17 JULY 2019

REACT AND HIS FRIENDS

REACT

A JavaScript library for **building user interfaces**

REACT PHILOSOPHY

- **UI = render(state)**
- If you want to change UI, change the state
- After you change the state, the UI should re-render (how to re-render efficiently is another question)

REACT PHILOSOPHY

- **UI = render(state)**
- Some UI are not part of state:
  - **Scrolling position**
  - **Focusing**
  - **Selection**
  - **Hovering**
  - **Animation**
- They are controlled by native DOM/CSS.

WHAT IS STATE?

- Figure out what may change in lifecycle
- In React, they come from **this.state**, or **this.props**.

INPUT

- <input value="Hello" type="text" />
  - In traditional HTML
  - In React

INPUT (REACT)

- Input value is **strictly** determined by its **value** attribute
- If you want to change the input value, change the state.

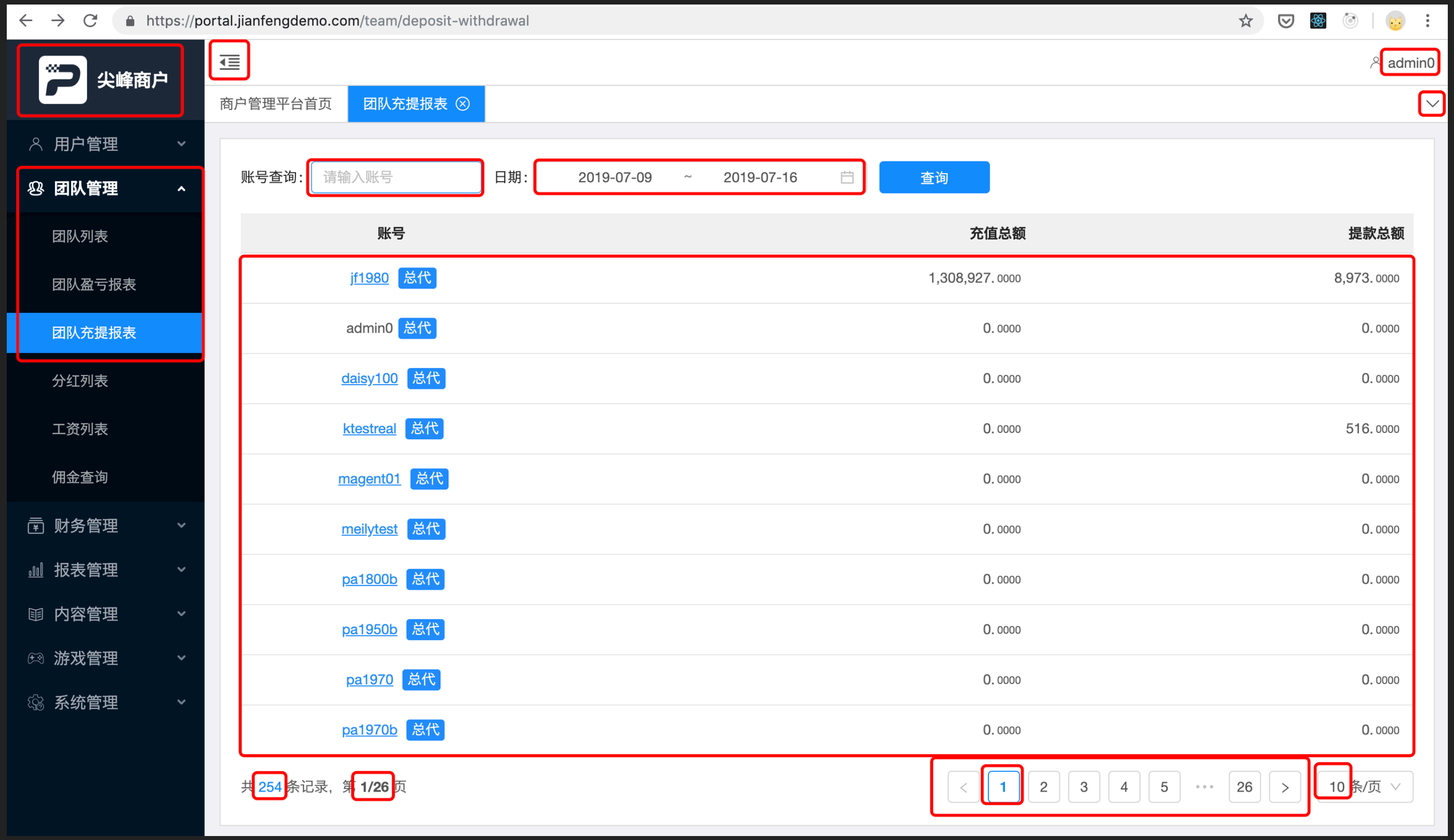
- 2-min demo for using **this.state** as input state

INPUT (REACT)

- Two way binding? Something like:
  <input bindValue={this.state.value} type="text" />
- React Philosophy does not allow so.
  But Angular/Vue does.
- Ref:
  https://reactjs.org/docs/two-way-binding-helpers.html
  https://www.zhihu.com/question/300849926

PORTAL WEBSITE PAGE

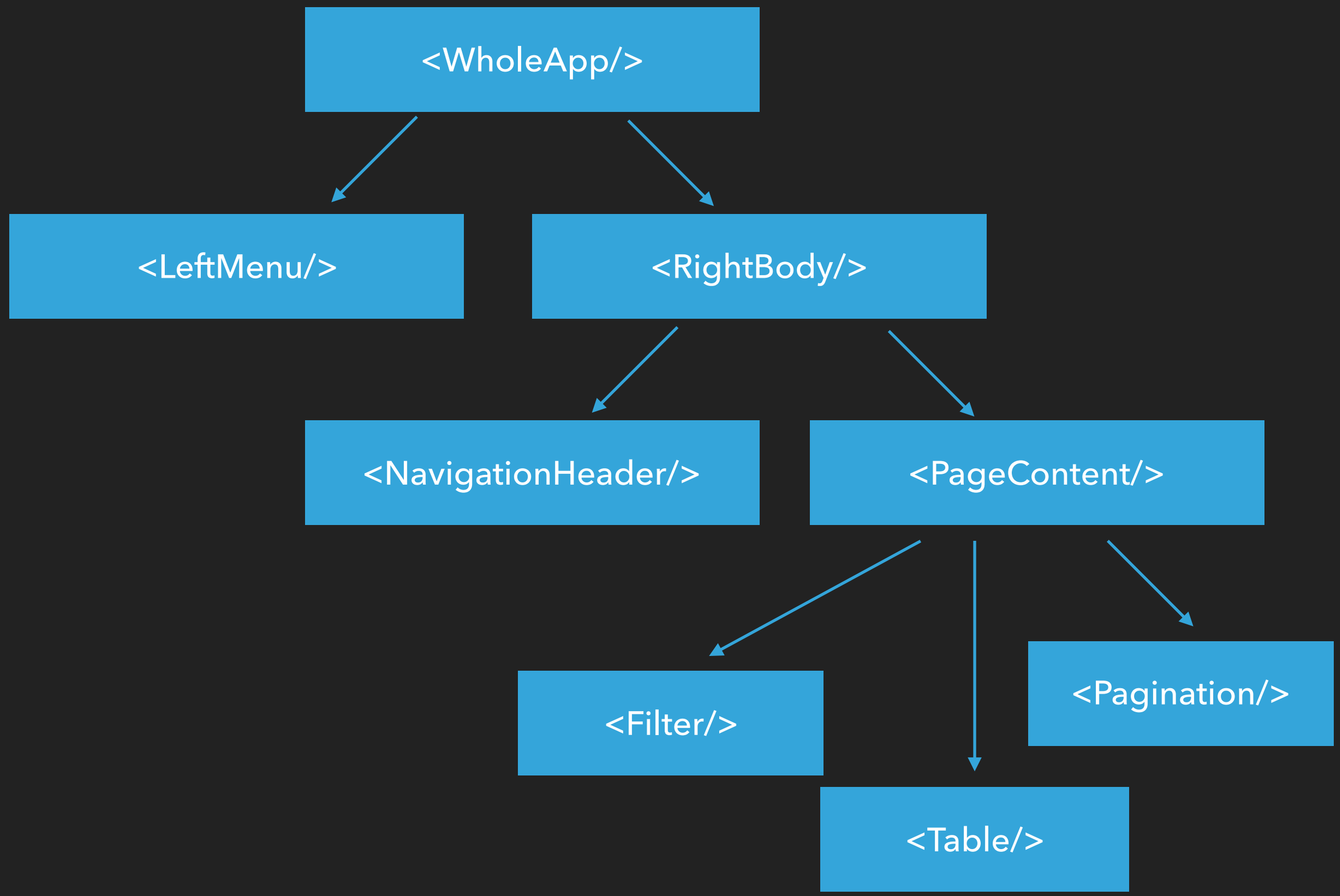
- https://portal.jianfengdemo.com/team/deposit-withdrawal
- Please find out **all states in this page**, as many as possible
- To simplify, suppose:
  - The user has logged in
  - The user has full permissions



{

logo, currentMenuGroup, currentMenuItem, isMenuCollapsed, currentUserName, showTabMenu
filterName, filterFromDate, filterEndDate,
tableRows, totalRecord, totalSize, pageIndex, pageSize

}


<WholeApp/>
<LeftMenu/>
<RightBody/>
<NavigationHeader/>
<PageContent/>
<Filter/>
<Table/>
<Pagination/>

REVIEW THE STATES

- **logo**
  should not be in states, it is fixed in lifecycle
- **currentMenuGroup/currentMenuItem**
  Actually, currentMenuItem is also determined by current URL.

REVIEW THE STATES

- **filterName/filterFromDate/filterEnd pageIndex/pageSize**
  Actually as one object, i.e. **APIRequest**
- **tableRows/totalRecord/totalSize**
  Actually as one object, i.e. **APIResponse**

REVIEW THE STATES

- **tableLoadingStatus**
  Boolean, true during API call, show a loading icon on top of table component

REVIEW THE STATES (SUMMARY)

- **WholeState**
  - **LoadingState**
    - **table**

  - **LocationState**
    - **location** (DOM-like interface)

  - **AppState**
    - **globalLayoutState**

      (isMenuCollapsed / currentMenuGroup / showTabMenu / currentUser)
    - **teamWithdrawalDepositPageState**

      (request / response)

WHERE TO SERVE THE STATE OBJECT 1

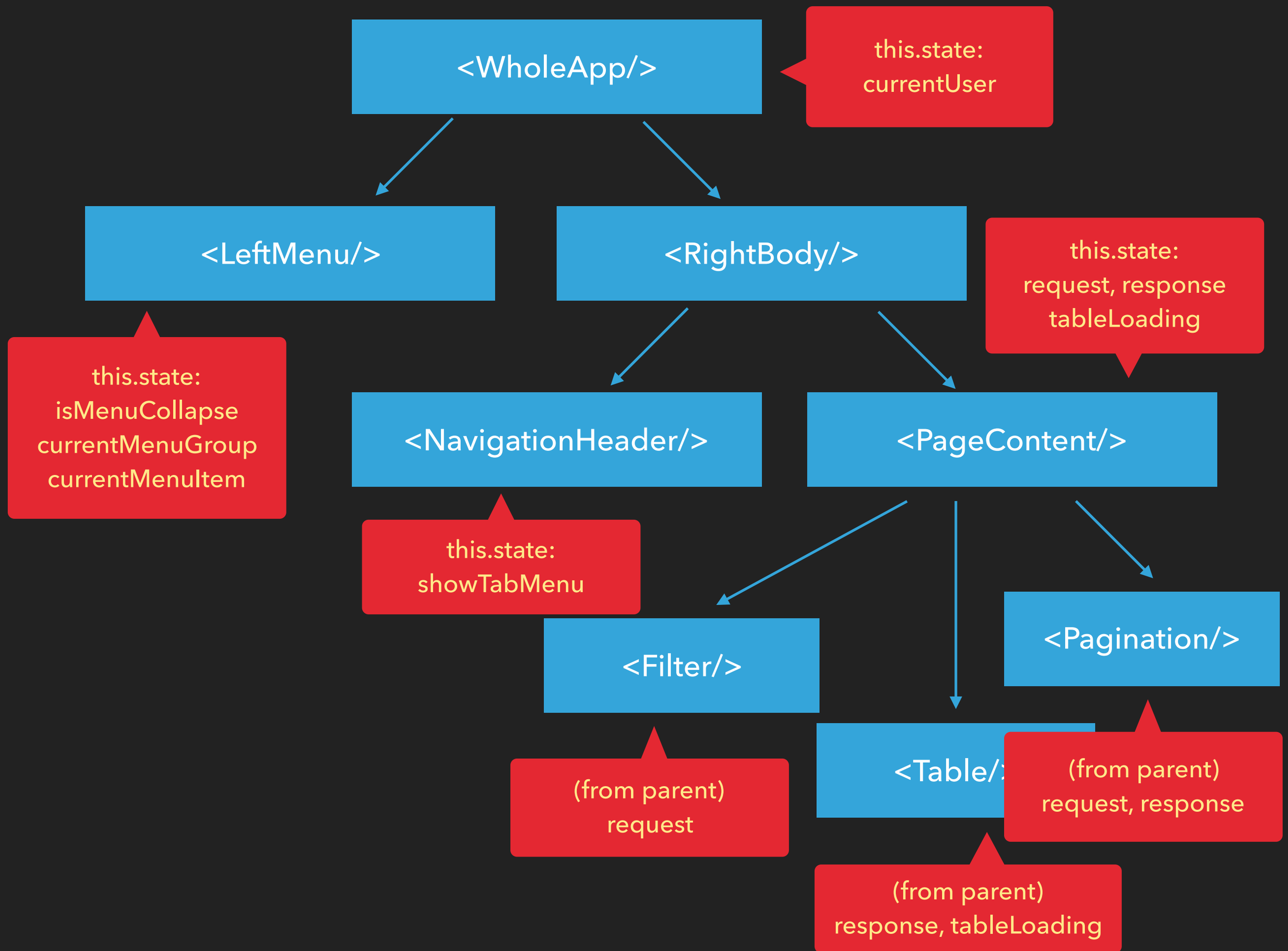
- Create this.state in the component where used.
  If the state is used in two components, create in their both parent.
- However, it is **hard** to determine **correctly** at one glance.

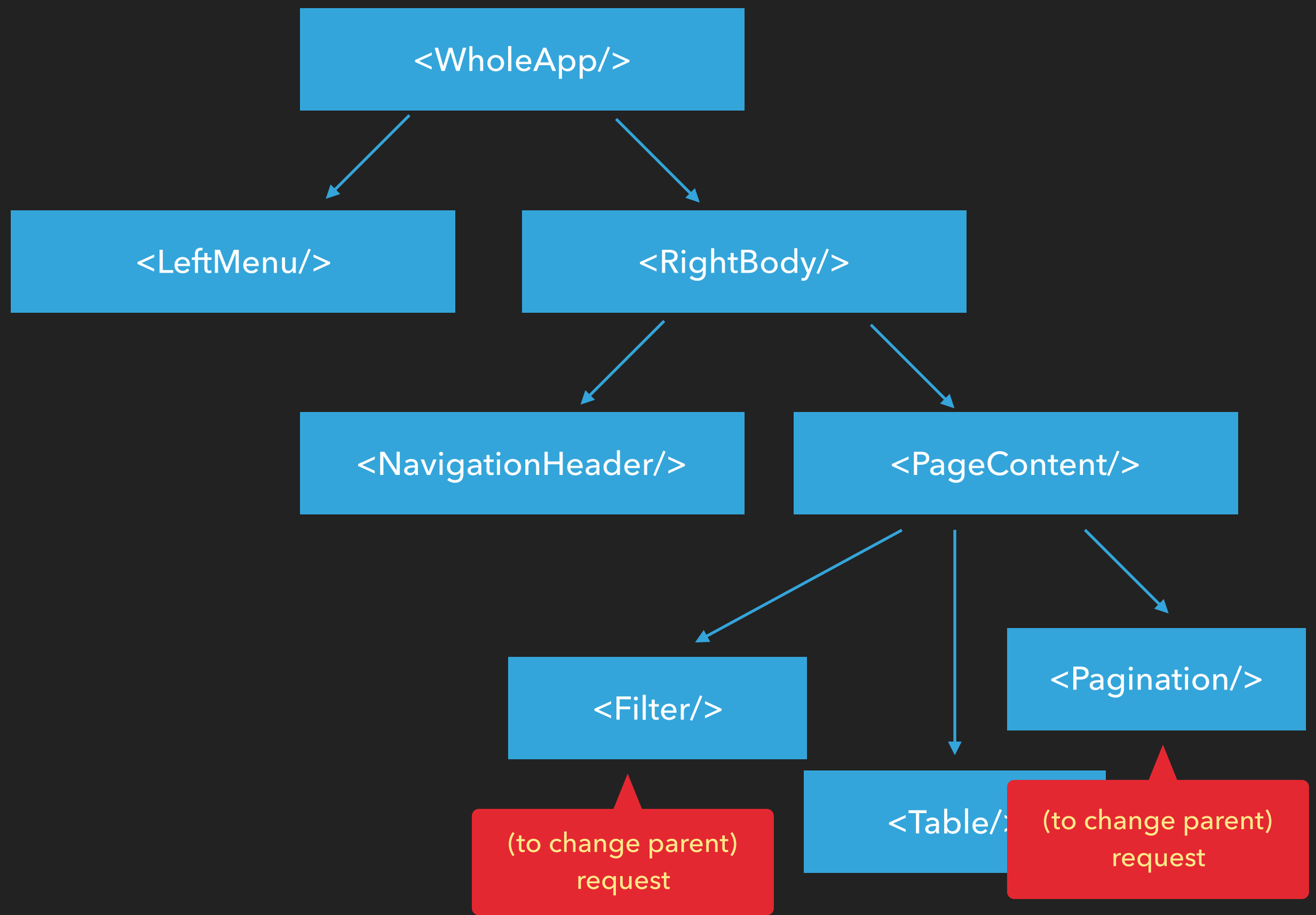
WHERE TO SERVE THE STATE OBJECT 1

- Create **this.state** in the component where used.

  But **this.state** can be only accessed/modified by current component (as well as its children, if passed down).

- However, it is **hard** to determine **correctly** at one glance.
- Demo for examples (both game/portal website): *pageIndex / showPopup / isMenuCollapse*


<WholeApp/>
this.state:
currentUser
<LeftMenu/>
<RightBody/>
this.state:
request, response
tableLoading
this.state:
isMenuCollapse
currentMenuGroup
currentMenuItem
<NavigationHeader/>
<PageContent/>
this.state:
showTabMenu
<Filter/>
<Pagination/>
(from parent)
request
<Table/
(from parent)
request, response
(from parent)
response, tableLoading


<WholeApp/>
<LeftMenu/>
<RightBody/>
<NavigationHeader/>
<PageContent/>
<Filter/>
<Table/>
<Pagination/>
(to change parent) request
(to change parent) request

WHERE TO SERVE THE STATE OBJECT 2

- We create the whole object, regardless of which component might use.

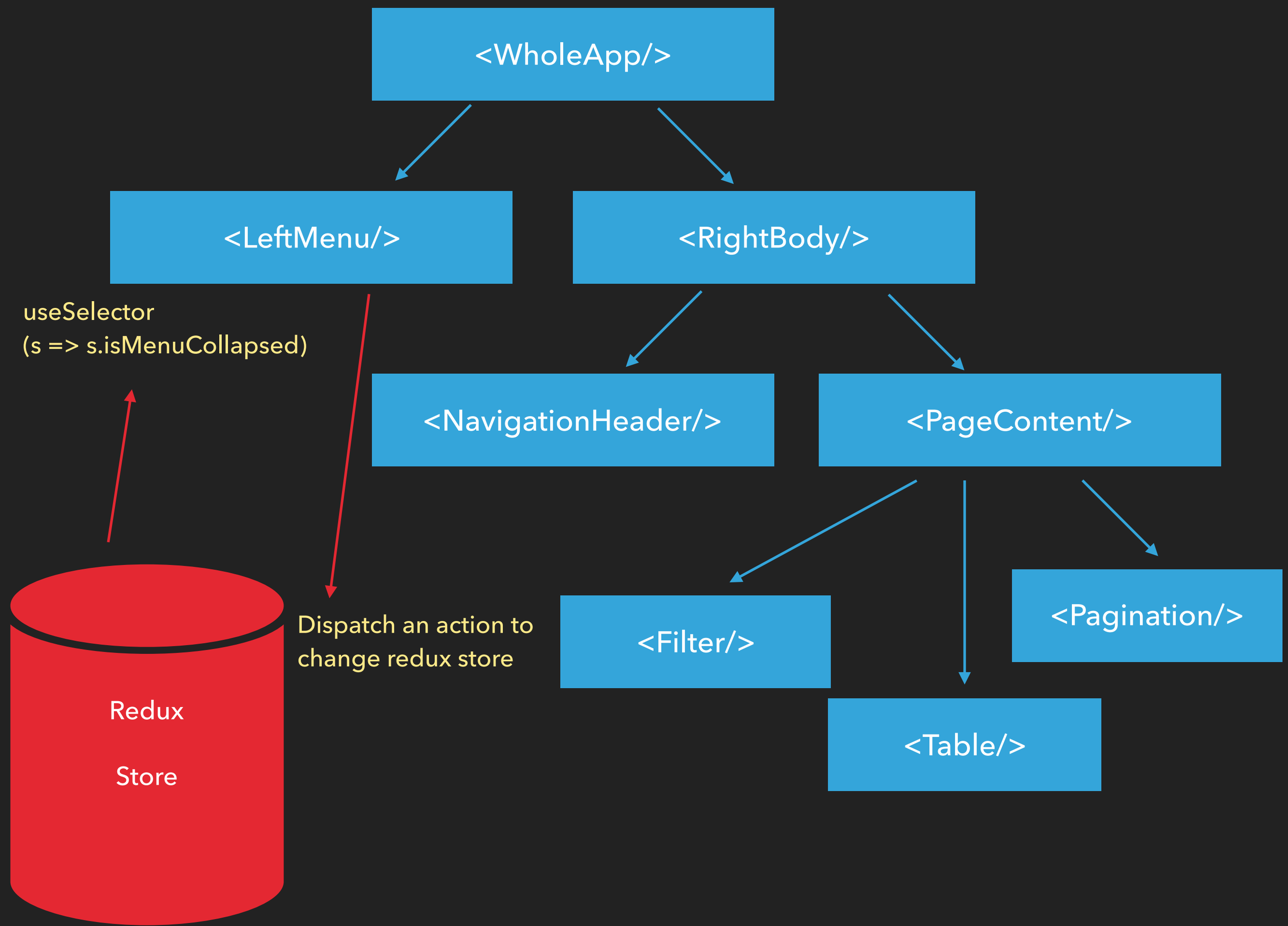
  All states from come one **Single Source of truth**.
- That's the **Redux** philosophy.
- Benefits:
  - Easy to debug/time-travel

REDUX

A JavaScript library of **predictable state container**

REDUX - HOW TO CHANGE STATE

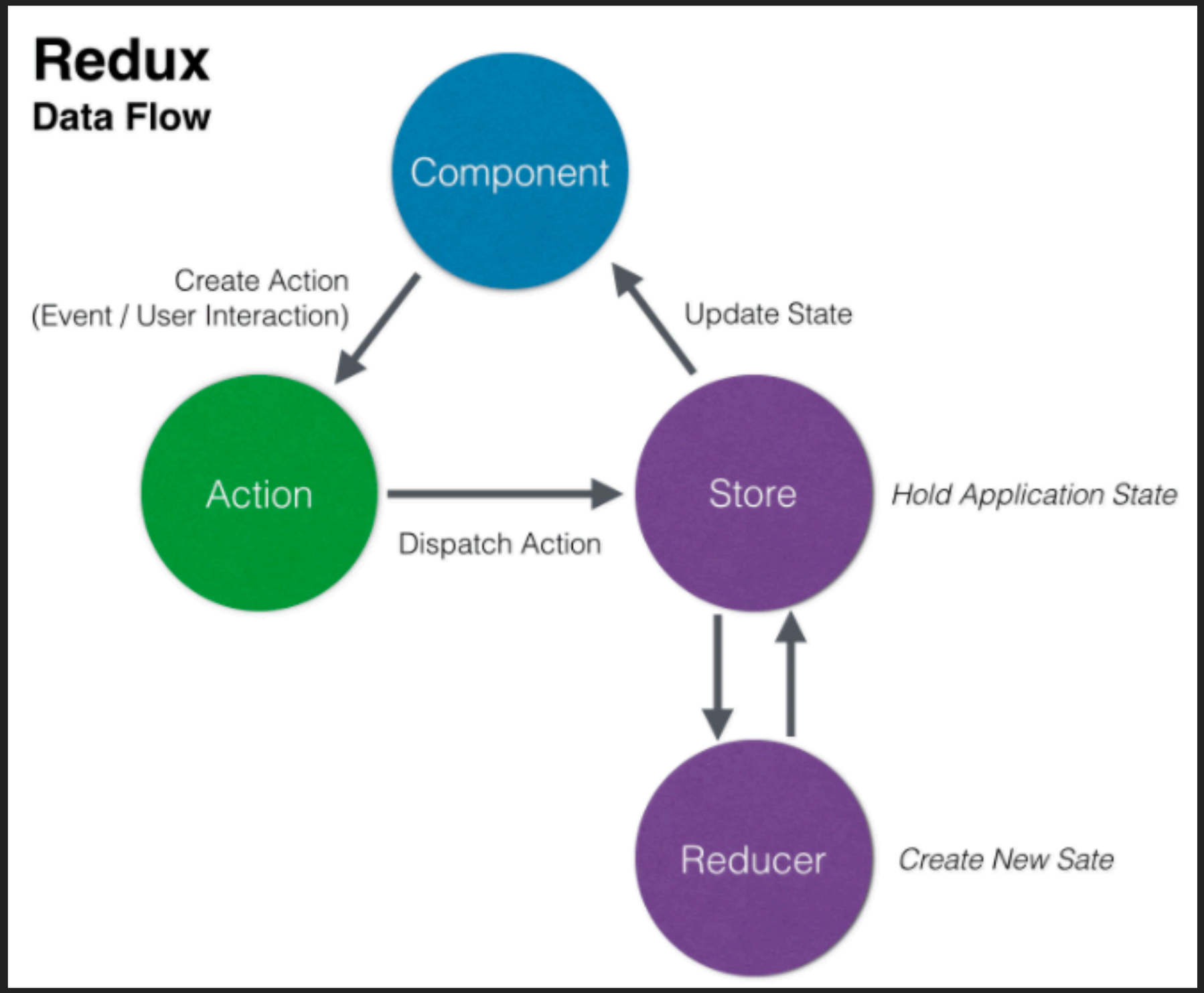
- **Store** as the whole state tree
- Dispatch an **action**
- Then **reducer** returns a **new state** after receiving the action.


<WholeApp/>
<LeftMenu/>
<RightBody/>
useSelector
(s => s.isMenuCollapsed)
<NavigationHeader/>
<PageContent/>
Dispatch an action to
change redux store
<Filter/>
<Pagination/>
<Table/>
Redux
Store

REDUX PHILOSOPHY

- **UI = render(state)**
- **NewState = reducer(PrevState, action)**

REDUX PHILOSOPHY



REDUX – TOGGLE MENU COLLAPSE STATE

- State: **S**
  Action: **{type: string; payload: any}**
  Reducer: **(prevState: S, action: Action) => S**
- Dispatch: **(action: Action) => void**

REDUX – TOGGLE MENU COLLAPSE STATE

```
function toggleMenuReducer(prevState, action) {
    if (action.type === "TOGGLE_MENU") {
        const newState = copy(prevState);
        newState.isMenuCollapsed = Boolean(action.payload);
        return newState;
    }

    return prevState;
}
```

REDUX – TOGGLE MENU COLLAPSE STATE

- onOpenMenu = () => {
  this.props.dispatch({type: "TOGGLE_MENU", payload: true});
  }

- onCloseMenu = () => {
  this.props.dispatch({type: "TOGGLE_MENU", payload: false});
  }

REDUX - TOGGLE MENU COLLAPSE STATE

- const openMenuAction =
  () => {type: "TOGGLE_MENU", payload: true};
  const closeMenuAction =
  () => {type: "TOGGLE_MENU", payload: false}

  // Above are called ActionCreator, to simplify code

- onOpenMenu = () => this.props.dispatch(**openMenuAction()**);
- onCloseMenu = () => this.props.dispatch(**closeMenuAction()**);

REDUX

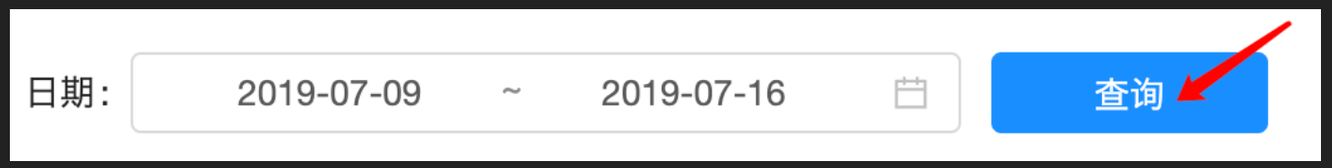
- Initial state (as reducer first default parameter)
- Put all reducers together

- <Provider store={**createStore(.....all reducers.....)**}>
  <WholeAppComponent>
  </Provider>

REDUX DISADVANTAGES

- Too many boilerplates
- Action/reducer not hierarchy
- **Async issue**

CALL API ISSUE



- What should **the reducer look like** after clicking “查询”? Do not consider error case now.
- 1, Change **LoadingState.table = true**
- 2, Call API, but it is async
- 3, After a while, API returns, change **AppState.XXPageState.response = API Response**
- 4, Change **LoadingState.table = true**

CALL API ISSUE

- At least 3 actions dispatched:
  set loading true / set API response / set loading false
- Where to call API?
  Redux reducer cannot do this

CALL API ISSUE

- An ideal solution
- function callAPI(request) {

```
    dispatch({type: “TABLE_LOADING”, payload: true});
    XXAPIService.search(request).then(response => {
        dispatch({type: “FILL_RESPONSE”, payload: response});
        dispatch({type: “TABLE_LOADING”, payload: false});
    });
}
```

CALL API ISSUE

- Dispatch only exists in **this.props.dispatch** (inside component)
- Traditional reducer should not access dispatch.
  By redux philosophy, these code should be written outside React component.
- **UI / logic separation principle**
- A good thing is, Redux provides **Middleware** mechanism, which can inject dispatch/getState object into the middleware functions.

REDUX THUNK

- https://github.com/reduxjs/redux-thunk

CALL API ISSUE USING THUNK

```
function callAPI(request) {
  return (dispatch) => {
    dispatch({type: “TABLE_LOADING”, payload: true});
    XXAPIService.search(request).then(response => {
        dispatch({type: “FILL_RESPONSE”, payload: response});
        dispatch({type: “TABLE_LOADING”, payload: false});
    });
  }
}
```

CALL API ISSUE USING THUNK (USING ES6 STYLE)

```
function callAPI(request) {
  return async (dispatch) => {
    dispatch({type: “TABLE_LOADING”, payload: true});
    const response = await XXAPIService.search(request);
    dispatch({type: “FILL_RESPONSE”, payload: response});
    dispatch({type: “TABLE_LOADING”, payload: false});
  }
}
```

REDUX THUNK

- onSearchButtonClick = () => {
  this.props.dispatch(**callAPI(request)**)
  }

CALL API ISSUE USING THUNK

- Is callAPI() a reducer?
  Is callAPI() an ActionCreator?
- Both no.

  callAPI() returns **a function**.

  But the black magic is, redux-thunk makes it **like an ActionCreator**.

REDUX THUNK

```
1  function createThunkMiddleware() {
2    return store => next => action => {
3      if (typeof action === 'function') {
4        return action(store.dispatch, store.getState);
5      }
6      return next(action);
7    };
8  }
```

REDUX THUNK

- If you are using JavaScript, everything works!
- However, in TypeScript, it does not match Redux's definition.

- dispatch() requires {type: string; payload: any} parameter
- But callAPI() returns a function!

REDUX THUNK HACKS IT

- Using its own type definition to overwrite Redux's.

  https://github.com/reduxjs/redux-thunk/blob/master/index.d.ts

REDUX SAGA

- Redux Saga is a JavaScript library that aims to make application side effects easier to manage.

REDUX SAGA

- Saga is a generator function, you can use following operators to do many things.
- It uses **yield** over **async/await**

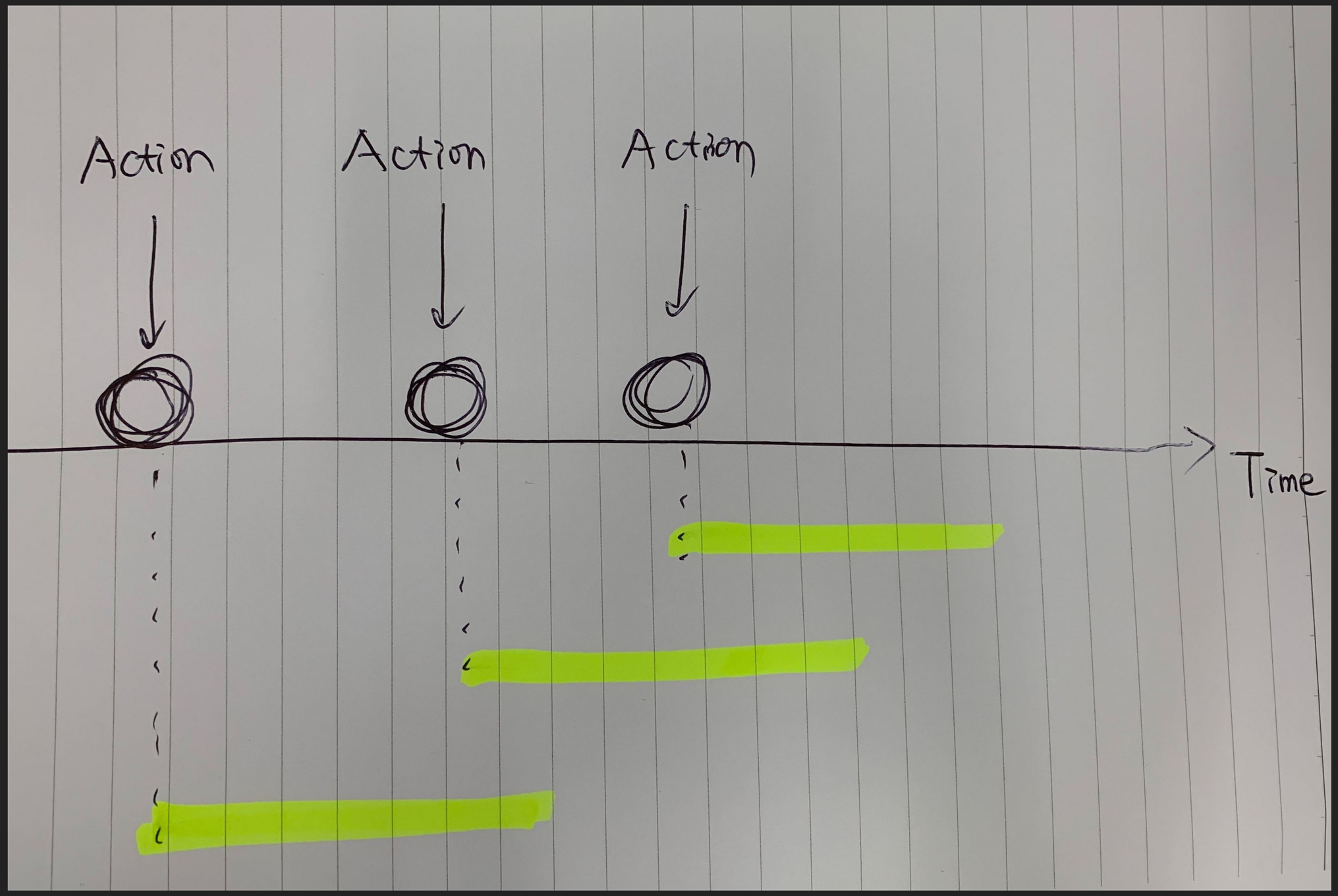
REDUX SAGA

- Saga is much more powerful (in async flow handling)
- Built-in operators:
  call: invoke sync/async functions
  put: store.dispatch
  select: store.getState
  delay: A Promise resolved after some time

WHAT IS TAKE?

- When an action comes, Redux Saga uses **take** to handle this action, if the parameter matches.
- **takeLead**: Only handle the first, until it completes
- **takeLatest**: Only handle the latest, terminate the previous if it is already handling one
- **takeEvery**: Handle every
- **throttle**: Handle in a limited frequency

WHAT IS TAKE?



CALL API ISSUE USING SAGA (USING ES6 STYLE)

```
function *callAPI(request) {
    yield put({type: "TABLE_LOADING", payload: true});
    const response = yield call(XXAPIService.search, request);
    yield put({type: "FILL_RESPONSE", payload: response});
    yield put({type: "TABLE_LOADING", payload: false});
}
```

CALL API ISSUE USING SAGA (USING ES6 STYLE)

- onSearchButtonClick = () => {
  this.props.dispatch({**type: "WATCH_CALL_API", payload: request**})
  }

```
function *watchCallAPI() {
  yield takeEvery("WATCH_CALL_API", callAPI);
}
```

CALL API ISSUE USING SAGA (USING ES6 STYLE)

- const sagaMiddleware = createSagaMiddleware();

  sagaMiddleware.run(....all watchXXXX....)

  <Provider store={**createStore(..all reducers.., sagaMiddleware)**}>
  <WholeAppComponent>
  </Provider>

REDUX SAGA DISADVANTAGES

- Too many boilerplates, like pure Redux

EMMMMM…

- That's why we created our **core-fe**
- Based on React, Redux, Redux Saga
  - Module based structure
  - Easy way to handle async/sync flow
  - Easy way to handle all errors
  - Utility
    e.g: **network, routing, event logging, decorators**

CORE-FE

```
class XXXModule extends Module {
    @loading("table")
    *callAPI(request) {
        const response = yield call(XXAPIService.search, request);
        this.setState({response})
    }
}

export const xxxActions = register(XXXModule,
initialState, "xxx");
```

CORE-FE

- onSearchButtonClick = () => {
  this.props.dispatch(**xxxActions.callAPI(request)**)
  }

TYPE ANALYSIS

- **XXXModule.callAPI** is a generator function
- After register(), xxxActions has exact the same interface of XXXModule, but **xxxActions.callAPI** is an ActionCreator.
- So, we need no type hacking for Redux.

EXTENDING MODULE

- In the module, we can access/update state via:
  - **this.state**
  - **this.setState({....})**
- A React-like API for business logic module, to handle redux state.

UNDER THE HOOD

- **register()** is important:
  - Transform every generator function into ActionCreator
    (...args: T) => {type: string; payload: T}
  - Generator:
    UserModule.login(name, password)
    ActionCreator:
    (name, password) => {type: “user/login”, payload: [name, password]}
  - Dynamic add initial state to redux store
  - Add every ActionCreator into global map (HandlerMap)
    Key: <ActionType string> / **Value: Generator function**

UNDER THE HOOD

- A global saga for watching all actions

```
function* globalSaga() {
    yield takeEvery("*", handleSagaByAction)
}

function* handleSagaByAction(action) {
    if (HandlerMap.has(action.type)) {
      yield* HandlerMap.get(action.type)
    }
}
```

CATCHING ALL ERRORS

- 3 error sources:
  - createSagaMiddleware(onError)
  - React lifecycle error
  - window.onError
- In the framework, we transform all of above:

1, Error object to our own exception, e.g:
API /JS Runtime / Network Connection / Lifecycle
2, {type: "@ERROR", payload: exception}

UNDER THE HOOD

- A global saga for watching all errors

```
function* globalSaga() {
    yield takeEvery("*", handleSagaByAction)
    yield takeLead("@ERROR", handleSagaByError)
}

function* handleSagaByError(action) {
    yield* globalErrorHandler(action.payload)
}
```

QUESTION

- Why **takeLead**?

DECORATORS

- Java annotation only **modifies the meta data**.
  Then framework (core-ng/SpringMVC) can add behaviour after reading the annotation.
- JavaScript decorator **modifies the behaviour** directly.

@LOADING

```
/**
 * To mark state.loading[identifier] during Saga execution
 */
export function Loading(identifier: string = "global"): HandlerDecorator {
    return createActionHandlerDecorator(function*(handler) {
        try {
            yield put(loadingAction(true, identifier));
            yield* handler();
        } finally {
            yield put(loadingAction(false, identifier));
        }
    });
}
```

VISION

- Keep shipping, next version **core-fe 2.0** coming soon
- New features:
  - Using async/await to replace yield
  - A side-effect free module
  - More type safe for module

THANK YOU