TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



**Software Detail Design**

**Đồ án Project I – IT3150**

**Đề tài: Trang web quản lý công việc - workflow management**

**Giáo viên hướng dẫn: TS. Đào Thành Chung**

**Sinh viên thực hiện: Nguyễn Thanh Tú MSSV: 20184002**

**Hà Nội, Tháng 1 năm 2021**

Nội dung

[**CHƯƠNG 1: GIỚI THIỆU** 3](#_Toc61021972)

[1.1. Mục đích 3](#_Toc61021973)

[1.2. Cấu trúc tài liệu 3](#_Toc61021974)

[**CHƯƠNG 2: MÔ HÌNH HÓA CẤU TRÚC HỆ THỐNG** 4](#_Toc61021975)

[2.1. Các tầng và mối quan hệ trên hệ thống 4](#_Toc61021976)

[Cấu trúc hướng điều kiển trong redux: 4](#_Toc61021977)

[Flow tương tác giữa redux và react: 4](#_Toc61021978)

[Flow khi tương tác redux, react application với api server: 4](#_Toc61021979)

[Các thành phần Redux 5](#_Toc61021980)

[**CHƯƠNG 3: THIẾT KẾ GIAO DIỆN** 9](#_Toc61021981)

[5.1. Giao diện đăng nhập 9](#_Toc61021982)

[5.2. Giao diện quản lý dự án 10](#_Toc61021983)

[5.3. Giao diện thêm công việc 10](#_Toc61021984)

[5.4. Giao diện cập nhật công việc 11](#_Toc61021985)

[5.5. Giao diện xem thông tin cá nhân 12](#_Toc61021986)

# **CHƯƠNG 1: GIỚI THIỆU**

## 1.1. Mục đích

* Tài liệu này là bản thiết kế chi tiết hệ thống, bao gồm các thiết kế: phân tích chi tiết lớp (analyst class), giao diện người dùng (graphic userinterface).
* Dựa trên phân tích có trong tài liệu đặc tả SRS đưa ra quy trình hoạt động của hệ thống thông qua các biểu đồ phân tích chi tiết lớp và biểu đồ tuần tự (sequence diagram) tương ứng cho các hành vi của từng ca sử dụng.
* Đưa ra thiết kế cơ sở dữ liệu cụ thể, cơ sở dữ liệu cho phép lưu trữ thông tin về người dùng, thông tin công việc.
* Từ tài liệu thiết kế này, nhóm phát triển hệ thống có cơ sở để lập trình phát triển, hình thành hệ thống cho người dùng. Thuận tiện cho việc kết nối các module của hệ thống với nhau.

## 1.2. Cấu trúc tài liệu

Bản thiết kế này chia làm 5 phần chính:

* 1. Mô hình hóa cấu trúc hệ thống: Cấu trúc hệ thống tổng quan
  2. Thiết kế lớp chi tiết: các lớp tham gia hệ thống
  3. Mô hình hóa hành vi hệ thống: Trình tự hoạt động
  4. Thiết kế cơ sở dữ liệu
  5. Thiết kế giao diện người dùng

# **CHƯƠNG 2: MÔ HÌNH HÓA CẤU TRÚC HỆ THỐNG**

## 2.1. Các tầng và mối quan hệ trên hệ thống

### Cấu trúc hướng điều kiển trong redux:

* Redux dựa theo cấu trúc flux do facebook đề xuất, theo mô hình data flow một hướng, nhằm tránh có cuộc gọi chồng chéo.
* Tất các trạng thái đều được lưu giữ một nơi là store, các view sẽ lấy các thay đổi từ store để hiển thị.
* Các sự kiện từ view phải gửi action để update trạng thái trong store.
* Reducer dựa vào các action để update state trong store.
* State là plain javascript object, không được biến đổi nó trực tiếp mà phải thông tạo object mới cho reducer trả về.

### Flow tương tác giữa redux và react:

Một điều dễ nhận ra là khi áp dụng Redux, mặc dù nó tốt thiệt – giúp cho ứng dụng dễ hiểu, dễ bảo trì, nhưng lượng code phải viết sẽ nhiều lên kha khá. Cứ mỗi chức năng, ta cần phải viết Actions/Events -> Reducers -> Container Component -> Presentation Component, tương ứng bạn cũng sẽ viết Test cho chừng đó bước, vì vậy việc sử dụng Application State một cách hợp lý là điều cực kỳ quan trọng, giúp cho ứng dụng gọn nhẹ về mặt cấu trúc, code, và tài nguyên (bộ nhớ để lưu trữ Store).

### Flow khi tương tác redux, react application với api server:

Việc tương tác với api server phải thông qua các cuộc gọi từ xa và phải được xử lí bất đồng bộ: Để có thể chờ các cuộc gọi bất đồng bộ trước khi xử lí và update view ta cần middleware trong redux cho việc xử lí các action này, dùng thư viện redux-thunk trong các bộ api hay các action phải chờ thời gian xử lí.

### Các thành phần Redux

**Actions**

Action là những đối tượng mô tả cách chúng ta muốn thay đổi state. Bạn có thể hình dung action như là những API cho state của bạn.

VD: ta có TodoActions định nghĩa ra các thao tác thêm sửa xóa... công việc. Các action mang type và các thông tin id, text.

// actions/TodoActions.js

import \* as types from '../constants/ActionTypes';

export function addTodo(text) {

return {

type: types.ADD\_TODO,

text

};

}

export function deleteTodo(id) {

return {

type: types.DELETE\_TODO,

id

};

}

export function editTodo(id, text) {

return {

type: types.EDIT\_TODO,

id,

text

};

}

Note: Action trong Redux là một object mà bắt buộc phải có ít nhất 1 property là "type". Mục đích của prop Type này là để Reducer nhận biết được là Action nào. Action ngoài prop type nó còn có thể bao gồm nhiều props khác để truyền data qua Reducer.

**Reducers**

Chúng ta tạo ra các sub-reducers và một root-reducer quản lý chung, root-reducer là tham số để truyền vào khi tạo store. Reducers là các pure function hoạt động theo nguyên lý : (state, action) => (new state)

Vì là pure function nên các reducers sẽ không trực tiếp thay đổi state mà nó nhận được, mà tạo ra các bản copy và thay đổi trên đó. Để thực hiện điều này chúng ta có thể dùng các function filter() map() Object.assign()

// reducers/todosReducers.js

import { ADD\_TODO, DELETE\_TODO, EDIT\_TODO, MARK\_TODO, MARK\_ALL, CLEAR\_MARKED } from '../constants/ActionTypes';

const initialState = [{

text: 'Use Redux',

marked: false,

id: 0

}];

export default function todos(state = initialState, action) {

switch (action.type) {

case ADD\_TODO:

return [{

id: (state.length === 0) ? 0 : state[0].id + 1,

marked: false,

text: action.text

}, ...state];

case DELETE\_TODO:

return state.filter((todo) => todo.id !== action.id);

case EDIT\_TODO:

return state.map((todo) => todo.id === action.id ? { ...todo, text: action.text } : todo);

default:

return state;

}

}

Root-reducer sẽ tập hợp các sub-reducers lại thông qua combineReducers() của Redux.

// reducers/rootReducers.js

import { combineReducers } from 'redux';

import todosReducers from './todosReducers';

const rootReducer = combineReducers({

todosReducers

});

export default rootReducer;

**Containers**

Chúng ta có 1 container là TodoApp:

// containers/TodoApp.js

import React, { Component } from 'react';

import { connect } from 'react-redux';

import { bindActionCreators } from 'redux';

import Header from '../components/Header';

import MainSection from '../components/MainSection';

import \* as TodoActions from '../actions/TodoActions';

class TodoApp extends Component {

render() {

const { todos, actions } = this.props;

return (

<div>

<Header addTodo={actions.addTodo} />

<MainSection todos={todos} actions={actions} />

</div>

);

}

}

function mapStateToProps(state) {

return {

todos: state.todosReducers

};

}

function mapDispatchToProps(dispatch) {

return {

actions: bindActionCreators(TodoActions, dispatch)

};

}

export default connect(mapStateToProps, mapDispatchToProps)(TodoApp);

**Component**

Component thông thường, chúng không giao tiếp với Redux, chỉ nhận giá trị và thao tác thông qua props.

Các xử lý hiển thị dữ liệu sẽ thực thi ở đây và các action nhận được từ container sẽ sử dụng như callback.

Store

Chúng ta sử dụng hàm createStore() với tham số là root-reducer

import { createStore } from 'redux';

import rootReducer from './reducers/rootReducer';

// initialState

const initialState = {}

// Create store

const store = createStore(rootReducer, initialState);

**Khởi tạo Root Component**

App sử dụng Redux root component đảm nhận thêm việc khởi tạo store và bao các component lại với Provider của react-redux giúp component có thể giao tiếp với redux.

// index.js

import 'todomvc-app-css/index.css';

import React from 'react';

import ReactDOM from 'react-dom';

import { createStore } from 'redux';

import { Provider } from 'react-redux'

import TodoApp from './containers/TodoApp';

import rootReducer from './reducers/rootReducer';

// initialState

const initialState = {}

// Create store

const store = createStore(rootReducer, initialState);

const appRoot = (

<Provider store={store}>

<div>

<TodoApp />

</div>

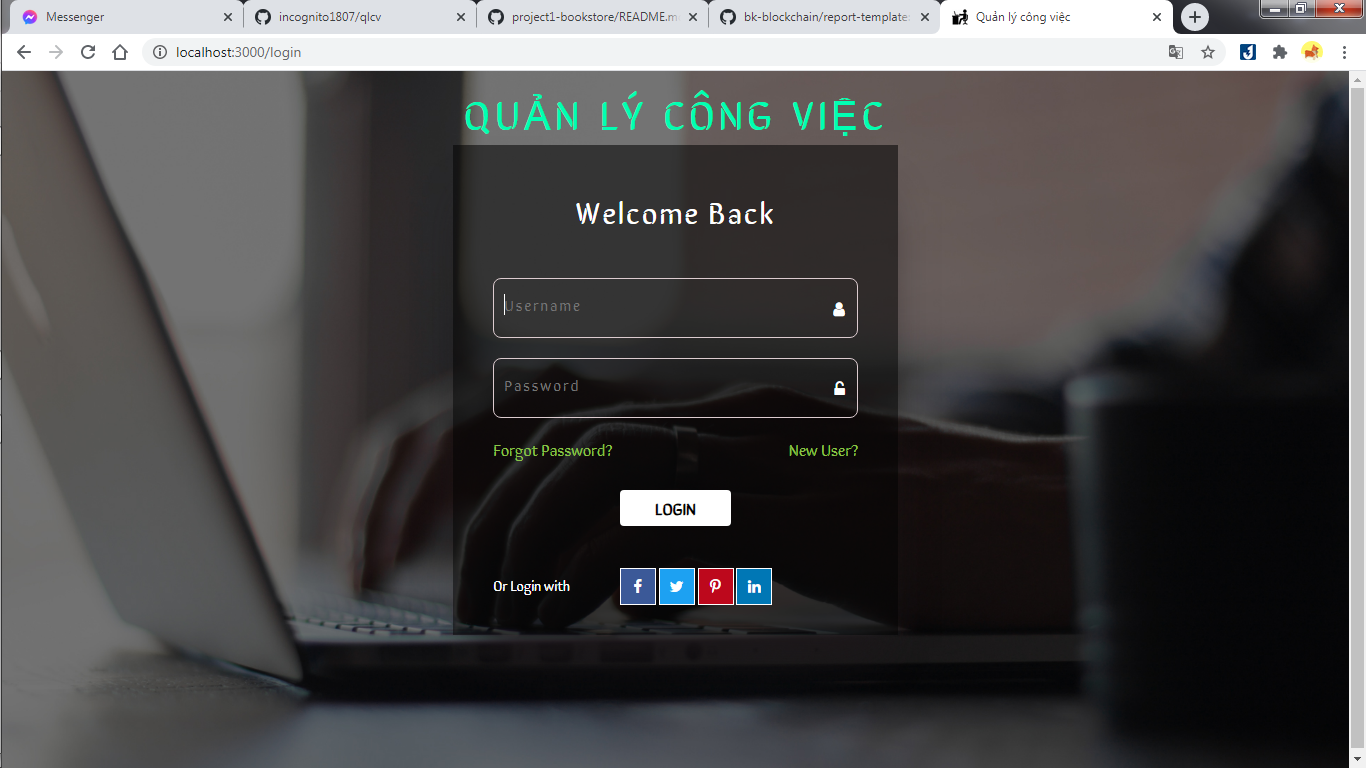
</Provider>

)

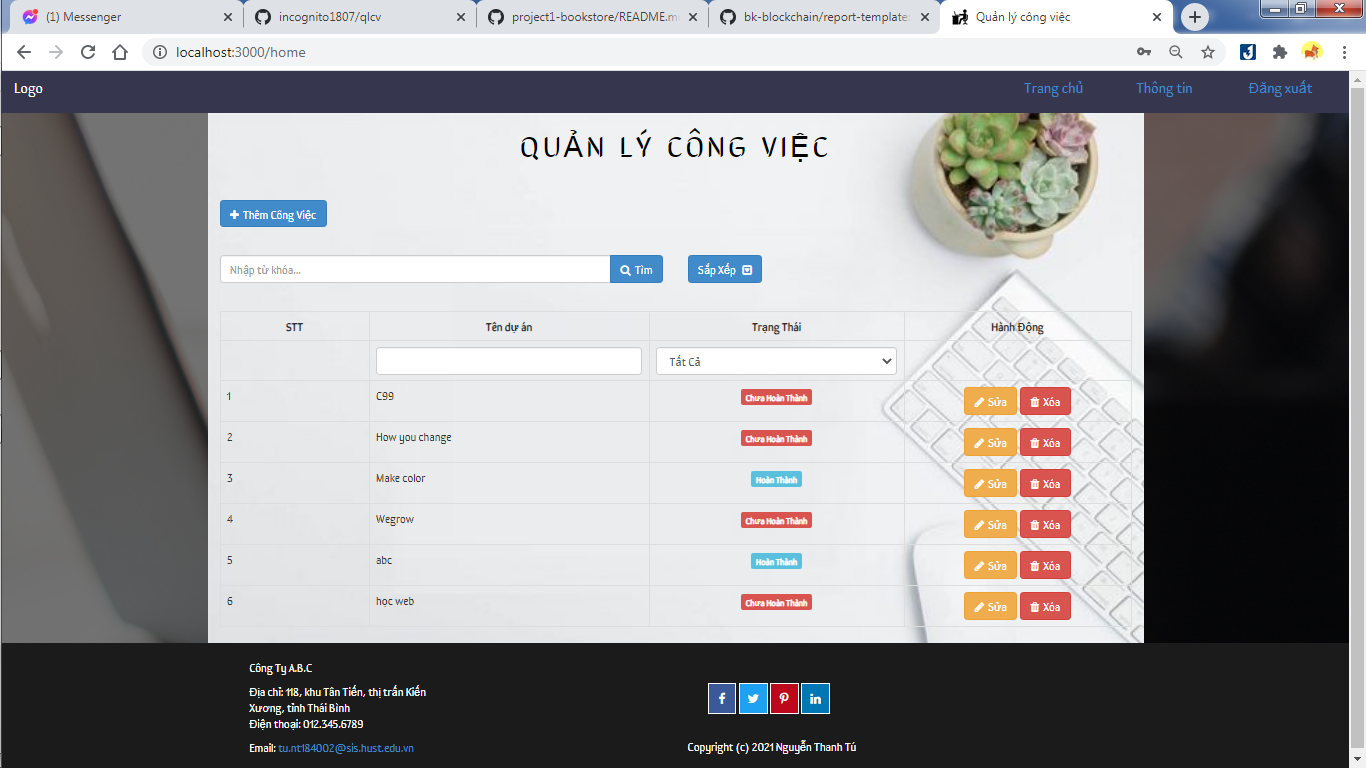
ReactDOM.render(appRoot, document.getElementById('root'))

# **CHƯƠNG 3: THIẾT KẾ GIAO DIỆN**

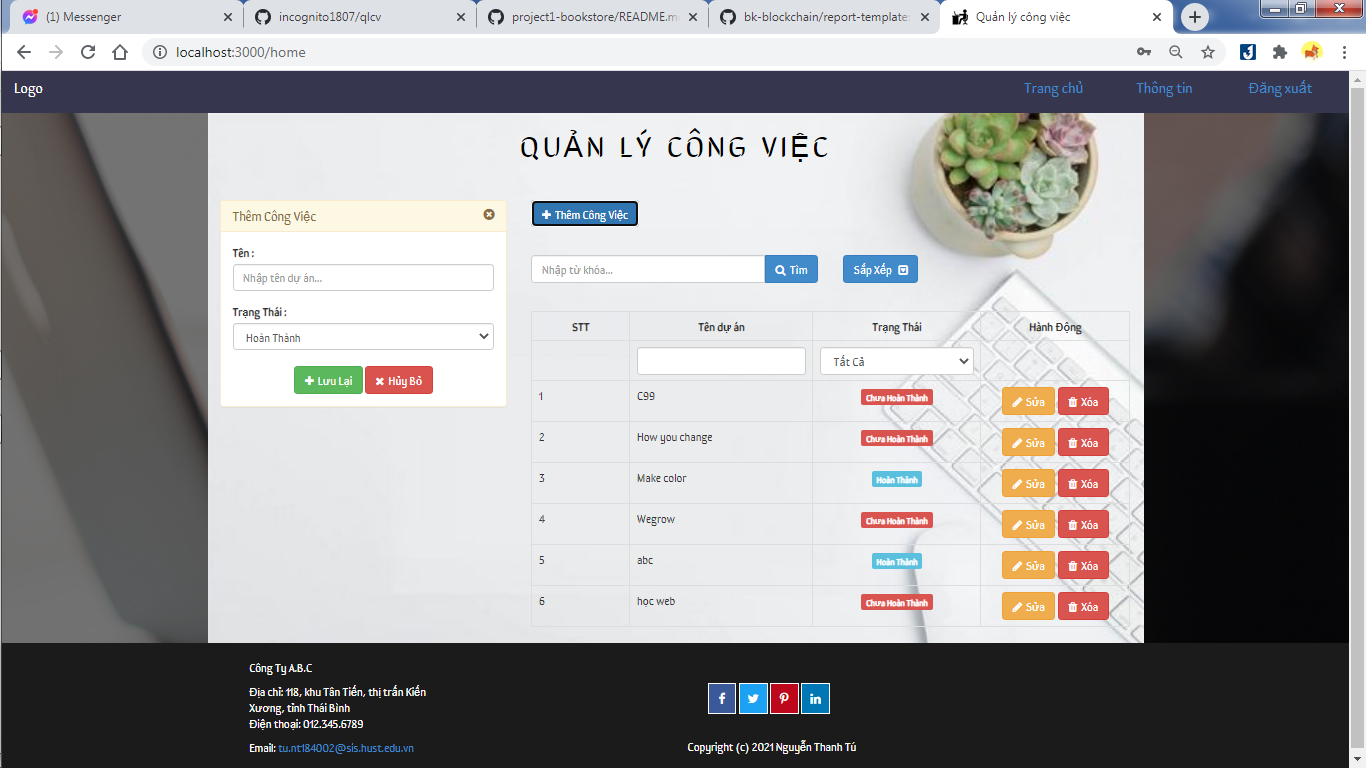
## 5.1. Giao diện đăng nhập



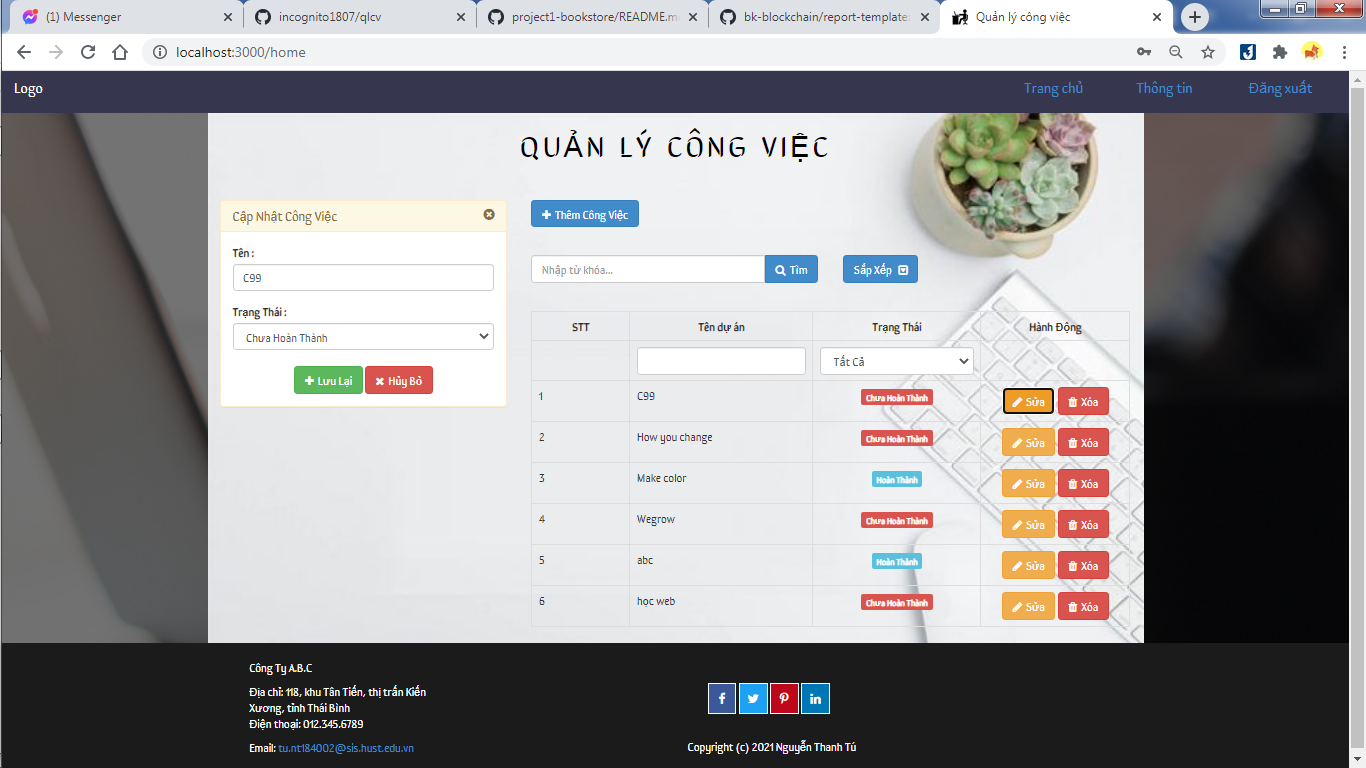
## 5.2. Giao diện quản lý dự án



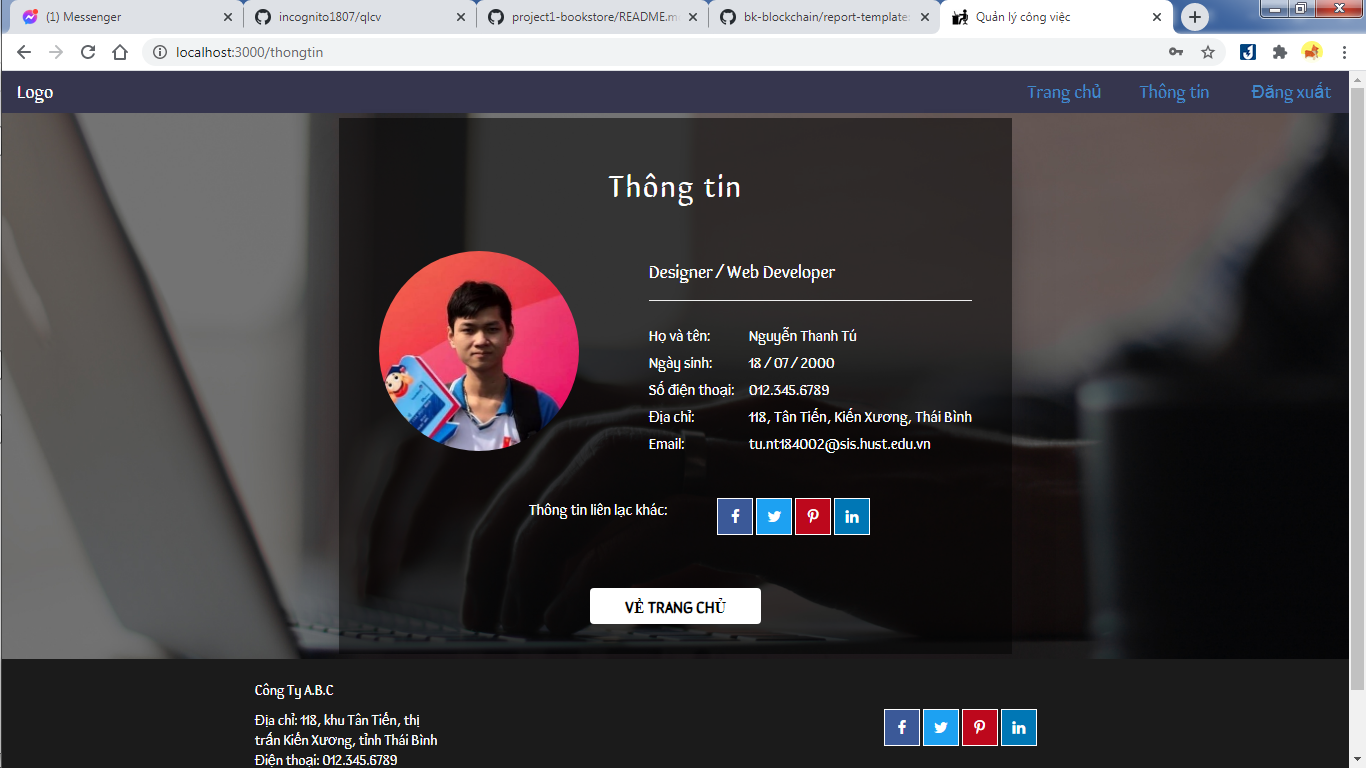
## 5.3. Giao diện thêm công việc



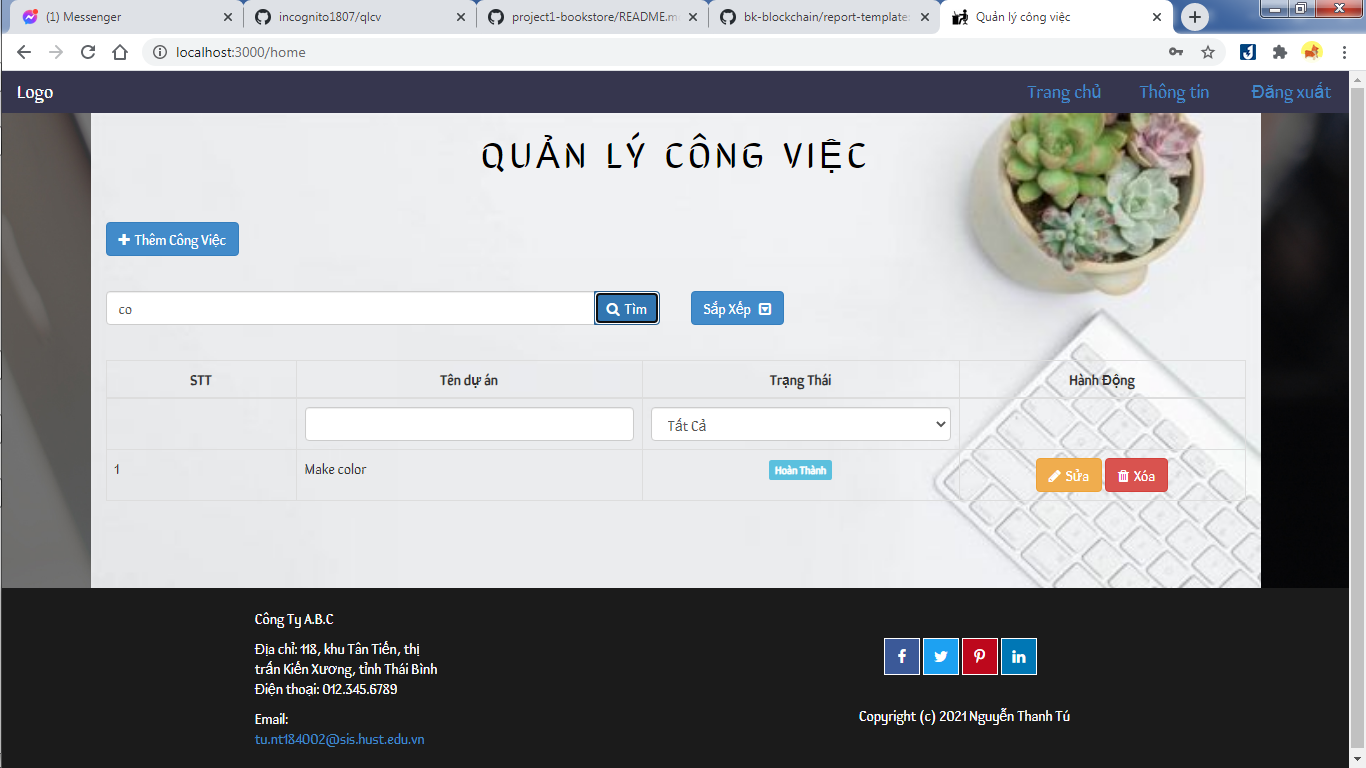
## 5.4. Giao diện cập nhật công việc



## 5.5. Giao diện xem thông tin cá nhân



## 5.6. Giao diện tìm công việc



## 5.7. Giao diện sắp xếp công việc

