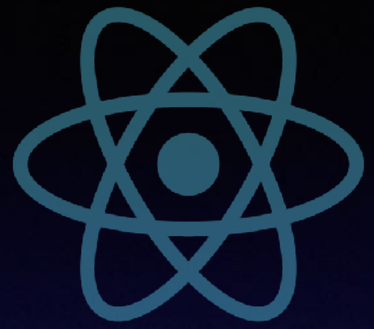


React and Redux

a comprehensive introduction

https://github.com/WisdomFusion

contents

- What’s React?
- Project Best Practice
- JSX
- ES6 need to know
- JavaScript Hight-Order Functions
- Functional and Class Component
- Presentational and Container Components
- React Lifecycle Methods
- state and props
- Redux
- Redux - the three principles of redux
- Why Redux?
- Redux Data Flow
- axios
- `import { Form } from 'antd';`
- misc

What’s React?

- A JavaScript library for building user interfaces
- the V in MVC
- A way to generate html from JavaScript
- A JavaScript view library (similar to a template engine) that enables fast DOM updates via a 'virtual DOM'
- React renders to a virtual DOM first, then it compares that to the real DOM, and only makes changes to the real DOM as necessary

Project Best Practice

```
$ npm i -g create-react-app yarn

$ create-react-app myapp

$ cd myapp

$ yarn add antd axios react-router-dom history prop-types redux

$ yarn add --dev react-app-rewired babel-plugin-import
```

Project Best Practice

- .env and `process.env.REACT_APP_<env variable name>`
- `yarn start`, use .env.development.local, serve on http://localhost:3000
- `yarn build`, use .env.production.local
- config BASENAME, PUBLIC_URL, history basename to build to specific folder in stead of /build
- Project Structure
  /assets, /components, /services, /store, /shared, etc.
- My React starter kit: https://github.com/WisdomFusion/react-starter-kit

JSX

- JavaScript syntax extension, produces React "elements"
- `return (<Fragment><a>{this.getUser().name}</a><br /></Fragment>);`
- Empty tags should close with `/>`, like XML
- JSX prevents injection attacks
- After compilation, JSX expressions become regular JavaScript objects

JSX

```
const IconText = ({ type, text }) => (
    <Fragment><Icon type={type} style={{ marginRight: 8 }} />{text}</Fragment>
);

const renderData = data => {
    return data.length
        ? (
            <ul className="events">
                {data.map(event => (
                    <li key={event.key}>
                        <Badge
                            status={moment(`${event.date} ${event.started_at}`) < moment() ?
'default' : 'success'}
                            text={`${event.date} ${event.started_at}-${event.ended_at}`}
                        />
                    </li>
                ))}
            </ul>
        )
        : null;
};
```

JSX Prevents Injection Attacks

- By default, React escapes any values embedded in JSX before rendering them
- Everything is converted to a string before being rendered. This helps prevent XSS (cross-site-scripting) attacks
- You can never inject anything that's not explicitly written in your application

ES6 need to know

- var, let or const? NO var.
- Export and Import statements
- Destructuring assignment
- Spread operator
- Object Property shorthand
- Fat arrow functions
- Template literals

ES6 - Export and Import

```
// class example
class Utils {
  // TODO
}
export default new Utils();

// import examples
import React, { Fragment } from 'react';
import { connect } from 'react-redux';
import { Link } from 'react-router-dom';
import { Icon, Calendar, Card, Timeline, Popconfirm, Badge, Modal } from 'antd';
import * as scheduleActions from '../../store/actions/schedule.actions';
import classroomSrv from '../../services/classroom.service';
import moment from 'moment/moment';
import util from '../../shared/utils';
import './Schedule.css';
```

ES6 - Destructuring assignment

```
// antd
import { Layout, Menu, Icon, Avatar, Dropdown } from 'antd';
const { Header } = Layout;

// props injections
<RenameReplayForm
    wrappedComponentRef={this.saveFormRef}
    replayId={replayId}
    fields={fields}
    visible={this.state.modalRenameVisible}
    onCancel={this.cancelRename}
    onSubmit={this.submitRename}
    onChange={this.onFormChange}
/>

// destructuring from props
const { replayId, visible, onCancel, onSubmit, form } = this.props;
const { getFieldDecorator, getFieldsError, getFieldError, isFieldTouched } = form;
```

ES6 - Spread operator

```
case USER_ADD:
    return {
        ...state, // object property
        users: [
            ...state.users, // array element
            action.user
        ]
    };
case USER_UPDATE:
    return {
        ...state,
        users: state.users.map(
            user => (user.user_id === action.id) ? { ...action.user } : user
        )
    };
```

ES6 - Object Property Shorthand

```
export const deleteUser = id => {
    return dispatch => {
        userService.destroy(id)
            .then(response => {
                if (response.status === 200) {
                    dispatch({
                        type: USER_DELETE,
                        id   // handy shorthand :)
                    });
                }
                util.success(response.message);
            })
...
```

ES6 - Fat arrow functions

```
onSubmit = event => {
    event.preventDefault();
    const form = this.formRef.props.form;

    form.validateFields((err, values) => {
        if (!err) {
            this.props.renameReplay(this.state.id, form.getFieldsValue());

            this.setState({
                modalVisible: false
            });
        }
    });
};

// DO NOT NEET BIND LIKE THIS
// this.onSubmit = this.onSubmit.bind(this);
```

https://reactjs.org/docs/handling-events.html

JavaScript Hight-Order Functions

- Array.prototype.map()
- Array.prototype.forEach()
- Array.prototype.filter()
- Array.prototype.every()
- Array.prototype.some()

JavaScript Hight-Order Functions

```
// map
{(!!schoolList && schoolList.length) && schoolList.map(school => (
    <Select.Option key={school.id} value={school.id}>{school.title}</Select.Option>
))}

// forEach
users.forEach(user => user.checked = ($event ? true : false));

// filter
users: state.users.filter(user => user.user_id !== action.id)

// every
const allChecked = users.every(user => user.checked === true)

// some
const hasSocialite = socialites.some(socialite => socialite.driver === socialiteDriver);
```

Functional and Class Components

- Local state is a feature available only to classes.
  https://reactjs.org/docs/state-and-lifecycle.html
- Converting a Function to a Class
  - Create an ES6 class, with the same name, that extends React.Component.
  - Add a single empty method to it called render().
  - Move the body of the function into the render() method.
  - Replace props with this.props in the render() body.
  - Delete the remaining empty function declaration.

Presentational and Container Components

	Presentational	Container
Purpose	How things look (markup, styles)	How things work (data fetching, state updates)
Aware of Redux	No	Yes
To read data	Read data from props	Subscribe to Redux state
To change data	Invoke callbacks from props	Dispatch Redux actions

read more

React Lifecycle Methods

- Apply only to class components, not to functional components
- mounting phase: `constructor()`, `componentWillMount()`, `render()`, `componentDidMount()`
- updating phase: `componentReceiveProps()`, `shouldComponentUpdate()`, `componentWillUpdate()`, `render()`, `componentDidUpdate()`
- unmounting phase: `componentWillUnmount()`

state and props

- Never update state directly, always use `this.setState()` method.
- Only time we use `state=` is in the constructor when initializing state.
- React `setState()` is asynchronous.
- Props are Read-Only.

Redux

- Redux is a predictable state container for JavaScript apps.
- Redux is an implementation of a standard Flux developed by facebook, describing the best way to handle state in a React application.

Redux - the three principles of redux

- Single source of truth
  The state of your whole application is stored in an object tree within a single store.

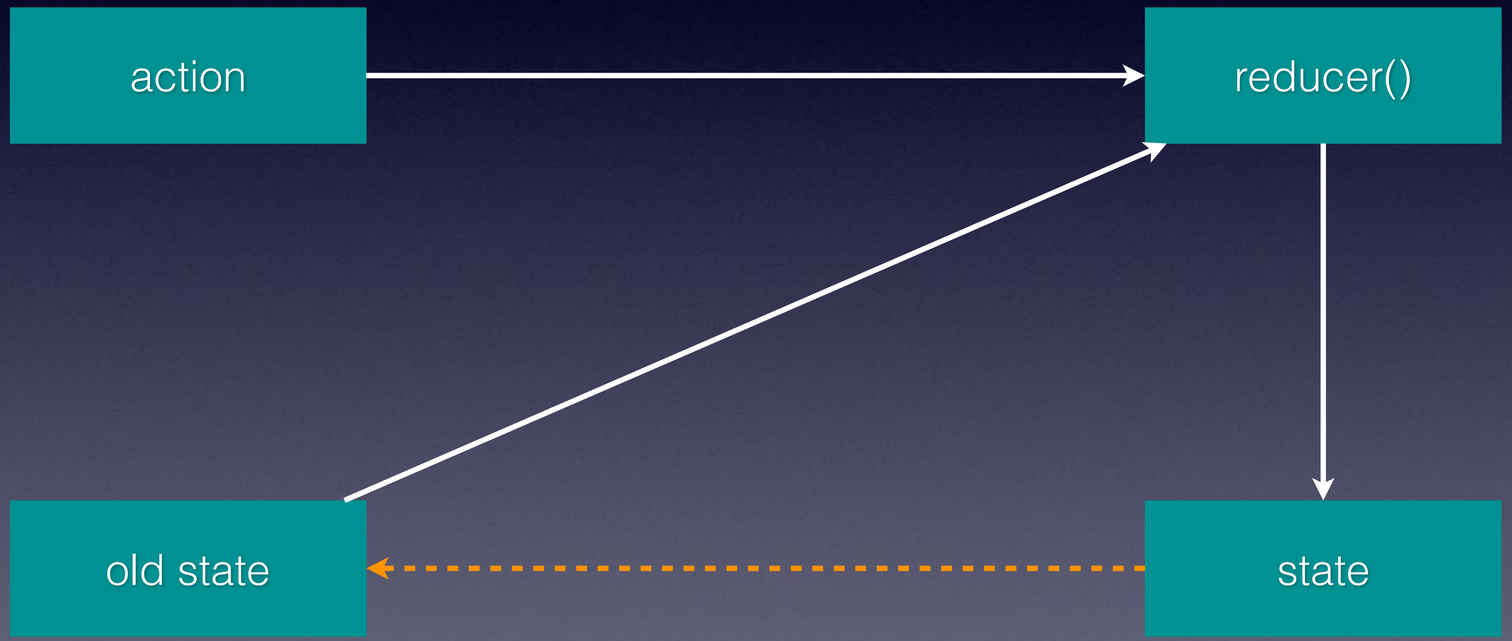
- State is read-only
  The only way to change the state is to emit an action, an object describing what happened.

- Changes are made with pure functions
  To specify how the state tree is transformed by actions, you write pure reducers.

Why Redux?

- Good structure in our project
- Separation of concerns
- Get access to the wonderful world of middleware
- No more sending parameters/attributes from parent components to child components with:
  - fragments of state
  - call-back functions to update state

Redux Data Flow



Redux Data Flow

- A user's click or input operations trigger events, which calls action creators,
- action creators return some actions,
- the action dispatch to reducers,
- the reducer updates states in store.

Redux Data Flow

```
export const REPLAYS_LOAD_REQUEST = 'REPLAYS_LOAD_REQUEST';
export const REPLAYS_LOAD_SUCCESS = 'REPLAYS_LOAD_SUCCESS';
export const REPLAYS_LOAD_FAILURE = 'REPLAYS_LOAD_FAILURE';
export const REPLAYS_FILTER = 'REPLAYS_FILTER';
export const REPLAY_RENAME = 'REPLAY_RENAME';

export const renameReplay = (id, data) => {
    return dispatch => {
        replayService.rename(id, data)
            .then(response => {
                if (response.status === 200) {
                    const replay = response.data;
                    const updatedReplay = {
                        id:    replay.id,
                        title: replay.title
                    };

                    dispatch({
                        type: REPLAY_RENAME,
                        id,
                        replay: updatedReplay
                    });
                    util.success(response.message);
...
```

Actions

Action Creators

Redux Data Flow

```
const replayReducer = (state = initialState, action) => {
    switch (action.type) {
        case REPLAYS_LOAD_SUCCESS:
            return {
                ...state,
                replays: action.replaysData.replays,
                total:   action.replaysData.total,
                loading: false,
                error:   null
            };
        case REPLAY_RENAME:
            return {
                ...state,
                replays: state.replays.map(
                    replay => (replay.id === action.id)
                        ? {
                            ...replay,
                            title: action.replay.title
                        }
                        : replay
                )
            };
...
```

Reducers

Redux Data Flow

```
// retrieve state from redux store to current component
const mapStateToProps = state => {
    return {
        replays: state.replay.replays,
        total:   state.replay.total,
        filter:  state.replay.filter
    };
};

// retrieve action dispatches from redux store to current component
const mapDispatchToProps = (dispatch) => {
    return {
        loadReplays:      filter     => dispatch(replayActions.loadReplays(filter)),
        setReplaysFilter: filter     => dispatch(replayActions.filterReplays(filter)),
        renameReplay:     (id, data) => dispatch(replayActions.renameReplay(id, data))
    };
};

// connect them together
export default connect(mapStateToProps, mapDispatchToProps)(ReplayList);
```

demonstration

axios

```
// create axios, and config baseURL
let httpClient = axios.create({
    baseURL: config.api.url_prefix
});

// add jwt token to request header
httpClient.interceptors.request.use(
    config => {
        const user = sessionService.getUser();
        const token = user ? user.token : null;
        if (token) {
            config.headers['Authorization'] = 'Bearer ' + token;
        } else {
            sessionService.logout();
        }
        return config;
    },
    error => Promise.reject(error)
);
```

import { Form } from 'antd';

```
Form.create({
    onFieldsChange(props, changedFields) {
        props.onChange(changedFields);
    },
    mapPropsToFields(props) {
        let data = {};
        Object.keys(props.fields).forEach(prop => {
            data[prop] = Form.createFormField({
                ...props.fields[prop],
                value: props.fields[prop].value
            });
        });
       return data;
    },
    // onValuesChange(_, values) {
    //     console.log(values);
    // }
})(
    class extends React.Component {
        render() {
...
```

import { Form } from 'antd';

```
// antd Form provides a lot of methods to made react form a little easy
const { getFieldDecorator, getFieldsError, getFieldError, isFieldTouched } =
this.props.form;
const userNameError = isFieldTouched('name') && getFieldError('name');
const passwordError = isFieldTouched('password') && getFieldError('password');

// Form Item example with field validation
<Form.Item validateStatus={userNameError ? 'error' : ''} help={userNameError || ''}>
    {getFieldDecorator('name', {
        rules: [{ required: true, message: 'admin name required'}]
    })(
        <Input size="large"
               prefix={<Icon type="user" style={{ color: 'rgba(0,0,0,.25)' }} />}
               placeholder="please input admin name"
        />
    )}
</Form.Item>
```

Misc

Warning: Each record in dataSource of table should have a unique `key` prop, or set `rowKey` to an unique primary key, see https://u.ant.design/table-row-key

```
{error.map((err, i) => <p key={i}>{err}</p>)}
```

NEVER USE `onClick={this.toggle()}`
USE `onClick={this.toggle}`
OR with parameters like this `onClick={() => this.toggle(params)}`

Install React DevTools and Redux DevTools

and more...

Study Resource

- http://es6-features.org/
- http://reactjs.org/
- http://redux.js.org/
- http://www.ruanyifeng.com/blog/2016/09/redux_tutorial_part_three_react-redux.html
- https://reacttraining.com/react-router/web/
- https://ant.design/docs/react/introduce