**Trước khi push code thì nhớ rảo qua các page liên quan để xem có error j ko!!!**

Contents

[**1.** **CSS TRICKS** 4](#_Toc70072019)

[**2.** **REACTJS TRICKS** 4](#_Toc70072020)

[**3.** **Ref trong reactjs** 4](#_Toc70072021)

[**4.** **Search** 4](#_Toc70072022)

[**5.** **SyntheticEvent (các sự kiện reactjs support)** 4](#_Toc70072023)

[**6.** **Pin/unpin groups or threads** 4](#_Toc70072024)

[**7.** **Các trạng thái Reaction** 4](#_Toc70072025)

[**8.** **Bắt vị trí của scrollbar => ko bị scrollTop khi loadmore** 4](#_Toc70072026)

[**9.** **Setstate mà ko hiện** 4](#_Toc70072027)

[**10.** **Add hiệu ứng cho những div ở cuối (div được loadmore thêm) – từ cuối lên Hoặc xử lý dấu , ở element cuối** 4](#_Toc70072028)

[**11.** **hàm tính loadmore khi scroll / cuộn trang đến cuối màn hình, trình duyệt (window)** 5](#_Toc70072029)

[**12.** **Hàm loadmore cho 1 element/ 1 div** 5](#_Toc70072030)

[**13.** **Css cho các pt tử - trừ phần tử đầu tiên** 5](#_Toc70072031)

[**14.** **Cách thêm hay xóa 1 input** 5](#_Toc70072032)

[**15.** **Tự lấy input của hàm bằng tay** 6](#_Toc70072033)

[**16.** **new Map()** 6](#_Toc70072034)

[**17.** **Click outside and change state** 6](#_Toc70072035)

[**18.** **Bubbling trigger event** 6](#_Toc70072036)

[**19.** **Bug disabled** 6](#_Toc70072037)

[**20.** **Các trường hợp gây crash App** 6](#_Toc70072038)

[**21.** **Bug ko hiện hover box-shadow** 7](#_Toc70072039)

[**22.** **Check thuộc tính có null/ undifined** 7](#_Toc70072040)

[**23.** **Cách tạo dấu ...** 7](#_Toc70072041)

[**24.** **Lưu ý khi thêm box-shadow –tạo đổ bóng** 7](#_Toc70072042)

[**25.** **Default export and named export** 7](#_Toc70072043)

[**26.** **Các methods cho Object** 7](#_Toc70072044)

[**27.** **Chaining Methods Array** 7](#_Toc70072045)

[**28.** **Filter in Array** 7](#_Toc70072046)

[**29.** **Các bước trước khi Refactor/update cái mới** 7](#_Toc70072047)

[**30.** **Khi dropdown bị đè bởi 1 parent khác** 8](#_Toc70072048)

[**31.** **Đưa 1 Object để call API hoặc đưa lên URL** 8](#_Toc70072049)

[**32.** **Xử lý [Promise, Promise, Pro..] 1 mảng chứa promise(s)** 8](#_Toc70072050)

[**33.** **Cách setState hiệu quả khi có onChange** 9](#_Toc70072051)

[**34.** **Tạo dấu ... Sử dụng css direction** 9](#_Toc70072052)

[35. **Tìm item của mảng này có chứa trong mảng khác hay ko/ có trong mảng** 10](#_Toc70072053)

[**36.** **Remove Array duplicate/trùng trong ES6** 10](#_Toc70072054)

[**37.** **Đệ qui/recursion** 13](#_Toc70072055)

[**38.** **vài trường hợp nên xài store/state chung (Mobx, Redux)** 14](#_Toc70072056)

[**39.** **Lợi dụng biến/item trong các method: map, filter,...** 14](#_Toc70072057)

[**40.** **Lỗi trả ra đường URL sai** 14](#_Toc70072058)

[**41.** **Check duplicate trong 1 Arr chứa các items as Obj** 14](#_Toc70072059)

[**42.** **Gộp hàm Add và Delete vào 1 function và xử lý Key bị trùng** 15](#_Toc70072060)

[**43.** **Cách tạo 1 function debounce (trong class component)** 15](#_Toc70072061)

[**44.** **Trường họp dùng componentWillReceiveProps** 16](#_Toc70072062)

[**45.** **Có thể check API trả về như thế nào để có thể thay đổi cho phù hợp.** 17](#_Toc70072063)

[**46.** **SSH – kết nối với gitlab** 17](#_Toc70072064)

[**47.** **Call/gọi API rồi gán vào Obj** 17](#_Toc70072065)

[**48.** **Tạo 1 overlay khi cần click outside div** 18](#_Toc70072066)

[**49.** **C#(C sharp)** 18](#_Toc70072067)

[**50.** **Convert number to Currency (chuyển tiền có phân cách bằng dấu , . )** 18](#_Toc70072068)

[**51.** **Đặt const/let trong Switch... case(not Error: variable has been declared)** 19](#_Toc70072069)

[**52.** **CSS textcut/truncate in Flex container** 19](#_Toc70072070)

[**53.** **Cách check xem các item trong array đã check hết hay chưa!** 20](#_Toc70072071)

[**54.** **Property does not exist on type 'JSX.IntrinsicElements'** 20](#_Toc70072072)

[**55.** **Chunk array. Chia/ tách array thành các array nhỏ hơn** 20](#_Toc70072073)

[**56.** **Tạo internal call api/ vòng lặp gọi api trong 1 thời gian** 22](#_Toc70072074)

[**57.** **Tính tổng cho array.** 22](#_Toc70072075)

[**58.** **Ref undifined lần đầu tiên render useRef, current = undifined** 22](#_Toc70072076)

[**59.** **Dùng Method get để lấy specific detail a thread/budget** 23](#_Toc70072077)

[**60.** **Chuyển string về url string** 23](#_Toc70072078)

[**61.** **cách copy 1 string** 23](#_Toc70072079)

[**62.** **flatten/mảng phẳng array** 23](#_Toc70072080)

[**63.** **Kiểm tra có phần tử/item trong 1 mảng /array hay không** 24](#_Toc70072081)

[**64.** **Css: fix bug ko hiện scrollbar ngang** 24](#_Toc70072082)

[**65.** **Css: tạo 1 cái menu responsive ko cần dùng javascript/js/jquery** 24](#_Toc70072083)

[**66.** **cách làm dark theme/ dark mode** 25](#_Toc70072084)

[**67.** **Css: tạo overlay before/after để tăng vùng chọn cho button hoặc 1 link** 25](#_Toc70072085)

[**68.** **Search/ tìm kiếm ở phía client/ giao diện** 25](#_Toc70072086)

[**69.** **Các trường hợp/ case gây hack não với điều kiện query** 26](#_Toc70072087)

[**70.** **Dùng async await để xử lý loading showWait** 26](#_Toc70072088)

[**71.** **Cách đưa về 1 mảng rỗng, ko tốn bộ nhớ RAM** 27](#_Toc70072089)

[**72.** **Nếu có phân trang/ pagination, search/ tìm kiếm ở 1 trang list** 27](#_Toc70072090)

[**73.** **How to find out if the browser tab of the page is focused? Tìm xem tab của trình duyệt có focus hay ko?** 28](#_Toc70072091)

[**74.** **dùng ref để focus hoặc blur trong modal** 28](#_Toc70072092)

[**75.** **cắt string substr để có thể làm ...** 28](#_Toc70072093)

[**76.** **Component ko re-render lại khi props thay đổi** 29](#_Toc70072094)

[**77.** **Cách xử lý 1 bug khi prevState === this.state (prevState same this.state)** 29](#_Toc70072095)

[**78.** **Thay thế if, else if, if else bằng object** 29](#_Toc70072096)

[**79.** **Kiểm tra có số có liên tục khôngconsecutive/** **continuously** 29](#_Toc70072097)

[**80.** **Serialize obj. Chuyển obj thành quer url** 30](#_Toc70072098)

[**81.** **Đưa obj lên url** 30](#_Toc70072099)

[**82.** **Lấy query từ url và chuyển sang obj** 30](#_Toc70072100)

[**83.** **Loadmore với Element div** 30](#_Toc70072101)

[**84.** **Vài trường hợp stretch với flexbox** 31](#_Toc70072102)

[**85.** **Mock up JSON API** 31](#_Toc70072103)

[**86.** **Scrollbar not take space** 31](#_Toc70072104)

[**87.** **For Multiple Conditions, use Array.includes (Or operator ||) Kiểm tra nhiều điều kiện** 31](#_Toc70072105)

[**88.** **Sử dụng ... spread operator** 31](#_Toc70072106)

[**89.** **Cách bật Modal/ Popup với các cells,** 32](#_Toc70072107)

[**90.** **Format money thành đơn vị** 32](#_Toc70072108)

[**91.** **Tạo div dummy để làm thanh scrollbar trên top** 32](#_Toc70072109)

[**92.** **Svg- Text Layout - No Word Wrap** 33](#_Toc70072110)

[**93.** **Thay thế cho tab network** 33](#_Toc70072111)

[**94.** **c# - aggregate mongodb** 33](#_Toc70072112)

[**95.** **Các trường hợp tìm search, query 1 cái structure, array, list** 33](#_Toc70072113)

[**96.** **Short-hand for fetch API** 33](#_Toc70072114)

[**97.** **Vài thủ thuật xử lý String** 34](#_Toc70072115)

[**98.** **Chuyển đổi một array object sang object** 34](#_Toc70072116)

[**99.** **Tìm phần tử khác nhau giữa 2 array** 35](#_Toc70072117)

[**100.** **CSS aspect-ratio: thuộc tính theo tỷ lệ màn hình** 35](#_Toc70072118)

[**101.** **CSS cho cursor khi focus input** 36](#_Toc70072119)

[**102.** **useReducer in React** 36](#_Toc70072120)

[**103.** **useMemo in React** 36](#_Toc70072121)

[**104.** **useCallback in React** 36](#_Toc70072122)

[**105.** **useImperativeHandle in React** 37](#_Toc70072123)

[**106.** **useLayoutEffect in React** 37](#_Toc70072124)

[**107.** **useDebugValue in React** 37](#_Toc70072125)

[**108.** **vài case cần lưu ý nếu muốn push lên url** 37](#_Toc70072126)

[**109.** **Bị re-render khi dùng hook (do dùng default props)** 37](#_Toc70072127)

[**110.** **Vấn đề Lưu name chung với id** 38](#_Toc70072128)

[**111.** **Loadmore: sẽ load theo thứ tự và ko thay đổi pageSize.** 38](#_Toc70072129)

[**112.** **UseDebounce** 38](#_Toc70072130)

1. **CSS TRICKS**

Khi muốn custom các Elements thì nên đặt Class ở Div/elements cao nhất để có thể dễ tùy biến.

1. **REACTJS TRICKS**

NẾU CÓ BUNCH OF LIST. ĐỂ CLICK/ HOVER CHỈ CHANGE CHÍNH CÁI ĐƯỢC CLICK/HOVER THÔI THÌ GÁN STATE LÀ NUMBER. STATE = INDEX CỦA LIST => CHANGE

1. **Ref trong reactjs**

Nên đặt ref trong hàm **render()**

Ko nên để chung với điều kiện **isloading ? <Skeleton /> : <div ref={...} mà để**  bên ngoài lun

Để bắt dc ref trong Component đó thì có thể dùng

**componentDidUpdate(prevState)**{this.ref\_abc.addEventListener(‘scroll’, function)}

ref: có thể ref cho 1 **component**.

1. **Search**

Nếu có giá trị search thì đưa vào pageSize. Nếu ko có giá trị search thì cũng phải đưa pageSize

1. **SyntheticEvent (các sự kiện reactjs support)**
2. **Pin/unpin groups or threads**

Khi muốn pin/unpin vào menu thì phải **reload** lại menu. Dùng làm **reload menu** thì mới thấy dc thay đổi

1. **Các trạng thái Reaction**

Gồm các trạng thái chính + trạng thái mặc định

Vd: like, praise, dislike + default.

1. **Bắt vị trí của scrollbar => ko bị scrollTop khi loadmore**

Xác định window.scrollY. set vào 1 state: scrollPos. Dùng callback để gọi 1 hàm xử lý Scroll: { window.scrollTo(0, this.state.scrollPos)}

1. **Setstate mà ko hiện**

Có thể do thiếu **Return. Nhớ Return**

1. **Add hiệu ứng cho những div ở cuối (div được loadmore thêm) – từ cuối lên Hoặc xử lý dấu , ở element cuối**
2. **Dùng:** nth-last-child(2/3/4/5-n)
3. **So sánh trong TempArr**
4. **Dùng index (arr.length – 5)**
5. **hàm tính loadmore khi scroll / cuộn trang đến cuối màn hình, trình duyệt (window)**

 componentDidMount() {

        window.addEventListener('scroll', this.handleOnScroll)

    }

    componentWillUnmount() {

        window.removeEventListener('scroll', this.handleOnScroll)

    }

    private handleOnScroll = () => {

        if (this.state.isLoadMore) return

        if (window.innerHeight + window.scrollY >= document.body.offsetHeight - 10) {

            this.setState({

                isLoadMore: this.state.totalCount != this.state.newsList.length

            }, () => {

                if (this.state.isLoadMore)

                    this.loadMores(this)

            })

        }

    }

    private loadMores(component: News) {

//call API here

    }

Note: biến **isLoadMore** là 1 cái cờ để bắt ko có call API quá nhiều lần khi scroll, cuộn trang. pageLoadMore: cờ để đến số page cần load.

1. **Hàm loadmore cho 1 element/ 1 div**

Trong React có hàm onScroll nên giờ ko cần dùng window.addEventListener('scroll', this.handleOnScroll) nữa.

const ref\_tableList = React.useRef<HTMLDivElement>(null)

const onScrollTable = () => {

        if (!ref\_tableList || !ref\_tableList.current) return

(ref\_tableList.current.scrollTop,

ref\_tableList.current.clientHeight,

ref\_tableList.current.scrollHeight)

   }

<div className="list" ref={el => ref\_tableList.current = el} onScroll={onScrollTable}>

.....

</div>

1. **Css cho các pt tử - trừ phần tử đầu tiên**

Cộng 2 phần từ giống nhau => pt đầu tiên của ul sẽ ko dc css (.hrv-pagination--item+.hrv-pagination—item)

1. **Cách thêm hay xóa 1 input**

Tạo 1 Array trong state. **Array bao gồm các Object** có các trường cơ bản id, name, value,...

Thêm 1 input: dùng push() để thêm. Push 1 Object vào array đó.

Xóa 1 input: dùng filter() để xóa. Lọc các phần tử trong array đó mà khác cái id được truyền vào.

1. **Tự lấy input của hàm bằng tay**

onClick={this.onClick}. để biết có input nào ko thì ghi rõ : onClick={(val, prop) => {this.onClick(val, prop)}

1. **new Map()**

dùng **new Map()** để tạo 1 cái **ref** có dạng là **1 array** == document. querySelectorAll.

Bước 1: this.ref\_\_div.set()

Bước 2: this.ref\_\_div.get()

1. **Click outside and change state**

<https://stackoverflow.com/questions/32553158/detect-click-outside-react-component>

componentDidMount() {document.addEventListener('mousedown', this.handleClickOutside); }

componentWillUnmount() {document.removeEventListener('mousedown', this.handleClickOutside); }

setWrapperRef(node) {this.wrapperRef = node;}

handleClickOutside(event) {

if (this.wrapperRef && !this.wrapperRef.contains(event.target)) {

alert('You clicked outside of me!');

}

}

render() {

return <div ref={this.setWrapperRef}>{this.props.children}</div>;

}

}

1. **Bubbling trigger event**

Khi click vào button con thì div cha ko thay đổi trạng thái(ko gọi hàm ở div cha)

Hàm cha ko có dấu () =>. Mà gọi lun. Để khi con click thì sẽ chạy lun hàm của cha(có event.stopPropagation)

1. **Bug disabled**

Khi check điều kiện disabled với 1 biến ở stay.

Nếu biến đó có giá trị : 0, null, undefined >>>>> Boolean(biến) = false

* Disabled = false > ko disabled cái button/component đó.

1. **Các trường hợp gây crash App**

Các hàm/tham chiếu(length,map,filter,find) Xử lý với **Array, Object =>** đưa điều kiện có hay ko Array/Object trước khi xử lý Object/Array

1. **Bug ko hiện hover box-shadow**

Khi set background là mã màu **hex (#212121**) thì khi box-shadow tràn xuống cái background hex này sẽ ko thấy được box-shadow. => dùng **rgba(41, 121, 254, 0.1)**

1. **Check thuộc tính có null/ undifined**

Nếu **biến đó == 0** suy ra **false** hoặc **biến: false**. Nên **ko thể check if(ten\_bien)** được. Mà phải check if(ten\_bien !== null)

1. **Cách tạo dấu ...**

.textcut-1-line,.textcut-2-line,.textcut-3-line{

    text-overflow: ellipsis;

    -webkit-box-orient: vertical;

    display: -webkit-box; ////// có thể bỏ display ở đây

    overflow: hidden;

}

.textcut-1-line{

    -webkit-line-clamp: 1;

}

1. **Lưu ý khi thêm box-shadow –tạo đổ bóng**

Nên dùng hệ màu background: rgba. Ko chơi hệ màu Hex > ko có hiệu ứng box-shadow

1. **Default export and named export**

**Default export**: khi 1 file/module chỉ cần xuất 1 object duy nhất. ( import Name from ‘./../.../’) ko có ngoặc nhọn {}

**Named export**: khi 1 file/module có nhiều Object cần xuất. ( import { Name1, Name2, Name3, ...} from ‘./../../’ ) có dấu ngoặc nhọn {}

Note: trong 1 file/module có thể có **Default export** và **named export**

1. **Các methods cho Object**

Object.keys(obj) > return array[string] of keys

Object.values(obj) > return array[string] of values

Object.entries(obj) > return [key, value]

1. **Chaining Methods Array**

Arr.filter().map(): filter trước, map sau

1. **Filter in Array**

.Filter() Luôn trả về 1 mảng mới => **phải đặt nó là 1 biến**, ko để 1 expression trống

**let** dataFromTable = [...this.state.dataFromTable].filter(item => item.id !== id) **🗸**

[...this.state.dataFromTable].filter(item => item.id !== id) **X**

1. **Các bước trước khi Refactor/update cái mới**

Kiểm tra xem component dc update này có dùng ở đâu ko? (có thể tìm trong scope trước)

Kiểm tra xem giá trị trả ra là gì > save/remember để so sánh với component sau khi update.

1. **Khi dropdown bị đè bởi 1 parent khác**



Solve: đưa pos-relative lên thằng ông nội/hơn 1 cấp so với cấp parent > dropdown sẽ lấy pos-absolute theo thằng ông nội này > đè dc 1 parent khác

1. **Đưa 1 Object để call API hoặc đưa lên URL**

Cần phải chuyển OBJ đó sang string hết rồi mới đưa lên URL hay call API. **JSON.stringify(OBJ)**

1. **Xử lý [Promise, Promise, Pro..] 1 mảng chứa promise(s)**

Phải lun có Promise.all([arr of Promises]) để trả ra kết quả. Nếu ko sẽ là các promises pending.



1. **Cách setState hiệu quả khi có onChange**

Nên chọn 1 state- object mặc định rồi lấy giá trị của state đó, xong setState cho property của Object khác.

Vd: defaultTos > gán vào value của SearchEntity.

setState = onChange > thay đổi mailfilter.tos

mỗi lần bật Modal thì gán defaultTos(dc convert từ mailfilter.tos) vào value của SearchEntity.

Cuối cùng afterCloseModal > set defaultTos = []

1. **Tạo dấu ... Sử dụng css direction**

Dùng để chuyển đổi dấu ... từ cuối câu sang đầu câu. Tạo dấu ...



1. **Tìm item của mảng này có chứa trong mảng khác hay ko/ có trong mảng**

Dùng for cho mảng A. Dùng find để tìm xem item trong mảng A có trong mảng B ko!



1. **Remove Array duplicate/trùng trong ES6**

Dùng new Set(arr)



Dùng Filter. Trong Filter dùng indexOf(item)



Sử dụng Reduce. Trong reduce dùng includes(return true/false). Condition: nếu trong mảng đã bao gồm item rồi thì return chính mảng đó(ko thêm, xóa). Ngược lại thì thêm item. [...acc, item]



Dùng lodash **\_.uniqBy** check duplicate với 1 điều kiện/ 1 property nếu là 1 arr chứa các objects

**Remove Duplicates from an Array using forEach() with Object {}**

const array = [1, 1, 1, 3, 3, 2, 2];

function removeDups(names) {

let unique = {};

names.forEach(function(i) {

if(!unique[i]) {

unique[i] = true;

}

});

return Object.keys(unique);

}

console.log(removeDups(array));

fn removeDups : nhận 1 array. Trong fn tạo 1 {}.

Nếu key(là item of array) trong {} này ko có thì set giá trị = true. Rồi Object.keys đó để lấy ra array các key trong {} đó

\* Cách này phần tử trước sẽ bị ghi đè bởi phần tử sau > lấy phần tử sau

**Use Filter and indexOf**

const array = [1, 1, 1, 3, 3, 2, 2];

const unique = array.filter((element, index) => {

return array.indexOf(element) === index;

});

console.log(unique); *//return [1, 3, 2]*

\* Cách này sẽ lấy phần tử trước. Các phần tử sau khác index nên ko lấy

1. **Đệ qui/recursion**

Là hàm gọi hàm. Và lập đến thì thỏa dk thì out. 

Hoặc tạo 1 component thì dễ hiểu hơn



1. **vài trường hợp nên xài store/state chung (Mobx, Redux)**

Khi là các process cần thực hiện theo các bước(steps).

Hoặc 1 quy trình nào đó => dùng store/state chung để có thể quản lý ở phía Client

1. **Lợi dụng biến/item trong các method: map, filter,...**

Users.map(**user** => <Component data={**user**} onChangeUser={(id) => onChangeUser(this, **user**, id)

Lợi dụng biến user trong map để truyền xuống hàm => để set lại giá trị trong Object user.(ko cần dùng find)



1. **Lỗi trả ra đường URL sai**



sẽ trả ra ‘budgets/budgets/manage/create\_v2

cách fix

 thêm / vào trước chuỗi string này

1. **Check duplicate trong 1 Arr chứa các items as Obj**

Cách 1: Dùng filter để lọc ra những items trùng. Dùng filter(x => for()). If true => break Để nó out for



Cách 2: gộp các mảng vào 1. Dùng điều kiện check && thêm điều kiện **inx !== i && inx > i** để lấy ra 1 phần tử thôi

let newArr = [...a, ...b].filter((n, inx) => {

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

        if (n.name === e[i].name && inx !== i && inx > i) return true

    }

})

Cách 3:

   // check if chosenObj in ObjList push in existArr, not push in newArr

    let existArr = [] as Interfaces.ILimitPerson[]

    let newArr = [] as Interfaces.ILimitPerson[]

    listOfAmountLimitObj.forEach(item => {

        let isInList = objList.some(obj => obj.numberId === item.numberId)

        if(isInList) {

            existArr.push(item)

        }else{

            newArr.push(item)

        }

    })

1. **Gộp hàm Add và Delete vào 1 function và xử lý Key bị trùng**

Nên viết Add và Delete chung 1 hàm và thêm 1 biến để làm flag

Key bị trùng dẫn đến việc delete sẽ ko đúng item mong muốn và nó sẽ delete cái cuối cùng.

1. **Cách tạo 1 function debounce (trong class component)**

Define 1 biến global trong class( dưới constructor)



Gán function ngoài class vào biến global này.(bên trong constructor và dưới this.state). nhớ có bind(this)



Chỗ nào cần dùng thì thay bằng this.[tên function]



Cách này linh hoạt: chỗ nào cần debounce thì gán thôi. Ko apply all

* Hoặc dùng debounce tại hàm đó lun
* 

1. **Trường họp dùng componentWillReceiveProps**

Là khi đưa cái prop đó gán vào state thì cần. Ko thì khỏi



Cách ko dùng isLoading khi nhập query (để hiện thông báo ko tìm thấy)



1. **Có thể check API trả về như thế nào để có thể thay đổi cho phù hợp.**

API trả về list of mails. Nhưng nếu API list chỉ có 1 mail và mail đó có isDraft: true. Thì dùng dk này để setState và xử lý.



1. **SSH – kết nối với gitlab**
2. Tạo 1 key trong máy (dùng sourceTree tạo cho dễ). lưu lại với password
3. Add key vào gitlab. (vào setting của acc mình)
4. Dùng URL SSH của project để clone về. Nếu đã dùng Url thường rồi thì setting SourceTree edit thành SSH
5. **Call/gọi API rồi gán vào Obj**



1. **Tạo 1 overlay khi cần click outside div**



div.overlay phải ngang cấp với div.box(để z-index luôn nằm dưới div.box)

tạo overlay bao hết screen để **tránh click nhầm vào link khác**. Nhớ tạo ref để có thể click được vùng div.overlay



Tạo Flag khi div.box show thì div.overlay cũng show

Khi click vào div.overlay thì sẽ tắt popup của div.box

1. **C#(C sharp)**

Dotnet restore –no-cache

Run debug

1. **Convert number to Currency (chuyển tiền có phân cách bằng dấu , . )**

**Note**: input: string. Nên phải toString() số khi truyền vào.



export const parseNumberToCurrency = (str: string) => {

    return str.replace(/\B(?=(\d{3})+(?!\d))/g, ',')

}

function formatNumber(num) {

return num.toString().replace(/(\d)(?=(\d{3})+(?!\d))/g, '$1,')

}

The $1 is a special replacement pattern which holds a value of the first parenthesized sub-match string

1. **Đặt const/let trong Switch... case(not Error: variable has been declared)**

Sử dụng curly brace {} vì const/let là block-scope nên phải có {}



1. **CSS textcut/truncate in Flex container**



Với Parent/flex container. Mặc định flex để min-width: auto > phải đổi min-width: 0 thì mới hiện ...

Nếu mún 1 dòng thôi thì dùng white-space:nowrap thay cho line-clamp.

Nếu dùng line-clamp thì có thêm 2 thuộc tính phía dưới nữa.

overflowWidth/ check có bật tootip hay ko

nếu overflow div width thì sẽ bật tootip



1. **Cách check xem các item trong array đã check hết hay chưa!**

Có thể dùng array mới và .length để check. Nhưng dùng for loop để dễ xử lý.



1. **Property does not exist on type 'JSX.IntrinsicElements'**

Vì dùng **camelCase** nên bị lỗi. Fix: dùng **PascalCase** thay thế.

1. **Chunk array. Chia/ tách array thành các array nhỏ hơn**

var i,temparray,chunk = 10;

for (i=0; i < array.length; i+=chunk) {

temparray = array.slice(i ,i+chunk);

// do whatever

}

Set chunk là 1 số. Dựa vào chunk để for loop

Cách 1: dùng chunk-lodash

Cách 2:



Cách 3: 

Cách 4: Recursion



Cách 5: dùng For. Trong for dùng slice().map()

Note: slice bao nhiêu thì bước nhảy for bấy nhiêu. Vd: slice(i, i + 100) thì for(, , i += 100)

const sendMailForUsers = async (users) => {

const usersLength = users.length

for (let i = 0; i < usersLength; i += 100) {

const requests = users.slice(i, i + 100).map((user) => {

// Mỗi đợt 100 email. và xử lý chúng

return triggerMailForUser(user)

// Async function to send the mail.

.catch(e => console.log(`Error in sending email for ${user} - ${e}`))

})

// requests sẽ có 100 hoặc ít hơn các promise đang chờ xử lý.

// [**Promise.all**](https://anonystick.com/blog-developer/Promise.all) sẽ đợi cho đến khi tất cả các promise

//đã được giải quyết và sau đó thực hiện 100 lần tiếp theo.

await Promise.all(requests)

.catch(e => console.log(`Error in sending email for the batch ${i} - ${e}`))

// Catch the error.

}

}

sendMailForUsers(userLists)

1. **Tạo internal call api/ vòng lặp gọi api trong 1 thời gian**

Nhớ chunk trước khi for



1. **Tính tổng cho array.**

Dùng reduce để sum các phần tử.

Acc: accumulate/ nơi chứa cộng dồn

Val: phần tử trong mảng

[1,2,3].reduce(function(acc, val) { return acc + val; }, 0)

1. **Ref undifined lần đầu tiên render useRef, current = undifined**



Tạo ra 1 biến để giữ ref này.

Current undifined vì ref chạy trước khi render. Nên đưa nó vào didMount hoặc đưa nó vào UseEffect {..,[]}



1. **Dùng Method get để lấy specific detail a thread/budget**

Nên dùng id từ url. Vì khi mới vào sẽ ko có id của state

1. **Chuyển string về url string**

encodeURIComponent(*str*);

1. **cách copy 1 string**

const listener = (e: ClipboardEvent) => {

e.clipboardData.setData('text/plain', text);

e.preventDefault();

};

document.addEventListener('copy', listener);

document.execCommand('copy');

document.removeEventListener('copy', listener);

1. **flatten/mảng phẳng array**

cách 1: Array.prototype.flat()

Cách 2:



Cách 3:



Cách 4: 

Cách 5:



1. **Kiểm tra có phần tử/item trong 1 mảng /array hay không**

const onCheckIsInMainList = (mainList: IFlatData[], deptIds: any[]) => {

    if(mainList && mainList.length > 0){

      let isInArr = false

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

          isInArr = mainList.some(each => each.id === deptIds[i])

          if(!isInArr) break;

      }

      return isInArr

    }

}

Muốn kiểm tra phần tử trong mảng(deptIds) nào thì dùng for loop cho mảng đó. Rồi check nó có trong một mảng khác hay ko(mainList) thì dùng some

1. **Css: fix bug ko hiện scrollbar ngang**

Đó là do để overflow ở div con. Nên phải để ở div Cha/ div cao nhất.

1. **Css: tạo 1 cái menu responsive ko cần dùng javascript/js/jquery**

những phần cần dc click dùng thẻ <label for=’check’>

tạo ra 1 cái <input type=’checkbox’ id=’check’ />

input phải nằm đầu hoặ trước element để dùng **input:checked ~ .element**

khi mún click thì dùng label click vào thì thay đổi checkbox

1. **cách làm dark theme/ dark mode**

const toggleSwitch = document.querySelector('.theme-switch input[type="checkbox"]');

const currentTheme = localStorage.getItem('theme');

if (currentTheme) {

**document.documentElement**.setAttribute('data-theme', currentTheme);

    if (currentTheme = = = 'dark') {

        toggleSwitch.checked = true;

    }

}

function switchTheme(e) {

    if (e.target.checked) {

**document.documentElement**.setAttribute('data-theme', 'dark');

        localStorage.setItem('theme', 'dark');

    } else {

**document.documentElement**.setAttribute('data-theme', 'light');

        localStorage.setItem('theme', 'light');

    }

}

toggleSwitch.addEventListener('change', switchTheme);

note: **document.documentElement sẽ gọi vào cái dom cao nhất là html**

1. **Css: tạo overlay before/after để tăng vùng chọn cho button hoặc 1 link**

.btn-overlay {

    position: relative;

}

.btn-overlay:before {

    content: '';

    position: absolute;

    top: 0;

    right: 0;

    left: 0;

    bottom: 0;

}

1. **Search/ tìm kiếm ở phía client/ giao diện**

export const stringToASCII = (str: string) => {

    try {

        return str.replace(/[àáảãạâầấẩẫậăằắẳẵặ]/g, 'a')

            .replace(/[èéẻẽẹêềếểễệ]/g, 'e')

            .replace(/[đ]/g, 'd')

            .replace(/[ìíỉĩị]/g, 'i')

            .replace(/[òóỏõọôồốổỗộơờớởỡợ]/g, 'o')

            .replace(/[ùúủũụưừứửữự]/g, 'u')

            .replace(/[ỳýỷỹỵ]/g, 'y')

    } catch {

        return ''

    }

}

export const searchASCII = (OGText: string, query: string) => {

// truyền vào 2 string to compare

    const src = OGText && OGText.trim().length > 0

        ? OGText.toLowerCase()

        : ''

    const input = query && query.trim().length > 0

        ? query.toLowerCase()

        : ''

    let strOriginalToASCII = stringToASCII(src)

    let strSearchToASCII = stringToASCII(input)

    let result = strOriginalToASCII.includes(strSearchToASCII)

    return result

}

Các regular expression này dùng để convert/ parse ký tự việt nam/ có dấu sang ko dấu

1. **Các trường hợp/ case gây hack não với điều kiện query**

const isExist = listQuery && listQuery.length > 0 || (listDefault && listDefault.length > 0 && !query)

listDefault: là list nhận dc từ API/ props.

listQuery: nhận dc khi có query/ search

query: giá trị search/ tìm kiếm/ query

condition: (có listQuery) => true **HOẶC || (**có listDefault và ko có query) => true

note: đi theo dk trái sang phải để đúng với logic

how to render()

const renderTableBody = () => {

        if (isExistDepart) {

            return query && query.trim()

                ? listDepartmentByQuery && listDepartmentByQuery.length > 0 && listDepartmentByQuery.map((item, index) => renderRowDepart(item))

                : listDepartment && listDepartment.length > 0 && listDepartment.map((item, index) => renderRowDepart(item))

        } else {

            return <td className='common-not-found' colSpan={10}>

                {Utils.Icon.empty(200)}

                <p>Hiện tại chưa có đơn vị nào sử dụng khoản mục này.</p>

            </td>

        }

    }

1. **Dùng async await để xử lý loading showWait**

async (component: FlowCreateBudget) => {

  HrvComponents.ShowWait.onStart()

        await getBudgetDefault(component) // cái này phải return là Promise<void>

    HrvComponents.ShowWait.onEnd()

}

const getBudgetDefault = async (component: FlowCreateBudget) => { async => Promise<void>

    let budgetDefault = { // will refactor

        id: null,

        name: '',

        description: '',

        from: null,

        to: null,

        cycle: null,

        allocateType: null,

        isDraft: true

    } as Interfaces.IBudgetV5

}

Note: dùng ko có dùng await trong hàm getBudgetDefault() nhưng vẫn phải để async trên đầu để nó chạy đồng bộ. ( onStart > await getBudgetDefault() > onEnd )

1. **Cách đưa về 1 mảng rỗng, ko tốn bộ nhớ RAM**



1. **Nếu có phân trang/ pagination, search/ tìm kiếm ở 1 trang list**

Nên gọi API từ URL, để dễ xử lý và đồng bộ nếu có các thông tin query/ search/ tìm kiếm. Để các thông tin query trên URL > bookmark dc.

Đặt các giá trị query vào filterValue/ filterOption (gồm: page, pageSize, query, date, createdBy,...)

didMount hoặc WillReceiveProps gọi api và set query lên URL

onInit: (component: BudgetList) => {

        events.onChangeQueryStringParameters(component, component.props)

    },

Convert cục query sang 1 object

convertQueryToObject: (paging: IPaging, filterOption: IBudgetFilterOption => {

        return {

            page: paging.page,

            page\_size: paging.page\_size,

            created\_by: filterOption.created\_by,

        } as IBudgetQueryStringParameter

    },

convertOBJ sang string

export const convertOBJ = (model: object): string => {

    if (model === null || model === undefined) return ''

    return '?' + Object.keys(model)

.filter(key => model[key] != null)

.map(key => key + '=' + model[key])

.join('&')

}

Đưa cục filterOptions lên URL

handleChangeRoute: (component: BudgetList, query: IBudgetQueryStringParameter) => {

        const queryString = convertOBJ(query) // hàm phía trên

        component.props.history.push(`${Paths.Budgets\_Manage}${queryString}`)

    },

Đưa cục obj từ state / URL vào call API

 handleGetBudgetPaging: async (component: BudgetList, query: IBudgetQueryStringParameter) => {

        const { paging, filterOption } = component.state

        const rsp = await SBudget.SManage.getBudgets(query);

}

1. **How to find out if the browser tab of the page is focused? Tìm xem tab của trình duyệt có focus hay ko?**

const isBrowserTabFocused = () => !document.hidden;

// Example

isBrowserTabFocused(); // true

1. **dùng ref để focus hoặc blur trong modal**

khi toggle modal, sẽ có 1 khoảng thời gian (ms) để modal đó hiện. Nên setTimeOut cho ref\_input.focus() để nó bắt khi modal đã bật

1. **cắt string substr để có thể làm ...**

export const substrName = (value: string, from?: number = 0, length: number  = 20) => {

    if (value != null && typeof value != "undefined") {

        if (value.length > length) {

            value = value.substr(from, length) + '...'

        }

    }

    return value;

}

1. **Component ko re-render lại khi props thay đổi**

Phải thêm key (là item.id/item.idGen) vào thì nó mới thay đổi dc. Và key phải là Unique. Dùng uuid/shortid để generate id

**Bad Practice**: ko nên đặt key = index.



1. **Cách xử lý 1 bug khi prevState === this.state (prevState same this.state)**

Dùng \_.cloneDeep(obj) để xử lý : prevState !== this.state

1. **Thay thế if, else if, if else bằng object**



1. **Kiểm tra có số có liên tục khôngconsecutive/** **continuously**

Note: phải đưa **newArr** là **number[]**

Nếu liên tục thì cho vào 1 mảng. Nếu ko thì cho ra 1 mảng riêng,

convertArr = newArr.reduce((acc, numItem) => {

        const lastArr = acc[acc.length - 1];

        const lastItem = lastArr[lastArr.length - 1];

        if (!lastItem || ((numItem - lastItem) === 1)) {

            lastArr.push(numItem)

        } else {

            acc.push([numItem]);

        }

        return acc

    }, [[]])

1. **Serialize obj. Chuyển obj thành quer url**

export const serializeTicketFilter = (obj) => {

    var str = [];

    for (var p in obj)

        if (obj.hasOwnProperty(p) && obj[p] != null) {

            str.push(p + '=' + obj[p]);

        }

    return str.join('&');

}

1. **Đưa obj lên url**

export const onPushtoUrl = (input: any, rootPath: string) => {

    if (!input) return

    let parseToSerialize = serializeTicketFilter(input) // hàm chuyển obj sang query url

    Utils.history.push(rootPath + '?' + parseToSerialize)

}

1. **Lấy query từ url và chuyển sang obj**

export const onPullObjFromUrl = (input: string) => {

    if (!input) return

    let strParam = input.split('?')[1]

    let getParamFromUrl = new URLSearchParams(strParam) // lấy phần sau ?, có thể dùng query-string

    let obj = {}

    getParamFromUrl.forEach((value, key) => {

        obj[key] = value

    })

    return obj

}

// cách 2:

let c = 'https://test/com?x=a&y=b#old&p=8397543#dsf43'

function getUrlVars(url) {

var vars = {};

var parts = url.replace(/[?&]+([^=&]+)=([^&]\*)/gi, function(m,key,value) {

vars[key] = value;

});

return vars;

}



Cách này sẽ lấy giá trị (gồm các ký tự đặc biệt lun) nên ko lo bị sai

1. **Loadmore với Element div**

  componentDidMount() {

        if (this.ref\_body\_tabs)

this.ref\_body\_tabs.addEventListener('scroll', (e) => onScrollLoadmore(this, e))

    }

    componentWillUnmount() {

        if (this.ref\_body\_tabs)

this.ref\_body\_tabs.removeEventListener('scroll', (e) => onScrollLoadmore(this, e));

    }

const onScrollLoadmore = async (component: TicketFeedbackList, e) => {

    if (component.state.isLoadMore

|| component.state.totalCount === component.state.ticketList.length) return

    if (e) e.preventDefault();

    let element = e.target as HTMLDivElement;

    if (element.scrollTop > 0 //chặn khi chuyển tab ko call api nữa

&& (element.scrollTop + element.clientHeight >= element.scrollHeight - 10)) {

        //setState and call API

}

1. **Vài trường hợp stretch với flexbox**

Muốn cho child stretch to Parent:

Parent phải là display: flex, child là flex: 1/ width: 100%

Nếu child là flex-direction: column, thì thêm cho child height: 100%.

\* vì khi parent có thuộc tính display: flex thì các thằng con sẽ chỉ lấy width, height là độ dài nội dung thôi.

1. **Mock up JSON API**

<https://designer.mocky.io/>

1. **Scrollbar not take space**

overflow-x/overflow-y: overlay

1. **For Multiple Conditions, use Array.includes (Or operator ||) Kiểm tra nhiều điều kiện**

const checkCarModel = (model) => {

const models = ['vinfast fadil', 'Hyundai Accent']; // can change here.

if(models.includes(model)) {

console.log('model valid');

}

}

1. **Sử dụng ... spread operator**

Nếu arr đó là null hoặc undifined thì khi sử dụng ... bị trắng trang vì lỗi slice



1. **Cách bật Modal/ Popup với các cells,**

Khi xử lý click vào 1 cell để bật Modal của cell đó lên thì nên dùng state cục bộ của component để tránh việc khi state global = true thì tất cả modal của các cell đều bật.



modal/popup nên nằm tương ứng trong từng cell.

1. **Format money thành đơn vị**

export const moneyFormatter = (num: number) => {

    if (num == null || isNaN(num)) return

    if (num >= 1000000000) { //billion

        return (num / 1000000000).toFixed(1).replace(/\.0$/, '') + ' tỷ';

    }

    if (num >= 1000000) { // million

        return (num / 1000000).toFixed(1).replace(/\.0$/, '') + ' triệu';

    }

    if (num >= 1000) { // thousand

        return (num / 1000).toFixed(1).replace(/\.0$/, '') + ' ngàn';

    }

    return num;

}

1. **Tạo div dummy để làm thanh scrollbar trên top**

Lưu ý: phải lam dummy y như format để cùng độ dài và hiện thanh scroll

private \_\_renderContentRight = () => {

        const { selectedCycles, settingColNames } = AvailableReportStore

        return <ScrollSync horizontal={true}>

            <div className="col-md-7 pl-0">

                <ScrollSyncPane group='horizontal'> 1

                    <div className="available-rp\_wrapper-dummy">

                        {selectedCycles && selectedCycles.length > 0 && selectedCycles.map(cyc => <div key={cyc} className="available-rp\_dummy\_child">

                 {settingColNames.map(name => <div key={name} className="dummy\_child\_item"></div>)}

                        </div>)}

                    </div>

                </ScrollSyncPane>

                <ScrollSyncPane group='horizontal'> 2

                    <div className="available-rp\_wrapper-right">

                        {/\* map through cycle \*/}

                        {selectedCycles && selectedCycles.length > 0

                            ? selectedCycles.map(cyc => <CycleGroupSection key={cyc} cycleId={cyc} />)

                            : 'Chưa có chu kỳ'

                        }

                    </div>

                </ScrollSyncPane>

            </div>

        </ScrollSync>

    }

1. **Svg- Text Layout - No Word Wrap**

There is no automatic word wrapping in SVG. You will have to position the text yourself, and break it into multiple lines. You can get some help from the relative positioning possible with the **<tspan>** element.

It is also possible to layout text along a path, for instance along a circle or spline. You can do this using the **<textpath>** element.

1. **Thay thế cho tab network**

<https://www.charlesproxy.com/>

1. **c# - aggregate mongodb**

test trên robo 3T trước

    var bsonDocumentGroup = new BsonDocument()

                .Add("\_id", "$CycleId")

                .Add("Total", new BsonDocument()

                    .Add("$sum", "$Amount"));

   var result = await collection.Aggregate()

                .Match(filter)

                .Group(bsonDocumentGroup)

                .ToListAsync();

            return result

                .Select(r => BsonSerializer.Deserialize<GroupBySingleFieldCommonModel<long>>(r)).ToList();

1. **Các trường hợp tìm search, query 1 cái structure, array, list**

Ban đầu dùng 1 cái list/arr từ api (ko có search). Sau đó dùng search để tìm kiếm trên cái list/arr đó.

Nên tính trước việc sẽ có search, tạo ra 2 state vd: structures:[], queriedStructures: []

  this.state = {

            accounts: null,

            queryAccounts: null,

}

Sau đó ở phần render check điều kiện. Nếu query.trim().length > 0 ? queriedStructures : structures

Note: làm vậy thì chỉ cần thay thế cái structures ngoài cùng là xong. Còn phần renderItem vẫn như cũ

1. **Short-hand for fetch API**

async function makeRequest(url: string, log?: (msg: string) => void) {

log?.(`Request started at ${new Date().toISOString()}`);

// roughly equivalent to

// if (log != null) {

// log(`Request started at ${new Date().toISOString()}`);

// }

const result = (await fetch(url)).json();

// const rsp = await fetch(url)

// const data = rsp.json()

log?.(`Request finished at at ${new Date().toISOString()}`);

return result;

}

Gồm 2 phần: **(await fetch(url))** và **.json()**

**(await fetch(url)):** trả về 1 promise, nên phải có await

.json(): sau khi **await fetch(url)** thì sẽ gọi hàm json();

1. **Vài thủ thuật xử lý String**

* Capitalize/ LowerCase

const capitalize = ([first, ...rest]) =>

first.toUpperCase() + rest.join('');

// first. toLowerCase() + rest.join('');

capitalize('fooBar'); // 'FooBar'

capitalize('fooBar', true); // 'Foobar'

* capitalizeEveryWord

const capitalizeEveryWord = str => str.replace(/\b[a-z]/g, char => char.toUpperCase());

capitalizeEveryWord('hello world!'); *// 'Hello World!'*

* stripHTMLTags

const stripHTMLTags = str => str.replace(/<[^>]\*>/g, '');

stripHTMLTags('<p><em>lorem</em> <strong>ipsum</strong></p>'); *// 'lorem ipsum'*

* convert String thành một Array (bỏ các ký tự đặc biệt)

const words = (str, pattern = /[^a-zA-Z-]+/) => str.split(pattern).filter(Boolean);

words('I love javaScript!!'); // ["I", "love", "javaScript"]

words('python, javaScript & coffee'); // ["python", "javaScript", "coffee"]

1. **Chuyển đổi một array object sang object**

const people = [

{

area: "GTZ",

name: "ABC",

age: 30,

},

{

area: "SZR",

name: "DEF",

age: 29,

}

];

const map = people.reduce((arr, item) => {

const { name, ...rest } = item;

arr[name] = rest;

return arr;

}, {});

console.log(map);

// { ABC: { area: "GTZ", age: 30 }, DEF: { area: "SZR", age: 29 } }

Học thêm dc 1 trò là destructuring 1 cái field cụ thể trong object const { name, ...rest } = item

1. **Tìm phần tử khác nhau giữa 2 array**

const diffItem = (arr = [], otherArr = []) =>

arr.reduce((t, v) => (!otherArr.includes(v) && t.push(v), t), []);

const a = [1, 2, 3, 4, 5];

const b = [2, 3, 6];

console.log(diffItem(a, b)); // > Array [1, 4, 5]

1. **CSS aspect-ratio: thuộc tính theo tỷ lệ màn hình**

Từ trước đến giờ mỗi lần muốn tạo 1 khối (hình ảnh/video) theo 1 tỷ lệ 16:9, 4:3, 3:2 mình thường làm Aspect Ratio bằng cách sử dụng padding-top. Ví dụ:

Tỉ lệ 1:1 -> padding-top: 100%

Tỉ lệ 4:3 -> padding-top: 75%

Tỉ lệ 3:2 -> padding-top: 66.67%

Tỉ lệ 16:9 -> padding-top: 56.25%

khả năng responsive -> padding-top: (height\_image/width\_image) \* 100%

div {

aspect-ratio: 16 / 9;

@supports not (aspect-ratio: 16 / 9) {

padding-top: 56.25%;

}

}

@support để dành cho fallbacks(nếu ko hỗ trợ thì sẽ chạy css trong @support)

1. **CSS cho cursor khi focus input**

input {

caret-color: red;

}

1. **useReducer in React**

const [state, dispatch] = useReducer(reducer, initialArg, init);

nhận vào reducer, initState

return state và dispatch

dispatch nhận vào 1 action: dispatch({type: 'decrement'})} (action là 1 obj và luôn có thuộc tính mặc định là type)

reducer(input của useReducer) dựa vào action.type để quyết định return state như thế nào

1. **useMemo in React**

const memoizedValue = React.useMemo(() => computeExpensiveValue(a, b), [a]);

Returns a [memoized](https://en.wikipedia.org/wiki/Memoization) **value**. (trả về là 1 giá trị)

useMemo dựa vào dependancy [a]: nếu a changes thì chạy lại hàm computeExpensiveValue.

1. **useCallback in React**

const increment = useCallback(() => {

setCount(count + 1)

}, [count])

giống shouldComponentUpdate

Returns a [memoized](https://en.wikipedia.org/wiki/Memoization) **callback**. (trả về là 1 hàm)

1. **useImperativeHandle in React**



Wrap component trong forwardRef (thường là ở dưới cùng với cái export)

Chỗ input của Component thêm 1 input ref

Trong Component dùng useImperativeHandle để chìa cái function ra bên ngoài

Note: case này dc dùng khi các component ông, cha cần sử dụng hàm trong component con (giống như public 1 hàm trong component con và dùng ref để trỏ đến component con)

1. **useLayoutEffect in React**

useLayoutEffect(() => {

........ logic

// tính toán các width,size,height của elements

}, [])

(chạy sau khi render và trước khi vẽ ra màn hình)

Caution: sẽ block visual update cho đến khi callback trong useLayoutEffect xong.

1. **useDebugValue in React**

dùng khi sử dụng custom hook (trong react-devtool). Để tạo label cho custom hook

useDebugValue(value)

value sẽ là label cho custom hook

1. **vài case cần lưu ý nếu muốn push lên url**

Khi push các thông tin lên url, ta chỉ push list of ids , date, string. (ngoại trừ dạng object).

Vì ko push dc object lên nên chỉ có id,string,date,... thì phải gọi thêm api để lấy thông tin về id đó. Vd: dùng id đó để get thông tin user(để lấy name,chức danh, phòng ban,...).

1. **Bị re-render khi dùng hook (do dùng default props)**

Ko nên để giá trị mặc định list = [], hay obj= {} trong {...destructuring} props. Vì khi re-render nó sẽ gán mới cho list/obj đó. Vì vậy sẽ trigger useEffect nào có dependancy là obj/list (nếu prop list, prop obj đó để gán vào State)

Nguyên nhân: do props.list/props.obj: undefined thì luôn !== giá trị mặc định []/{}.

Khắc phục: có thể để giá trị mặc định list = [], hay obj= {} trong {...destructuring} props. Và useEffect **check dependancy là listState/objState**. Cách này giúp gọn hơn và chỉ cần check State thôi. Ko cần check props

 const { array = [], boolean, children, number, obj, string } = props

    //props.array: undefined !== [] => setState loop

    const [arrayState, setArrayState] = React.useState<any[]>(array)

    // const [objState, setobjState] = React.useState<any>(obj)

    React.useEffect(() => {

        if (array == null) return

        console.log('arrayState changed')

        setArrayState(array)

    }, [arrayState])

1. **Vấn đề Lưu name chung với id**

Nếu HR database thay đổi tên (có cơ chế thay đổi tên user) thì việc lưu name chung với id sẽ bị sai

Khác biệt loadmore và pagination(paging:1,2,3 pageSize:10,20,50)

1. **Loadmore: sẽ load theo thứ tự và ko thay đổi pageSize.**

* Ưu: có thể loadmore trên client(get all data rồi load dựa theo state – 1,2,3 ở client) => ko cần chunk data
* Nhược: phải addlistener. Khi đến cuối trang tự gọi, dễ spam.

Pagination: có thể nhảy đến số trang bất kỳ, và có thể thay đổi pageSize

* Ưu: linh hoạt trong paging và pageSize.
* Nhược: nếu làm ở Client thì phải chunk ra theo pageSize để có thể dùng paging gọi đến chunk đó

1. **UseDebounce**

const useDebounce = (value, delay) => {

const [debouncedValue, setDebouncedValue] = React.useState(value);

React.useEffect(() => {

const handler = setTimeout(() => {

setDebouncedValue(value);

}, delay);

return () => {

clearTimeout(handler);

};

}, [value]);

return debouncedValue;

};

1. **Tab list/according**

Dùng NavLink để đưa lên URL. Ko cần push lên URL.

