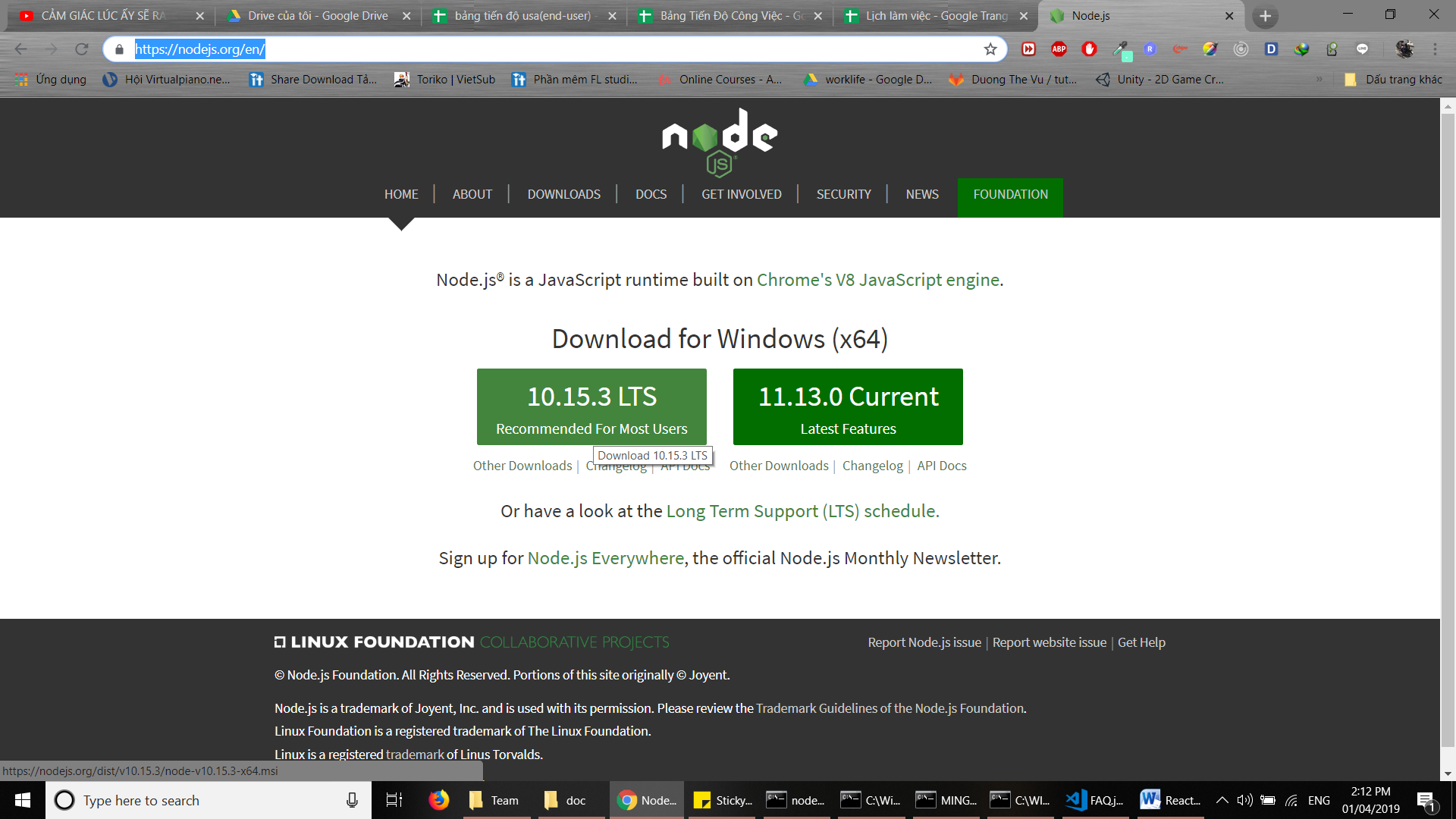
Doc này sẽ hướng dẫn các bạn cài đặt react native và tạo một app project với 4 chức năng cơ bản: thêm, sửa, xóa, đọc (CRUD).

1. Cài đặt môi trường:  
   +) Cài nodejs: <https://nodejs.org/en/>



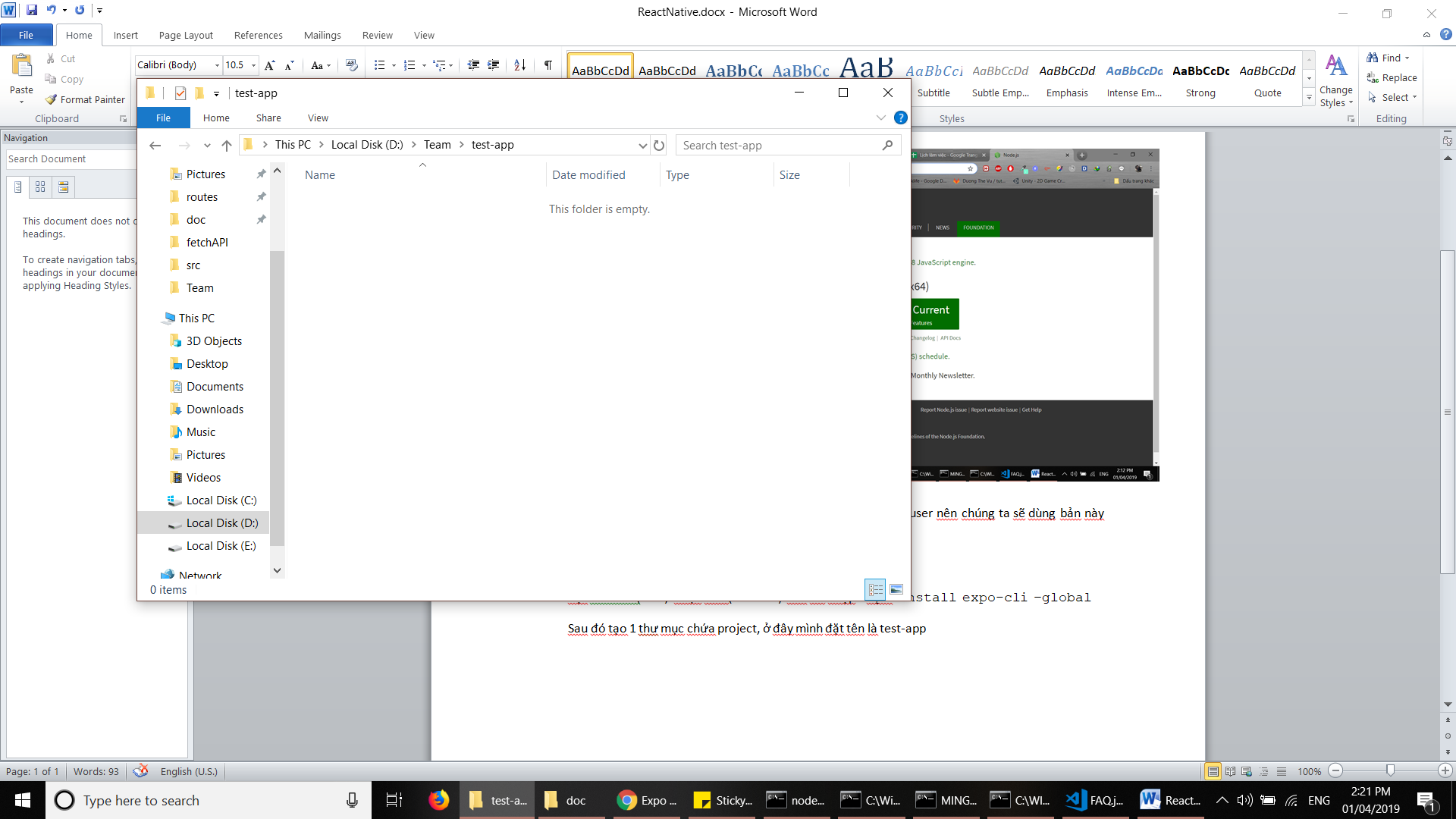
Tải bản 10.15.3 nhé. ở đây người ta recommended for most user nên chúng ta sẽ dùng bản này cho đảm bảo tính ổn định của version.

+) Cài expo:

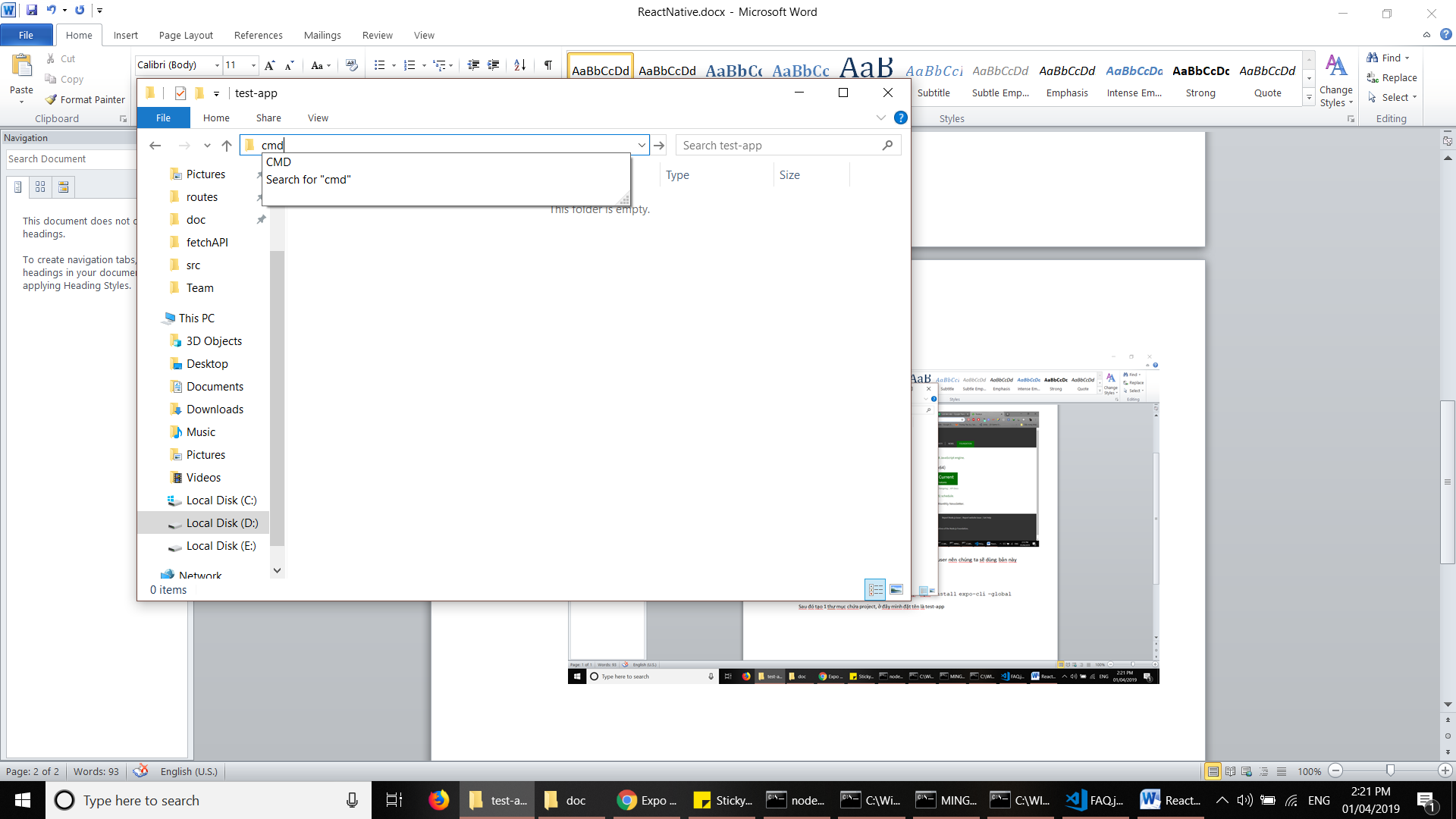
Bật terminal(Mac) hoặc cmd(Window) sau đó nhập : npm install expo-cli –global

1. Tạo project:

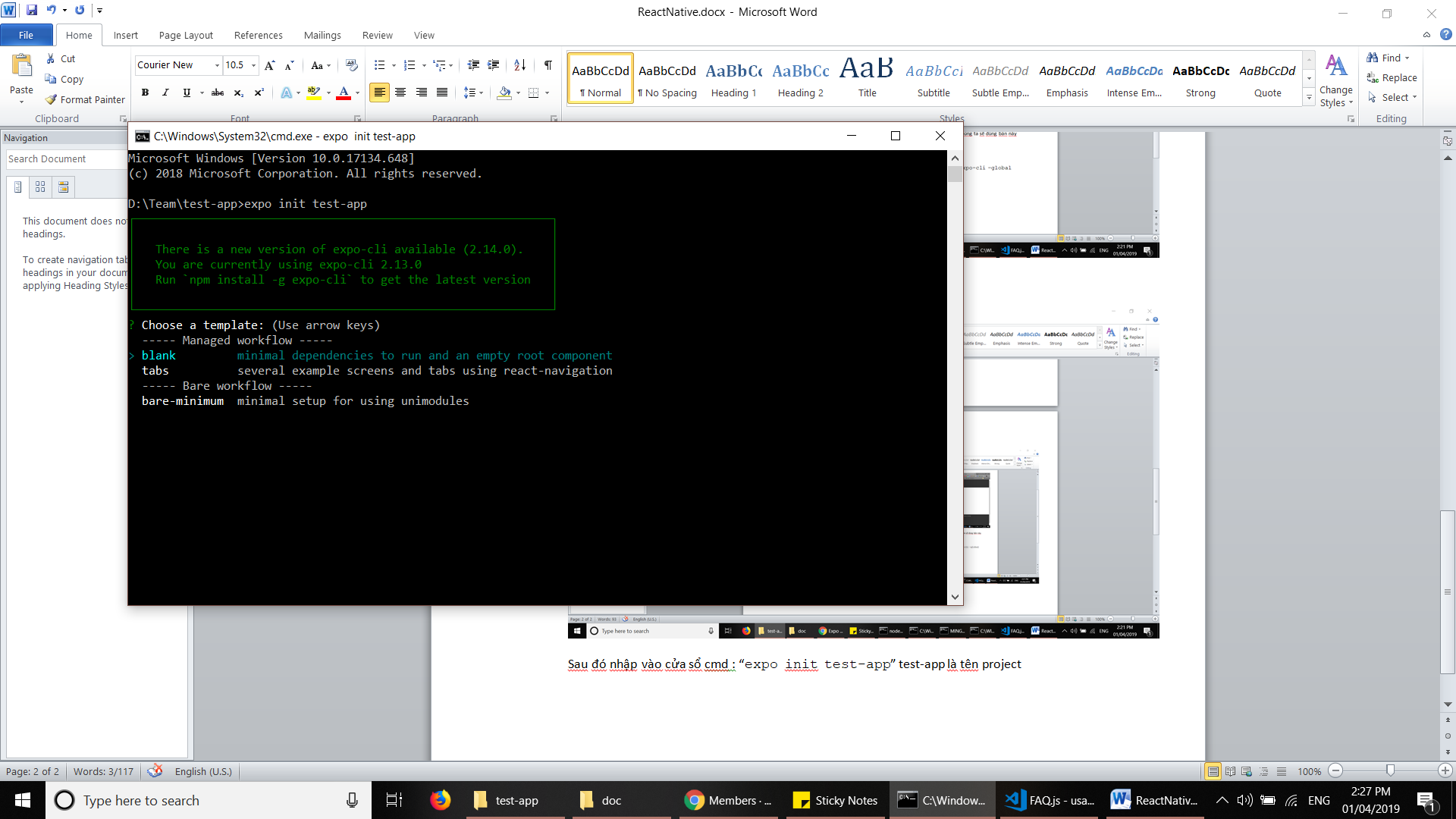
Tạo 1 thư mục chứa project, ở đây mình đặt tên là test-app



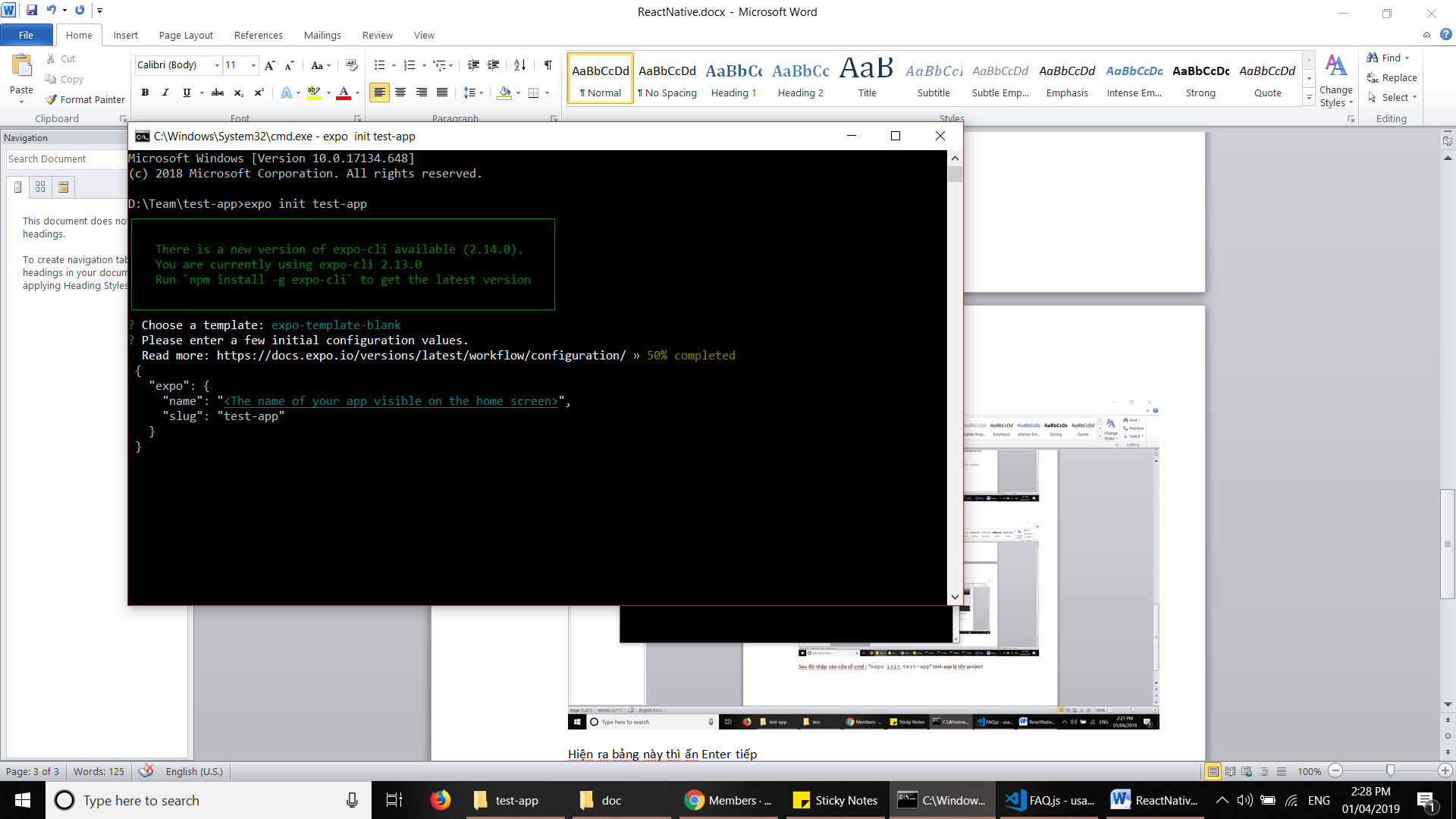
Nhập “cmd” trên đường dẫn của thư mục đó



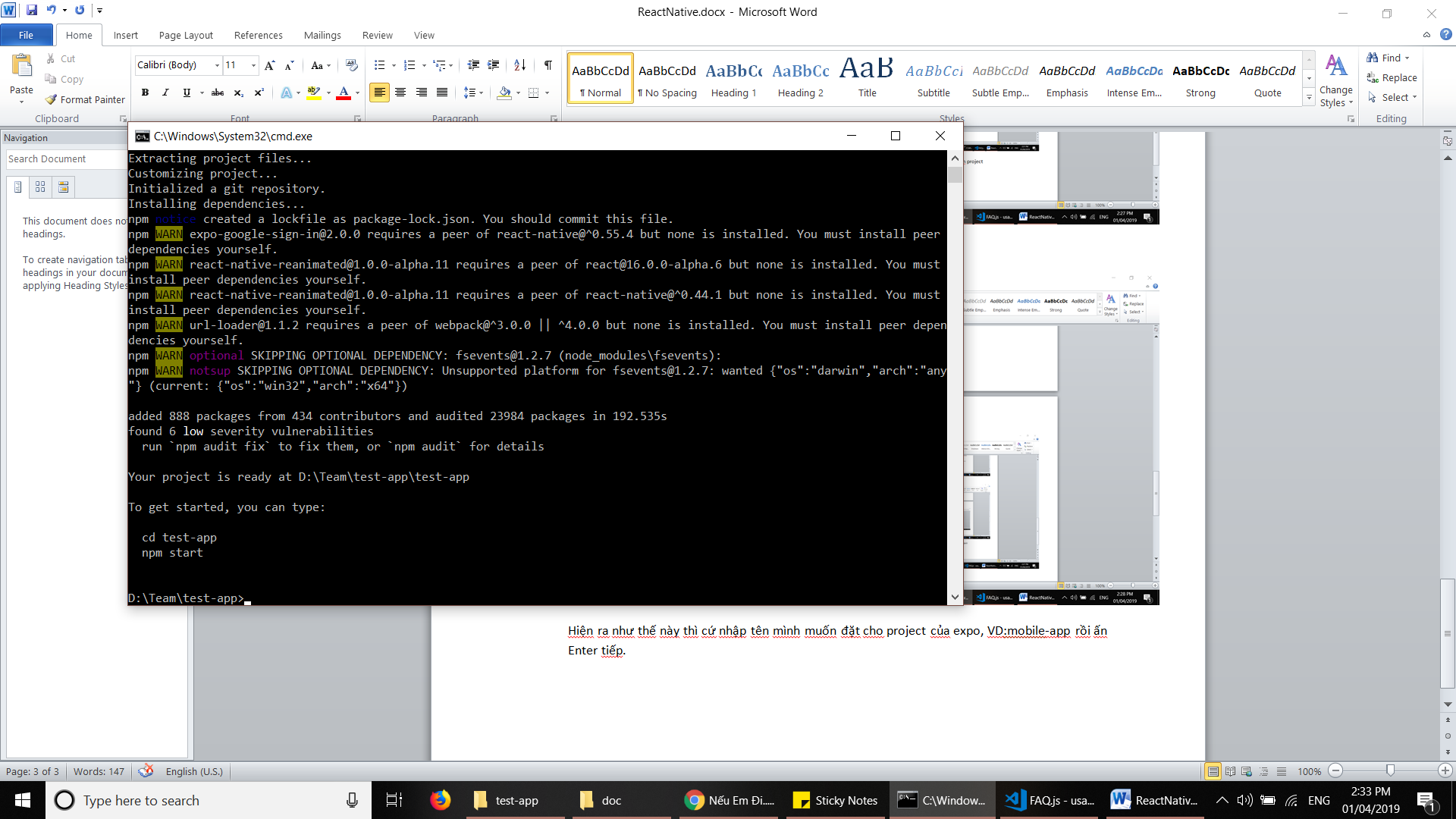
Sau đó nhập vào cửa sổ cmd : “expo init test-app” test-app là tên project

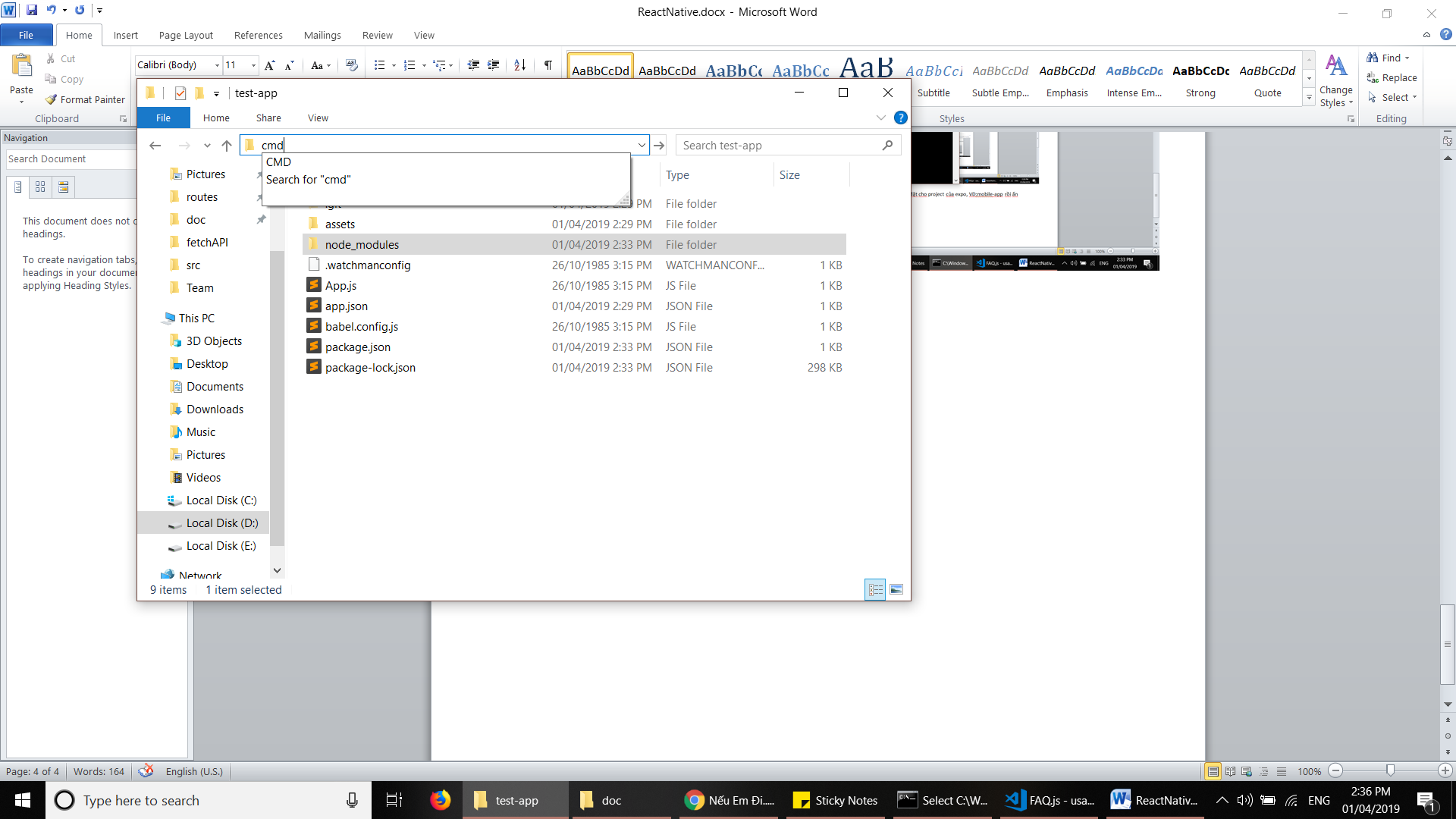


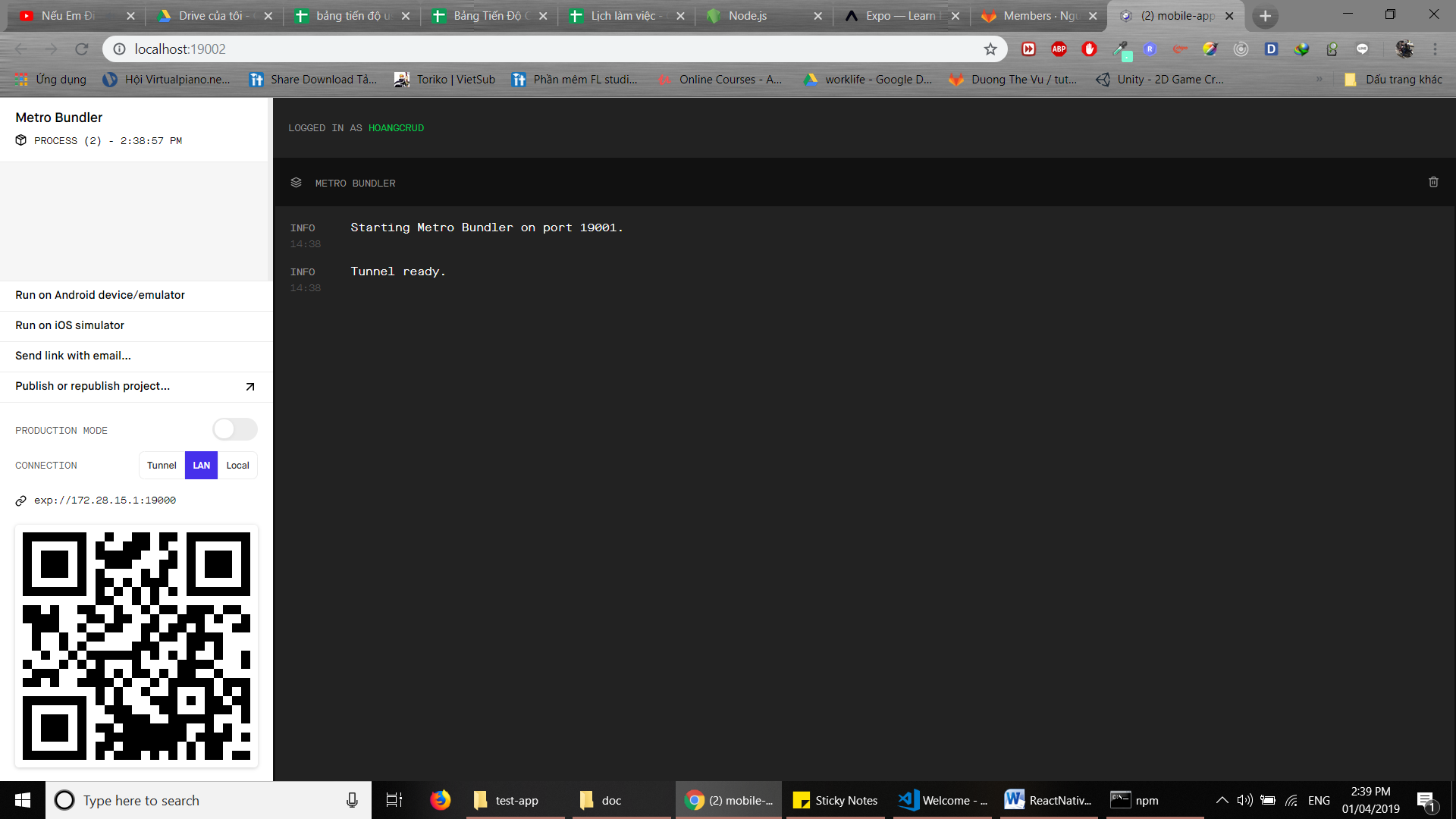
Hiện ra như thế này thì ấn Enter tiếp

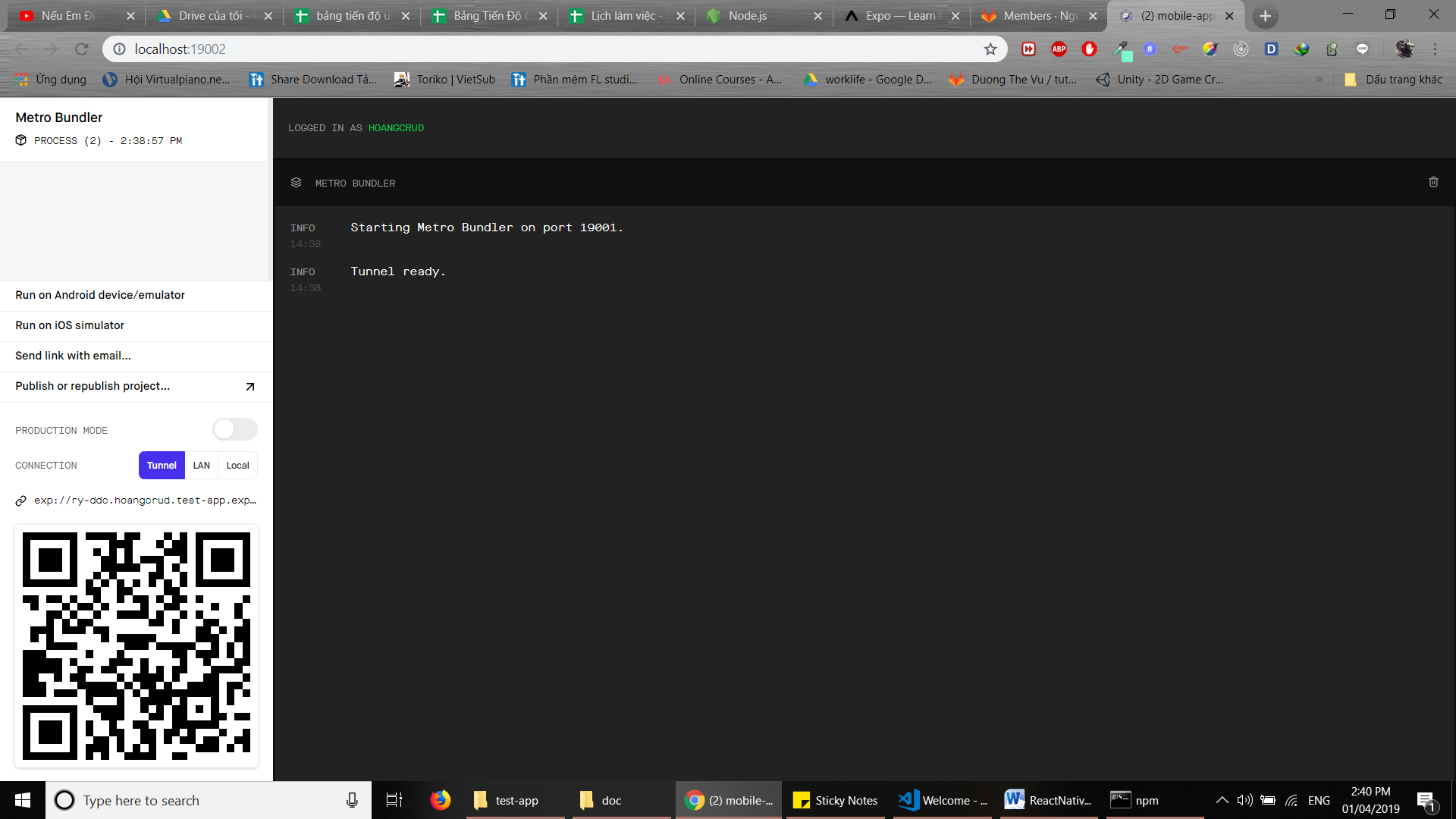


Hiện ra như thế này thì cứ nhập tên mình muốn đặt cho project của expo, VD:mobile-app rồi ấn Enter tiếp.

Tạo project xong sẽ hiển thị như thế này 

Trỏ vào thư mục chứa thư mục node\_modules và bật cmd: 

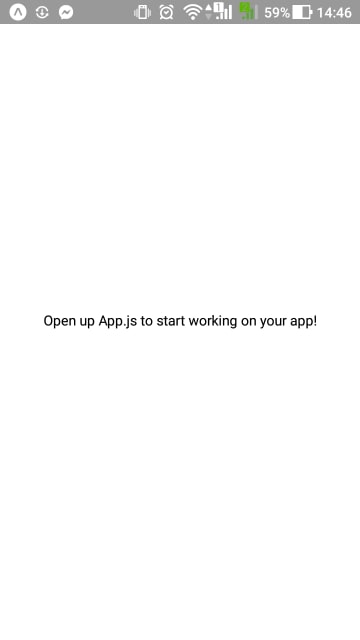
Sau đó gõ “expo start”, chạy xong sẽ bật lên cửa sổ như sau: 

Chuyển Connection từ LAN thành Tunnel: 

OK, giờ các bạn tải app expo về điện thoại. cứ search “expo“ trên appstore hoặc chplay là có:

Nếu dùng iphone thì quét mã bằng máy ảnh, hệ thống sẽ tự động gợi ý bật expo. Còn android thì vào app expo sẽ có thao tác “Scan QR Code”. Chúng ta sẽ quét cái hình ở góc dưới màn hình bên trái kia nhé.

Quá trình quét rồi chạy project bằng expo này sẽ mất 1 chút thời gian load. Sau khi chạy xong sẽ có màn hình này:



Ok vậy là đã chạy được 1 blank project. Giờ chúng ta bắt đầu tạo project crud.

Về cơ bản thì react-native khá giống reactjs. Về bố trí thư mục, mình sẽ bố trí như sau:  
-actions: chứa các action được định nghĩa trước

-screens: chứa các màn hình trong ứng dụng(ở reactjs sẽ là pages chứa các trang trong project)

-containers: chứa các container, nơi connect redux

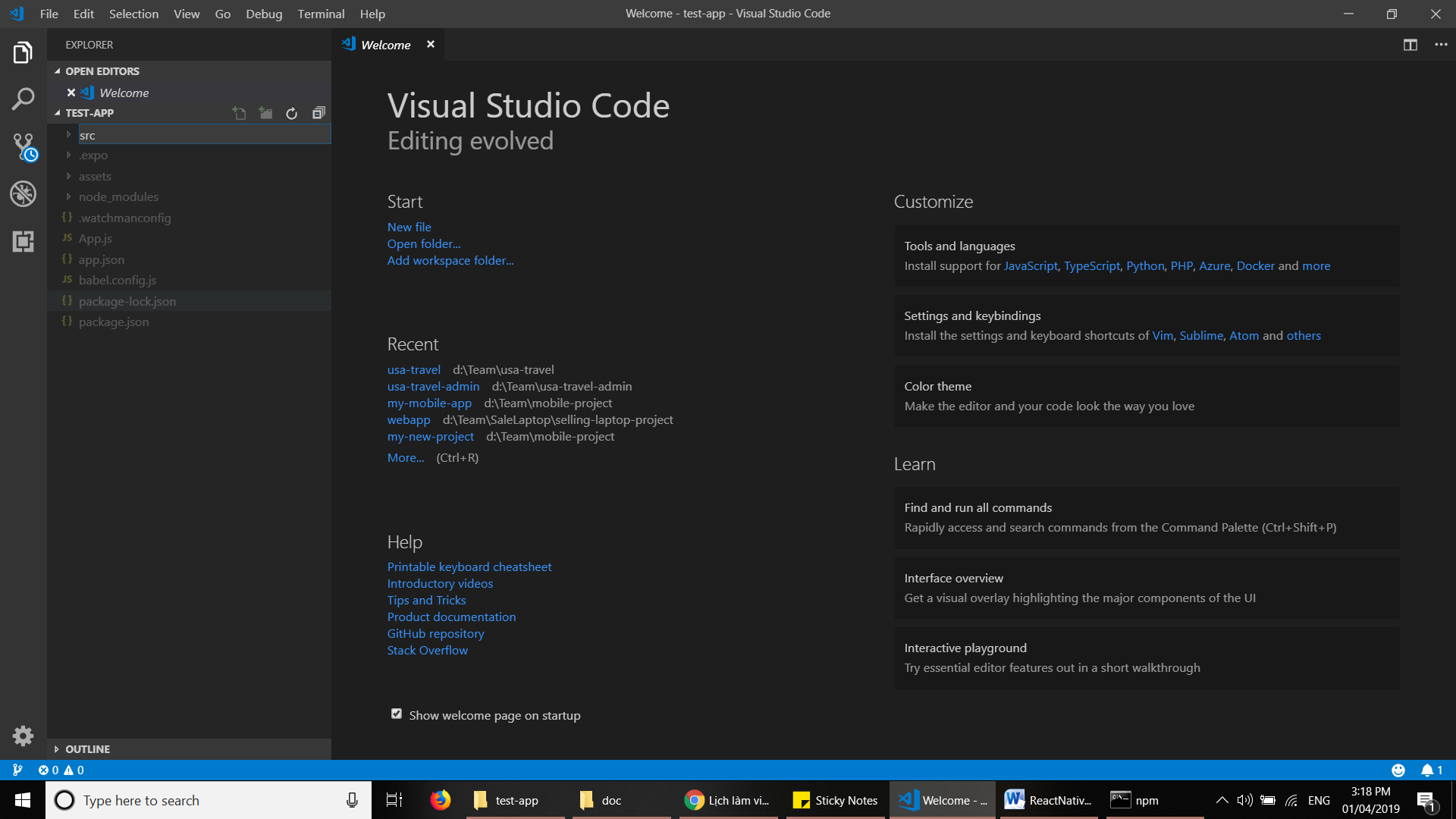
-components: chứa các component, các thành phần hiển thị trên ứng dụng

-sagas: chứa các file saga để nhận action gọi api  
-fetchAPIs: chứa các hàm call API (gọi sang backend để lấy dữ liệu từ database)

-reducers: chứa các file reducer, nơi sẽ nhận các action và cập nhật store

BẮT TAY VÀO LÀM:

Trong ví dụ này mình sẽ làm một ứng dụng của crud đó là một To do list app. App này sẽ cho phép chúng ta tạo ra một list công việc cần làm, và có thể thêm sửa xóa trong list đó.

Đầu tiên, vào project ban nãy chúng ta đã tạo ra, tạo 1 thư mục “src” 

Tạo 1 file constants.js trong thư mục src, file này sẽ chứa tất cả constant của project, ví dụ là các type của action. File constants có nội dung như sau:

//type of actions

export const GET\_TODOS\_REQUEST = 'GET\_TODOS\_REQUEST';

export const GET\_TODOS\_SUCCESS = 'GET\_TODOS\_SUCCESS';

export const GET\_TODOS\_FAILURE = 'GET\_TODOS\_FAILURE';

export const ADD\_TODOS\_REQUEST = 'ADD\_TODOS\_REQUEST';

export const ADD\_TODOS\_SUCCESS = 'ADD\_TODOS\_SUCCESS';

export const ADD\_TODOS\_FAILURE = 'ADD\_TODOS\_FAILURE';

export const UPDATE\_TODOS\_REQUEST = 'UPDATE\_TODOS\_REQUEST';

export const UPDATE\_TODOS\_SUCCESS = 'UPDATE\_TODOS\_SUCCESS';

export const UPDATE\_TODOS\_FAILURE = 'UPDATE\_TODOS\_FAILURE';

export const DELETE\_TODOS\_REQUEST = 'DELETE\_TODOS\_REQUEST';

export const DELETE\_TODOS\_SUCCESS = 'DELETE\_TODOS\_SUCCESS';

export const DELETE\_TODOS\_FAILURE = 'DELETE\_TODOS\_FAILURE';

Tiếp tục tạo 1 file db.json trong thư mục src với nội dung như sau:

{

"todos": [

{

"id": 1,

"content": "Task 1"

},

{

"id": 2,

"content": "Task 2"

},

{

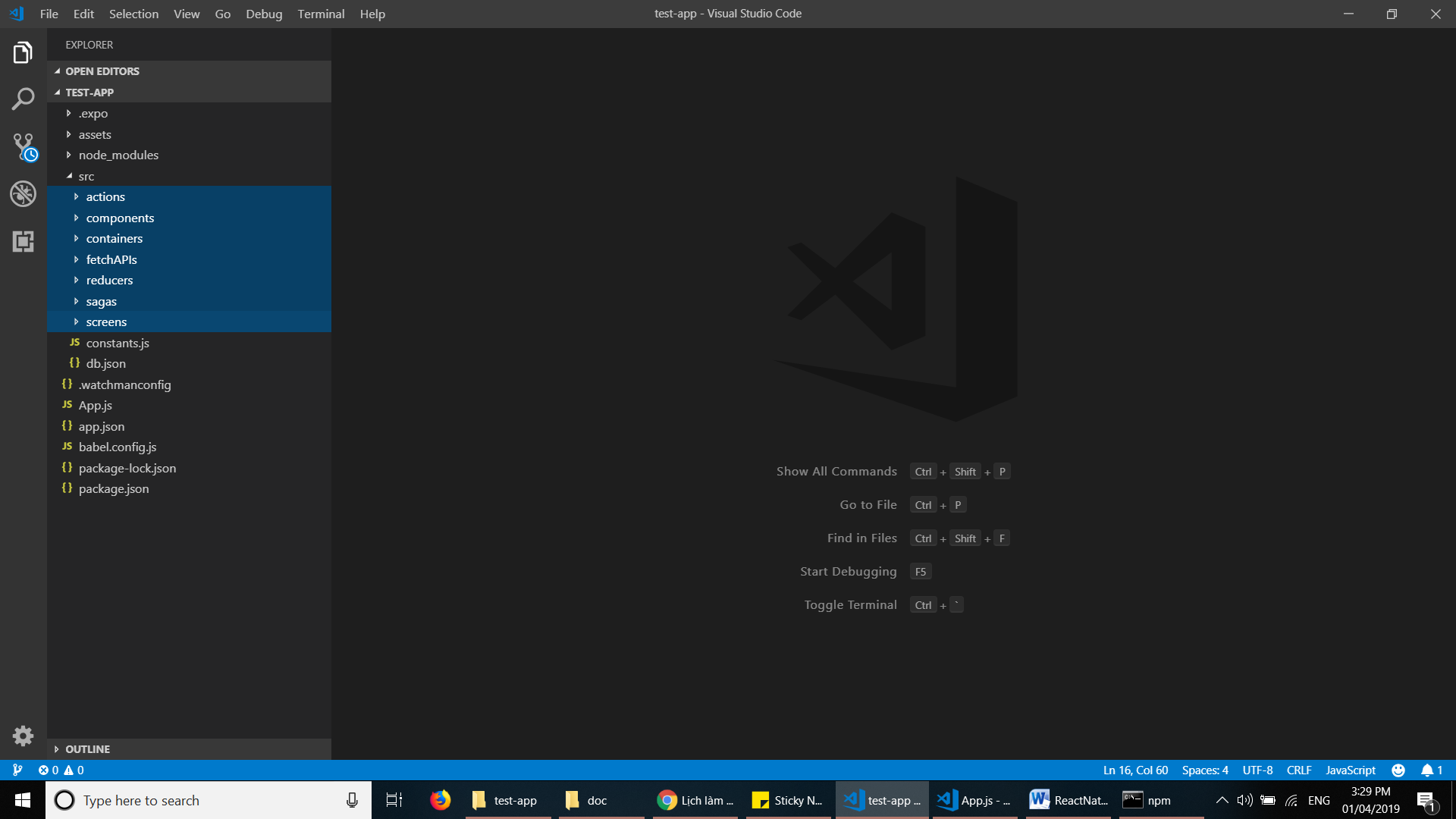
"id": 3,

"content": "Task 3"

}

]

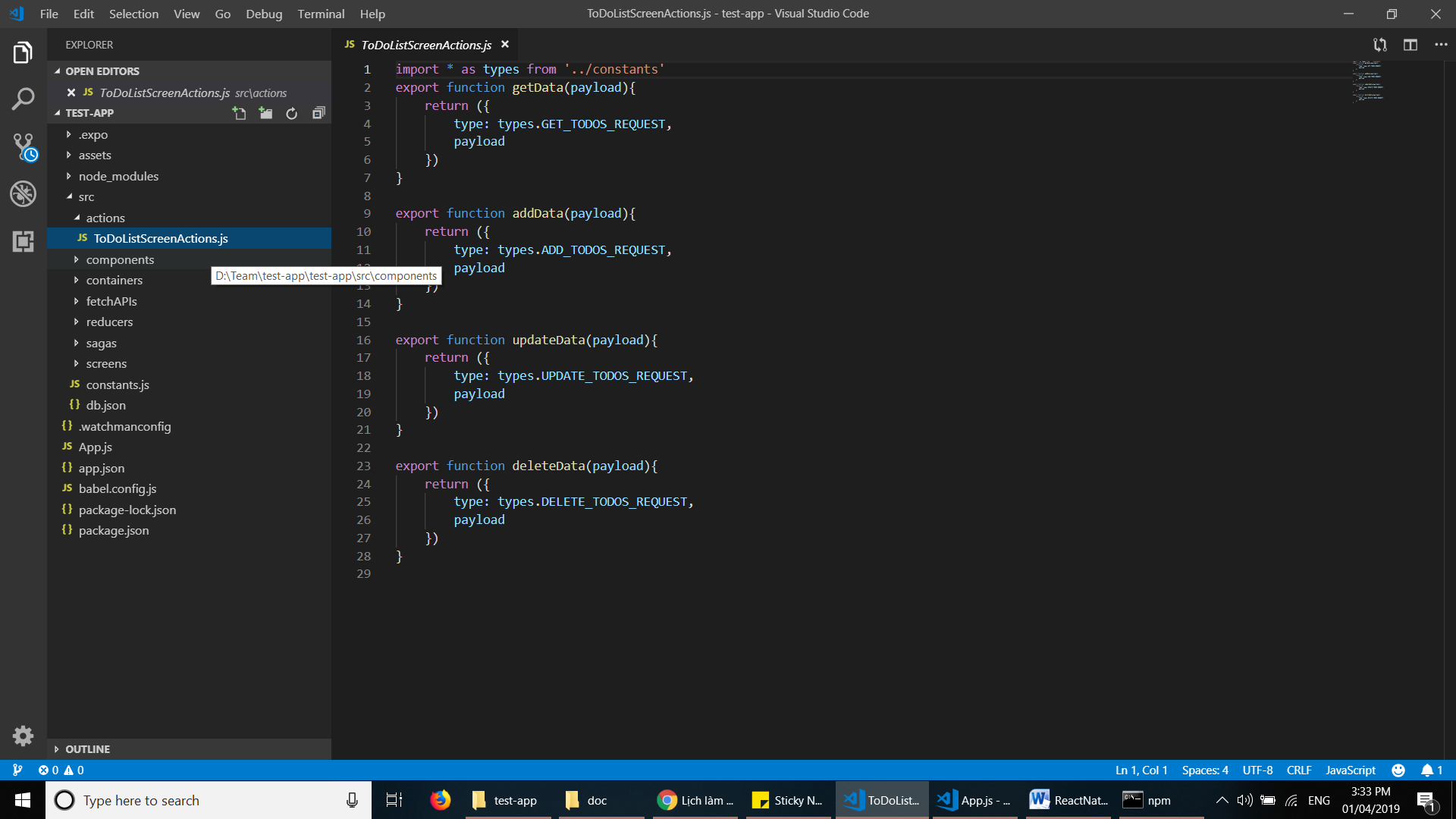
}

Ok giờ chúng ta tạo các thư mục như đã bố trí bên trên 

Các bạn bật lại cửa sổ terminal/cmd trước đó và sử dụng tổ hợp phím “ctrl + c” để dừng ứng dụng đang chạy. Giờ chúng ta cần cài thêm redux, redux-saga, react-redux để sử dụng.

Các bạn cài redux, redux-saga bằng câu lệnh “npm install redux redux-saga react-redux”

Cài react-navigation bằng câu lệnh “npm install --save react-navigation”

Trong thư mục actions, chúng ta tạo 1 file ToDoListScreenActions.js

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

export function getData(payload){

return ({

type: types.GET\_TODOS\_REQUEST,

payload

})

}

export function addData(payload){

return ({

type: types.ADD\_TODOS\_REQUEST,

payload

})

}

export function updateData(payload){

return ({

type: types.UPDATE\_TODOS\_REQUEST,

payload

})

}

export function deleteData(payload){

return ({

type: types.DELETE\_TODOS\_REQUEST,

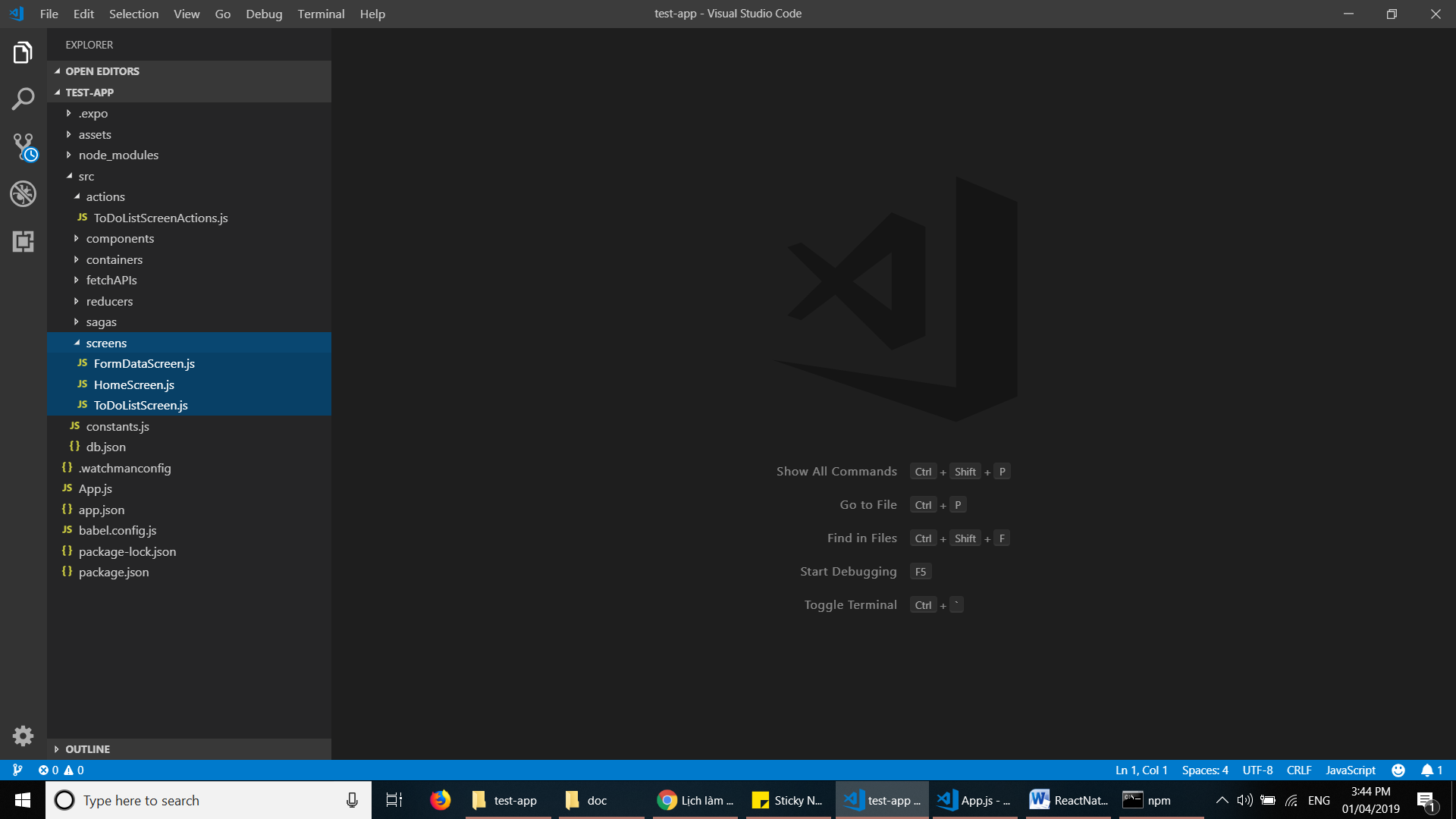
payload

})

}

Ở đây mình sẽ định nghĩa ra 4 hàm tương đương với 4 action: đọc, thêm, sửa, xóa

Ở trong thư mục screens chúng ta sẽ phân chia màn hình như sau:  
Trong ví dụ này mình sẽ làm một màn hình home sau đó khi ấn nút chuyển trang sẽ hiển thị ra To Do List. Khi ấn vào nút sửa hoặc thêm thì sẽ chuyển sang trang nhập form. Vậy tổng cộng mình sẽ có 3 screen là Home, ToDoList và FormData



Sửa file App.js thành nội dung như sau:

import React from "react";

import { createStackNavigator, createAppContainer } from "react-navigation";

import ToDoListScreen from './src/screens/ToDoListScreen'

import HomeScreen from './src/screens/HomeScreen'

import FormDataScreen from './src/screens/FormDataScreen'

import { Provider } from 'react-redux'

import store from './store'

const AppNavigator = createStackNavigator(

{

Home: HomeScreen,

List: ToDoListScreen,

Form: FormDataScreen,

},

{

initialRouteName: 'Home',

/\* The header config from HomeScreen is now here \*/

defaultNavigationOptions: {

headerStyle: {

backgroundColor: '#f4511e',

},

headerTintColor: '#fff',

headerTitleStyle: {

fontWeight: 'bold',

},

},

}

);

const AppContainer = createAppContainer(AppNavigator);

export default class App extends React.Component {

render() {

return <Provider store={store}>

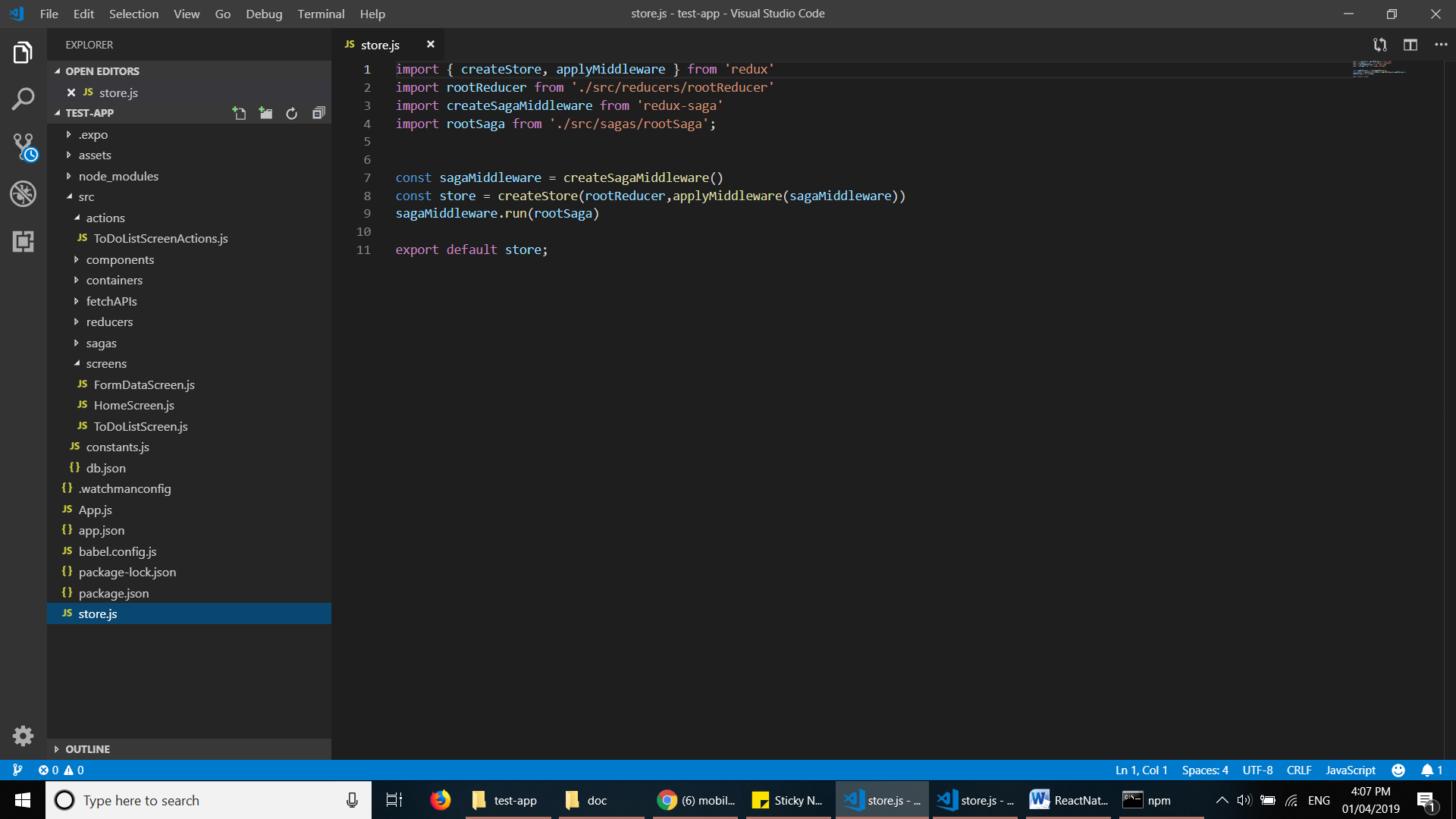
<AppContainer />

</Provider>;

}

}

Tạo 1 file store.js ở cùng tầng thư mục với App.js và có nội dung như sau:



import { createStore, applyMiddleware } from 'redux'

import rootReducer from './src/reducers/rootReducer'

import createSagaMiddleware from 'redux-saga'

import rootSaga from './src/sagas/rootSaga';

const sagaMiddleware = createSagaMiddleware()

const store = createStore(rootReducer,applyMiddleware(sagaMiddleware))

sagaMiddleware.run(rootSaga)

export default store;

Trong HomeScreen.js, nhập nội dung như sau:

import React from "react";

import { View, Text, Button } from "react-native";

class HomeScreen extends React.Component {

static navigationOptions = {

title: 'Home',

};

render() {

return (

<View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>

<Text>Home Screen</Text>

<Button

title="To Do List"

onPress={() => {

this.props.navigation.navigate('List', {

titleParam: 'To Do List',

});

}}

/>

</View>

);

}

}

export default HomeScreen

Trong ToDoListScreen.js, ta thêm nội dung:

import React from "react";

import { View, Text, Button } from "react-native";

class ToDoListScreen extends React.Component {

static navigationOptions = ({ navigation }) => {

return {

title: navigation.getParam('titleParam', '!!!No Title!!!'),

};

};

render() {

return (

<View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>

<Text>To Do List Screen</Text>

<Button

title="Form Data"

onPress={() => {

this.props.navigation.navigate('Form', {

titleParam: 'Form Data',

});

}}

/>

</View>

);

}

}

export default ToDoListScreen

Trong FormDataScreen.js ta thêm nội dung:

import React from "react";

import { View, Text } from "react-native";

class FormDataScreen extends React.Component {

static navigationOptions = ({ navigation }) => {

return {

title: navigation.getParam('titleParam', 'Form Data'),

};

};

render() {

return (

<View>

<Text>Form-Data</Text>

</View>

);

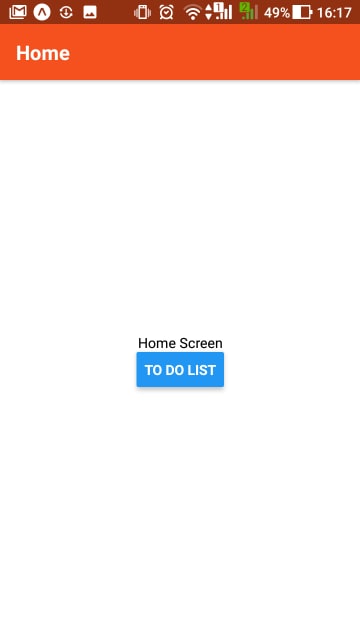
}

}

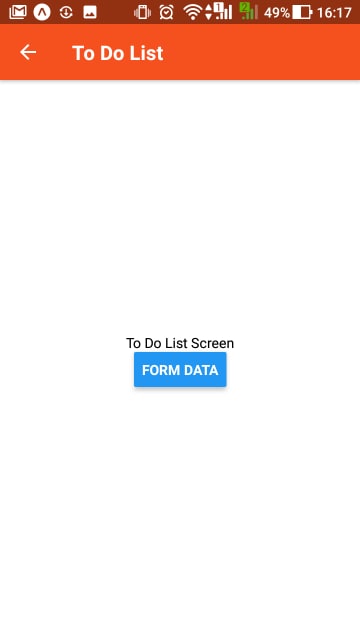
export default FormDataScreen

Ta sẽ tạo 1 ứng dụng với 3 màn hình home, to do list và form data như sau:

--Home Screen:



--To Do List Screen:

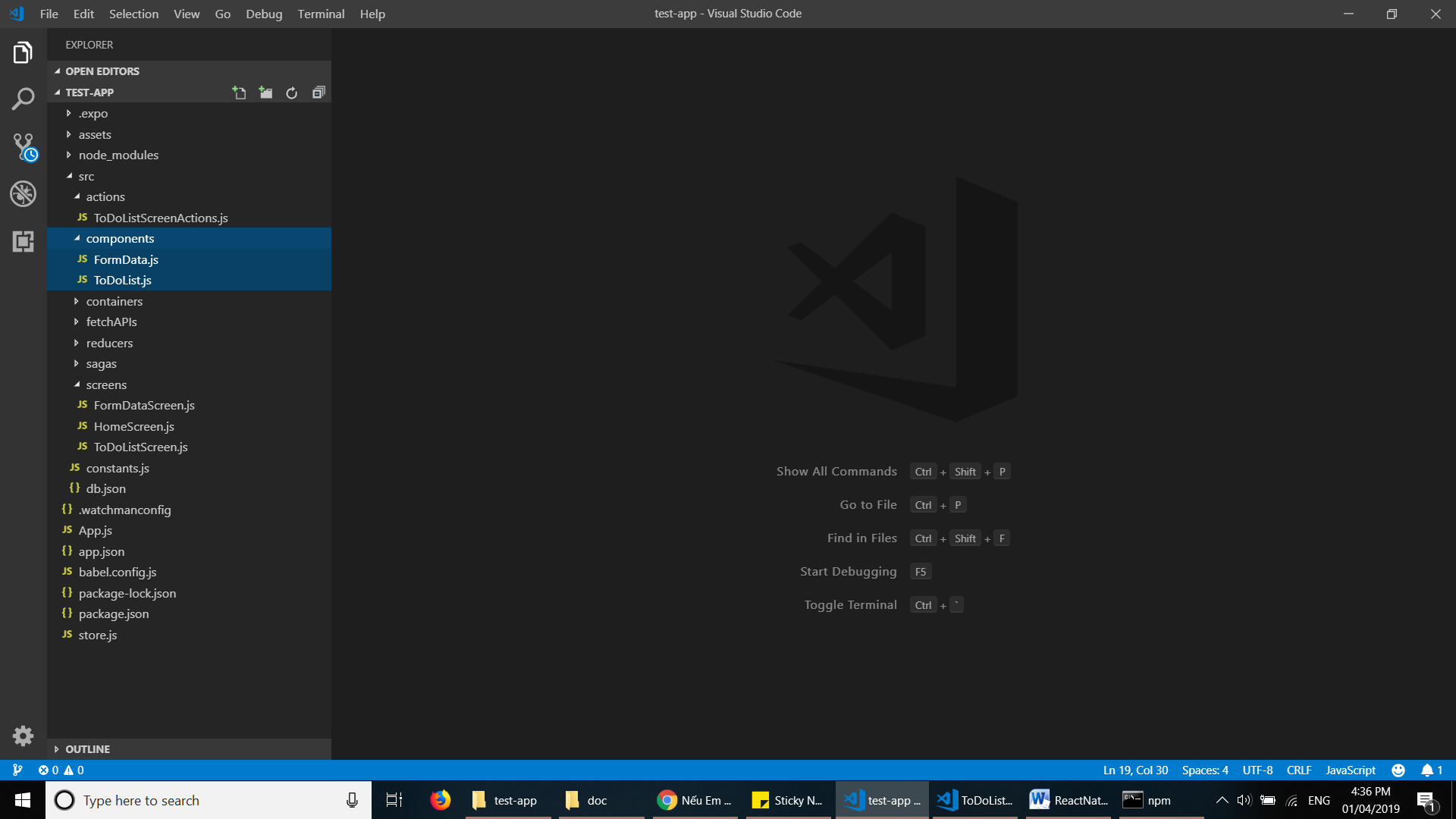


--Form Data Screen:



Tiếp tục làm luồng redux và hiển thị lên dữ liệu từ database nhé.

Tạo 2 file ToDoList.js và FormData.js trong thư mục components:



Trong file ToDoList.js thêm nội dung như sau:

import React, { Component } from 'react';

import { View, Text, Button, ScrollView } from "react-native";

class ToDoList extends Component {

updateTask(id, content){

this.props.navigation.navigate('Form', {

titleParam: 'Update Form With ID: '+ id,

id: id,

content: content

});

}

addTask(){

this.props.navigation.navigate('Form', {

titleParam: 'Add Form',

});

}

deleteTask(id){

this.props.delete(id);

}

render() {

let listTask = [];

if (this.props.todos) {

console.log('this.props.todos', this.props.todos)

listTask = this.props.todos.map((item, key) => {

return (

<View key={key} style={{ flex: 1, flexDirection: 'row'}}>

<View style={{ width: '70%', height: 'auto', backgroundColor: 'powderblue', borderColor :"white", borderWidth:2 }}>

<Text>

{item.content}

</Text>

</View>

<View style={{ width: "30%", height: 'auto', backgroundColor: 'steelblue', borderColor :"white", borderWidth:2 }}>

<Button color="blue" title="Sửa" onPress={() => this.updateTask(item.id, item.content)} />

<Button color="pink" title="Xóa" onPress={() => this.deleteTask(item.id)} />

</View>

</View>

)

})

} else {

console.log('no props')

}

return (

<ScrollView className="toDoList" >

<Button color="green" title="Thêm" onPress={() => this.addTask()} />

{listTask}

</ScrollView>

);

}

}

export default ToDoList

Trong file FormData.js thêm nội dung như sau:

import React, { Component } from 'react';

import { View, TextInput, Button, StyleSheet, Text } from "react-native";

class FormData extends Component {

constructor(props) {

super(props);

this.state = {

content: ''

};

}

handlePress() {

const { navigation } = this.props;

let itemId = navigation.getParam('id', null);

console.log('id of form', itemId)

if (itemId !== null) {

if (this.state.content !== '') {

let data = {

id: itemId,

content: this.state.content

}

this.props.update(data)

}

} else {

this.props.add(this.state);

}

this.props.navigation.navigate('List', {

titleParam: 'To Do List',

});

}

render() {

let itemId = null;

let color = 'green';

let textBtn = 'Thêm';

let content = '';

if (this.props) {

let { navigation } = this.props;

itemId = navigation.getParam('id', null);

content = navigation.getParam('content', '');

if (itemId !== null) {

color = 'blue';

textBtn = 'Sửa';

}

};

return (

<View style={styles.container}>

<Text>Nhập nội dung Task:</Text>

<TextInput style={styles.input}

underlineColorAndroid="transparent"

placeholder="Nhập vào đây..."

autoCapitalize="none"

onChangeText={(text) => this.setState({ content: text })}

defaultValue={content} />

<Button color={color} title={textBtn} onPress={() => this.handlePress()} />

</View>

);

}

}

export default FormData

const styles = StyleSheet.create({

container: {

paddingTop: 23

},

input: {

margin: 15,

height: 40,

borderColor: 'grey',

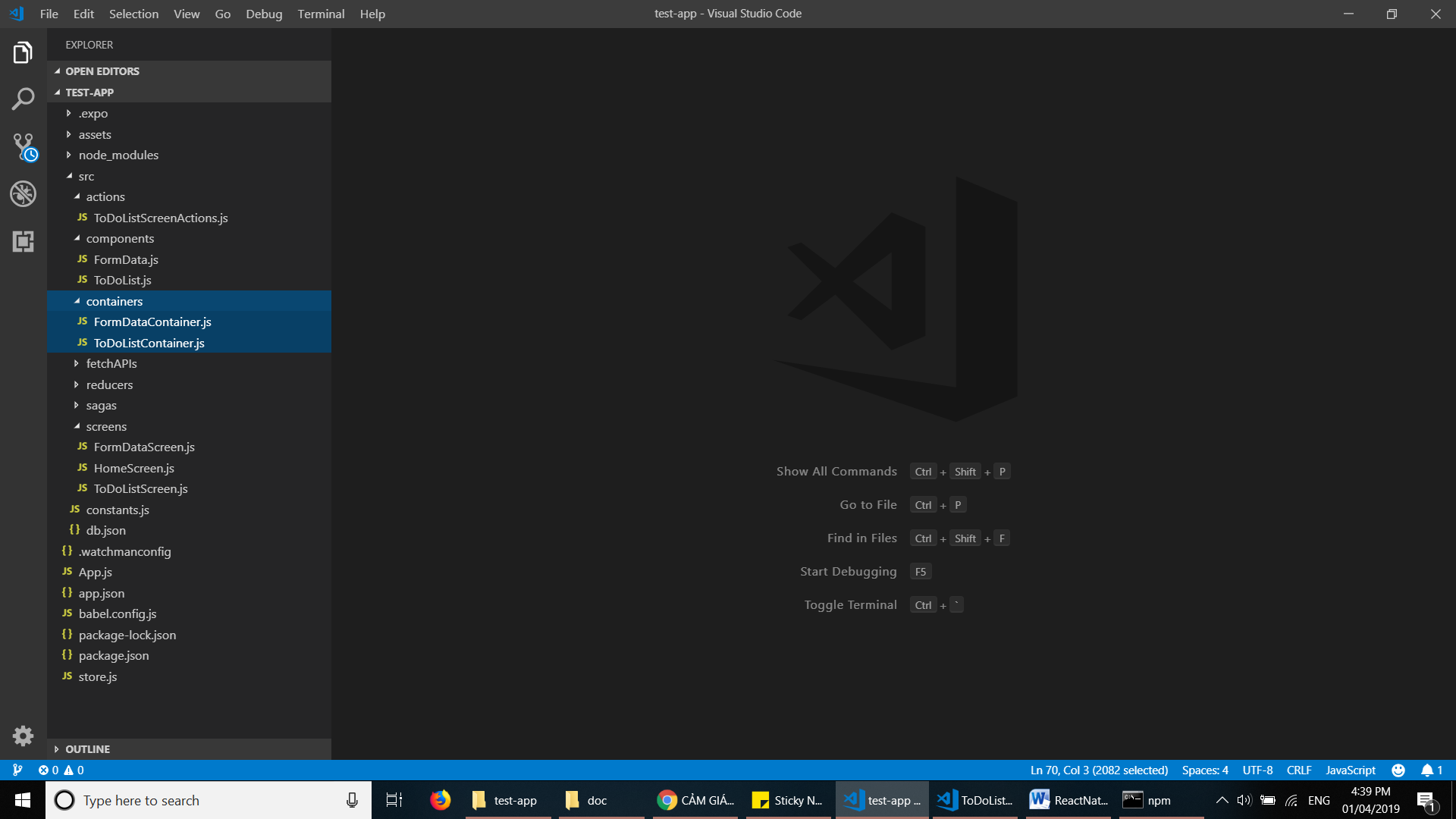
borderWidth: 1,

paddingLeft: 5

}

})

Trong thư mục containers, tạo 2 file FormDataContainer.js và ToDoListContainer.js



Trong file ToDoListContainer.js thêm nội dung:

import React, { Component } from 'react';

import {connect} from 'react-redux'

import { View } from "react-native";

import ToDoList from '../components/ToDoList'

import \* as actions from '../actions/ToDoListScreenActions'

class ToDoListContainer extends Component {

componentDidMount(){

this.props.initLoad()

}

render() {

return (

<View className="toDoListContainer" >

<ToDoList {...this.props}/>

</View>

);

}

}

//mapStateToProp

const mapStateToProps = (state) => {

return {

todos: state.todos.listToDo

}

}

const mapDispatchToProps = (dispatch) => {

return {

initLoad : () => {

dispatch(actions.getData())

},

delete: (id) => {

dispatch(actions.deleteData(id))

}

}

}

export default connect(mapStateToProps,mapDispatchToProps)(ToDoListContainer)

Trong file FormDataContainer.js thêm nội dung:

import React, { Component } from 'react';

import {connect} from 'react-redux'

import { View } from "react-native";

import FormData from '../components/FormData'

import \* as actions from '../actions/ToDoListScreenActions'

class FormDataContainer extends Component {

render() {

return (

<View className="formDataContainer" >

<FormData {...this.props}/>

</View>

);

}

}

const mapDispatchToProps = (dispatch) => {

return {

add : (data) => {

dispatch(actions.addData(data))

},

update: (data) => {

dispatch(actions.updateData(data))

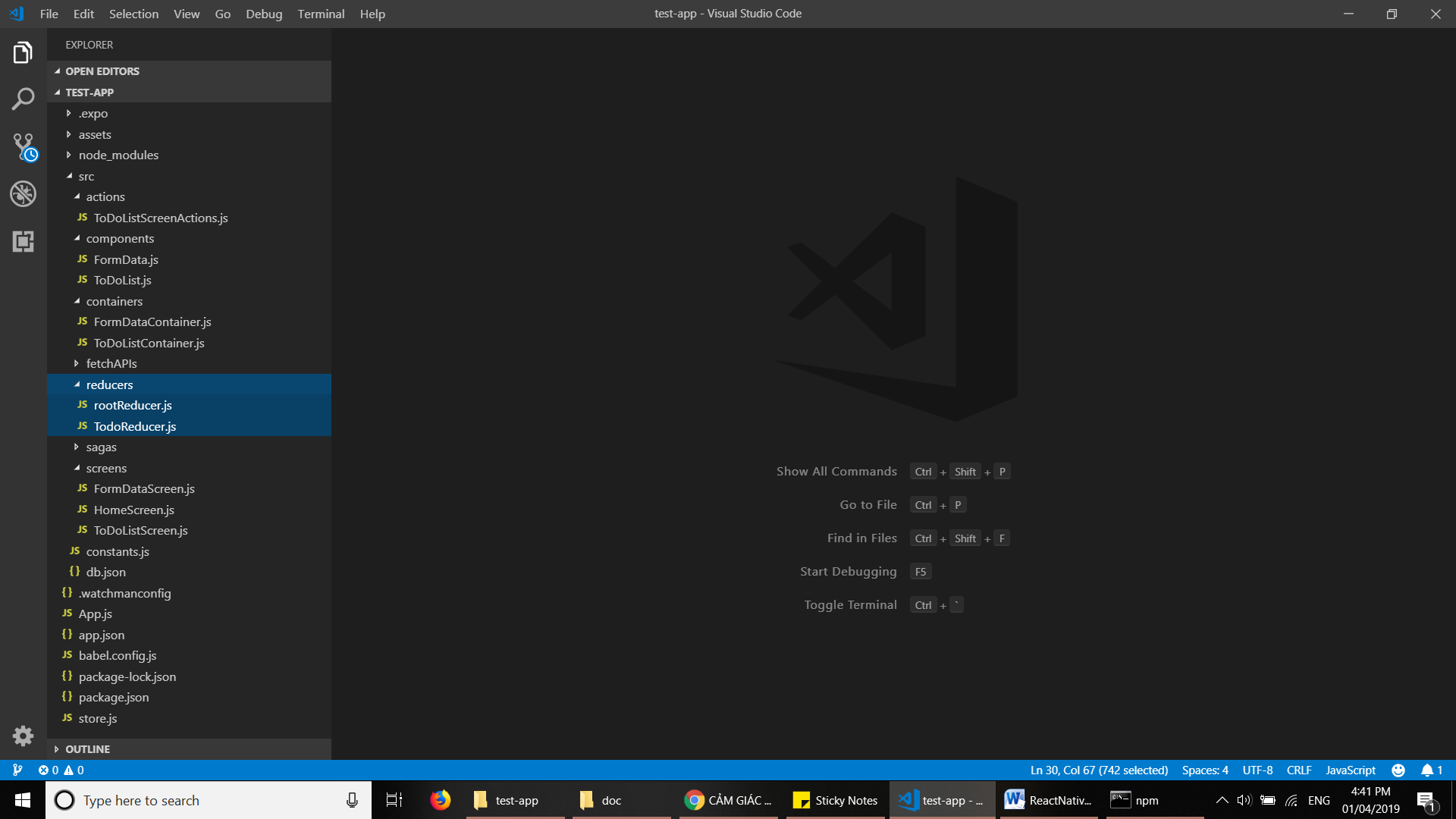
}

}

}

export default connect(null,mapDispatchToProps)(FormDataContainer)

Trong thư mục reducers, thêm 2 file rootReducer.js và ToDoReducer.js



Trong file ToDoReducer.js thêm nội dung:

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

const DEFAULT\_STATE = {

listToDo : null,

dataFetched: false,

isFetching: false,

error: false,

errorMessage: null

}

export default (state = DEFAULT\_STATE, action) => {

switch (action.type) {

case types.GET\_TODOS\_REQUEST:

return {

...state,

isFetching: true,

}

case types.GET\_TODOS\_SUCCESS:

return {

...state,

isFetching: false,

dataFetched: true,

error: false,

errorMessage:null,

listToDo: action.payload,

}

case types.GET\_TODOS\_FAILURE:

return {

...state,

isFetching: false,

error: true,

errorMessage: action.payload.errorMessage

}

case types.ADD\_TODOS\_REQUEST:

return {

...state,

isFetching: true,

}

case types.ADD\_TODOS\_SUCCESS:

return {

...state,

isFetching: false

}

case types.ADD\_TODOS\_FAILURE:

return {

...state,

isFetching: false,

error: true,

errorMessage: action.payload.errorMessage

}

case types.UPDATE\_TODOS\_REQUEST:

return {

...state,

isFetching: true,

}

case types.UPDATE\_TODOS\_SUCCESS:

return {

...state,

isFetching: false

}

case types.UPDATE\_TODOS\_FAILURE:

return {

...state,

isFetching: false,

error: true,

errorMessage: action.payload.errorMessage

}

default:

return state;

}

}

Trong file rootReducer.js, thêm nội dung:

import { combineReducers } from 'redux'

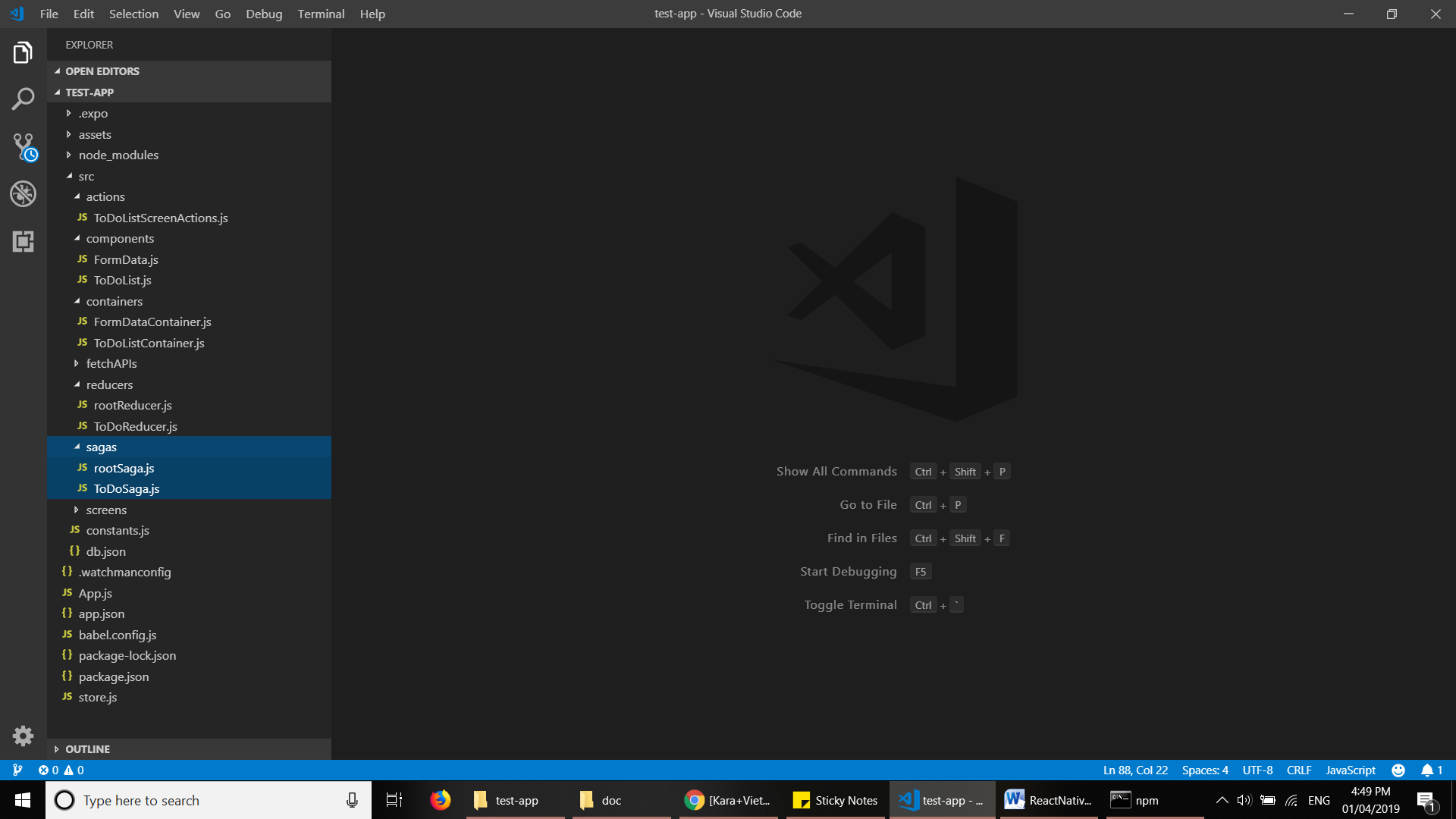
import ToDoReducer from './ToDoReducer'

export default combineReducers({

todos: ToDoReducer

})

Trong thư mục sagas, thêm 2 file rootSaga.js và ToDoSaga.js



Trong file ToDoSaga.js, thêm nội dung:

import { put, takeEvery } from 'redux-saga/effects'

import getDataApi from '../fetchAPIs/getDataApi'

import addDataApi from '../fetchAPIs/addDataApi'

import deleteDataApi from '../fetchAPIs/deleteDataApi'

import updateDataApi from '../fetchAPIs/updateDataApi'

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

function\* getData(action) {

try {

const listData = yield getDataApi(action.payload)

yield put({

type: types.GET\_TODOS\_SUCCESS,

payload: listData

})

} catch (error) {

yield put({

type: types.GET\_TODOS\_FAILURE,

payload: {

errorMessage: error.message

}

})

}

}

function\* addData(action) {

try {

const listData = yield addDataApi(action.payload)

yield put({

type: types.ADD\_TODOS\_SUCCESS,

payload: listData

})

yield put({

type: types.GET\_TODOS\_REQUEST,

payload: listData

})

} catch (error) {

yield put({

type: types.ADD\_TODOS\_FAILURE,

payload: {

errorMessage: error.message

}

})

}

}

function\* updateData(action) {

try {

const listData = yield updateDataApi(action.payload)

yield put({

type: types.UPDATE\_TODOS\_SUCCESS,

payload: listData

})

yield put({

type: types.GET\_TODOS\_REQUEST,

payload: listData

})

} catch (error) {

yield put({

type: types.UPDATE\_TODOS\_FAILURE,

payload: {

errorMessage: error.message

}

})

}

}

function\* deleteData(action) {

try {

const listData = yield deleteDataApi(action.payload)

yield put({

type: types.DELETE\_TODOS\_SUCCESS,

payload: listData

})

yield put({

type: types.GET\_TODOS\_REQUEST,

payload: listData

})

} catch (error) {

yield put({

type: types.DELETE\_TODOS\_FAILURE,

payload: {

errorMessage: error.message

}

})

}

}

export const ToDoSaga = [

takeEvery(types.GET\_TODOS\_REQUEST, getData),

takeEvery(types.ADD\_TODOS\_REQUEST, addData),

takeEvery(types.UPDATE\_TODOS\_REQUEST, updateData),

takeEvery(types.DELETE\_TODOS\_REQUEST, deleteData)

];

Trong file rootSaga.js, thêm nội dung:

import { all } from 'redux-saga/effects'

import {ToDoSaga} from './ToDoSaga'

function\* rootSaga() {

    yield all([

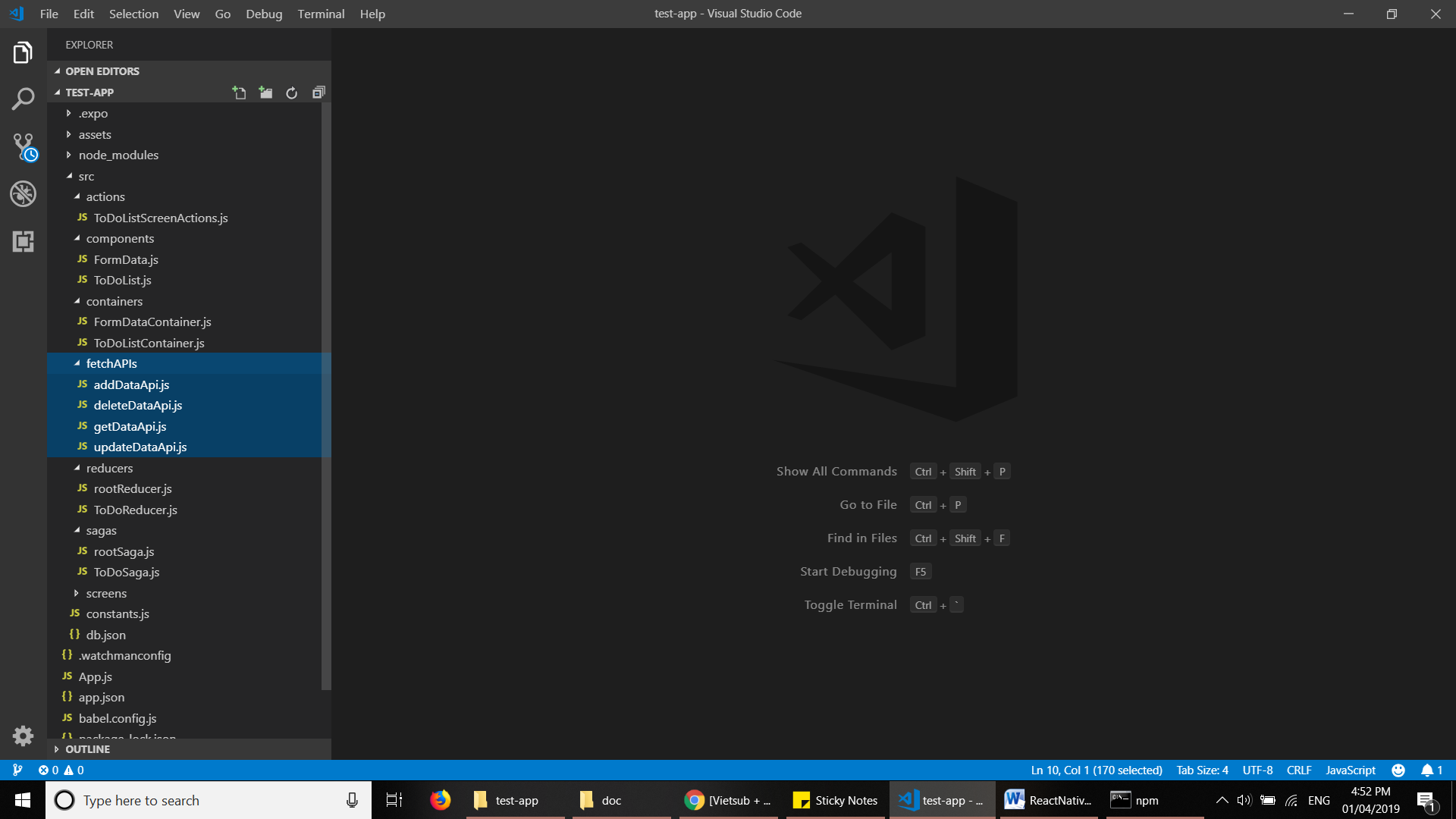
        ...ToDoSaga,

    ]);

}

export default rootSaga;

Trong thư mục fetchAPIs ta thêm file như sau:



Trong addDataApi.js, ta thêm nội dung:

export default function addDataApi(data) {

return new Promise((resolve, reject) => {

const url = 'http://192.168.100.15:3001/todos'

fetch(url, {

method: "POST",

headers: {"Content-type": "application/json"},

body: JSON.stringify(data)

})

.then((response) => response.json())

.then((res) => {

console.log('res', res)

resolve(res);

})

.catch((error) => {

console.log('error', error)

reject(error);

});

});

}

Trong deleteDataApi.js, ta thêm nội dung:

export default function deleteDataApi(data) {

return new Promise((resolve, reject) => {

const url = 'http://192.168.100.15:3001/todos/'+data

fetch(url, {

method: "DELETE"

})

.then((response) => response.json())

.then((res) => {

console.log('res', res)

resolve(res);

})

.catch((error) => {

console.log('error', error)

reject(error);

});

});

}

Trong getDataApi.js, ta thêm nội dung:

export default function getDataApi() {

return new Promise((resolve, reject) => {

const url = 'http://192.168.100.15:3001/todos'

fetch(url, {

method: "GET"

})

.then((response) => response.json())

.then((res) => {

console.log('res', res)

resolve(res);

})

.catch((error) => {

console.log('error', error)

reject(error);

});

});

}

Trong updateDataApi.js, ta thêm nội dung:

export default function updateDataApi(data) {

const dataContent = {

content: data.content

}

return new Promise((resolve, reject) => {

const url = 'http://192.168.100.15:3001/todos/'+data.id

fetch(url, {

method: "PUT",

headers: {"Content-type": "application/json"},

body: JSON.stringify(dataContent)

})

.then((response) => response.json())

.then((res) => {

console.log('res', res)

resolve(res);

})

.catch((error) => {

console.log('error', error)

reject(error);

});

});

}

Sửa lại file ToDoListScreen.js trong thư mục screens như sau:

import React from "react";

import { View } from "react-native";

import ToDoListContainer from '../containers/ToDoListContainer'

class DetailsScreen extends React.Component {

static navigationOptions = ({ navigation }) => {

return {

title: navigation.getParam('titleParam', '!!!No Title!!!'),

};

};

render() {

return (

<View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>

<ToDoListContainer {...this.props}/>

</View>

);

}

}

export default DetailsScreen

Sửa lại file FormDataScreen.js trong thư mục screens như sau:

import React from "react";

import { View } from "react-native";

import FormDataContainer from "../containers/FormDataContainer";

class FormDataScreen extends React.Component {

static navigationOptions = ({ navigation }) => {

return {

title: navigation.getParam('titleParam', 'Form Data'),

};

};

render() {

return (

<View>

<FormDataContainer {...this.props}/>

</View>

);

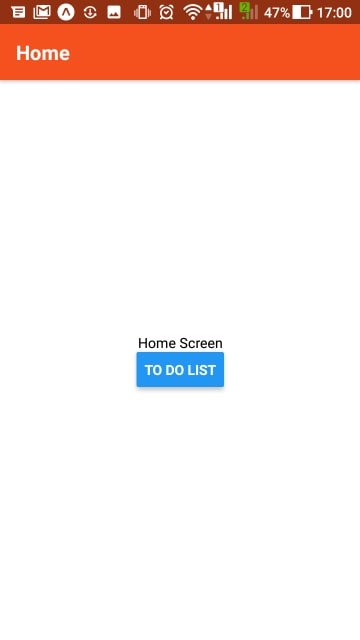
}

}

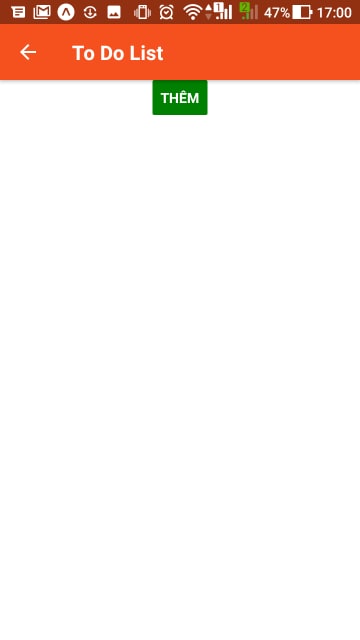
export default FormDataScreen

Giờ bật ứng dụng trong expo bằng câu lệnh “expo start”. Nếu trình duyệt không tự động mở ra thì chúng ta sẽ mở một trang có url như sau: <http://localhost:19002/> . Sau đó quét QR code bằng ứng dụng expo trên điện thoại.(trên iphone thì sau khi cài expo chúng ta sẽ quét bằng máy ảnh mặc định nhé, sẽ có 1 thông báo cho mình để vào ứng dụng)

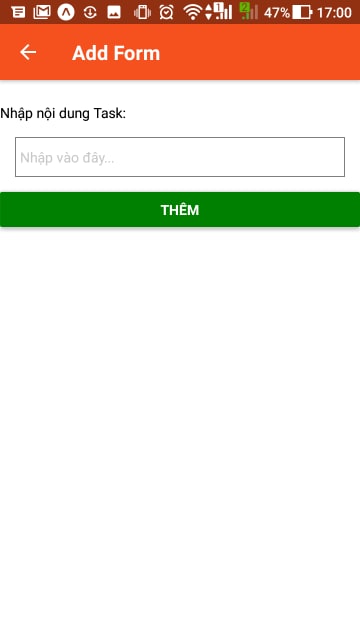
Vừa vào ứng dụng, chúng ta sẽ thấy màn hình home:



Ấn vào nút TO DO LIST để chuyển sang trang to do list:



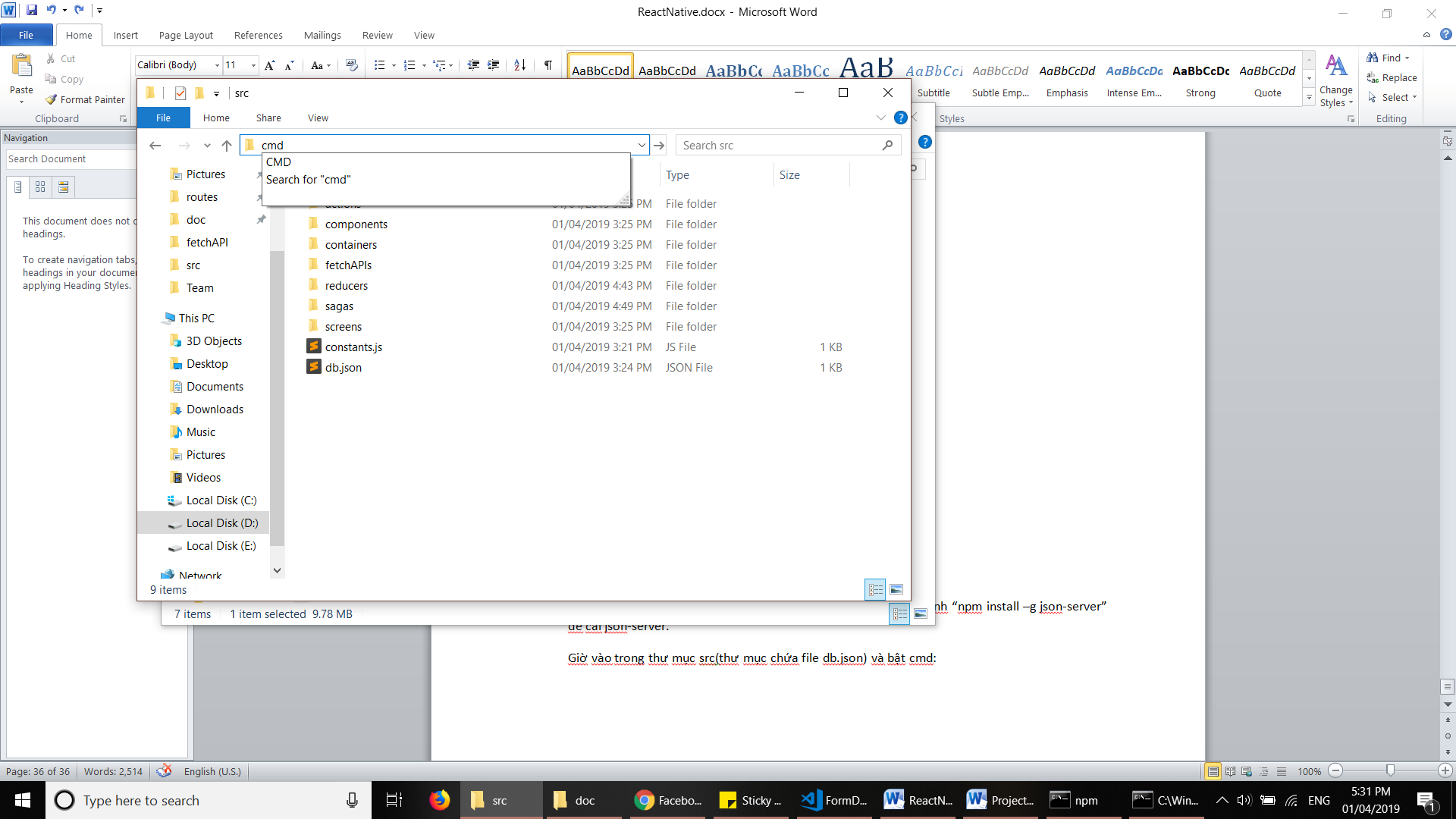
Ở đây chưa hiển thị được ra dữ liệu từ database vì ta chưa bật lên. Chúng ta dùng json-server để mock API.  
Ấn vào nút thêm vẫn ra form data cho phần thêm:



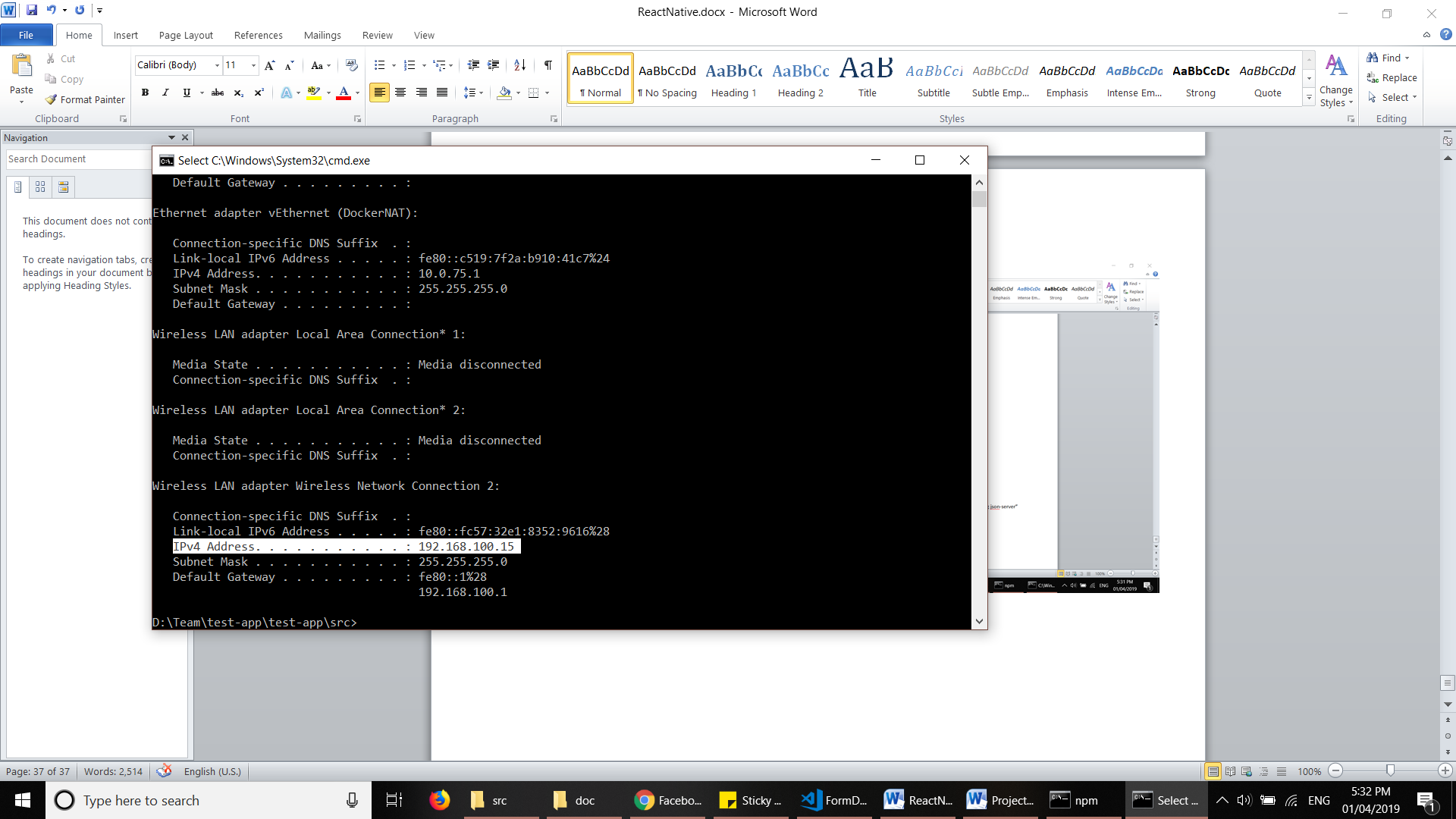
Giờ chúng ta thử bật json-server nhé.

Nếu bạn chưa cài json-server thì bật terminal/cmd và chạy câu lệnh “npm install –g json-server” để cài json-server.

Giờ vào trong thư mục src(thư mục chứa file db.json) và bật cmd:

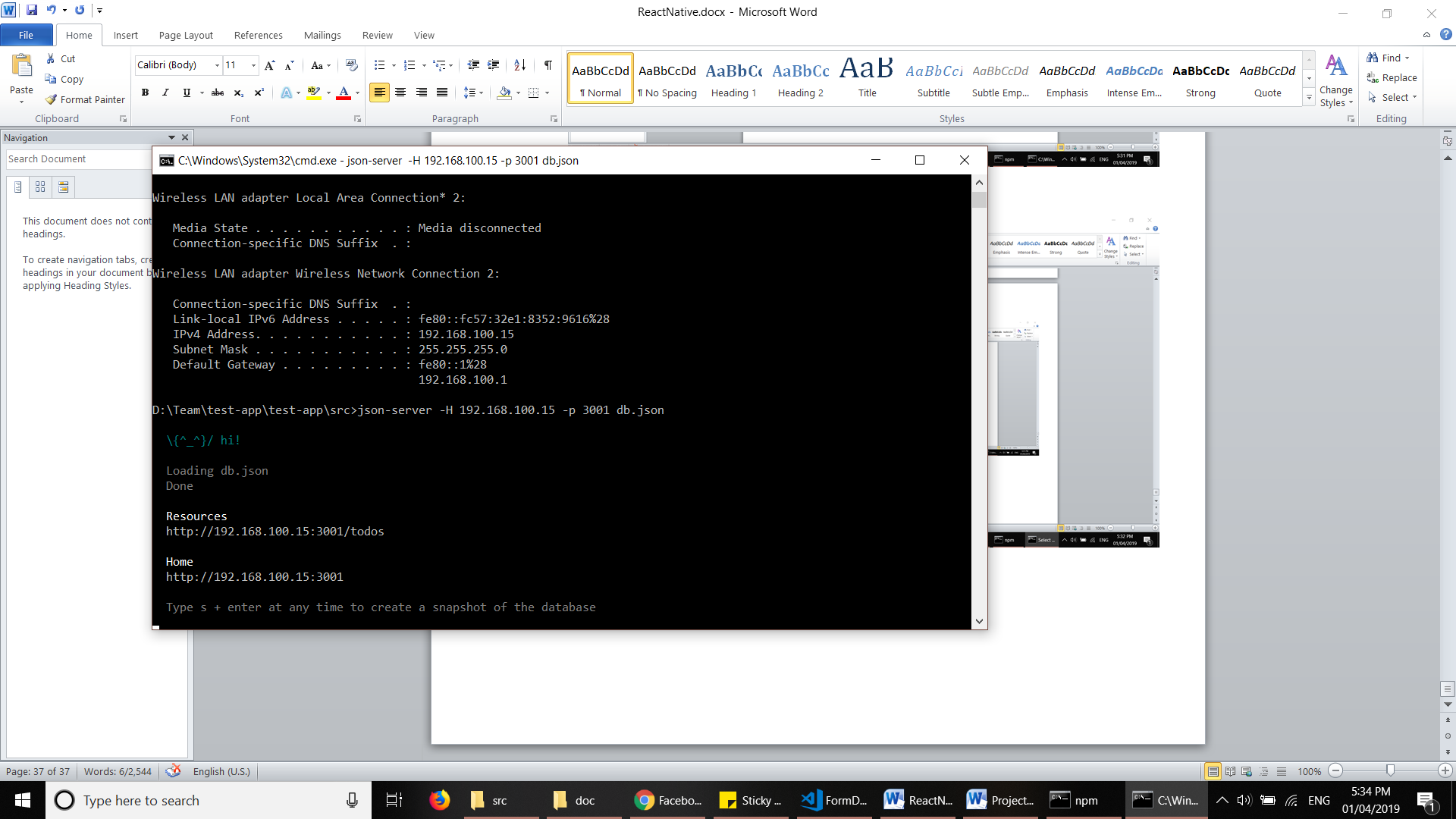


Nhập “ipconfig” sẽ ra màn hình như sau:



Lấy thông tin dòng IPv4, như máy mình thì sẽ nhập tiếp câu lệnh là :  
“json-server –H 192.168.100.15 –p 3001 db.json”

Sau khi chạy xong server sẽ như sau:



Giờ chúng ta sẽ sửa lại url trong các file fetchAPI:

Ví dụ: 