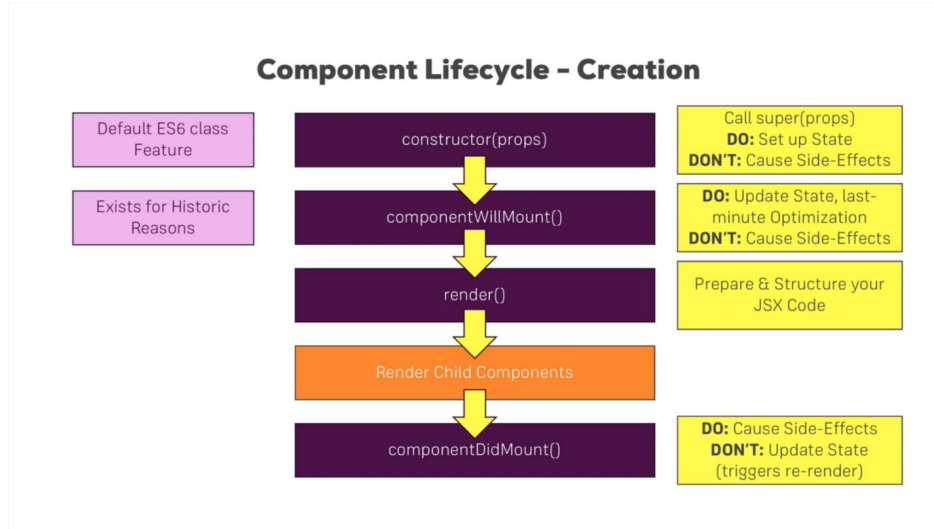


# LIFECYCLE

**Gv: Đặng Trung Hiếu**

## Vòng đời component - Creation

- ❑ Vòng đời Creation bao gồm các phương thức sẽ được khởi chạy lần lượt khi component được khởi tạo và chỉ sử dụng được với statefull



- ❑ Trong đó, componentWillMount sắp bị loại bỏ ở phiên bản reactjs 17 nên ta sẽ ko dùng tới
- ❑ componentDidMount là nơi để ta thực hiện các hành động khi load component lên , ví dụ như gọi API load dữ liệu...

# Vòng đời component - Creation

```
import React, { Component } from 'react';

class demo extends Component {
  constructor(props){
    super(props);
    console.log('Constructor')
  }
  componentWillMount(){
    console.log('componentWillMount')
  }
  render() {
    console.log('render');
    return (
      <div>
        <h1>DEMO CREATION LIFECYCLE</h1>
      </div>
    );
  }
  componentDidMount(){
    console.log('componentDidMount');
  }
}

export default demo;
```

Constructor	<a href="#">demo.js:6</a>
componentWillMount	<a href="#">demo.js:9</a>
render	<a href="#">demo.js:12</a>
componentDidMount	<a href="#">demo.js:20</a>

⚠ ./src/App.js [webpackHotDevClient.js:120](#)

Line 1: 'Fragment' is defined but never used no-unused-vars

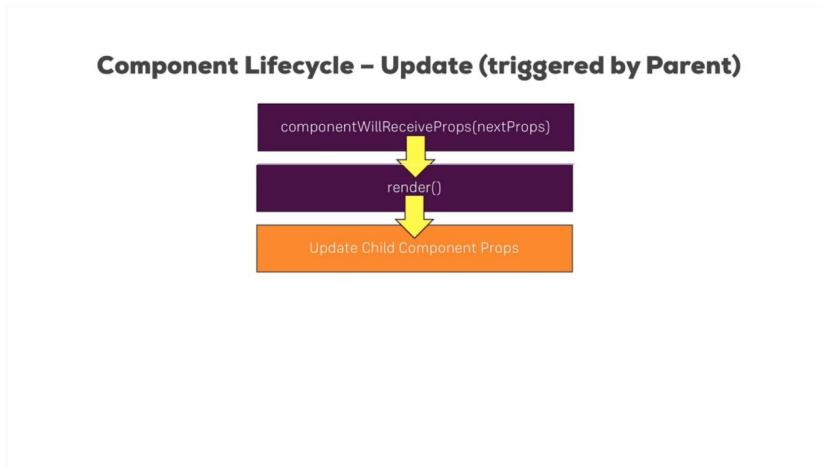
Line 2: 'DanhSachSinhVien' is defined but never used no-unused-vars

Line 3: 'Test' is defined but never used no-unused-vars

=> Thứ tự console.log đã chứng minh thứ tự chạy của các lifecycle creation

## *Vòng đời component - Update*

- ❑ Khi giá trị props bị thay đổi, sẽ dẫn tới việc component được update, các hàm bên dưới sẽ chạy lượt khi component update



- ❑ Vậy thì khi nào sự thay đổi của component thực sự xảy ra ?

## Vòng đời component - Update

❑ Ví dụ, ta tạo 2 component DemoParent và DemoChild như sau:

```
import React, { Component } from 'react';
import DemoChild from './demo-child';
class DemoParent extends Component {
  state = {
    name: 'Hiếu'
  }
  onChangeName = () => {
    this.setState({
      name: 'Dũng'
    })
  }
  render() {
    return (
      <div>
        <button onClick={this.onChangeName}>Change Name</button>
        <DemoChild name={this.state.name} />
      </div>
    );
  }
}
export default DemoParent;
```

```
import React, { Component } from 'react';

class DemoChildComponent extends Component {
  render() {
    return (
      <div>
        <p>Tên: {this.props.name}</p>
      </div>
    );
  }
}

export default DemoChildComponent;
```

- ❑ Ở component DemoParent, ta truyền props *this.state.name* vào trong DemoChild component
- ❑ Khi nhấn nút, ta thay đổi *this.state.name*, nghĩa là props truyền vào DemoChild component bị thay đổi => DemoChild component sẽ được update => các lifecycle update sẽ chạy

## Vòng đời component - Update

❑ Ví dụ, ta tạo 2 component DemoParent và DemoChild như sau:

```
import React, { Component } from 'react';
import DemoChild from './demo-child';
class DemoParent extends Component {
  state = {
    name: 'Hiếu'
  }
  onChangeName = () => {
    this.setState({
      name: 'Dũng'
    })
  }
  render() {
    return (
      <div>
        <button onClick={this.onChangeName}>Change Name</button>
        <DemoChild name={this.state.name} />
      </div>
    );
  }
}
export default DemoParent;
```

```
import React, { Component } from 'react';

class DemoChildComponent extends Component {
  render() {
    return (
      <div>
        <p>Tên: {this.props.name}</p>
      </div>
    );
  }
}

export default DemoChildComponent;
```

- ❑ Ở component DemoParent, ta truyền props *this.state.name* vào trong DemoChild component
- ❑ Khi nhấn nút, ta thay đổi *this.state.name*, nghĩa là props truyền vào DemoChild component bị thay đổi => DemoChild component sẽ được update => các lifecycle update sẽ chạy

# Vòng đời component - Update

❑ Tại component DemoChild :

```
import React, { Component } from 'react';

class DemoChildComponent extends Component {
  componentWillReceiveProps(nextProps){
    console.log('componentWillReceiveProps',nextProps);
  }
  render() {
    return (
      <div>
        <p>Tên: {this.props.name}</p>
      </div>
    );
  }
  componentDidUpdate(){
    console.log('componentDidUpdate')
  }
}

export default DemoChildComponent;
```

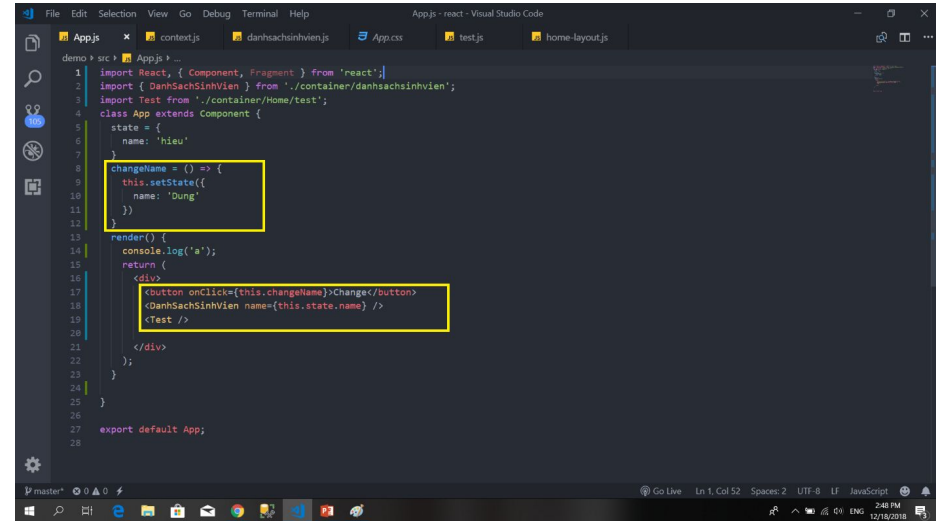
```
componentWillReceiveProps
componentDidUpdate
```

```
{name: "Dũng"}
```

nextProps được in  
ra chính là props  
mới sau khi được  
thay đổi

# PureComponent

- ❑ Trong một vài trường hợp, dù props của component không hề thay đổi nhưng vẫn bị update, dẫn tới ảnh hưởng performance của app
- ❑ Ví dụ :
- ❑ Ở đây ta có thể thấy, khi click vào nút change, state sẽ được set lại, dẫn tới hàm render() sẽ chạy lại và các component con bên trong app cũng được render lại.
- ❑ Vấn đề ở đây là state chỉ ảnh hưởng tới component DanhSachSinhVien, còn component Test vốn ko hề thay đổi, do đó việc render lại nó là ko cần thiết



```
1 import React, { Component, Fragment } from 'react';
2 import { DanhSachSinhVien } from '../container/danhachsinhvien';
3 import Test from '../container/home/test';
4 class App extends Component {
5   state = {
6     name: 'hieu'
7   }
8   changeName = () => {
9     this.setState({
10       name: 'Dung'
11     })
12   }
13   render() {
14     console.log('a');
15     return (
16       <div>
17         <button onClick={this.changeName}>Change</button>
18         <DanhSachSinhVien name={this.state.name} />
19         <Test />
20       </div>
21     );
22   }
23 }
24 export default App;
```



## *PureComponent*

- ❑ Để xử lý vấn đề này, ta có cách giải quyết sau
- ❑ Ở component Test, thay vì class Test extends Component, ta sẽ cho nó extends từ PureComponent, lúc này component Test chỉ render lại khi mà props của nó thật sự thay đổi
- ❑ Lưu ý: chỉ sử dụng PureComponent, ko nên lạm dụng vì có thể dẫn tới lỗi . Bản chất của PureComponent là tự động kiểm tra xem nếu props và state của component đó thay đổi thì sẽ render lại, không thì thôi. Nhưng sự so sánh thay đổi của react là so sánh tham chiếu , nếu như ta truyền một object dưới dạng props, và thay đổi một thuộc tính nào đó thì react ko so sánh đc, vì căn bản là cùng 1 object

```
src > container > Home > test.js > ...  
import React, { PureComponent } from 'react'  
  
export default class test extends PureComponent {  
  
  render() {  
  
    return (  
      <div>  
        aaa  
      </div>  
    )  
  }  
}
```

# AXIOS

**Gv: Đặng Trung Hiếu**

# *Thư viện Axios*



CYBERSOFT  
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH



- ❖ Axios là thư viện hỗ trợ ta gọi API lấy dữ liệu
- ❖ Cách sử dụng tương tự như ajax của jquery
- ❖ Bài tập thực hành : Lấy danh sách khóa học thông qua API

# Bài tập : lấy danh sách khóa học



- ❖ Đầu tiên, tạo ra 2 component : DanhSachKhoaHoc và KhoaHoc
- ❖ Tiếp theo , tại app.js, gọi component DanhSachKhoaHoc để hiển thị ra màn hình

## DanhSachKhoaHoc

```
1 import React, { Component } from 'react';
2
3 import KhoaHoc from './khoaHoc';
4
5 class DanhSachKhoaHoc extends Component {
6   render() {
7     return (
8       <div className="container">
9         <div className="row">
10           <div className="w-100 text-center mb-4">
11             <h1 className="display-4">Danh Sách Khóa Học</h1>
12           </div>
13           <div className="col-4">
14             <KhoaHoc />
15           </div>
16         </div>
17       </div>
18     );
19   }
20 }
21
22 export default DanhSachKhoaHoc;
```

## KhoaHoc

```
import React from 'react';

const khoaHoc = () => {
  return (
    <div class="border p-2 text-center">
      
      <p className="lead font-weight-bold">VueJS</p>
      <p className="lead">Người tạo : Đặng Trung Hiếu</p>
      <button className="btn btn-success">Chi Tiết</button>
    </div>
  );
};

export default khoaHoc;
```

# Bài tập : lấy danh sách khóa học



- ❖ Tiếp theo, tại component DanhSachKhoaHoc, ta sử dụng axios gọi lên api lấy danh sách khóa học về và hiển thị ra màn hình
- ❖ Bước 1 : cài đặt axios **npm install axios –save**
- ❖ Bước 2 : tại component **DanhSachKhoaHoc**, gọi và sử dụng axios

```
1 import React, { Component } from 'react';
2
3 import Axios from 'axios';
4 import KhoaHoc from './khoaHoc';
5
6 class DanhSachKhoaHoc extends Component {
7   /* Một trong những vòng đời của component khi được khởi tạo.
8    ComponentDidMount sẽ chạy sau khi hàm render chạy
9    Đây là nơi lý tưởng để gọi request API
10  */
11   componentDidMount() {
12     Axios({
13       method: "GET",
14       url: 'http://sv.myclass.vn/api/QuanLyTrungTam/DanhSachKhoaHoc'
15     }).then(res => {
16       console.log(res);
17     })
18     .catch(err => {
19       console.log(err)
20     })
21   }
22
23   render() {
24     return (
25       <div className="container">
26         <div className="row">
27           <div class="w-100 text-center mb-4">
28             <h1 className="display-4">Danh Sách Khóa Học</h1>
29           </div>
30           <div className="col-4">
31             <KhoaHoc />
32           </div>
33         </div>
34       </div>
35     );
36   }
37 }
```

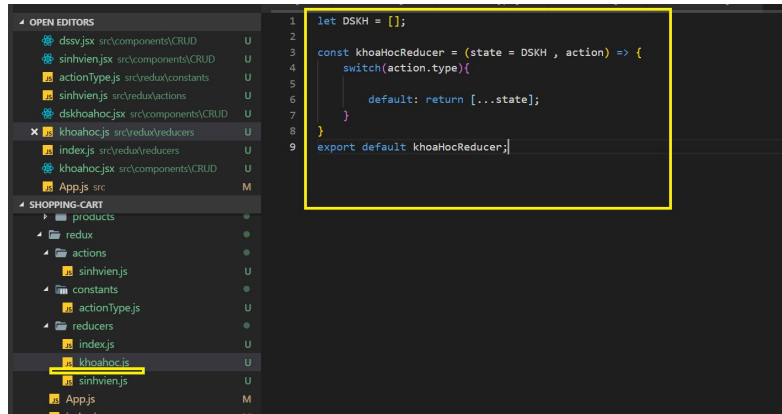
- ❖ Đầu tiên, ta import phương thức Axios từ package axios đã install
- ❖ Tiếp theo, ta gọi hàm Axios chạy và trả ra một đối tượng promise (hỗ trợ tác vụ bất đồng bộ). Một đối tượng promise sẽ có 2 trạng thái chính ; Thành công hoặc thất bại
- ❖ Nếu thành công thì nhảy vào then với res là kết quả server trả về
- ❖ Nếu thất bại thì nhảy vào catch, với err là lỗi bắt được
- ❖ Kiểm tra cửa sổ console và thấy kết quả. Nếu thành công , server trả về object Response, để lấy cái ta cần là danh sách khóa học,

## *Bài tập : lấy danh sách khóa học*

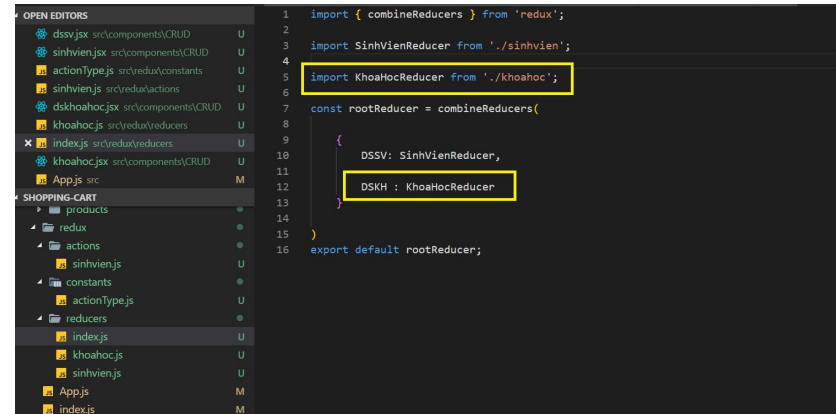
- ❖ Tuy nhiên, componentDidMount chạy sau render, cho nên dù ta lấy được kết quả từ api, giao diện vẫn không load lại được giao diện
  - ⇒ phải sử dụng state để lưu danh sách khóa học, khi lấy được danh sách về, ta setState lại thì giao diện sẽ render lại
  - ⇒ V state nên lưu ở đâu, nên để tại component hay lưu trên store ?
  - ⇒ Tùy vào yêu cầu sử dụng, nếu ta chỉ sử dụng danh sách lấy được ở component DanhSachKhoaHoc thôi, hay sẽ dùng ở một số component khác nữa
  - ⇒ Ở đây ta sẽ lưu trên store, để có sử dụng ở nhiều component

# Bài tập : lấy danh sách khóa học

- ❖ Lưu danh sách khóa học lấy được trên store,
- ❖ Bước 1: tạo reducer quản lý danh sách khóa học
- ❖ Bước 2: tại reducers/index.js , ta thêm 1 thuộc tính vào state store đang lưu trữ



```
1 let DSKH = [];  
2  
3 const khoaHocReducer = (state = DSKH, action) => {  
4   switch(action.type){  
5     default: return [...state];  
6   }  
7 }  
8  
9 export default khoaHocReducer;
```



```
1 import { combineReducers } from 'redux';  
2  
3 import SinhVienReducer from './sinhvien';  
4  
5 import KhoaHocReducer from './khoaHoc';  
6  
7 const rootReducer = combineReducers(  
8   {  
9     DSSV: SinhVienReducer,  
10    DSKH : KhoaHocReducer  
11  }  
12 )  
13  
14 export default rootReducer;
```

## *Bài tập : lấy danh sách khóa học*

- ❖ Bước 4: tại component DanhSachKhoaHoc, ta cần dispatch action lên để lưu danh sách, đồng thời cần lấy danh sách đó xuống để sử dụng , do đó cần cả mapStateToProps lẫn mapDispatchToProps

```
2
3  const mapStateToProps = (state) => {
4    return {
5      DSKH : state.DSKH
6    }
7  }
8
9  const mapDispatchToProps = (dispatch) => {
10    return {
11      onSaveDSKH : (danhSachKhoaHoc) =>{
12        dispatch(actGetCourseList(danhSachKhoaHoc))
13      }
14    }
15  }
16  export default connect(mapStateToProps, mapDispatchToProps)(DanhSachKhoaHoc);
```



CYBERSOFT  
BẢO TÀO CHUYÊN GIA LẬP TRÌNH





## *Bài tập : lấy danh sách khóa học*

- ❖ Bước 5: Sau khi lấy được danh sách khóa học từ api về, ta tiến hành dispatch action gửi danh sách đó lên store để lưu trữ

```
componentDidMount(){
  Axios({
    method: "GET",
    url: 'http://sv.myclass.vn/api/QuanLyTrungTam/DanhSachKhoaHoc'
  }).then(res => {
    this.props.onSaveDSKH(res.data);
  })
  .catch(err => {
    console.log(err)
  })
}
```



CYBERSOFT  
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH



## Bài tập : lấy danh sách khóa học

- ❖ Bước 6: Tiến hành lập component KhoaHoc theo danh sách lấy được

```
renderCourse = () => {  
  return this.props.DSKH.map((khoaHoc, index) => {  
    return (  
      <div className="col-4" key={index}>  
        <KhoaHoc khoaHoc={khoaHoc} />  
      </div>  
    )  
  })  
}  
  
render() {  
  return (  
    <div className="container">  
      <div className="row">  
        <div class="w-100 text-center mb-4">  
          <h1 className="display-4">Danh Sách Khóa Học</h1>  
        </div>  
        <div>  
          {this.renderCourse()}  
        </div>  
      </div>  
    </div>  
  );  
}
```

## Bài tập : lấy danh sách khóa học



CYBERSOFT  
BẢO TẠO CHUYÊN GIA LẬP TRÌNH



- ❖ Tại component Danh Sách Khóa Học, khi lấy danh sách từ trên store về, ta sẽ tiến hành dispatch 1 action để đem danh sách đó lưu trên store
- ❖ Bước 1: tạo 1 biến const mới trong constants/actionType.js

```
1  
2 export const DELETE_SV = "DELETE_SV";  
3  
4 export const GET_COURSE_LIST = "GET_COURSE_LIST";
```

- ❖ Bước 2: tạo 1 action creator để tạo ra action sẽ được dispatch trong file actions/khoahoc.js

```
import {GET_COURSE_LIST} from '../constants/actionType'  
  
export const actGetCourseList = (danhSachKhoaHoc) => {  
  return {  
    type: GET_COURSE_LIST,  
    //danh sách khóa học gửi lên để lưu trên store  
    danhSachKhoaHoc  
  }  
}
```

- ❖ Bước 3: tiến hành connect component DanhSachKhoaHoc với store

# Nâng cấp: tạo async action với middleware

- ❖ Bước 7: Xây dựng trong reducers/khoahoc.js để handle action

```
import { GET_COURSE_LIST } from "../constants/actionType";

let DSKH = [];

const khoaHocReducer = (state = DSKH , action) => {
  switch(action.type){
    case GET_COURSE_LIST :
      var updateState = [...action.danhSachKhoaHoc];
      return updateState;
    default: return [...state];
  }
}

export default khoaHocReducer;
```

## Bài tập : lấy danh sách khóa học

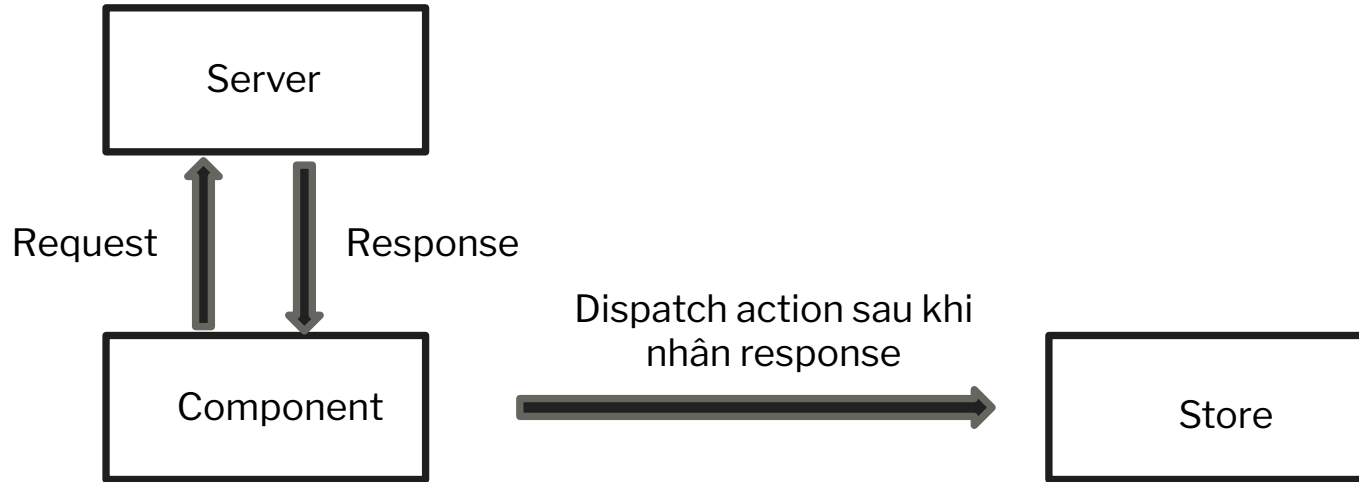
- ❖ Bước 8: Thay đổi trong component KhoaHoc để hiển thị thông tin khóa học tương ứng

```
1 import React from 'react';
2
3 const khoaHoc = (props) => {
4   return (
5     <div class="border p-2 text-center">
6       <img src={props.khoaHoc.HinhAnh} className="w-100" />
7       <p className="lead font-weight-bold">{props.khoaHoc.TenKhoaHoc}</p>
8       <p className="lead">Người tạo : {props.khoaHoc.NgườiTao}</p>
9       <button className="btn btn-success">Chi Tiết</button>
10     </div>
11   );
12 };
13
14 export default khoaHoc;
```

## Nâng cấp : tạo async action

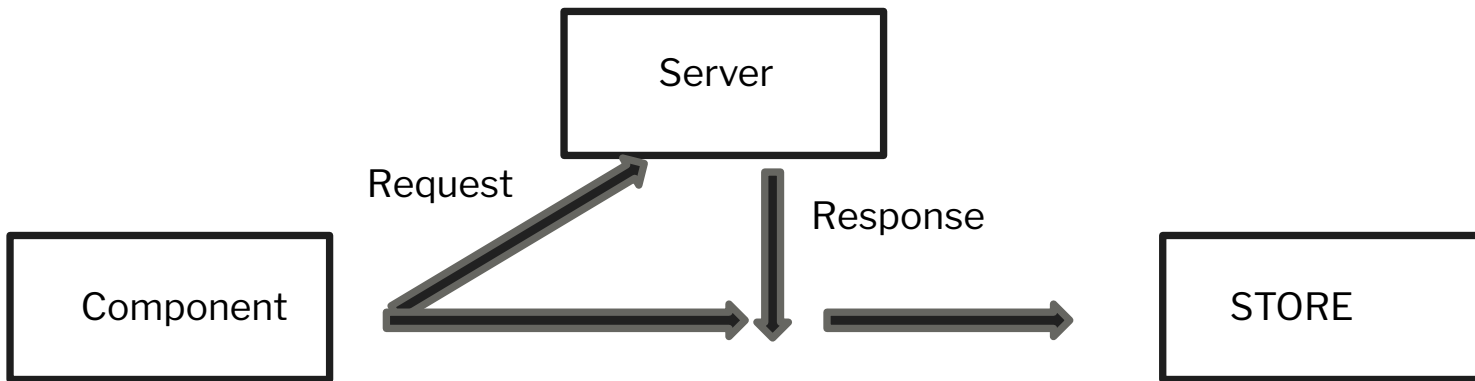


- ❖ Để có thể lấy được danh sách khóa học từ api về, lưu trữ trên store và sử dụng, ta phải trả qua khá nhiều công đoạn và thời gian



## Nâng cấp : tạo async action

- ❖ Để rút ngắn quãng đường, ta có dispatch 1 action lên store ngay lập tức, trên quãng đường từ component lên tới store, ta sẽ tiến hành gửi request và nhận reponse



# Nâng cấp : tạo *async action*

❖ Bước 1: tại actions/khoahoc.js , ta sẽ tạo thêm 1 action nữa.

```
import {GET_COURSE_LIST} from '../constants/actionType'

import Axios from 'axios';

export const saveCourseList = () => {
  return (dispatch) => {
    Axios({
      method: "GET",
      url: 'http://sv.myclass.vn/api/QuanLyTrungTam/DanhSachKhoaHoc'
    }).then(res => {
      dispatch(actGetCourseList(res.data))
    })
    .catch(err => {
      console.log(err)
    })
  }
}

export const actGetCourseList = (danhSachKhoaHoc) => {
  return {
    type: GET_COURSE_LIST,
    //danh sách khóa học gửi lên để lưu trên store
    danhSachKhoaHoc
  }
}
```

Ta sẽ tiến hành dispatch action này lên store, trên đường đi sẽ tiến hành gửi request lên server và nhận response, sau đó dispatch tiếp action GET\_COURSE\_LIST lên để reducer xử lý



# Nâng cấp : tạo async action



- ❖ Bước 2: Tại component DanhSachKhoaHoc, ta sẽ có 1 số thay đổi như sau

```
componentDidMount(){
  this.props.onSaveDSKH();
}

renderCourse = () => {
  return this.props.DSKH.map((khoaHoc, index) => {
    return (
      <div className="col-4" key={index}>
        <KhoaHoc khoaHoc={khoaHoc} />
      </div>
    )
  })
}

render() { ...
}

const mapStateToProps = (state) => { ...
}

const mapDispatchToProps = (dispatch) => {
  return {
    onSaveDSKH : () =>{
      dispatch(saveCourseList())
    }
  }
}

export default connect(mapStateToProps, mapDispatchToProps)(DanhSachKhoaHoc);
```

- Bây giờ ta sẽ dispatch action creator mới tạo ra đó là saveCourseList

- Ở componentDidMount sẽ không axios nữa.

# Nâng cấp : tạo async action



Tuy nhiên, cách này sẽ có một vấn đề xảy ra. Việc gọi api là bất đồng bộ, cho nên không chắc chắn là khi tới được reducer, response đã trả về hay chưa.

=> Phải sử dụng middleware

Error: Actions must be plain objects. Use custom middleware for async actions.



```
dispatch
D:/CYBERSOFT/cybersoft/FE11/reactjs/Shopping_cart/shopping-cart/node_modules/redux/es/redux.js:192

onSaveDSKH
D:/CYBERSOFT/cybersoft/FE11/reactjs/Shopping_cart/shopping-cart/src/components/CRUD/dskhoahoc.jsx:50
```

```
47 | const mapDispatchToProps = (dispatch) => {
48 |   return {
49 |     onSaveDSKH : () =>{
> 50 |       dispatch(saveCourseList())
    |
51 |     }
52 |   }
53 | }
```

View compiled

```
componentDidMount
D:/CYBERSOFT/cybersoft/FE11/reactjs/Shopping_cart/shopping-cart/src/components/CRUD/dskhoahoc.jsx:14
```

```
11 |   Đây là nơi lý tưởng để gọi request API
12 |   */
13 |   componentDidMount(){
> 14 |     this.props.onSaveDSKH();
    |     ^
15 |   }
16 |
17 |   renderCourse = () => {
```

View compiled

3 stack frames were collapsed.

This screen is visible only in development. It will not appear if the app crashes in production.  
Open your browser's developer console to further inspect this error.

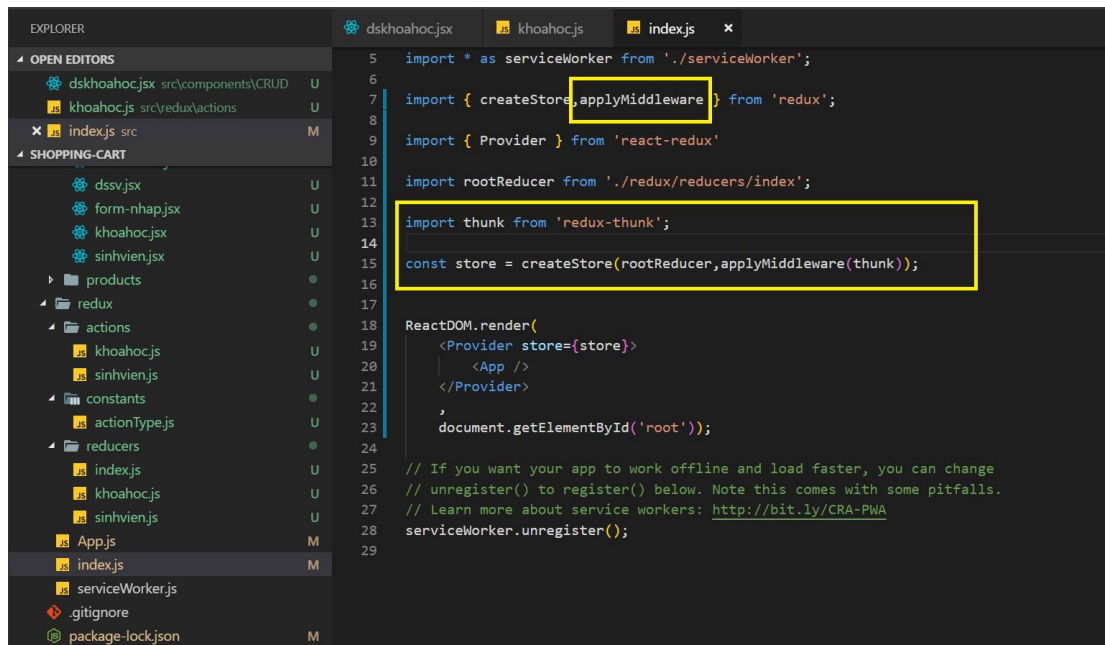
## *Nâng cấp : Sử dụng middleware*



- ❖ Middleware có thể xem như là lớp ngăn cách giữa component và reducer
- ❖ Action được dispatch lên reducer phải đi qua middleware
- ❖ Ta có thể sử dụng middleware để đảm bảo rằng khi tới được reducer, response từ server đã được trả về
- ❖ Có nhiều loại middleware, ở đây ta sử dụng Redux-thunk

# Nâng cấp : Sử dụng Redux-thunk

- ❖ Bước 1: cài đặt redux bằng lệnh npm install --save redux-thunk
- ❖ Bước 2: tại file index.js, ta sử dụng redux-thunk



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with files like dskhoahoc.jsx, khoahoc.jsx, index.js, and various reducers. The code editor shows the content of index.js, which imports Redux and Redux-thunk, creates a store, and renders the application. The following code is highlighted with yellow boxes:

```
import { createStore, applyMiddleware } from 'redux';  
  
import thunk from 'redux-thunk';  
  
const store = createStore(rootReducer, applyMiddleware(thunk));
```

The rest of the code in index.js is as follows:

```
import * as serviceWorker from './serviceWorker';  
  
import { Provider } from 'react-redux';  
  
import rootReducer from './redux/reducers/index';  
  
ReactDOM.render(  
  <Provider store={store}>  
    <App />  
  </Provider>  
,  
  document.getElementById('root'));
```

Comments at the bottom of the file mention service workers and provide a link to learn more about them.

# REACT HOOK

**Gv: Đặng Trung Hiếu**

➤ **React hook: Lý thuyết cơ bản**

- Các hook cơ bản và phổ biến được cung cấp bởi react mà ta sẽ tìm hiểu bao gồm:
  - `useState`
  - `useEffect`
  - `useCallback`
  - `useMemo`

➤ React hook: useState

- Ví dụ đơn giản: nhấn button và tăng number lên 1
- Để number tăng, đồng thời render lại giao diện, ta cần sử dụng state, useState cho phép ta làm điều đó, cho phép sử dụng state ở functional component

increase number

**Number: 0**

```
import React, { useState } from "react";

function ChildComponent(props) {
  const [number, setNumber] = useState(0);
  const increaseNumber = () => {
    setNumber(number + 1);
  };
  return (
    <React.Fragment>
      <button onClick={increaseNumber}>increase number</button>
      <h1>Number: {number}</h1>
    </React.Fragment>
  );
}

export default React.memo(ChildComponent);
```

1. **useState** phải được import từ react
2. **useState** trả về một mảng như hình **[number, setNumber]**, trong đó number là state của component, khi number thay đổi thì component render lại, **setNumber** là phương thức dùng để set lại **number**. Cả **number** lẫn **setNumber** đều là tên tự đặt tùy ý, chỉ cần đúng thứ tự
3. Một component có thể sử dụng nhiều **useState()**

## ➤ React hook: useEffect

- Ví dụ đơn giản: vẫn là nhấn button và tăng number lên 1
- useEffect giúp sử dụng được lifecycle trong functional component.
- useEffect tương ứng với 3 lifecycle đã học trong class component: componentDidMount, componentDidUpdate và componentWillUnmount

```
import React, { useState, useEffect } from 'react';

function ChildComponent(props) {
  const [number, setNumber] = useState(0);

  useEffect(() => {
    console.log('luôn chạy khi component did mount, did update và unmount');
  });

  useEffect(() => {
    console.log('khi component update, sẽ check thử nếu thực sự number');
  }, [number]);

  useEffect(() => {
    console.log('chạy khi component did mount, và chỉ chạy một lần duy nhất');
  }, []);

  const increaseNumber = () => {
    setNumber(number + 1);
  };

  return (
    <React.Fragment>
      <button onClick={increaseNumber}>increase number</button>
      <h1>Number: {number}</h1>
    </React.Fragment>
  );
}
```

1. Một component có thể sử dụng nhiều useEffect
2. useEffect có tham số thứ 2 là một mảng các giá trị, khi component được update, nếu 1 trong số các biến trong mảng này thay đổi, mới chạy lại useEffect, ngược lại sẽ không chạy
3. Mảng rỗng đồng nghĩa với việc useEffect đó chỉ chạy 1 lần duy nhất khi component được khởi tạo



- Trước khi thảo luận tiếp về 2 hook còn lại là useCallback và useEffect, ta hãy cùng đi qua một khái niệm mới, gọi là Memo (tương ứng với PureComponent trong class component)
- Đầu tiên, tạo ra 2 component: DemoHook và DemoChild

## ChildComponent

```
import React from "react";

function ChildComponent(props) {
  console.log("Child render");
  return <React.Fragment>Demo Hook Child Component</React.Fragment>;
}

export default ChildComponent;
```

## DemoComponent

```
import React, { useState } from "react";
import DemoChild from "../child";

function ChildComponent(props) {
  const [number, setNumber] = useState(0);

  const increaseNumber = () => {
    setNumber(number + 1);
  };

  return (
    <React.Fragment>
      <button onClick={increaseNumber}>increase number</button>
      <h1>Number: {number}</h1>
      <DemoChild />
    </React.Fragment>
  );
}

export default React.memo(ChildComponent);
```

- ở đây, khi ta nhấn nút increase number, ta thay đổi number, kéo theo **DemoComponent** render lại, dẫn tới là component con là ChildComponent cũng render lại theo, dù ko hề có sự thay đổi, do đó , react hỗ trợ **Memo**, giống với PureComponent ở class component, giúp component chỉ render lại khi props hoặc state của nó thực sự thay đổi
- Cách dùng: như vậy, childComponent chỉ thực sự re-render khi props hoặc state của nó thay đổi

```
import React, { memo } from "react";

function ChildComponent(props) {
  console.log("Child render");
  return <React.Fragment>Demo Hook Child Component</React.Fragment>;
}

export default memo(ChildComponent);
```

- Quay lại về **useCallback**, ta sẽ xét tới trường hợp dưới đây
- Ở component cha, ta có một hàm là **showNumber** với chức năng đơn giản là console log **number** ra, sau đó truyền vào component con **ChildComponent**.
- Về cơ bản, ở **ChildComponent** ta đã sử dụng **memo**, mà **showNumber** lại là hàm, nên đương nhiên nó sẽ ko đổi, dẫn tới component **ChildComponent** sẽ không được render lại dù component cha có render .
- Kiểm chứng kết quả, ta thấy **ChildComponent** vẫn bị render lại

```
import React, { useState } from "react";
import DemoChild from "../child";
function ParentComponent(props) {
  const [number, setNumber] = useState(0);

  const increaseNumber = () => {
    setNumber(number + 1);
  };

  const showNumber = () => {
    console.log(number);
  };

  return (
    <React.Fragment>
      <button onClick={increaseNumber}>increase number</button>
      <h1>Number: {number}</h1>
      <DemoChild showNumber={showNumber} />
    </React.Fragment>
  );
}

export default ParentComponent;
```

- Lý do: khi component Cha render, nó sẽ render lại tất cả trong function ParentComponent như ta thấy ở dưới, kéo theo **showNumber** sẽ được khai báo lại một lần nữa, tức là nó bị thay đổi, làm cho ChildComponent bị re-render.
- Do đó ở đây ta cần useCallback, **showNumberCallback** ở đây là một bản snapshot của **showNumber**, và nó chỉ được khai báo lại khi **number** thay đổi, nếu để mảng rỗng, có nghĩa là sẽ khai báo 1 lần duy nhất.
- Dẫn tới , khi component cha thay đổi state và render lại, nếu number không đổi, thì showNumberCallback sẽ không đổi, dẫn tới component con sẽ ko render lại

```
import React, { useState, useCallback } from "react";
import DemoChild from "../child";

function ParentComponent(props) {
  const [number, setNumber] = useState(0);

  const increaseNumber = () => {
    setNumber(number + 1);
  };

  const showNumber = () => {
    console.log(number);
  };

  const showNumberCallback = useCallback(showNumber, [number]);

  return (
    <React.Fragment>
      <button onClick={increaseNumber}>increase number</button>
      <h1>Number: {number}</h1>
      <DemoChild showNumber={showNumberCallback} />
    </React.Fragment>
  );
}
```

- Tiếp theo về **useMemo**, nó giống như **useCallback**, điểm khác là **useCallback** trả về một hàm, còn **useMemo** trả về một giá trị
- Xét ví dụ dưới đây: ở đây ta có hàm **numberUp** sẽ tiến hành tính toán cái gì đó và trả ra một giá trị như ví dụ, điều đặc biệt ta có thể nhận thấy, là ta chỉ cần tính một lần thôi, vì căn bản không có gì thay đổi cả. Nhưng khi click button để đổi state, component render lại, sẽ gọi lại **numberUp** chạy một lần nữa, tính toán và trả ra giá trị, đây là điều không cần thiết.

```
import React, { useState } from "react";

function ParentComponent(props) {
  const [number, setNumber] = useState(0);

  const increaseNumber = () => {
    setNumber(number + 1);
  };

  const numberUp = () => {
    let i = 0;
    while (i < 1000) i++;
    console.log(i);
    return i;
  };

  return (
    <React.Fragment>
      <button onClick={increaseNumber}>increase number</button>
      <h1>Number: {numberUp()}</h1>
    </React.Fragment>
  );
}
```

- useMemo sẽ giúp ta cache lại giá trị của numberUp, là chỉ thực hiện tính toán lại khi có sự thay đổi. Và đây là điểm khác biệt, useCallback trả ra một bản snapshot của 1 hàm, còn useMemo chỉ trả ra 1 giá trị thôi.
- **memoizedNumber** chính là giá trị được return từ **numberUp()**, và nó chỉ được tính toán một lần, vì ở đây là mảng rỗng, nếu muốn **memoizedNumber** được tính toán lại khi **number** thay đổi thì thêm **number** vào mảng là được

```
const increaseNumber = () => {
  setNumber(number + 1);
};

const numberUp = () => {
  let i = 0;
  while (i < 1000) i++;
  console.log(i);
  return i;
};

const memoizedNumber = useMemo(() => numberUp(), []);

return (
  <React.Fragment>
    <button onClick={increaseNumber}>increase number</button>
    <h1>Number: {memoizedNumber}</h1>
  </React.Fragment>
);

export default ParentComponent;
```

# MOVIE MANAGER APP

**Gv: Đặng Trung Hiếu**

- ❖ Một số Module react native dùng trong ứng dụng
  - Platform
  - Dimesion
  
- ❖ Một số Package dùng trong ứng dụng
  - React navigation
  - React native vector icon
  - React native element
  - React native image picker
  - Xử lý form với Formik
  - Validation form với Yup



## ❖ *Platform*

- Giúp detect platform của thiết bị đang sử dụng IOS hoặc Android
- Thay đổi giao diện tương thích theo từng OS
- Chi tiết: <https://reactnative.dev/docs/platform-specific-code>

## ❖ Các Functions chính của Platform

- Kiểm tra OS thiết bị : Platform.OS
- Kiểm tra Version thiết bị: Platform.Version
- Thay đổi style theo OS: Platform.select()
- Platform specific code file



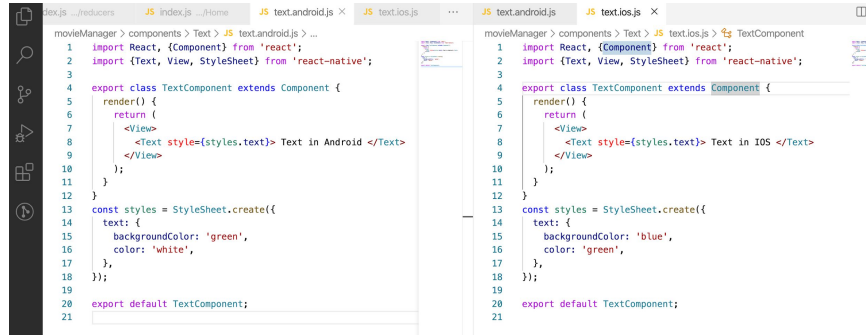
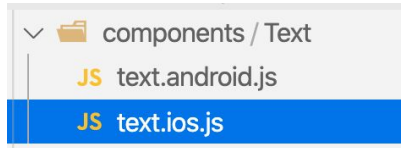
## Platform

- Đừng quên import vào nha: *import {Platform} from 'react-native'*
- Và đây là demo:

```
export class HomeScreen extends Component {  
  render() {  
    //Kiểm tra OS  
    console.log(Platform.OS);  
    //Kiểm tra Version  
    console.log(Platform.Version);  
  
    return (  
      <SafeAreaView>  
        {  
          /* Thay đổi style của text theo OS: IOS hiện màu xanh, android màu đỏ */  
          /* Tạo 2 object style cho IOS và Android, rồi sử dụng Platform.select() để chọn */  
          <Text  
            style={Platform.select({ios: styles.textIOS, android: styles.text})}>  
            Home screen  
          </Text>  
        </SafeAreaView>  
      );  
    }  
}  
  
const styles = StyleSheet.create({  
  textIOS: {  
    backgroundColor: 'green',  
    color: 'white',  
  },  
  textAndroid: {  
    backgroundColor: 'red',  
    color: 'blue',  
  },  
});
```

## ❖ Platform specific code file

- Trong trường hợp muốn chỉnh sửa giao diện của IOS và Android khác nhau, ngoại trừ việc dùng Platform.select(), ta có thể tách ra 2 file riêng biệt cho Android và IOS
- Trong folder components, tạo 2 file riêng cho ios và android



Cách dùng: ở screen home , chúng ta sẽ import vào để dùng, nhưng chỉ import tên là **text** thôi nha, không có .android hay .ios, react native sẽ tự detect OS và dùng file thích hợp

## ❖ *Dimension*

- Hỗ trợ chỉnh sửa giao diện cho các kích thước màn hình khác nhau
  - Sẽ có 2 trường hợp xảy ra
    - TH1: người dùng sử dụng nhiều thiết bị khác nhau, kích thước khác nhau do đó giao diện hiển thị cũng phải tương ứng khác nhau
    - TH2: cùng một thiết bị , nhưng người dùng thay đổi chiều hiển thị

## ❖ *Dimension*

- Hỗ trợ chỉnh sửa giao diện cho các kích thước màn hình khác nhau
  - Sẽ có 2 trường hợp xảy ra
    - TH1: người dùng sử dụng nhiều thiết bị khác nhau, kích thước khác nhau do đó giao diện hiển thị cũng phải tương ứng khác nhau
    - TH2: cùng một thiết bị , nhưng người dùng thay đổi chiều hiển thị

# Form handling

---

# *Tìm hiểu thư viện formik*



CYBERSOFT  
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH



- ❖ Ở các slide đầu, mình đã hướng dẫn các bạn xử lý form thông qua state và handle sự kiện **onChangeText**. Ở phần này, chúng ta sẽ xử lý form thông qua 1 thư viện khá phổ biến hiện tại, có tên là Formik, hỗ trợ xử lý form và validation form.
- ❖ Cách sử dụng:
  - Cài đặt formik vào dự án: ***npm install formik -S***
  - Bài tập demo : xử lý form đăng nhập với formik

# Tìm hiểu thư viện formik



```
import { Formik, Form } from "formik";

class SignInScreen extends Component {
  render() {
    return (
      <div className="container">
        <Formik
          initialValues={{
            taiKhoan: "",
            matKhai: ""
          }}
          onSubmit={values => {
            this.props.dispatch(login(values, this.props.history.replace));
          }}
          render={(({ handleChange }) => (
            <Form>
              <h4 className="display-4">Đăng Nhập</h4>
              <div className="form-group">
                <label htmlFor="Tai Khoan">Tai Khoan</label>
                <input type="text" name="taiKhoan" onChange={handleChange} className="form-control" />
              </div>
              <div className="form-group">
                <label htmlFor="Mật Khẩu">Mật Khẩu</label>
                <input type="password" className="form-control" name="matKhai" onChange={handleChange} />
              </div>
              <div className="form-group text-center">
                <button type="submit" className="btn btn-success">
                  Đăng nhập
                </button>
              </div>
            </Form>
          )) />
        </div>
      </div>
    );
  }
}
```

Giá trị khởi tạo của formik, cũng là định dạng value mà formik sẽ trả ra

Sự kiện submit, xảy ra khi nhấn nút submit form sẽ chạy hàm gì, trong đó tham số values chính là object mà formik trả ra

Giao diện form được in ra màn hình

Đặt cho mỗi ô input một name tương ứng với tên thuộc tính trên initialValues, và 1 sự kiện onChange như trên, handleChange là hàm có sẵn của formik cung cấp

Sau khi set up xong, công việc của còn lại của chúng ta hết sức đơn giản, nhập dữ liệu vào ô input nào, thì hàm **handleChange** có sẵn của formik sẽ giúp ta thay đổi lại giá trị tương ứng.

Khi submit form, thì tham số values chính là object mà formik trả ra sau khi người dùng nhập dữ liệu xong. Ta chỉ cần call api, gửi request lên server để tiến hành đăng nhập là đc.



# Xét giá trị mặc định cho form

- ❖ Trong một số chức năng, ví dụ như update User, các bạn sẽ cần phải fill thông tin của user đó lên form để cập nhật, v làm sao làm được điều đó với formik?



```
class SignInScreen extends Component {  
  render() {  
    return (  
      <div className="container">  
        <div className="row">  
          <div className="col-5 mx-auto">  
            <Formik  
              initialValues={{  
                taiKhoan: "trunghieuv",  
                matKhai: "123"  
              }}  
              onSubmit={values => {  
                console.log(values);  
              }}  
            <Form>  
              <h4 className="display-4">Đăng Nhập</h4>  
              <div className="form-group">  
                <label htmlFor="taiKhoan">Tên Tài Khoản</label>  
                <input  
                  type="text"  
                  name="taiKhoan"  
                  onChange={handleChange}  
                  value={values.taiKhoan}  
                  className="form-control"/>  
                </div>  
              <div className="form-group">  
                <label htmlFor="matKhai">Mật Khẩu</label>  
                <input  
                  type="password"  
                  name="matKhai"  
                  onChange={handleChange}  
                  value={values.matKhai}  
                  className="form-control"/>  
                </div>  
            </Form>  
          </div>  
        </div>  
      </div>  
    );  
  }  
}
```

ở đây mọi người có thể xét giá trị mặc định cho initialValues như bên, hoặc có thể gán initialValues={object của các bạn},

Ngoài handleChange, ta có thể lấy thêm được values, chính là giá trị mà formik đang lưu (

# Validation with formik and yup



CYBERSOFT  
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH



- ❖ Yup là một thư viện hỗ trợ chúng ta validation form , các bạn có thể sử dụng nó với JS thuần, angular, react...
- ❖ Yup và formik là bộ đôi kết hợp khá ăn ý, giúp việc validation và hiển thị error message sẽ dễ dàng hơn rất nhiều
- ❖ Bản chất của **yub** là giúp ta tạo ra 1 **schema** (bản mẫu) , ví dụ như bản mẫu có bao nhiêu thuộc tính, mỗi thuộc tính có required hay không, có minlength hay maxlength gì không... Sau đó ta sẽ lấy object ta muốn validation so sánh với **schema** , nếu khớp thì là hợp lệ, còn không thì ngược lại.
- ❖ Cài đặt yup : ***npm i yup -S***
- ❖ Document: <https://github.com/jquense/yup>

# Validation with formik and yup

- ❖ Tạo ra 1 schema để validation form đăng nhập

```
import * as yup from "yup";

const schema = yup.object().shape({
  taiKhoan: yup
    .string()
    .required("Vui lòng nhập tài khoản")
    .matches(/^[A-Za-z]+$/, "Tài khoản không đúng định dạng"),
  matKhau: yup
    .string()
    .required("Vui lòng nhập mật khẩu")
    .min(6, "Mật khẩu phải tối thiểu 6 kí tự")
    .max(18, "Mật khẩu phải tối đa 18 kí tự")
});
```

- ❖ Áp dụng vào formik: khi ta nhấn submit, formik sẽ tự check, nếu đúng schema mới được submit.

```
<div className="row">
  <div className="col-5 mx-auto">
    <Formik
      validationSchema={schema}
      initialValues={{
        taiKhoan: "trunghieuv",
        matKhau: "123"
      }}
      onSubmit={values => {
        console.log(values);
      }}
      render={({ handleChange, values }) => (
```

# Validation with formik and yup



- ❖ Hiện thị error message ra màn hình
- ❖ Lưu ý : chuyển input thành component Field của Formik để sử dụng được đối tượng touched

```
render=({ handleChange, values, errors, touched }) => (  
  <Form>  
    <h4 className="display-4">Đăng Nhập</h4>  
    <div className="form-group">  
      <label htmlFor="Tài Khoản">Tài Khoản</label>  
      <Field  
        type="text"  
        name="taiKhoan"  
        onChange={handleChange}  
        value={values.taiKhoan}  
        className="form-control"  
      />  
      {errors.taiKhoan && touched.taiKhoan ? <div>{errors.taiKhoan}</div> : ''}  
    </div>  
    <div className="form-group">  
      <label htmlFor="Mật Khẩu">Mật Khẩu</label>  
      <Field  
        type="password"  
        className="form-control"  
        name="matKhau"  
        onChange={handleChange}  
        value={values.matKhau}  
      />  
      {errors.matKhau && touched.matKhau ? <div>{errors.matKhau}</div> : ''}  
    </div>  
    <div className="form-group text-center">  
      <button type="submit" className="btn btn-success">  
        Đăng nhập  
      </button>  
    </div>  
  </Form>  
)}
```