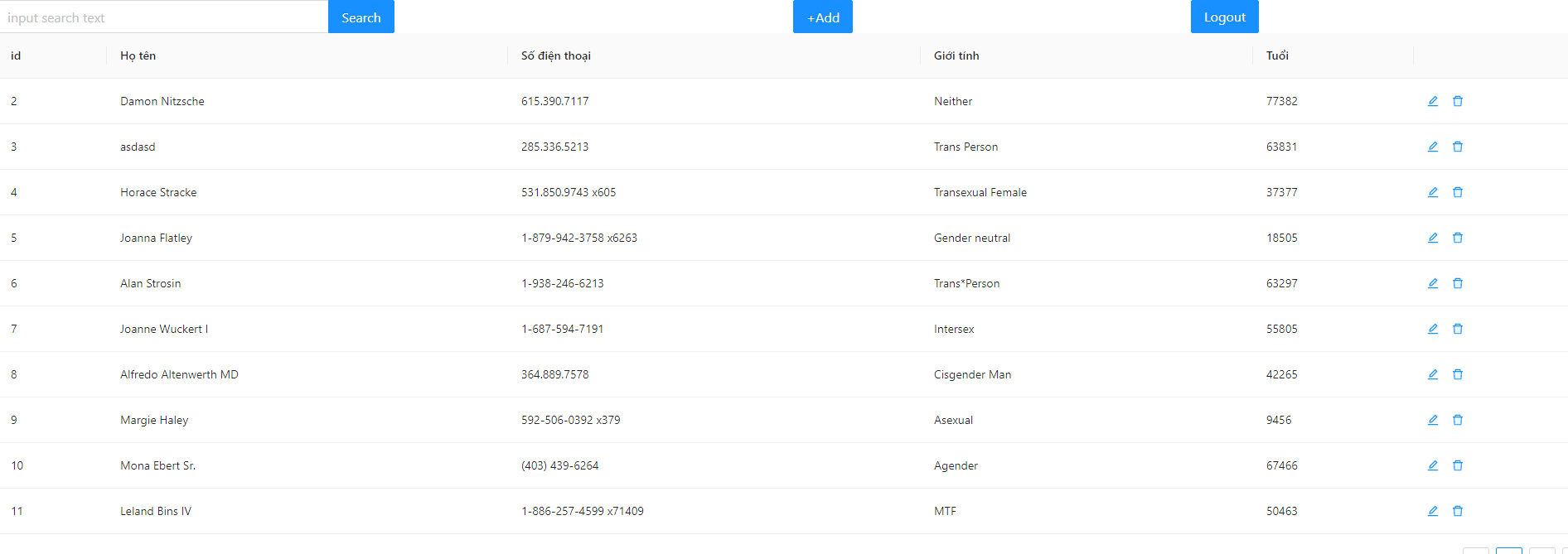
}

* 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.
  + 1. **Source code**
* Tất cả source code nằm trong link github sau:

[internSCC/React/redux-saga-ts at main · uytran36/internSCC (github.com)](https://github.com/uytran36/internSCC/tree/main/React/redux-saga-ts)

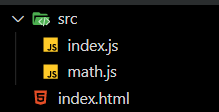
* + 1. **Demo**

****

****

# CHƯƠNG 2: TÌM HIỂU VỀ WEBPACK VÀ JEST

* 1. **Webpack**
     1. **Webpack là gì?**
* Webpack là công cụ giúp compile các module Javascript. Nó hay được gọi là “module bundler”.
  + 1. **Ưu điểm**
* Dành cho các dự án lớn, dễ dàng phát triển, quản lý và customize
* Tăng tốc độ cho project
* Phân chia các module và chỉ load khi thật sự cần
* Đóng gói tất cả các file nguồn thành 1 file duy nhất. Nhờ vào loader mà nó có thể biên dịch các loại file khác nhau.
* Biến các tài nguyên tĩnh (image, css,..) thành module.
* Biến đổi các mã nguồn: js, less, sass -> js, css or es6 -> es5 (nhằm hỗ trợ các trình duyệt)
* Đóng gói các module => sẽ giải duyết được các đường dẫn trong react
* JSX -> JS ( Browser sẽ không hiểu JSX nên webpack sẽ giúp chuyển đổi)
* Áp dụng Css preprocessor để xây dựng stylesheet (ví dụ: SASS)
  + 1. **Cài đặt**
       1. **Khởi tạo các file cơ bản:**
* Đầu tiên, em sẽ init npm vào project và tạo ra các file như sau:

****

* File ***index.html***chỉ có head và body rỗng, đồng thời import file ***bundle.js*** mà lát nữa em sẽ tạo sau thông qua webpack.
* File ***math.js*** em sẽ viết một hàm cơ bản trả về 1 số và export nó ra:

const sum = (a, b) => {

    return a + b;

}

export default sum;

* File ***index.js*** sẽ import file ***math.js*** và sẽ đưa kết quả tính toán vào trong trang html:

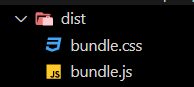
import sum from "./math";

const total = sum(3, 5);

document.write(total);

console.log(total);

* Bây giờ em sẽ cài đặt webpack và viết file config cho nó
  + - 1. **Thiết lập file config**
* Em sẽ sử dụng các module sau:
  + Babel: dùng để chuyển đổi ES6, 7, 8,.. thành ES5
  + styleLoader, cssLoader, MiniCssExtractPlugin: tải và tách file css trong webpack
* Em sẽ tạo file tên là ***webpack.config.js*** trong đó sẽ chứa thông tin các cài đặt cho webpack, các module, plugins của webpack. Ở đây em sẽ tạo tên folder chứa file là *dist* và file được tạo ra sau khi chạy webpack là ***bundle.js***
* Em sẽ chạy lệnh webpack và nó sẽ tạo ra folder như sau



* Trong đó bundle.css là những cài đặt về css trong project và bundle.js là tất cả xử lý logic cũng như khai báo trong js của project.
  + 1. **React với webpack**
* Tương tự với cách làm trên, em sẽ áp dụng webpack vào project em đã tạo với React sử dụng Ant Design.
* Để sử dụng được Ant Design, em sẽ thêm vào file ***.babelrc*** và ***webpack.config.js*** các khai báo như sau:
* ***.babelrc:***

{

  "presets": ["@babel/preset-env", "@babel/preset-react"],

  "plugins": [["import", { "libraryName": "antd", "style": true }]],

  "env": {

    "development" : {

      "compact": false

    }

  },

  "compact": true,

}

* ***webpack.config.js:*** em sẽ thêm plugin less loader để sử dụng được các file css trong Ant Design:

{

loader: "less-loader", // compiles Less to CSS

  options: {

   lessOptions: {

   javascriptEnabled: true,

   },

  },

},

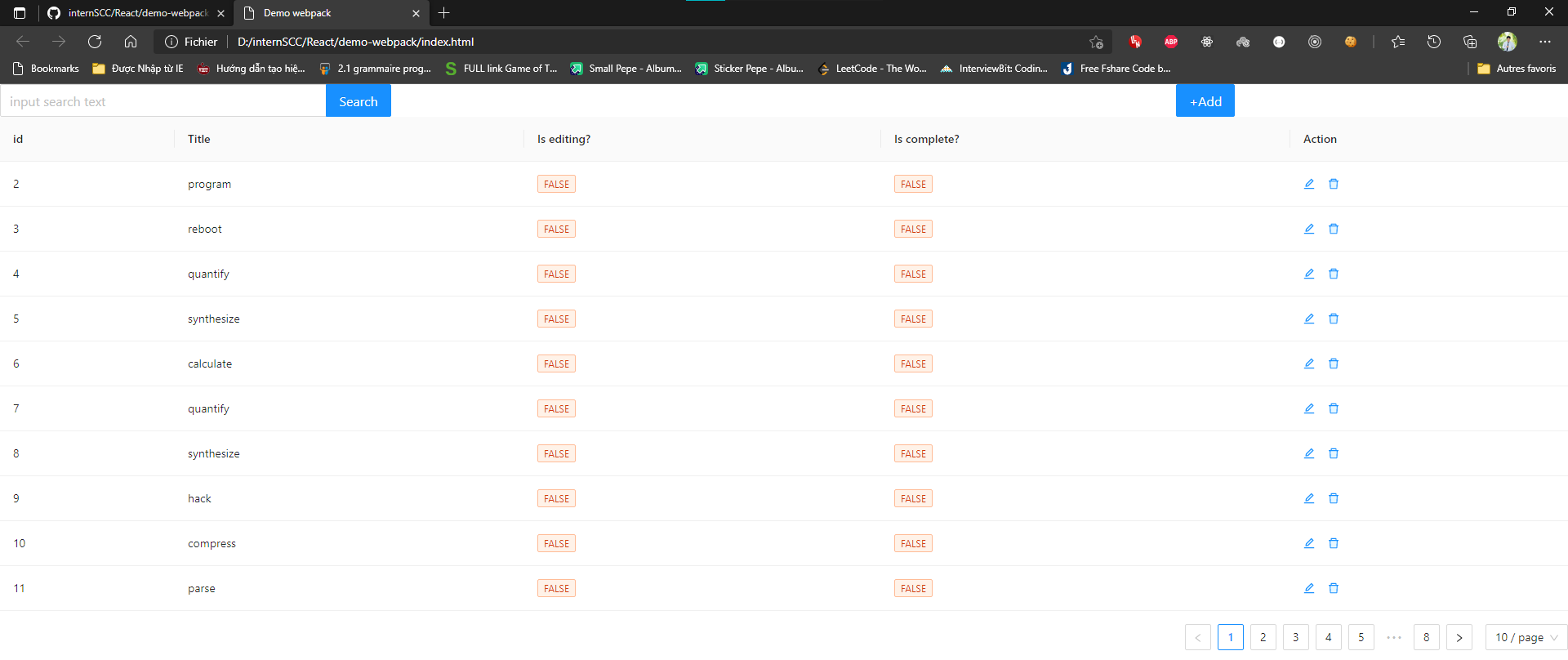
* Sau khi chạy lệnh webpack em được folder như sau :



* + 1. **Source code**
* Tất cả source code nằm trong link github sau:

[internSCC/React/demo-webpack at main · uytran36/internSCC (github.com)](https://github.com/uytran36/internSCC/tree/main/React/demo-webpack)

* + 1. **Demo**
* Khi em mở file index.html, em đã có thể sử dụng web giống như chạy server



* 1. **Jest**
     1. **Giới thiệu**
* Jest là công cụ dùng để unit testing trong React. Em sẽ sử dụng công cụ này để test các component trong React.
* Em sẽ viết các test case với yêu cầu như sau:
  + Kiểm tra xem các component đã render đúng chưa
  + Trong form đăng nhập, mật khẩu có ít nhất 6 kí tự và tối đa 20 kí tự
* Em sẽ sử dụng Jest với Enzyme để test component tiện hơn
  + 1. **Cài đặt và sử dụng Jest với Enzyme**
* Em sẽ cài đặt các thư viện sau:
  + @testing-library/jest-dom
  + Enzyme
  + enzyme-adapter-react-16
  + @babel/preset-env
  + @babel/preset-react
* Em sẽ tạo thư mục *\_\_test\_\_* để chứa file test ***FormLogin.test.js***, đồng thời em sẽ tạo file ***babel.config.js*** và ***jest.config.js*** để khai báo các cài đặt
* Trong file ***FormLogin.test.js*** em sẽ tạo 3 test cho 3 yêu cầu ở trên:
  + Kiểm tra xem component đã render đúng chưa:

describe("Form login component", () => {

  it("should render form", () => {

    const wrapper = shallow(<FormLogin />);

    const form = wrapper.find(".form-login");

    expect(form.hasClass("login-form-button"));

  });

});

* + Mật khẩu có ít nhất 6 kí tự :

describe("Password min 6 chars", () => {

  it("should have min 6 chars", () => {

    const wrapper = mount(

      <StaticRouter>

        <FormLogin />

      </StaticRouter>

    );

    let password = wrapper.find("input#normal\_login\_password");

    password.simulate("change", {

      target: { value: "12345" },

    });

    password = wrapper.find("input#normal\_login\_password")

    expect(password.props().value).toEqual("12345");

  });

});

* + Mật khẩu có tối đa 20 kí tự :

describe("Password max 20 chars", () => {

  it("should have max 20 chars", () => {

    const wrapper = mount(

      <StaticRouter>

        <FormLogin></FormLogin>

      </StaticRouter>

    );

    let password = wrapper.find("input#normal\_login\_password");

    password.simulate("change", {

      target: { value: "1234512312312123123131231231231231" },

    });

    password = wrapper.find("input#normal\_login\_password");

    expect(password.props().value).toEqual(

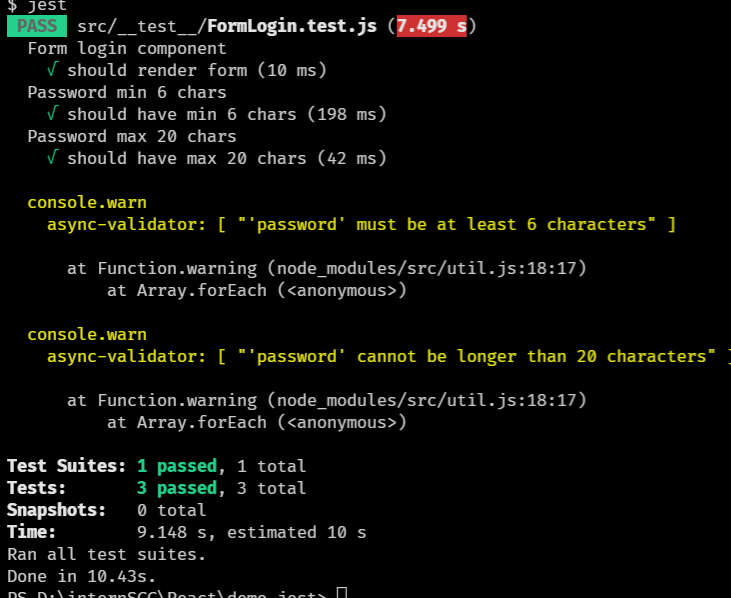
      "1234512312312123123131231231231231"

    );

  });

});

* + 1. **Kết quả**

****

* + 1. **Source code**
* Tất cả source code nằm trong link github sau:

[internSCC/React/demo-jest at main · uytran36/internSCC (github.com)](https://github.com/uytran36/internSCC/tree/main/React/demo-jest)

* + 1. **Hướng đi tiếp theo**
* Tìm hiểu Dva và UmiJS sau đó xây dựng app CRUD cơ bản thông qua UmiJS và Dva