**프로리액트**

**React.js를 이용한 모던 프런트엔드 구축**

**지은이: 카시우 지 소자 안토니우**

**옮긴이: 최민석**

**출판사: 위키북스**

**소스코드: https://github.com/pro-react/**

hybrid를 위해서는 build한 후 빌드된 내용을 www폴더에 넣는다

1. **시작하기**
   1. **시작하기 전에**
      1. **Node.js와 npm**
      2. **자바스크립트 ES6**

화살표 함수, 클래스, 스프레드 연산자

map,reduce,assign

* 1. **리액트의 정의**

ui 컴포넌트(확장성,재사용,유지보수 용이)

jsx 사용(변환단계 트랜스파일러가 필요)

* 1. **리액트의 장점**
     1. **편리한 반응형 렌더링**

가상DOM IN MEMORY 사용 - STATE값에 따라 바뀐 부분만

DOM에 반영

* + 1. **순수 자바스크립트를 이용한 컴포넌트 기반 개발**

자바스크립트,CSS,HTML이 하나의 컴포넌트를 생성하며

긴밀한 관계를 유지한다.

* + 1. **문서 모델의 유연한 추상화**
  1. **첫번째 리액트 앱 작성**

class Hello extends React.Component {

render(){

return (

<h1>Hello World</h1>

)

}

}

* + 1. **리액트 개발 워크플로**

<https://github.com/pro-react/react-app-boilerplate>

npm install npm start

source/App.js

index.html

<!DOCTYPE html>

<html>

<head>

<title>First React Component</title>

</head>

<body>

<div id=”root”></div>

<script type=”text/javascript” src=”bundle.js”></script>

</body>

</html>

npm init

package.json

{

“name”: “your-app-name”,

“version”: ”x.x.x”,

“description”: “Your app description”,

“author”: “You”,

“scripts”: {

“start”: “node\_modules/.bin/webpack-dev-server --progress”

},

“devDependencies”: {

“babel-core”: “^5.8.\*”,

“babel-loader”: “^5.3.\*”,

“webpack”: “^1.12.\*”,

“webpack-dev-server”: “^1.10.\*”

},

“dependencies”: {

“react”: “^0.13.\*”

}

}

webpack.config.js

module.exports = {

entry: [

‘./source/App.js’

],

output: {

path: \_\_dirname,

filename: “bundle.js”

},

module: {

loaders: [{

test: /\.jsx?$/,

loader: ‘babel’

}]

}

};

npm start

localhost:8080

아래는 bundle.js 생성시 아래 사용

node ./node\_modules\webpack\bin\webpack

빌드 생성

* + 1. **첫번째 컴포넌트 만들기**

import React from ‘react’;

class Hello extends React.Component {

render(){

return (

<h1>Hello World</h1>

);

}

}

React.render(<Hello />, document.getElementById(‘root’));

* + 1. **타이핑 수고를 약간 줄이는 방법**

구조분해 할당(destructuring assignment)

import React, { Component } from ‘react’;

class Hello extends Component {

render(){

return (

<h1>Hello World</h1>

);

}

}

React.render(<Hello />, document.getElementById(‘root’));

* + 1. **동적 값**

import React, { Component } from ‘react’;

class Hello extends Component {

render(){

var place = “World”;

return (

<h1>Hello {place}</h1>

);

}

}

React.render(<Hello />, document.getElementById(‘root’));

* 1. **컴포넌트 조합하기**
     1. **속성**

property는 부모에게서 전달되며 자식 컴포넌트안에서 변경 불가

import React, { Component } from ‘react;

//부모 컴포넌트

class GroceryList extends Component {

render(){

return (

<ul>

<ListItem quantity=”1” name=”Bread” />

<ListItem quantity=”6” name=”Eggs” />

<ListItem quantity=”2” name=”Milk” />

</ul>

);

}

}

//자식 컴포넌트

class ListItem extends Component {

render(){

return (

<li>

{this.props.quantity} x {this.props.name}

</li>

);

}

}

React.render(<GroceryList />, document.getElementById(‘root’));

import React, { Component } from ‘react;

//부모 컴포넌트

class GroceryList extends Component {

render(){

return (

<ul>

<ListItem quantity=”1”>Bread</ListItem>

<ListItem quantity=”6”>Eggs</ListItem>

<ListItem quantity=”2”>Milk</ListItem>

</ul>

);

}

}

//자식 컴포넌트

class ListItem extends Component {

render(){

return (

<li>

{this.props.quantity} x {this.props.children}

</li>

);

}

}

React.render(<GroceryList />, document.getElementById(‘root’));

* + 1. **칸반 보드 앱 소개**

칸반: 칸반스타일의 프로젝트 관리 툴

칸반 앱 데이터 모델

[

{

id: 1,

title: “Card one title”,

description: “Card detailed description”,

status: “todo”,

tasks: [

{id: 1, name: “Task one”, done: true},

{id: 2, name: “Task two”, done: false},

{id: 3, name: “Task three”, done: false},

]

},

{

id: 2,

title: “Card Two title”,

description: “Card detailed description”,

status: “in-progress”,

tasks: []

},

{

id: 3,

title: “Card Three title”,

description: “Card detailed description”,

status: “done”,

tasks: []

},

];

* + 1. **컴포넌트 계층 정의**

**App**

**KanbanBoard Component**

**To Do**

* **Write some code**

**code along with …**

□ContactList Example

□Kanban Example

□Test

**Done**

* **Buy a React Book**

**Pro: An app should be built…**

**In Progress**

* **Read the Book**

**I should Read… along with …**

**List Component**

**Card Component**

**CheckList Component**

* + 1. **속성의 중요성**
    2. **컴포넌트 만들기**

하향식으로 컴포넌트 만들기

각 컴포넌트를 별도의 자바스크립트 파일로 만듬

* + 1. **앱 모듈(App.js)**

app.js

import React from ‘react’;

import KanbanBoard from ‘./KanbanBoard’;

let cardsList = [

{

id: 1,

title: “Read the Book”,

description: “I should read the whole book”,

status: “in-progress”,

tasks: []

},

{

id: 2,

title: “Write some code”,

description: “Code along with the samples in the book”,

status: “todo”,

tasks: [

{

id: 1,

name: “ContactList Example”,

done: true

},

{

id: 2,

name: “Kanban Example”,

done: false

},

{

id: 3,

name: “My own experiments”,

done: false

},

]

},

];

React.render(<KanbanBoard cards={cardsList} />, document.getElementById(‘root’));

* + - 1. **KanbanBoard 컴포넌트(KanbanBoard.js)**

데이터를 속성을 통해 받고 상태를 필터링해 List 컴포넌트 세 개를 만든다

아래는 필터와 맵함수를 사용한다

import React, { Component } from ‘react’;

import List from ‘./List’;

class KanbanBoard extends Component {

render(){

return (

<div className=”app”>

<List id=”todo” title=”To Do” cards={

this.props.cards.filter(

(card)=> card.status == “todo”)

} />

<List id=”in-progress” title=”In Progress” cards={

this.props.cards.filter(

(card)=> card.status == “in-progress”)

} />

<List id=”done” title=”Done” cards={

this.props.cards.filter(

(card)=> card.status == “done”)

} />

</div>

);

}

}

export default KanbanBoard;

* + - 1. **List 컴포넌트(List.js)**

import React, { Component } from ‘react’;

import Card from ‘./Card’;

class List extends Component {

render(){

var cards = this.props.cards.map((card)=>{

return <Card id={card.id}

title={card.title}

description={card.description}

tasks={card.tasks} />

});

return (

<div className=”list”>

<h1>{this.props.title}</h1>

{cards}

</div>

);

}

}

export default List;

* + - 1. **Card 컴포넌트(Card.js)**

import React, { Component } from ‘react’;

import CheckList from ‘./CheckList’;

class Card extends Component {

render(){

return (

<div className=”card”>

<div className=”card\_\_title”>{this.props.title}</div>

<div className=”card\_\_details”>

{this.props.description}

<CheckList cardId={this.props.id} tasks={this.props.tasks} />

<div>

</div>

);

}

}

export default Card;

* + - 1. **Checklist 컴포넌트(CheckList.js)**

import React, { Component } from ‘react’;

class CheckList extends Component {

render(){

let tasks = this.props.tasks.map((task)=>{

<li className=”checklist\_\_task”>

<input type=”checkbox” defaultChecked={task.done} />{task.name}

<a href=”#” className=”checklist\_\_task--remove” />

</li>

});

return (

<div className=”checklist”>

<ul>{tasks}</ul>

</div>

);

}

}

export default CheckList;

* + - 1. **마무리 작업**

\*{

box-sizing: border-box;

}

html,body,#root {

height:100%;

margin: 0;

padding: 0;

}

body {

background: #eee;

font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;

}

h1{

font-weight: 200;

color: #3b414c;

font-size: 20px;

}

ul {

list-style-type: none;

padding: 0;

margin: 0;

}

.app {

white-space: nowrap;

height:100%;

}

.list {

position: relative;

display: inline-block;

vertical-align: top;

white-space: normal;

height: 100%;

width: 33%;

padding: 0 20px;

overflow: auto;

}

.list:not(:last-child):after{

content: "";

position: absolute;

top: 0;

right: 0;

width: 1px;

height: 99%;

background: linear-gradient(to bottom, #eee 0%, #ccc 50%, #eee 100%) fixed;

}

.card {

position: relative;

z-index: 1;

background: #fff;

width: 100%;

padding: 10px 10px 10px 15px;

margin: 0 0 10px 0;

overflow: auto;

border: 1px solid #e5e5df;

border-radius: 3px;

box-shadow: 0 1px 0 rgba(0, 0, 0, 0.25);

}

.card\_\_title {

font-weight: bold;

border-bottom: solid 5px transparent;

}

.card\_\_title:before {

display: inline-block;

width: 1em;

content: '▸';

}

.card\_\_title--is-open:before {

content: '▾';

}

.checklist\_\_task:first-child {

margin-top: 10px;

padding-top: 10px;

border-top: dashed 1px #ddd;

}

.checklist\_\_task--remove:after{

display: inline-block;

color: #d66;

content: "✖";

}

.checklist--add-task {

border: 1px dashed #bbb;

width: 100%;

padding: 10px;

margin-top: 5px;

border-radius: 3px;

}

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Pro-React Kanban</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div id='root'>

</div>

<script src="bundle.js"></script>

</body>

</html>

* 1. **상태 소개**

property는 immutable(변경불가)

this.state

this.setState

* + 1. **칸반 앱: 토글되는 카드**

import React, { Component } from ‘react’;

import CheckList from ‘./CheckList’;

class Card extends Component {

constructor(){

super(…arguments);

this.state = {

showDetails: false

}

}

render(){

let cardDetails;

if(this.state.showDetails){

cardDetails = (<div className=”card\_\_details”>

{this.props.description}

<CheckList cardId={this.props.id} tasks={this.props.tasks} />

<div>

);

};

return (

<div className=”card”>

<div className=”card\_\_title” onClick={()=>this.setState({showDetails: !this.state.showDetails})}>{this.props.title}</div>

{cardDetails}

</div>

);

}

}

export default Card;

1. **DOM 추상화의 내부**
   1. **리액트의 이벤트**

단 하나의 이벤트 리스너가 문서 루트에 연결되며, 이벤트가 발생하면 리액트가 이를 적절한 컴포넌트 요소로 맵핑 - 이벤트의 위임, 컴포넌트가 unmount때 이벤트도 해제

* + 1. **DOM 이벤트 리스너**

리액트의 속성은 낙타 표기법을 따른다.

**터치와 마우스 이벤트**

|  |  |  |  |
| --- | --- | --- | --- |
| onTouchStart | onTouchMove | onTouchEnd | onTouchCancel |
| onClick | **onDoubleClick** | **onMouseDown** | **onMouseUp** |
| onMouseOver | **onMouseMove** | **onMouseEnter** | **onMouseOut** |
| onContextMenu | **onDrag** | **onDragEnter** | **onDragLeave** |
| onDragExit | **onDragStart** | **onDragEnd** | **onDragOver** |
| onDrop |  |  |  |

**키보드 이벤트**

|  |  |  |
| --- | --- | --- |
| onKeyDown | onKeyUp | onKeyPress |

**포커스와 폼 이벤트**

|  |  |  |  |
| --- | --- | --- | --- |
| onFocus | onBlur |  |  |
| onChange | **onInput** | **onSubmit** |  |

**기타 이벤트**

|  |  |  |  |
| --- | --- | --- | --- |
| onScroll | onWheel | onCopy | onCut |
| onPaste |  |  |  |

* + 1. **칸반 앱: DOM 이벤트 관리**

class Card extends Component {

constructor(){

super(…arguments);

this.state = {

showDetails: false

}

}

toggleDetails(){

this.setState({showDetails: !this.state.showDetails});

}

render(){

let cardDetails;

if(this.state.showDetails){

cardDetails = (<div className=”card\_\_details”>

{this.props.description}

<CheckList cardId={this.props.id} tasks={this.props.tasks} />

<div>

);

};

return (

<div className=”card”>

<div className=”card\_\_title” onClick={this.toggleDetails.bind(this)}>{this.props.title}</div>

{cardDetails}

</div>

);

}

}

export default Card;

* 1. **JSX 자세히 살펴보기**

트랜스파일링을 거치면 리액트 라이브러리에 대한 함수 호출로 변환된다.

<h1>Hello World</h1>

* React.createElement(“h1”, null, “Hello World”);
  + 1. **JSX와 HTML 비교**
    2. **JSX와 HTML의 차이**

**태그 특성은 낙타 표기법으로 작성한다**

HTML: <input type=”text” maxlength=”30” />

JSX : <input type=”text” maxLength=”30” />

**모든 요소는 짝이 맞아야 한다**

<br/><img ../>

**특성 이름은 DOM API 기반이다**

document.getElementById(“box”).className에 기인

<div id=”box” className=”some-class”></div>

* + 1. **JSX의 특이점**

**단일 루트 노드 반환**

return (<div>

<h1>1</h1>

<h2>2</h2>

</div>)

🡺React.createElement(“div”, null,

React.createElement(“h1”, null, “1”),

React.createElement(“h2”, null, “2”),

)

**조건 절**

<div className={**if** (condition) {“salutation”}}>Hello JSX</div> 이런 형태는 불가하고

**삼항식을 사용**

<div className={condition ? “salutation” : “”}>

**조건을 밖으로 이동**

let className;

if(condition){

className = “salutation”;

}

<div className={className}>

* 1. **칸반 앱: 카드가 열려있는지 여부 확인**

class Card extends Component {

constructor(){

super(…arguments);

this.state = {

showDetails: false

}

}

toggleDetails(){

this.setState({showDetails: !this.state.showDetails});

}

render(){

let cardDetails;

if(this.state.showDetails){

cardDetails = (<div className=”card\_\_details”>

{this.props.description}

<CheckList cardId={this.props.id} tasks={this.props.tasks} />

<div>

);

};

return (

<div className=”card”>

<div className={this.state.showDetails? ”card\_\_title card\_\_title--is-open” : “card\_\_title”} onClick={this.toggleDetails.bind(this)}>{this.props.title}</div>

{cardDetails}

</div>

);

}

}

export default Card;

* + 1. **공백**

return (

<a href=”<http://google.com>”>Google</a>{“ “}

<a href=<http://facebook.com>>Facebook</a>

)

* + 1. **JSX의 주석**

<nav>

{/\* 자식 주석이므로 {}로 감싼다. \*/}

<person

/\* 다중

행

주석 \*/

name={window.isLoggedIn ? window.name : ‘’} // 행 끝 주석

/>

</nav>

* + 1. **동적 HTML렌더링**
    2. **칸반 앱: 마크다운 렌더링**

app.js

import React from ‘react’;

import KanbanBoard from ‘./KanbanBoard’;

let cardsList = [

{

id: 1,

title: “Read the Book”,

description: “I should read the **\*\*whole\*\*** book”,//마크다운포맷

status: “in-progress”,

tasks: []

},

{

id: 2,

title: “Write some code”,

description: “Code along with the samples in the book. The complete source can be found at **[github](**[**https://github.com/pro-react)”,//**](https://github.com/pro-react))마크다운포맷

status: “todo”,

tasks: [

{

id: 1,

name: “ContactList Example”,

done: true

},

{

id: 2,

name: “Kanban Example”,

done: false

},

{

id: 3,

name: “My own experiments”,

done: false

},

]

},

];

React.render(<KanbanBoard cards={cardsList} />, document.getElementById(‘root’));

마크다운포맷을 HTML로 변환하려면 마크드(<https://github.com/chjj/marked>) 라이브러리 이용

npm install --save marked

import React, { Component } from ‘react’;

import CheckList from ‘./CheckList’;

**import marked from ‘marked’;**

class Card extends Component {

constructor(){

super(…arguments);

this.state = {

showDetails: false

}

}

toggleDetails(){

this.setState({showDetails: !this.state.showDetails});

}

render(){

let cardDetails;

if(this.state.showDetails){

cardDetails = (<div className=”card\_\_details”>

<span **dangerouslySetInnerHTML={{\_\_html: marked(this.props.description)}}**

<CheckList cardId={this.props.id} tasks={this.props.tasks} />

<div>

);

};

return (

<div className=”card”>

<div className={this.state.showDetails? ”card\_\_title card\_\_title--is-open” : “card\_\_title”} onClick={this.toggleDetails.bind(this)}>{this.props.title}</div>

{cardDetails}

</div>

);

}

}

export default Card;

* 1. **JSX를 배제하고 리액트 이용**
     1. **일반 자바스크립트로 리액트 요소 만들기**

let child1 = React.createElement(‘li’, null, ‘First Text Content’);

let child2 = React.createElement(‘li’, null, ‘Second Text Content’);

let root = React.createElement(‘ul’, { className: ‘my-list’},

child1, child2);

React.render(root, document.getElementById(‘example’));

* + 1. **요소 팩토리**

React.DOM.form({className:”commentForm”},

React.DOM.input({type:”text”, placeholder:”Name”}),

React.DOM.input({type:”text”, placeholder:”Comment”}),

React.DOM.input({type:”submit”, value:”Post”})

);

import React, { Component } from ‘react’;

import {render} from ‘react-dom’;

**let {**

**form,**

**input**

**} = React.DOM //구조분해할당**

class CommentForm extends Component{

render(){

return form({className:”commentForm”},

input({type:”text”, placeholder:”Name”}),

input({type:”text”, placeholder:”Comment”}),

input({type:”submit”, value:”Post”})

)

}

}

* + 1. **커스텀 팩토리**

let Factory = React.createFactory(ComponentClass);

…

let root = Factory({ custom: ‘prop’ });

render(root, document.getElementById(‘example’));

* 1. **인라인 스타일링**
     1. **인라인 스타일 정의**

낙타표기법사용

import React, { Component } from ‘react’;

import {render} from ‘react-dom’;

class Hello extends Component {

let divStyle = {

width: 100,

height: 30,

padding: 5,

backgroundColor: ‘#ee9900’

};

return <div style={divStyle}>Hello World</div>

}

* + 1. **칸반 앱: 인라인 스타일링을 이용한 카드 색상 지정**

**app.js**

import React from ‘react’;

import KanbanBoard from ‘./KanbanBoard’;

let cardsList = [

{

id: 1,

title: “Read the Book”,

description: “I should read the **\*\*whole\*\*** book”,//마크다운포맷

**color: ‘#BD8D31’,**

status: “in-progress”,

tasks: []

},

{

id: 2,

title: “Write some code”,

description: “Code along with the samples in the book. The complete source can be found at **[github](**[**https://github.com/pro-react)”,//**](https://github.com/pro-react))마크다운포맷

**color: ‘#3A7E28’,**

status: “todo”,

tasks: [

{

id: 1,

name: “ContactList Example”,

done: true

},

{

id: 2,

name: “Kanban Example”,

done: false

},

{

id: 3,

name: “My own experiments”,

done: false

},

]

},

];

React.render(<KanbanBoard cards={cardsList} />, document.getElementById(‘root’));

**List.js**

import React, { Component } from ‘react’;

import Card from ‘./Card’;

class List extends Component {

render(){

var cards = this.props.cards.map((card)=>{

return <Card id={card.id}

title={card.title}

description={card.description}

**color={card.color}**

tasks={card.tasks} />

});

return (

<div className=”list”>

<h1>{this.props.title}</h1>

{cards}

</div>

);

}

}

export default List;

**Card.js**

class Card extends Component {

constructor(){

super(…arguments);

this.state = {

showDetails: false

}

}

toggleDetails(){

this.setState({showDetails: !this.state.showDetails});

}

render(){

let cardDetails;

if(this.state.showDetails){

cardDetails = (<div className=”card\_\_details”>

<span **dangerouslySetInnerHTML={{\_\_html: marked(this.props.description)}}**

<CheckList cardId={this.props.id} tasks={this.props.tasks} />

<div>

);

};

**let sideColor = {**

**position: ‘absolute’,**

**zIndex: -1,**

**top: 0,**

**bottom: 0,**

**left: 0,**

**width: 7,**

**backgroundColor: this.props.color**

**};**

return (

<div className=”card”>

**<div style={sideColor} />**

<div className={this.state.showDetails? ”card\_\_title card\_\_title--is-open” : “card\_\_title”} onClick={this.toggleDetails.bind(this)}>{this.props.title}</div>

{cardDetails}

</div>

);

}

}

export default Card;

* 1. **폼 처리**
     1. **제어 컴포넌트**

값이나 확인되는 속성을 가지는 폼 컴포넌트를 제어 컴포넌트

class Search extends Component {

constructor(){

super();

this.state = {

searchTerm: “React”

}

}

handleChange(event){

this.setState({searchTerm: event.target.value});

}

render(){

return (

<div>

Search Term:

<input type=”search” value={this.state.searchTerm}

onChange={this.handleChange.bind(this)} />

</div>

)

}

}

* + 1. **특수 사례**

**TextArea**

<textarea value=”this is desc\nthis is new line”/>

**Select**

<select value=”B”>

<option value=”A”>Mobile</option>

<option value=”B”>Work</option>

<option value=”C”>Home</option>

</select>

* + 1. **비제어 컴포넌트**

handleSubmit(event){

//event.target.name.value

//event.target.email.value

}

render(){

return (

<form onSubmit={this.handleSubmit}>

<div className=”formGroup”>

Name: <input name=”name” type=”text” />

</div>

<div className=”formGroup”>

Email: <input name=”email” type=”email” />

</div>

<button type=”submit”>Submit</button>

</form>

)

}

* + 1. **칸반 앱: 태스크 폼 만들기**

**CheckList.js**

import React, { Component } from ‘react’;

class CheckList extends Component {

render(){

let tasks = this.props.tasks.map((task)=>{

<li className=”checklist\_\_task”>

<input type=”checkbox” defaultChecked={task.done} />{task.name}

<a href=”#” className=”checklist\_\_task--remove” />

</li>

});

return (

<div className=”checklist”>

<ul>{tasks}</ul>

**<input type=”text”**

**className=”checklist--add-task”**

**placeholder=”Type then hit Enter to add a task” />//비제어 컴포넌트 추가**

</div>

);

}

}

export default CheckList;

.checklist--add-task {

border: 1px dashed #bbb;

width: 100%;

padding: 10px;

margin-top: 5px;

border-radius: 3px;

}

* 1. **가상 DOM의 작동 방식**

실제 DOM 이 있고 가상 DOM 이 있는 상태에서 상태값이 변하면 변하는 값만 실제 DOM에 반영한다.

* DOM 트리에서 노드 자체가 다르면(DIV 🡺SPAN) 리액트는 DIV를 제거하고 SPAN을 생성한 후 삽입한다.
* 컴포넌트유형이 다른 경우 첫번째 경우를 따른다
* 노드가 같은 경우

-<div id=”before” /> 🡺 <div id=”after” /> 이때는 attribute만 반영

**키**

반복리스트에 키를 준다

<li>Orange</li><li>Banana</li>

* <li>Apple</li><li>Orange</li>
* li에 키를 줘서 순서를 바꾸고 값 사용

**칸반 앱: 키**

**List.js**

import React, { Component } from ‘react’;

import Card from ‘./Card’;

class List extends Component {

render(){

var cards = this.props.cards.map((card)=>{

return <Card **key={card.id}**

id={card.id}

title={card.title}

description={card.description}

color={card.color}

tasks={card.tasks} />

});

return (

<div className=”list”>

<h1>{this.props.title}</h1>

{cards}

</div>

);

}

}

export default List;

**CheckList.js**

import React, { Component } from ‘react’;

class CheckList extends Component {

render(){

let tasks = this.props.tasks.map((task)=>{

<li **key={task.id}** className=”checklist\_\_task”>

<input type=”checkbox” defaultChecked={task.done} />{task.name}

<a href=”#” className=”checklist\_\_task--remove” />

</li>

});

return (

<div className=”checklist”>

<ul>{tasks}</ul>

<input type=”text”

className=”checklist--add-task”

placeholder=”Type then hit Enter to add a task” />//비제어 컴포넌트 추가

</div>

);

}

}

export default CheckList;

**ref**

렌더링되는 실제 DOM 객체의 reference

class FocusText extends Component {

handleClick(){

this.refs.myTextInput.focus();

}

render(){

return (

<div>

<input type=”text” ref=”myTextInput” />

<input

type=”button”

value=”Focus the text input”

onClick={this.handleClick.bind(this)}

</div>

);

}

}

1. **컴포넌트를 이용한 애플리케이션 구축**
   1. **속성 유효성 검사**

import React, { Component, PropTypes } from ‘react’;

import { render } from ‘react-dom’;

class Greeter extends Component {

render(){

return (

<h1>{this.props.salutation}</h1>

)

}

}

Greeter.propTypes = {

salutation: PropTypes.string.isRequired

}

render(<Greeter salutation=”Hello World” />, document.getElementById(‘root’));

* + 1. **속성 기본값**

import React, { Component, PropTypes } from ‘react’;

import { render } from ‘react-dom’;

class Greeter extends Component {

render(){

return (

<h1>{this.props.salutation}</h1>

)

}

}

**Greeter.propTypes = {**

**salutation: PropTypes.string**

**}**

**Greeter.defaultProps = {**

**salutation: “Hello World”**

**}**

render(<Greeter />, document.getElementById(‘root’));

* + 1. **기본 제공되는 propTypes 유효성 검사기**

**자바 스크립트 기본형 PropTypes**

|  |  |
| --- | --- |
| 유효성 검사기 | 설명 |
| PropTypes.array | **배열** |
| PropTypes.bool | **true or false** |
| PropTypes.func |  |
| PropTypes.number |  |
| PropTypes.object |  |
| PropTypes.string |  |

**조합 기본형 PropTypes**

|  |  |
| --- | --- |
| 유효성 검사기 | 설명 |
| PropTypes.oneOfType | **PropTypes.oneOfType([**  **PropTypes.string,**  **PropTypes.number,**  **PropTypes.instanceOf(Message)**  **])** |
| PropTypes.arrayOf | **PropTypes.arrayOf(PropTypes.number)** |
| PropTypes.objectOf | **속성이 특정 형식의 속성 값을 가진 객채여야 한다. PropTypes.objectOf(PropTypes.number)** |
| PropTypes.shape | **속성이 특정 형태를 가진 객채여야 한다. 동일한 속성의 집합이**  **PropTypes.shape({**  **color: PropTypes.string,**  **fontSize: PropTypes.number**  **})** |

**특수 PropTypes**

|  |  |
| --- | --- |
| 유효성 검사기 | 설명 |
| PropTypes.node | **속성이 렌더링가능 (숫자, 문자열, 요소, 배열)** |
| PropTypes.element | **속성이 리액트 요소** |
| PropTypes.instanceOf |  |
| PropTypes.node.oneOf | **PropTypes.node.oneOf([‘News’, ‘Photos’])** |

* + 1. **칸반 앱: 속성 형식 정의**

import React, { Component, **PropTypes** } from ‘react’;

import List from ‘./List’;

class KanbanBoard extends Component {

render(){

return (

<div className=”app”>

<List id=”todo” title=”To Do” cards={

this.props.cards.filter(

(card)=> card.status == “todo”)

} />

<List id=”in-progress” title=”In Progress” cards={

this.props.cards.filter(

(card)=> card.status == “in-progress”)

} />

<List id=”done” title=”Done” cards={

this.props.cards.filter(

(card)=> card.status == “done”)

} />

</div>

);

}

}

**KanbanBoard.propTypes = {**

**cards: PropTypes.arrayOf(PropTypes.object)**

**};**

export default KanbanBoard;

import React, { Component, **PropTypes** } from ‘react’;

import Card from ‘./Card’;

class List extends Component {

render(){

var cards = this.props.cards.map((card)=>{

return <Card **key={card.id}**

id={card.id}

title={card.title}

description={card.description}

color={card.color}

tasks={card.tasks} />

});

return (

<div className=”list”>

<h1>{this.props.title}</h1>

{cards}

</div>

);

}

}

**List.propTypes = {**

**title: PropTypes.string.isRequired,**

**cards: PropTypes.arrayOf(PropTypes.object)**

**}**

export default List;

class Card extends Component {

constructor(){

super(…arguments);

this.state = {

showDetails: false

}

}

toggleDetails(){

this.setState({showDetails: !this.state.showDetails});

}

render(){

let cardDetails;

if(this.state.showDetails){

cardDetails = (<div className=”card\_\_details”>

<span dangerouslySetInnerHTML={{\_\_html: marked(this.props.description)}}

<CheckList cardId={this.props.id} tasks={this.props.tasks} />

<div>

);

};

let sideColor = {

position: ‘absolute’,

zIndex: -1,

top: 0,

bottom: 0,

left: 0,

width: 7,

backgroundColor: this.props.color

};

return (

<div className=”card”>

<div style={sideColor} />

<div className={this.state.showDetails? ”card\_\_title card\_\_title--is-open” : “card\_\_title”} onClick={this.toggleDetails.bind(this)}>{this.props.title}</div>

{cardDetails}

</div>

);

}

}

**Card.propTypes = {**

**id: PropTypes.number,**

**title: PropTypes.string,**

**description: PropTypes.string,**

**color: PropTypes.string,**

**tasks: PropTypes.arrayOf(PropTypes.object)**

**};**

export default Card;

import React, { Component } from ‘react’;

class CheckList extends Component {

render(){

let tasks = this.props.tasks.map((task)=>{

<li key={task.id} className=”checklist\_\_task”>

<input type=”checkbox” defaultChecked={task.done} />{task.name}

<a href=”#” className=”checklist\_\_task--remove” />

</li>

});

return (

<div className=”checklist”>

<ul>{tasks}</ul>

<input type=”text”

className=”checklist--add-task”

placeholder=”Type then hit Enter to add a task” />//비제어 컴포넌트 추가

</div>

);

}

}

**CheckList.propTypes = {**

**cardId: PropTypes.number,**

**tasks: PropTypes.arrayOf(PropTypes.object)**

**}**

export default CheckList;

* + 1. **커스텀 propTypes 유효성 검사기**
       1. **칸반 앱: 커스텀 propTypes 유효성 검사기 정의**

**let titlePropType = (props, propName, componentName) => {**

**if(props[propName]){**

**let value = props[propName];**

**if(typeof value !== ‘string’ || value.length > 80){**

**return new Error(`${propName} in ${componentName} is longer than 80 characters`);**

**}**

**}**

**}**

class Card extends Component {

constructor(){

super(…arguments);

this.state = {

showDetails: false

}

}

toggleDetails(){

this.setState({showDetails: !this.state.showDetails});

}

render(){

let cardDetails;

if(this.state.showDetails){

cardDetails = (<div className=”card\_\_details”>

<span dangerouslySetInnerHTML={{\_\_html: marked(this.props.description)}}

<CheckList cardId={this.props.id} tasks={this.props.tasks} />

<div>

);

};

let sideColor = {

position: ‘absolute’,

zIndex: -1,

top: 0,

bottom: 0,

left: 0,

width: 7,

backgroundColor: this.props.color

};

return (

<div className=”card”>

<div style={sideColor} />

<div className={this.state.showDetails? ”card\_\_title card\_\_title--is-open” : “card\_\_title”} onClick={this.toggleDetails.bind(this)}>{this.props.title}</div>

{cardDetails}

</div>

);

}

}

Card.propTypes = {

id: PropTypes.number,

title: **titlePropType**,

description: PropTypes.string,

color: PropTypes.string,

tasks: PropTypes.arrayOf(PropTypes.object)

};

export default Card;

* 1. **컴포넌트 조합 전략과 모범 사례**
     1. **상태 저장 컴포넌트와 순수 컴포넌트**
     2. **어떤 컴포넌트가 상태 저장일까?**

ContactsApp.js

import React, { Component, PropTypes } from ‘react’;

import { render } from ‘react-dom’;

class ContactsApp extends Component {

constructor(){

super();

this.state = {

filterText: ‘’

}

}

handleUserInput(searchTerm){

this.setState({filterText:searchTerm});

}

render(){

return (

<div>

<SearchBar filterText={this.state.filterText}

onUserInput={this.handleUserInput.bind(this)} />

<ContactList contacts={this.props.contacts}

filterText={this.state.filterText}/>

</div>

);

}

}

ContactsApp.propTypes = {

contacts: PropTypes.arrayOf(PropTypes.object)

}

class SearchBar extends Component {

**handleChange(event){**

**this.props.onUserInput(event.target.value);**

**}**

render(){

**return** <input type=”search”

placeholder=”search”

value={this.props.filterText}

**onChange={this.handleChange.bind(this)}** />

}

}

SearchBar.propTypes = {

**onUserInput: PropTypes.func.isRequired,**

filterText: PropTypes.string.isRequired

}

class ContactList extends Component {

render(){

let filteredContacts = this.props.contacts.filter(

(contact) => contact.name.indexOf(this.props.filterT ext) !== -1

);

return (

<ul>

{filteredContacts.map(

(contact) => <ContactItem key={contact.email}

name={contact.name}

email={contact.email}

)}

</ul>

);

}

}

ContactList.propTypes = {

contacts: PropTypes.arrayOf(PropTypes.object)

}

class ContactItem extends Component {

render(){

return <li>{this.props.name} - {this.props.email}</li>

}

}

let contacts = [

{ name: “Cassio Yong”, email: “[cassio@gmail.com](mailto:cassio@gmail.com)” },

{ name: “Dan Abramov”, email: “[hetert@somewhere.com](mailto:hetert@somewhere.com)” },

{ name: “Pete hunt”, email: “ddd@ddd.com”},

{ name: “Paul oban”, email: “gggg@ccc.com”},

{ name: “Pyan Florence”, email: “hokk@jjj.com”},

{ name: “Sebastian”, email: “rrr@hhh,com”}

];

render(<ContactsApp contacts={contacts} />, document.getElementById(“root”));

* 1. **데이터 흐름과 컴포넌트 통신**
     1. **수명주기 단계와 메서드**

**마운팅 과정**

클래스 생성자 -> componentWillMount(최초 한번) -> render ->componentDidMount(DOM 생성됨)

**언마운팅**

componentWillUnMount

**속성 변경**

componentWillReceiveProps -> shouldComponentUpdate ->componentWillUpdate->render->componentDidUpdate

**상태 변경**

shouldComponentUpdate ->componentWillUpdate->render->componentDidUpdate

* + 1. **수명주기 함수의 실제 활용: 데이터 가져오기**

window.fetch(크롬과 파이어 팍스만 지원)

npm install --save whatwg-fetch 사용(IE Pollyfill)

public/contacts.json

[

{ name: “Cassio Yong”, email: “[cassio@gmail.com](mailto:cassio@gmail.com)” },

{ name: “Dan Abramov”, email: “[hetert@somewhere.com](mailto:hetert@somewhere.com)” },

{ name: “Pete hunt”, email: “ddd@ddd.com”},

{ name: “Paul oban”, email: “gggg@ccc.com”},

{ name: “Pyan Florence”, email: “hokk@jjj.com”},

{ name: “Sebastian”, email: “rrr@hhh,com”}

];

ContactsAppContainer

**import React, { Component, PropTypes } from ‘react’;**

**import { render } from ‘react-dom’;**

**import ‘whatwg-fetch’;**

**class ContactsAppContainer extends Component {**

**constructor(){**

**super();**

**this.state = {**

**contacts: []**

**};**

**}**

**componentDidMount(){**

**fetch(‘./contacts.json’)**

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

**.then((responseData) => {**

**this.setState({contacts: responseData});**

**})**

**.catch((error) => {**

**console.log(‘Error fetching and parsing data’, error);**

**});**

**}**

**render(){**

**return (**

**<ContactsApp contacts={this.state.contacts} />**

**);**

**}**

**}**

class ContactsApp extends Component {

constructor(){

super();

this.state={

filterText: ''

};

}

handleUserInput(searchTerm){

this.setState({filterText:searchTerm})

}

render(){

return(

<div>

<SearchBar filterText={this.state.filterText}

onUserInput={this.handleUserInput.bind(this)} />

<ContactList contacts={this.props.contacts}

filterText={this.state.filterText}/>

</div>

)

}

}

ContactsApp.propTypes = {

contacts: PropTypes.arrayOf(PropTypes.object)

}

class SearchBar extends Component {

handleChange(event){

this.props.onUserInput(event.target.value)

}

render(){

return <input type="search"

placeholder="search"

value={this.props.filterText}

onChange={this.handleChange.bind(this)} />

}

}

SearchBar.propTypes = {

onUserInput: PropTypes.func.isRequired,

filterText: PropTypes.string.isRequired

}

class ContactList extends Component {

render(){

let filteredContacts = this.props.contacts.filter(

(contact) => contact.name.indexOf(this.props.filterText) !== -1

);

return(

<ul>

{filteredContacts.map(

(contact) => <ContactItem key={contact.email}

name={contact.name}

email={contact.email} />

)}

</ul>

)

}

}

ContactList.propTypes = {

contacts: PropTypes.arrayOf(PropTypes.object),

filterText: PropTypes.string.isRequired

}

class ContactItem extends Component {

render() {

return <li>{this.props.name} - {this.props.email}</li>

}

}

ContactItem.propTypes = {

name: PropTypes.string.isRequired,

email: PropTypes.string.isRequired

}

**render(<ContactsAppContainer />, document.getElementById('root'));**

* 1. **불변성에 대한 개요**

this.setState사용

* + 1. **일반 스크립트에서의 불변성**

불변성의 주 개념은 객체를 변경하지 않고 대체하는 것

class Voucher extends Component {

constructor(){

super(…arguments);

this.state = {

**passengers:[**

**‘Simmon, Robert A. ‘,**

**‘Taylor, Kathleen R.’**

**],**

ticket: {

company: 'Dalta',

flightNo: '0990',

departure: {

airport: 'LAS',

time: '2016-08-21:19:20:10'

},

arrival: {

airport: 'MIA',

time: '2016-08-21:22:12:00'

},

codeshare: [

{company:'GL', flightNo:'9840'},

{company:'TM', flightNo:'5010'}

]

}

}

}

}

let updatedPassengers = this.state.passengers;//이 때

복사본이 만들어지는 것이 참조를 만든다

updatedPassengers.push(‘Mitchel, Vincent M.’);//참조된

원본을 직접 바꾸게 된다

this.setState({passengers:updatedPassengers});

이런 문제는

let updatedPassengers = this.state.passengers.concat(‘Mitchel, Vincent M.’);//복사후 새로 생성

this.setState({passengers:updatedPassengers});

Object.assign(target, source\_1, …, source\_n)

let updatedTicket = Object.assign({}, this.state.ticket, {flightNo:’1010’});

this.setState({ticket:updatedTicket});

Object.assign은 크롬과 파이어팍스만 지원

npm install --save babel-polyfill

import ‘babel-polyfill’

* + 1. **중첩된 객체**

자바스크립트 언어에서는 객체와 배열을 참조로 전달

Object.assign은 깊은 복사본을 만들지 않는다

let originalTicket = {

company: 'Dalta',

flightNo: '0990',

departure: {

airport: 'LAS',

time: '2016-08-21:19:20:10'

},

arrival: {

airport: 'MIA',

time: '2016-08-21:22:12:00'

},

codeshare: [

{company:'GL', flightNo:'9840'},

{company:'TM', flightNo:'5010'}

]

};

let newTicket = Object.assign({}, originalTicket, {flightNo: ‘5690’})

여기서 newTicket의 departure와 arrival 객체는 별도의 복사본이 아니라 원래 originalTicket을 참조한다

newTicket.arrival.airport=”MC0” 으로 바꾸면

originalTicket.arrival.airport도 “MC0”으로 바뀐다

* + 1. **리액트 불변성 도우미**

원본을 변하지 않고 새롭게 객체 생성

객체나 배열 복사시 새로운 객체 생성

npm install --save react-addons-update

import update from ‘react-addons-update’;

let student = {name:’John Caster’, grades:[‘A’,’C’,’B’]}

**let newStudent = update(student, {grades:{$push: [‘A’]}})**

**grades 키를 찾는다**

**새로운 값을 배열로 푸시한다**

**let newStudent = update(student, {grades:{$set: [‘A’,’A’,’B’]}})**

**완전히 새로운 배열 생성**

let originalTicket = {

company: 'Dalta',

flightNo: '0990',

departure: {

airport: 'LAS',

time: '2016-08-21:19:20:10'

},

arrival: {

airport: 'MIA',

time: '2016-08-21:22:12:00'

},

codeshare: [

{company:'GL', flightNo:'9840'},

{company:'TM', flightNo:'5010'}

]

};

let newTicket = update(originalTicket, {

arrival : {

airport: { $set: ‘MC0’}

}

});

이 때 originalTicket의 arrival의 airport의 값은 그대로 유지된다

* + - 1. **배열 인덱스**

let newTicket = update(originalTicket, {

codeshare : {

**0: {$set: {company:’AZ’, flightNo:’7320’}}**

}

})

* + - 1. **사용할 수 있는 명령**

|  |  |
| --- | --- |
| 명령 | 설명 |
| $push | 배열의 push와 비슷하게 배열 끝부분에 요소를 하나 이상 추가한다. 예:  let initialArray = [1,2,3];  let newArray = update(initialArray, {$push: [4]});  // => [1,2,3,4] |
| $unshift | 배열의 unshift와 비슷하게 배열 앞부분에 요소를 하나 이상 추가한다. 예:  let initialArray = [1,2,3];  let newArray = update(initialArray, {$unshift: [0]});  //=>[0,1,2,3] |
| $splice | 배열의 splice와 비슷하게 요소 제거 및/또는 추가를 통해 배열의 내용을 변경한다. 구문상의 주요 차이점은 배열을 처리하기 위한 splice 매개변수를 포함하는 배열의 배열을 제공해야 한다 예:  let initialArray = [1,2,’a’];  let newArray = update(initialArray, {$splice:[[2,1,3,4]]});  //=>[1,2,3,4] |
| $set | 대상을 완전히 대체한다 |
| $merge | 지정한 객체의 키를 대상과 병합한다. 예:  let obj = {a: 5, b: 3};  let newObj = update(obj, {$merge: {b: 6, c:7}});  //=>{a: 5, b: 6, c: 7} |
| $apply | 현재 값을 지정한 함수로 전달하고 새로 반환된 값으로 이를 업데이트한다. 예:  let obj = {a: 5, b: 3};  let newObj = update(obj, {b: $apply: (value) => value\*2 }});  //=> {a: 10, b: 6} |

* 1. **칸반 앱: 약간의 복잡성 추가**
     1. **외부 API에서 초기 카드 가져오기**

KanbanBoardContainer.js

import React, { Component } from ‘react’;

import KanbanBoard from ‘./KanbanBoard’;

class KanbanBoardContainer extends Component {

constructor(){

super(…arguments);

this.state = {

cards: [],

};

}

render(){

return <KanbanBoard cards={cards} />

}

}

export default KanbanBoardContainer;

npm install --save whatwg-fetch

<http://kanbanapi.pro-react.com> --> REST API

KanbanBoardContainer.js

import React, { Component } from ‘react’;

import KanbanBoard from ‘./KanbanBoard’;

import ‘whatwg-fetch’;

const API\_URL = ‘http://kanbanapi.pro-react.com’;

const API\_HEADERS = {

‘Content-Type’: ‘application/json’,

‘Authorization’: ‘any-string-you-like’

};

class KanbanBoardContainer extends Component {

constructor(){

super(…arguments);

this.state = {

cards: [],

};

}

componentDidMount(){

fetch(API\_URL+’/cards’, {headers: API\_HEADERS})

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

.then((responseData) => {

this.setState({cards: reponseData});

})

.catch((error) => {

console.log(‘Error fetching and parsing data’, error);

});

}

addTask(cardId, taskName){

}

deleteTask(cardId, taskId, taskIndex){

}

toggleTask(cardId, taskId, taskIndex){

}

render(){

return <KanbanBoard cards={this.state.cards}

taskCallbacks={{

toggle: this.toggleTask.bind(this),

delete: this.deleteTask.bind(this),

add: this.addTask.bind(this)

}}

/>

}

}

export default KanbanBoardContainer;

App.js

import React from ‘react’;

import {render} from ‘react-dom’;

import KanbanBoardContainer from ‘./KanbanBoardContainer’;

render(<KanbanBoardContainer />, document.getElementById(‘root’));

import React, { Component, PropTypes } from ‘react’;

import List from ‘./List’;

class KanbanBoard extends Component {

render(){

return (

<div className=”app”>

<List id=”todo” title=”To Do” taskCallbacks={this.props.taskCallbacks} cards={

this.props.cards.filter(

(card)=> card.status == “todo”)

} />

<List id=”in-progress” title=”In Progress” taskCallbacks={this.props.taskCallbacks} cards={

this.props.cards.filter(

(card)=> card.status == “in-progress”)

} />

<List id=”done” title=”Done” taskCallbacks={this.props.taskCallbacks} cards={

this.props.cards.filter(

(card)=> card.status == “done”)

} />

</div>

);

}

}

KanbanBoard.propTypes = {

cards: PropTypes.arrayOf(PropTypes.object),

taskCallbacks: PropTypes.object

};

export default KanbanBoard;

import React, { Component, PropTypes } from ‘react’;

import Card from ‘./Card’;

class List extends Component {

render(){

var cards = this.props.cards.map((card)=>{

return <Card key={card.id}

taskCallbacks={this.props.taskCallbacks}

{…card}

/>

});

return (

<div className=”list”>

<h1>{this.props.title}</h1>

{cards}

</div>

);

}

}

List.propTypes = {

title: PropTypes.string.isRequired,

cards: PropTypes.arrayOf(PropTypes.object),

taskCallbacks: PropTypes.object

}

export default List;

let titlePropType = (props, propName, componentName) => {

if(props[propName]){

let value = props[propName];

if(typeof value !== ‘string’ || value.length > 80){

return new Error(`${propName} in ${componentName} is longer than 80 characters`);

}

}

}

class Card extends Component {

constructor(){

super(…arguments);

this.state = {

showDetails: false

}

}

toggleDetails(){

this.setState({showDetails: !this.state.showDetails});

}

render(){

let cardDetails;

if(this.state.showDetails){

cardDetails = (<div className=”card\_\_details”>

<span dangerouslySetInnerHTML={{\_\_html: marked(this.props.description)}}

<CheckList cardId={this.props.id}

tasks={this.props.tasks}

taskCallbacks={this.props.taskCallbacks} />

<div>

);

};

let sideColor = {

position: ‘absolute’,

zIndex: -1,

top: 0,

bottom: 0,

left: 0,

width: 7,

backgroundColor: this.props.color

};

return (

<div className=”card”>

<div style={sideColor} />

<div className={this.state.showDetails? ”card\_\_title card\_\_title--is-open” : “card\_\_title”} onClick={this.toggleDetails.bind(this)}>{this.props.title}</div>

{cardDetails}

</div>

);

}

}

Card.propTypes = {

id: PropTypes.number,

title: titlePropType,

description: PropTypes.string,

color: PropTypes.string,

tasks: PropTypes.arrayOf(PropTypes.object),

taskCallbacks: PropTypes.object,

};

export default Card;

import React, { Component } from ‘react’;

class CheckList extends Component {

checkInputKeyPress(evt){

if(evt.key === ‘Enter’){

this.props.taskCallbacks.add(this.props.cardId, evt.target.value);

evt.target.value = ‘’;

}

}

render(){

let tasks = this.props.tasks.map((task, taskIndex)=>{

<li key={task.id} className=”checklist\_\_task”>

<input type=”checkbox” defaultChecked={task.done} onChange={this.props.taskCallbacks.toggle.bind(null, this.props.cardId, task.id, taskIndex)} />{task.name}

<a href=”#” className=”checklist\_\_task--remove” onClick={this.props.taskCallbacks.delete.bind(null, this.props.cardId, task.id, taskIndex)} />

</li>

});

return (

<div className=”checklist”>

<ul>{tasks}</ul>

<input type=”text”

className=”checklist--add-task”

placeholder=”Type then hit Enter to add a task”

onKeyPress={this.checkInputKeyPress.bind(this)}

/>//비제어 컴포넌트 추가

</div>

);

}

}

CheckList.propTypes = {

cardId: PropTypes.number,

tasks: PropTypes.arrayOf(PropTypes.object),

taskCallbacks: PropTypes.object

}

export default CheckList;

* + 1. **태스크 콜백을 속성과 연결**

KanbanBoardContainer.js

import React, { Component } from ‘react’;

import KanbanBoard from ‘./KanbanBoard’;

class KanbanBoardContainer extends Component {

constructor(){

super(…arguments);

this.state = {

cards: [],

};

}

render(){

return <KanbanBoard cards={cards} />

}

}

export default KanbanBoardContainer;

npm install --save whatwg-fetch

<http://kanbanapi.pro-react.com> --> REST API

KanbanBoardContainer.js

import React, { Component } from ‘react’;

import KanbanBoard from ‘./KanbanBoard’;

import ‘whatwg-fetch’;

const API\_URL = ‘http://kanbanapi.pro-react.com’;

const API\_HEADERS = {

‘Content-Type’: ‘application/json’,

‘Authorization’: ‘any-string-you-like’

};

class KanbanBoardContainer extends Component {

constructor(){

super(…arguments);

this.state = {

cards: [],

};

}

componentDidMount(){

fetch(API\_URL+’/cards’, {headers: API\_HEADERS})

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

.then((responseData) => {

this.setState({cards: reponseData});

})

.catch((error) => {

console.log(‘Error fetching and parsing data’, error);

});

}

addTask(cardId, taskName){

}

deleteTask(cardId, taskId, taskIndex){

}

toggleTask(cardId, taskId, taskIndex){

}

render(){

return <KanbanBoard cards={this.state.cards}

taskCallbacks={{

toggle: this.toggleTask.bind(this),

delete: this.deleteTask.bind(this),

add: this.addTask.bind(this)

}}

/>

}

}

export default KanbanBoardContainer;

App.js

import React from ‘react’;

import {render} from ‘react-dom’;

import KanbanBoardContainer from ‘./KanbanBoardContainer’;

render(<KanbanBoardContainer />, document.getElementById(‘root’));

import React, { Component, PropTypes } from ‘react’;

import List from ‘./List’;

class KanbanBoard extends Component {

render(){

return (

<div className=”app”>

<List id=”todo” title=”To Do” taskCallbacks={this.props.taskCallbacks} cards={

this.props.cards.filter(

(card)=> card.status == “todo”)

} />

<List id=”in-progress” title=”In Progress” taskCallbacks={this.props.taskCallbacks} cards={

this.props.cards.filter(

(card)=> card.status == “in-progress”)

} />

<List id=”done” title=”Done” taskCallbacks={this.props.taskCallbacks} cards={

this.props.cards.filter(

(card)=> card.status == “done”)

} />

</div>

);

}

}

KanbanBoard.propTypes = {

cards: PropTypes.arrayOf(PropTypes.object),

taskCallbacks: PropTypes.object

};

export default KanbanBoard;

import React, { Component, PropTypes } from ‘react’;

import Card from ‘./Card’;

class List extends Component {

render(){

var cards = this.props.cards.map((card)=>{

return <Card key={card.id}

taskCallbacks={this.props.taskCallbacks}

{…card}

/>

});

return (

<div className=”list”>

<h1>{this.props.title}</h1>

{cards}

</div>

);

}

}

List.propTypes = {

title: PropTypes.string.isRequired,

cards: PropTypes.arrayOf(PropTypes.object),

taskCallbacks: PropTypes.object

}

export default List;

let titlePropType = (props, propName, componentName) => {

if(props[propName]){

let value = props[propName];

if(typeof value !== ‘string’ || value.length > 80){

return new Error(`${propName} in ${componentName} is longer than 80 characters`);

}

}

}

class Card extends Component {

constructor(){

super(…arguments);

this.state = {

showDetails: false

}

}

toggleDetails(){

this.setState({showDetails: !this.state.showDetails});

}

render(){

let cardDetails;

if(this.state.showDetails){

cardDetails = (<div className=”card\_\_details”>

<span dangerouslySetInnerHTML={{\_\_html: marked(this.props.description)}}

<CheckList cardId={this.props.id}

tasks={this.props.tasks}

taskCallbacks={this.props.taskCallbacks} />

<div>

);

};

let sideColor = {

position: ‘absolute’,

zIndex: -1,

top: 0,

bottom: 0,

left: 0,

width: 7,

backgroundColor: this.props.color

};

return (

<div className=”card”>

<div style={sideColor} />

<div className={this.state.showDetails? ”card\_\_title card\_\_title--is-open” : “card\_\_title”} onClick={this.toggleDetails.bind(this)}>{this.props.title}</div>

{cardDetails}

</div>

);

}

}

Card.propTypes = {

id: PropTypes.number,

title: titlePropType,

description: PropTypes.string,

color: PropTypes.string,

tasks: PropTypes.arrayOf(PropTypes.object),

taskCallbacks: PropTypes.object,

};

export default Card;

import React, { Component } from ‘react’;

class CheckList extends Component {

checkInputKeyPress(evt){

if(evt.key === ‘Enter’){

this.props.taskCallbacks.add(this.props.cardId, evt.target.value);

evt.target.value = ‘’;

}

}

render(){

let tasks = this.props.tasks.map((task, taskIndex)=>{

<li key={task.id} className=”checklist\_\_task”>

<input type=”checkbox” defaultChecked={task.done} onChange={this.props.taskCallbacks.toggle.bind(null, this.props.cardId, task.id, taskIndex)} />{task.name}

<a href=”#” className=”checklist\_\_task--remove” onClick={this.props.taskCallbacks.delete.bind(null, this.props.cardId, task.id, taskIndex)} />

</li>

});

return (

<div className=”checklist”>

<ul>{tasks}</ul>

<input type=”text”

className=”checklist--add-task”

placeholder=”Type then hit Enter to add a task”

onKeyPress={this.checkInputKeyPress.bind(this)}

/>//비제어 컴포넌트 추가

</div>

);

}

}

CheckList.propTypes = {

cardId: PropTypes.number,

tasks: PropTypes.arrayOf(PropTypes.object),

taskCallbacks: PropTypes.object

}

export default CheckList;

* + 1. **태스크 조작**

npm install --save react-addons-update

array.prototype.find와 array.prototype.findIndex 메서드는 크롬과 파이어 팍스만 지원

npm install --save babel-polyfill

import ‘babel-polyfill’

KanbanBoardContainer.js

import React, { Component } from ‘react’;

import KanbanBoard from ‘./KanbanBoard’;

class KanbanBoardContainer extends Component {

constructor(){

super(…arguments);

this.state = {

cards: [],

};

}

render(){

return <KanbanBoard cards={cards} />

}

}

export default KanbanBoardContainer;

npm install --save whatwg-fetch

<http://kanbanapi.pro-react.com> --> REST API

KanbanBoardContainer.js

import React, { Component } from ‘react’;

import update from ‘react-addons-update’;

import KanbanBoard from ‘./KanbanBoard’;

import ‘whatwg-fetch’;

import ‘babel-polyfill’;

const API\_URL = ‘http://kanbanapi.pro-react.com’;

const API\_HEADERS = {

‘Content-Type’: ‘application/json’,

‘Authorization’: ‘any-string-you-like’

};

class KanbanBoardContainer extends Component {

constructor(){

super(…arguments);

this.state = {

cards: [],

};

}

componentDidMount(){

fetch(API\_URL+’/cards’, {headers: API\_HEADERS})

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

.then((responseData) => {

this.setState({cards: reponseData});

})

.catch((error) => {

console.log(‘Error fetching and parsing data’, error);

});

}

addTask(cardId, taskName){

let prevState = this.state;

let cardIndex = this.state.cards.findIndex((card)=>card.id==cardId);

let newTask = {id:Date.now(), name:taskName, done: false};

let nextState = update(this.state.cards, {

[cardIndex]: {

tasks: {$push: [newTask]}

}

});

this.setState({cards:nextState});

fetch(‘${API\_URL}/cards/${cardId}/tasks’,{

method: ‘post’,

headers: API\_HEADERSM

body: JSON.stringify(newTask)

})

.then((response)=>{

if(response.ok){

return response.json();

}else{

thow new Error(“Server response wasn’t OK”)

}

})

.then((responseData)=>{

newTask.id = responseData.id;

this.setState({cards:nextState});

})

.catch((error)=>{

this.setState(prevState);

})

}

deleteTask(cardId, taskId, taskIndex){

let prevState = this.state;

let cardIndex = this.state.cards.findIndex((card)=>card.id == cardId);

let nextState = update(this.state.cards, {

[cardIndex]: {

tasks: {$splice: [[taskIndex,1]]}

}

});

this.setState({cards:nextState});

fetch(‘${API\_URL}/cards/${cardId}/tasks/${taskId}’,{

method: ‘delete’,

headers: API\_HEADERS

})

.then((response) => {

if(!response.ok){

throw new Error(“Server response wasn’t OK”)

}

})

.catch((error)=>{

console.error(“Fetch error:”, error)

this.setState(prevState);

});

}

toggleTask(cardId, taskId, taskIndex){

let prevState = this.state;

let cardIndex = this.state.cards.findIndex((card)=>card.id == cardId);

let newDoneValue;

let nextState = update(this.state.cards, {

[cardIndex]:{

tasks: {

[taskIndex]:{

done: {

$apply: (done) =>{

newDoneValue = !done;

return newDoneValue;

}

}

}

}

}

});

this.setState({cards:nextState});

fetch(‘${API\_URL}/cards/${cardId}/tasks/${taskId}’,{

method: ‘PUT’,

headers: API\_HEADERS,

body: JSON.stringify({done:newDoneValue})

});

}

render(){

return <KanbanBoard cards={this.state.cards}

taskCallbacks={{

toggle: this.toggleTask.bind(this),

delete: this.deleteTask.bind(this),

add: this.addTask.bind(this)

}}

/>

}

}

export default KanbanBoardContainer;

App.js

import React from ‘react’;

import {render} from ‘react-dom’;

import KanbanBoardContainer from ‘./KanbanBoardContainer’;

render(<KanbanBoardContainer />, document.getElementById(‘root’));

import React, { Component, PropTypes } from ‘react’;

import List from ‘./List’;

class KanbanBoard extends Component {

render(){

return (

<div className=”app”>

<List id=”todo” title=”To Do” taskCallbacks={this.props.taskCallbacks} cards={

this.props.cards.filter(

(card)=> card.status == “todo”)

} />

<List id=”in-progress” title=”In Progress” taskCallbacks={this.props.taskCallbacks} cards={

this.props.cards.filter(

(card)=> card.status == “in-progress”)

} />

<List id=”done” title=”Done” taskCallbacks={this.props.taskCallbacks} cards={

this.props.cards.filter(

(card)=> card.status == “done”)

} />

</div>

);

}

}

KanbanBoard.propTypes = {

cards: PropTypes.arrayOf(PropTypes.object),

taskCallbacks: PropTypes.object

};

export default KanbanBoard;

import React, { Component, PropTypes } from ‘react’;

import Card from ‘./Card’;

class List extends Component {

render(){

var cards = this.props.cards.map((card)=>{

return <Card key={card.id}

taskCallbacks={this.props.taskCallbacks}

{…card}

/>

});

return (

<div className=”list”>

<h1>{this.props.title}</h1>

{cards}

</div>

);

}

}

List.propTypes = {

title: PropTypes.string.isRequired,

cards: PropTypes.arrayOf(PropTypes.object),

taskCallbacks: PropTypes.object

}

export default List;

let titlePropType = (props, propName, componentName) => {

if(props[propName]){

let value = props[propName];

if(typeof value !== ‘string’ || value.length > 80){

return new Error(`${propName} in ${componentName} is longer than 80 characters`);

}

}

}

class Card extends Component {

constructor(){

super(…arguments);

this.state = {

showDetails: false

}

}

toggleDetails(){

this.setState({showDetails: !this.state.showDetails});

}

render(){

let cardDetails;

if(this.state.showDetails){

cardDetails = (<div className=”card\_\_details”>

<span dangerouslySetInnerHTML={{\_\_html: marked(this.props.description)}}

<CheckList cardId={this.props.id}

tasks={this.props.tasks}

taskCallbacks={this.props.taskCallbacks} />

<div>

);

};

let sideColor = {

position: ‘absolute’,

zIndex: -1,

top: 0,

bottom: 0,

left: 0,

width: 7,

backgroundColor: this.props.color

};

return (

<div className=”card”>

<div style={sideColor} />

<div className={this.state.showDetails? ”card\_\_title card\_\_title--is-open” : “card\_\_title”} onClick={this.toggleDetails.bind(this)}>{this.props.title}</div>

{cardDetails}

</div>

);

}

}

Card.propTypes = {

id: PropTypes.number,

title: titlePropType,

description: PropTypes.string,

color: PropTypes.string,

tasks: PropTypes.arrayOf(PropTypes.object),

taskCallbacks: PropTypes.object,

};

export default Card;

import React, { Component } from ‘react’;

class CheckList extends Component {

checkInputKeyPress(evt){

if(evt.key === ‘Enter’){

this.props.taskCallbacks.add(this.props.cardId, evt.target.value);

evt.target.value = ‘’;

}

}

render(){

let tasks = this.props.tasks.map((task, taskIndex)=>{

<li key={task.id} className=”checklist\_\_task”>

<input type=”checkbox” defaultChecked={task.done} onChange={this.props.taskCallbacks.toggle.bind(null, this.props.cardId, task.id, taskIndex)} />{task.name}

<a href=”#” className=”checklist\_\_task--remove” onClick={this.props.taskCallbacks.delete.bind(null, this.props.cardId, task.id, taskIndex)} />

</li>

});

return (

<div className=”checklist”>

<ul>{tasks}</ul>

<input type=”text”

className=”checklist--add-task”

placeholder=”Type then hit Enter to add a task”

onKeyPress={this.checkInputKeyPress.bind(this)}

/>//비제어 컴포넌트 추가

</div>

);

}

}

CheckList.propTypes = {

cardId: PropTypes.number,

tasks: PropTypes.arrayOf(PropTypes.object),

taskCallbacks: PropTypes.object

}

export default CheckList;

* + 1. **기본적인 낙관적 업데이트 롤백**

1. **정교한 상호 작용**
   1. **리액트의 애니메이션**
      1. **CSS 트랜지션과 애니메이션의 기초**

**CSS 트랜지션**

transition 속성 사용

transition: 에니메이션을 적용할 속성 이름(예: color 또는 width) 생략하면 애니메이션 가능한 모든 속성이 대상

애니메이션 지속 시간

가속 곡선 타이밍 함수 ease-in, ease-out

애니메이션 시작 전 지연 시간

<!DOCTYPE html>

<html>

<head>

<meta charset=”utf-8”>

<title>Hover Transition</title>

<style media=”screen”>

a{

font-family: Helvetica, Arial, sans-serif;

text-decoration: none;

color:#ffffff;

}

.button{

padding: 0.75rem 1rem;

border-radius: 0.3rem;

box-shadow: 0;

background-color: #bbbbbb;

}

.button:hover{

background-color:#ee2222;

box-shadow: 0 4px #990000;

webkit-transition: 0.5s;

transition: 0.5s;

}

</style>

</head>

<body>

<a href=”#” class=”button”>Hover Me! </div>

</body>

</html>

**접두사 관련 사항**

webkit-transition: 0.5s;

**키프레임 애니메이션**

@keyframes pulsing-heart {

0% { transform: none; }

50% { transform: scale(1.4); }

100% { transform: none; }

}

<!DOCTYPE html>

<html>

<head>

<meta charset=”utf-8”>

<title>Pulsing Heart</title>

<style media=”screen”>

body{

text-align: center;

}

@keyframes pulsing-heart {

0% { transform: none; }

50% { transform: scale(1.4); }

100% { transform: none; }

}

.heart{

font-size: 10rem;

color: #FF0000;

}

.heart:hover{

animation: pulsing-heart .5s infinite;

transform-origin: center

}

</style>

</head>

<body>

<div>

<div class=”heart”>&hearts; </div>

</div>

</body>

</html>

**프로그래밍 방식으로 CSS 트랜지션과 애니메이션 시작**

/\* 사이드바 기본 스타일 \*/

.sidebar{

background-color:#eee;

box-shadow: 1px 0 3px #888888;

position:absolute;

width: 15rem;

height: 100%;

}

.sidebar-transition{

opacity: 0;

left: -15rem;

}

.sidebar-transition-active{

opacity: 1;

left: 0;

transition: ease-in-out 0.5s;

}

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Hacked together sidebar Transition</title>

<style media="screen">

ul{

list-style-type: none;

padding: 0;

}

li{

padding: 15px;

border-bottom: solid 1px #eee;

background-color: #ddd;

}

.sidebar{

background-color: #eee;

box-shadow: 1px 0 3px #888888;

position: absolute;

width: 15rem;

height: 100%;

}

.sidebar-transition{

opacity: 0;

left: -15rem;

}

.sidebar-transition-active{

opacity: 1;

left: 0;

transition: 0.5s;

}

</style>

</head>

<body>

<header>

<button onClick="document.querySelector('.sidebar').classList.add('sidebar-transition-active');"> &#9776; </button>

</header>

<div class="sidebar sidebar-transition">

<ul>

<li>Some Content</li>

<li>Content B</li>

<li>Content C</li>

<li>Content X</li>

</ul>

</div>

</body>

</html>

* + 1. **ReactCSSTransitionGroup**

npm install --save react-addons-css-transition-group

**리액트 애니메이션: 쇼핑 리스트**

새로운 프로젝트 생성 혹은 <https://github.com/pro-react/react-app-boilerplate> 사용

AnimatedShoppingList

import React, { Component } from ‘react’;

import { render } from ‘react-dom’;

import ReactCSSTransitionGroup from ‘react-addons-transition-group’;

class AnimatedShoppingList extends Component {

constructor(){

super(…arguments);

this.state={

items: [

{id:1, name: ‘Milk”},

{id:2, name:’Yogurt’},

{id:3, name:’Orange Juice’},

]

}

}

handleChange(evt){

if(evt.key === ‘Enter’){

let newItem = {id:Date.now(), name:evt.target.value}

let newItems = this.state.items.concat(newItem);

evt.target.value = ‘’;

this.setState({items: newItems});

}

}

handleRemove(i){

var newItems = this.state.items;

newItems.splice(i, 1);

this.setState({items: newItems});

}

render(){

let shoppingItems = this.state.items.map((item, 1)=>{

<div key={item.id}

className=”item”

onClick={this.handleRemove.bind(this, i)}>

{item.name}

</div>

});

return (

<div>

<ReactCSSTransitionGroup

transitionName=”example” //클래스이름 요소가 추가될 때마다 example-enter와 example-enter-active클래스가 추가된다

transitionEnterTimeout={300}// enter가 완료된 후 이 시간이 지나면 example-enter와 example-enter-active 클래스가 제거된다

transitionLeaveTimeout={300}>

{shoppingItems}

</ReactCSSTransitionGroup>

<input type=”text” value={this.state.newItem}

onKeyDown={this.handleChange.bind(this)} />

</div>

);

}

};

render(<AnimatedShoppingList />, document.getElementById(‘root’));

**기본 애니메이션 설정**

input{

padding: 5px;

width: 120px;

margin-top: 10px;

}

.item{

background-color: #efefef;

cursor: pointer;

display: block;

margin-bottom: 1px;

padding: 8px 12px;

width: 120px;

}

**ReactCSSTransitionGroup 요소 추가**

애미메이션을 적용하려는 자식 요소는 ReactCSSTransitionGroup으로 매핑해야 한다

.example-enter{

opacity: 0;

transform: translateX(-250px)

}

.example-enter-active{

opacity: 1;

transform: translateX(0);

transition: 0.3s;

}

.example-leave{

opacity: 1;

transform: translateX(0);

}

.example-leave.example-leave-active{

opacity: 0;

transform: translateX(250px)

transition: 0.3s

}

**초기 마운팅 애니메이션**

초기 로딩시는 애니메이션이 적용되지 않는다

<ReactCSSTransitionGroup

transitionAppear={true}//마운팅시

transitionAppearTimeout={300}

transitionName=”example” transitionEnterTimeout={300}

transitionLeaveTimeout={300}>

{shoppingItems}

</ReactCSSTransitionGroup>

.example-appear{

opacity: 0;

transform: translateX(-250px)

}

.example-appear.example-appear-active{

opacity: 1;

transform: translateX(0);

transition: 0.5s;

}

* 1. **드래드 앤드 드롭**

npm install --save [react-dnd@2.x.x](mailto:react-dnd@2.x.x) [react-dnd-html5-backend@1.x.x](mailto:react-dnd-html5-backend@1.x.x)

**리액트 DnD 구현 개요**

고차(higher-order) 컴포넌트는 매개변수로 컴포넌트를 매개변수로 받고 여기에 기능을 추가한 향상된 버전의 컴포넌트를 반환하는 자바스크립트 함수이다.

DragSource는 드래그 원본 컴포넌트 반환

DropTarget은 드래그 대상 컴포넌트 반환

DragDropContext는 드래그 앤드 드랍 이 수행되는 부모 컴포넌트 래핑 내부적으로 공유하는 DnD상태 설정

**리액트 DnD 예제 구현**

간식 앱 구현

**Drag here to order!**

App

import React, { Component } from ‘react’;

import {render} from ‘react-dom’;

import Container from ‘./Container’;

class App extends Component {

render(){

return (

<Container />

)

}

}

render(<App />, document.getElementById(‘root’));

**Container 컴포넌트**

import React, { Component } from ‘react’;

import ShoppingCart from ‘./ShoppingCart’;

import Snack from ‘./Snack’;

import { DragDropContext } from ‘react-dnd’;

import HTML5Backend from ‘react-dnd-html5-backend’;

class Container extends Component {

render(){

return (

<div>

<Snack name=’Chips’ />

<Snack name=’Cupcake’ />

<Snack name=’Donut’ />

<Snack name=’Doritos’ />

<Snack name=’Popcorn’ />

<ShoppingCart/>

</div>

);

}

}

export default DragDropContext(HTML5Backend)(Container)

**DragSource와 DropTarget 고차 컴포넌트**

고차 컴포넌트 생성시 아래 3개 매개변수 필요

**타입**

컴포넌트의 이름

**사양 객체**

드래그앤드랍시 사용되는 함수를 포함하는 객체

DragSource: beginDrag, endDrag

DropTarget: canDrag, onDrop

**콜렉팅 함수**

속성 주입 함수

현재 Drag나 Drop 할 시 공유되는 상태값이나 속성 핸들러 함수

**ShoppingCart 컴포넌트**

import React, { PropTypes, Component } from ‘react’;

import { DropTarget } from ‘react-dnd’;

//ShoppingCart 드래그 앤드 드롭 사양

// - DropTarget 메서드(모두 선택적)

// - drop: 호환되는 항목이 드랍되면 호출

// - hover: 항목으로 컴포넌트를 가리키면 호출

// - canDrop: 드랍 대상이 항목을 수락할 수 있는지 여부를 지정하는 데 사용

const ShoppingCartSpec = {//사양 객체

drop(){//드랍이 되면

return { name: ‘ShoppingCart’ };

}

};

//collect 함수 리액트 DnD 커넥터와 상태를 컴포넌트의 속성과 연결

//connect: 드롭 대상 역할을 DOM 노드에 할당

//monitor: 리액트 DnD에서 속성으로 상태 연결

//canDrop(), isOver(), didDrop()

let collect = (connect, monitor) => {

return {

connectDropTarget: connect.dropTarget(),

isOver: monitor.isOver(),

canDrop: monitor.canDrop()

};

}

class ShoppingCart extends Component {

render(){

const { canDrop, isOver, connectDropTarget } = this.props;//구조분해 할당

const isActive = canDrop && isOver;

let backgroundColor = ‘#FFFFFF’;

if(isActive){

backgroundColor = ‘#F7F7BD’;

}else if (canDrop) {

backgroundColor =’#F7F7F7’;

}

const style = {

backgroundColor: backgroundColor

};

return connectDropTarget(

<div className=’shopping-cart’ style={style}>

{isActive ?

‘Hummm, snack!’ :

‘Drag here to order!’

}

</div>

);

}

}

ShoppingCart.propTypes = {

connectDropTarget: PropTypes.func.isRequired,

isOver: PropTypes.**bool**.isRequired,

canDrop: PropTypes.bool.isRequired

}

export default DropTarget(“snack”, ShoppingCartSpec, collect)(ShoppingCart);

**Snack 컴포넌트**

import React, { Component, PropTypes } from ‘react’;

import { DragSource } from ‘react-dnd’;

//snack 드래그 앤드 드롭 사양

//

// - 필수: beginDrag

// - 선택: endDrag

// - 선택: canDrag

// - 선택: isDragging

const snackSpec = {

beginDrag(props){

return {

name: props.name

};

},

endDrag(props, monitor){

const dragItem = monitor.getItem();

const dropResult = monitor.getDropResult();

if(dropResult){

console.log(‘You dropped ${dragItem.name} into ${dropResult.name}’);

}

}

};

//Snack DragSource collect 콜렉팅 함수

// - connect: DragSourceConnector의 인스턴스

// 드래그 원본 역할을 DOM 노드에 할당

// - monitor: DragSourceMonitor의 인스턴스

// 리액트 DnD에서 속성으로 컴포넌트의 상태를 연결

// 상태를 얻을 수 있는 함수로는 canDrag(), isDragging(),

// getItemType(), getItem(), didDrop()

let collect = (connect, monitor) => {

return {

connectDragSource: connect.dragSource(),

isDragging: monitor.isDragging()

};

}

class Snack extends Component {

render(){

const { name, connectDragSource, isDragging } = this.props;

const opacity = isDragging ? 0.4 : 1;

const style = {

opacity: opacity

};

return (

connectDragSource(

<div className=’snack’ style={style}>

{name}

</div>

)

);

}

}

Snack.propTypes = {

connectDragSource: PropTypes.func.isRequired,

name: PropTypes.string.isRequired,

isDragging: PropTypes.bool.isRequired

};

**스타일링**

body {

font: 16px/1 sans-serif;

}

#root {

height: 100%;

}

h1 {

font-weight: 200;

color: #3b414c;

font-size: 20px;

}

.app {

white-space: nowrap;

height: 100%;

}

.snack {

display: inline-block;

padding: .5em;

margin: 0 1em 1em 0.25em;

border: 4px solid #d9d9d9;

background: #f7f7f7;

height: 5rem;

width: 5rem;

border-radius: 5rem;

cursor: pointer;

line-height: 5em;

text-align: center;

color: #333;

}

.shopping-cart {

border: 5px dashed #d9d9d9;

border-radius: 10px;

padding: 5rem 2rem;

text-align: center;

}

**리팩터: 상수 이용**

constants.js

export default {

SNACK: ‘snack’

};

ShoppingCart

…

import constants from ‘../constants’;

…

export default DropTarget(constants.SNACK, ShoppingCartSpec, collect)(ShoppingCart);

Snack

…

import constants from ‘../constants’;

…

export default DragSource(constants.SNACK, snackSpec, collect)(Snack);;

* 1. **칸반 앱: 애니메이션과 드래그 앤드 드랍 지원**

**카드 토글 애니메이션**

npm install --save react-addons-css-transition-group

Card

import React, { Component, PropTypes } from ‘react’;

import ReactCSSTransitionGroup from ‘react-addons-css-transition-group’;

import marked from ‘marked’;

import CheckList from ’./CheckList’;

let titlePropType =m(props, propName, componentName) =>{

if(props[propName]){

let value = props[propName];

if(typeof value !== ‘string’ || value.length > 80){

return new Error(

‘${propName} in ${componentName} is longer than 80 characters’

);

}

}

};

class Card extends Component {

constructor(){

super(…arguments);

this.state = {

showDetails: false

}

}

toggleDetails(){

this.setState({showDetails: !this.state.showDetails});

}

render(){

let cardDetails;

if(this.state.showDetails){

cardDetails = (<div className=”card\_\_details”>

<span dangerouslySetInnerHTML={{\_\_html: marked(this.props.description)}}

<CheckList cardId={this.props.id}

tasks={this.props.tasks}

taskCallbacks={this.props.taskCallbacks} />

<div>

);

};

let sideColor = {

position: ‘absolute’,

zIndex: -1,

top: 0,

bottom: 0,

left: 0,

width: 7,

backgroundColor: this.props.color

};

return (

<div className=”card”>

<div style={sideColor} />

<div className={this.state.showDetails? ”card\_\_title card\_\_title--is-open” : “card\_\_title”} onClick={this.toggleDetails.bind(this)}>{this.props.title}</div>

<ReactCSSTransitionGroup transitionName=”toggle”

transitionEnterTimeout={250}

transitionLeaveTimeout={250}>

{cardDetails}

</div>

</ReactCSSTransitionGroup>

);

}

}

Card.propTypes = {

id: PropTypes.number,

title: titlePropType,

description: PropTypes.string,

color: PropTypes.string,

tasks: PropTypes.arrayOf(PropTypes.object),

taskCallbacks: PropTypes.object,

};

export default Card;

.toggle-enter{

max-height: 0;

overflow: hidden;

}

.toggle-enter.toggle-enter-active{

max-height: 300px;

overflow: hidden;

transition: max-height .25s ease-in;

}

.toggle-leave{

max-height: 300px;

overflow: hidden;

}

.toggle-leave.toggle-leave-active{

max-height: 0;

overflow: hidden;

transition: max-height .25s ease-in;

}

**카드 드래그**

npm install --save [react-dnd@2.x.x](mailto:react-dnd@2.x.x) [react-dnd-html5-backend@1.x.x](mailto:react-dnd-html5-backend@1.x.x)

KanbanBoardContainer.js

import React, { Component } from ‘react’;

import update from ‘react-addons-update’;

import KanbanBoard from ‘./KanbanBoard’;

import ‘whatwg-fetch’;

import ‘babel-polyfill’;

const API\_URL = ‘http://kanbanapi.pro-react.com’;

const API\_HEADERS = {

‘Content-Type’: ‘application/json’,

‘Authorization’: ‘any-string-you-like’

};

class KanbanBoardContainer extends Component {

constructor(){

super(…arguments);

this.state = {

cards: [],

};

}

componentDidMount(){

fetch(API\_URL+’/cards’, {headers: API\_HEADERS})

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

.then((responseData) => {

this.setState({cards: reponseData});

})

.catch((error) => {

console.log(‘Error fetching and parsing data’, error);

});

}

addTask(cardId, taskName){

let prevState = this.state;

let cardIndex = this.state.cards.findIndex((card)=>card.id==cardId);

let newTask = {id:Date.now(), name:taskName, done: false};

let nextState = update(this.state.cards, {

[cardIndex]: {

tasks: {$push: [newTask]}

}

});

this.setState({cards:nextState});

fetch(‘${API\_URL}/cards/${cardId}/tasks’,{

method: ‘post’,

headers: API\_HEADERSM

body: JSON.stringify(newTask)

})

.then((response)=>{

if(response.ok){

return response.json();

}else{

thow new Error(“Server response wasn’t OK”)

}

})

.then((responseData)=>{

newTask.id = responseData.id;

this.setState({cards:nextState});

})

.catch((error)=>{

this.setState(prevState);

})

}

deleteTask(cardId, taskId, taskIndex){

let prevState = this.state;

let cardIndex = this.state.cards.findIndex((card)=>card.id == cardId);

let nextState = update(this.state.cards, {

[cardIndex]: {

tasks: {$splice: [[taskIndex,1]]}

}

});

this.setState({cards:nextState});

fetch(‘${API\_URL}/cards/${cardId}/tasks/${taskId}’,{

method: ‘delete’,

headers: API\_HEADERS

})

.then((response) => {

if(!response.ok){

throw new Error(“Server response wasn’t OK”)

}

})

.catch((error)=>{

console.error(“Fetch error:”, error)

this.setState(prevState);

});

}

toggleTask(cardId, taskId, taskIndex){

let prevState = this.state;

let cardIndex = this.state.cards.findIndex((card)=>card.id == cardId);

let newDoneValue;

let nextState = update(this.state.cards, {

[cardIndex]:{

tasks: {

[taskIndex]:{

done: {

$apply: (done) =>{

newDoneValue = !done;

return newDoneValue;

}

}

}

}

}

});

this.setState({cards:nextState});

fetch(‘${API\_URL}/cards/${cardId}/tasks/${taskId}’,{

method: ‘PUT’,

headers: API\_HEADERS,

body: JSON.stringify({done:newDoneValue})

});

}

updateCardStatus(cardId, listId){

let cardIndex = this.state.cards.findIndex((card)=>card.id == cardId);

let card = this.state.cards[cardIndex]

if(card.status !== listId){

this.setState(update(this.state, {

cards: {

[cardIndex]: {

status: {$set: listId }

}

}

}

);

}

}

updateCardPosition(cardId, afterId){

if(cardId !== afterId){

let cardIndex = this.state.cards.findIndex((card)=>card.id == cardId);

let card = this.state.cards[cardIndex];

let afterIndex = this.state.cards.findIndex((card)=>card.id == afterId);

this.setState(update(

this.state, {

cards: {

$splice: [

[cardIndex, 1],

[afterIndex, 0, card]

]

}

}

));

}

}

render(){

return <KanbanBoard cards={this.state.cards}

taskCallbacks={{

toggle: this.toggleTask.bind(this),

delete: this.deleteTask.bind(this),

add: this.addTask.bind(this)

}}

cardCallbacks ={{

updateStatus: this.updateCardStatus.bind(this),

updatePosition: this.updateCardPosition.bind(this)

}}

/>

}

}

export default KanbanBoardContainer;

import React, { Component, PropTypes } from ‘react’;

import List from ‘./List’;

class KanbanBoard extends Component {

render(){

return (

<div className=”app”>

<List id=”todo” title=”To Do” taskCallbacks={this.props.taskCallbacks}

cardCallbacks={this.props.cardCallbacks}

cards={

this.props.cards.filter(

(card)=> card.status == “todo”)

} />

<List id=”in-progress” title=”In Progress” taskCallbacks={this.props.taskCallbacks}

cardCallbacks={this.props.cardCallbacks}

cards={

this.props.cards.filter(

(card)=> card.status == “in-progress”)

} />

<List id=”done” title=”Done” taskCallbacks={this.props.taskCallbacks}

cardCallbacks={this.props.cardCallbacks}

cards={

this.props.cards.filter(

(card)=> card.status == “done”)

} />

</div>

);

}

}

KanbanBoard.propTypes = {

cards: PropTypes.arrayOf(PropTypes.object),

taskCallbacks: PropTypes.object,

cardCallbacks: PropTypes.object

};

export default KanbanBoard;

import React, { Component, PropTypes } from ‘react’;

import Card from ‘./Card’;

class List extends Component {

render(){

var cards = this.props.cards.map((card)=>{

return <Card key={card.id}

taskCallbacks={this.props.taskCallbacks}

cardCallbacks={this.props.cardCallbacks}

{…card}

/>

});

return (

<div className=”list”>

<h1>{this.props.title}</h1>

{cards}

</div>

);

}

}

List.propTypes = {

title: PropTypes.string.isRequired,

cards: PropTypes.arrayOf(PropTypes.object),

taskCallbacks: PropTypes.object,

cardCallbacks: PropTypes.object

}

export default List;

constants.js

export default {

CARD: ‘card’

}

**다른 리스트 사이로 드래그**

import React, { Component, PropTypes } from ‘react’;

import ReactCSSTransitionGroup from ‘react-addons-css-transition-group’;

import marked from ‘marked’;

import CheckList from ’./CheckList’;

import { DragSource } from ‘react-dnd’

import constants from ‘./constants’;

let titlePropType =m(props, propName, componentName) =>{

if(props[propName]){

let value = props[propName];

if(typeof value !== ‘string’ || value.length > 80){

return new Error(

‘${propName} in ${componentName} is longer than 80 characters’

);

}

}

};

const cardDragSpec = {

beginDrag(props){

return {

id: props.id

}

}

}

let collectDrag = (connect, monitor) => {

return {

connectDragSource: connect.dragSource()

};

}

class Card extends Component {

constructor(){

super(…arguments);

this.state = {

showDetails: false

}

}

toggleDetails(){

this.setState({showDetails: !this.state.showDetails});

}

render(){

const { connectDragSource } = this.props;

let cardDetails;

if(this.state.showDetails){

cardDetails = (<div className=”card\_\_details”>

<span dangerouslySetInnerHTML={{\_\_html: marked(this.props.description)}}

<CheckList cardId={this.props.id}

tasks={this.props.tasks}

taskCallbacks={this.props.taskCallbacks} />

<div>

);

};

let sideColor = {

position: ‘absolute’,

zIndex: -1,

top: 0,

bottom: 0,

left: 0,

width: 7,

backgroundColor: this.props.color

};

return connectDragSource(

<div className=”card”>

<div style={sideColor} />

<div className={this.state.showDetails? ”card\_\_title card\_\_title--is-open” : “card\_\_title”} onClick={this.toggleDetails.bind(this)}>{this.props.title}</div>

<ReactCSSTransitionGroup transitionName=”toggle”

transitionEnterTimeout={250}

transitionLeaveTimeout={250}>

{cardDetails}

</div>

</ReactCSSTransitionGroup>

);

}

}

Card.propTypes = {

id: PropTypes.number,

title: titlePropType,

description: PropTypes.string,

color: PropTypes.string,

tasks: PropTypes.arrayOf(PropTypes.object),

taskCallbacks: PropTypes.object,

cardCallbacks: PropTypes.object,

connectDragSource: PropTypes.func.isRequired

};

export default DragSource(constants.CARD, cardDragSpec, collectDrag)(Card);

import React, { Component, PropTypes } from ‘react’;

import Card from ‘./Card’;

import { DropTarget } from ‘react-dnd’;

import constants from ‘./constants’;

const listTargetSpec = {

hover(props, monitor){

const draggedId = monitor.getItem().id;

props.cardCallbacks.updateStatus(draggedId, props.id)

}

};

function collect(connect, monitor){

return {

connectDropTarget: connect.dropTarget()

}

}

class List extends Component {

render(){

const { connectDropTarget } = this.props;

var cards = this.props.cards.map((card)=>{

return <Card key={card.id}

taskCallbacks={this.props.taskCallbacks}

cardCallbacks={this.props.cardCallbacks}

{…card}

/>

});

return connectDropTarget(

<div className=”list”>

<h1>{this.props.title}</h1>

{cards}

</div>

);

}

}

List.propTypes = {

title: PropTypes.string.isRequired,

cards: PropTypes.arrayOf(PropTypes.object),

taskCallbacks: PropTypes.object,

cardCallbacks: PropTypes.object,

connectDropTarget: PropTypes.func.isRequired

}

export default DropTarget(constants.CARD, listTargetSpec, collect)(List);

import React, { Component, PropTypes } from ‘react’;

import { DragDropContext } from ‘react-dnd’;

import HTML5Backend from ‘react-dnd-html5-backend’;

import List from ‘./List’;

class KanbanBoard extends Component {

render(){

return (

<div className=”app”>

<List id=”todo” title=”To Do” taskCallbacks={this.props.taskCallbacks}

cardCallbacks={this.props.cardCallbacks}

cards={

this.props.cards.filter(

(card)=> card.status == “todo”)

} />

<List id=”in-progress” title=”In Progress” taskCallbacks={this.props.taskCallbacks}

cardCallbacks={this.props.cardCallbacks}

cards={

this.props.cards.filter(

(card)=> card.status == “in-progress”)

} />

<List id=”done” title=”Done” taskCallbacks={this.props.taskCallbacks}

cardCallbacks={this.props.cardCallbacks}

cards={

this.props.cards.filter(

(card)=> card.status == “done”)

} />

</div>

);

}

}

KanbanBoard.propTypes = {

cards: PropTypes.arrayOf(PropTypes.object),

taskCallbacks: PropTypes.object,

cardCallbacks: PropTypes.object

};

export default DragDropContext(HTML5Backend)(KanbanBoard);

**카드 정렬**

import React, { Component, PropTypes } from ‘react’;

import ReactCSSTransitionGroup from ‘react-addons-css-transition-group’;

import marked from ‘marked’;

import CheckList from ’./CheckList’;

import { DragSource, DropTarget } from ‘react-dnd’

import constants from ‘./constants’;

let titlePropType =m(props, propName, componentName) =>{

if(props[propName]){

let value = props[propName];

if(typeof value !== ‘string’ || value.length > 80){

return new Error(

‘${propName} in ${componentName} is longer than 80 characters’

);

}

}

};

const cardDragSpec = {

beginDrag(props){

return {

id: props.id

}

}

}

const cardDropSpec = {

hover(props, monitor){

const draggedId = monitor.getItem().id;

props.cardCallbacks.updatePosition(draggedId, props.id);

}

}

let collectDrag = (connect, monitor) => {

return {

connectDragSource: connect.dragSource()

};

}

let collectDrop = (connect, monitor) => {

return {

connectDropTarget: connect.dropTarget(),

};

}

class Card extends Component {

constructor(){

super(…arguments);

this.state = {

showDetails: false

}

}

toggleDetails(){

this.setState({showDetails: !this.state.showDetails});

}

render(){

const { connectDragSource, connectDropTarget } = this.props;

let cardDetails;

if(this.state.showDetails){

cardDetails = (<div className=”card\_\_details”>

<span dangerouslySetInnerHTML={{\_\_html: marked(this.props.description)}}

<CheckList cardId={this.props.id}

tasks={this.props.tasks}

taskCallbacks={this.props.taskCallbacks} />

<div>

);

};

let sideColor = {

position: ‘absolute’,

zIndex: -1,

top: 0,

bottom: 0,

left: 0,

width: 7,

backgroundColor: this.props.color

};

return connectDropTarget(connectDragSource(

<div className=”card”>

<div style={sideColor} />

<div className={this.state.showDetails? ”card\_\_title card\_\_title--is-open” : “card\_\_title”} onClick={this.toggleDetails.bind(this)}>{this.props.title}</div>

<ReactCSSTransitionGroup transitionName=”toggle”

transitionEnterTimeout={250}

transitionLeaveTimeout={250}>

{cardDetails}

</div>

</ReactCSSTransitionGroup>

));

}

}

Card.propTypes = {

id: PropTypes.number,

title: titlePropType,

description: PropTypes.string,

color: PropTypes.string,

tasks: PropTypes.arrayOf(PropTypes.object),

taskCallbacks: PropTypes.object,

cardCallbacks: PropTypes.object,

connectDragSource: PropTypes.func.isRequired,

connectDropTarget: PropTypes.func.isRequired

};

const dragHighOrderCard = DragSource(constants.CARD, cardDragSpec, collectDrag)(Card);

const dragDropHighOrderCard = DropTarget(constants.CARD, cardDropSpec, collectDrop)(dragHighOrderCard)

export default dragDropHighOrderCard;

**콜백에 스로틀 적용**

updatePosition이나 updateStatus 함수의 호출수를 제한

utils.js

export const throttle = (func, wait) => {

let context, args, prevArgs, argsChanged, result;

let previous = 0;

return function(){

let now, remaining;

if(wait){

now = Date.now();

remaining = wait - (now - previous);

}

context = this;

args = arguments;

argsChanged = JSON.stringify(args) != JSON.stringify(prevArgs);

prevArgs = {…args};

if(argsChanged || wait && (remaining <= 0 || remaining > wait)){

if(wait){

previous = now;

}

result = func.apply(context, args);

context = args = null;

}

return result;

};

};

KanbanBoardContainer.js

import React, { Component } from ‘react’;

import update from ‘react-addons-update’;

import {throttle} from ‘./utils’;

import KanbanBoard from ‘./KanbanBoard’;

import ‘whatwg-fetch’;

import ‘babel-polyfill’;

const API\_URL = ‘http://kanbanapi.pro-react.com’;

const API\_HEADERS = {

‘Content-Type’: ‘application/json’,

‘Authorization’: ‘any-string-you-like’

};

class KanbanBoardContainer extends Component {

constructor(){

super(…arguments);

this.state = {

cards: [],

};

//인수가 변경된 경우에만 호출

this.updateCardStatus = throttle(this.updateCardStatus.bind(this));

this.updateCardPosition = throttle(this.updateCardPosition.bind(this), 500);

}

componentDidMount(){

fetch(API\_URL+’/cards’, {headers: API\_HEADERS})

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

.then((responseData) => {

this.setState({cards: reponseData});

})

.catch((error) => {

console.log(‘Error fetching and parsing data’, error);

});

}

addTask(cardId, taskName){

let prevState = this.state;

let cardIndex = this.state.cards.findIndex((card)=>card.id==cardId);

let newTask = {id:Date.now(), name:taskName, done: false};

let nextState = update(this.state.cards, {

[cardIndex]: {

tasks: {$push: [newTask]}

}

});

this.setState({cards:nextState});

fetch(‘${API\_URL}/cards/${cardId}/tasks’,{

method: ‘post’,

headers: API\_HEADERSM

body: JSON.stringify(newTask)

})

.then((response)=>{

if(response.ok){

return response.json();

}else{

thow new Error(“Server response wasn’t OK”)

}

})

.then((responseData)=>{

newTask.id = responseData.id;

this.setState({cards:nextState});

})

.catch((error)=>{

this.setState(prevState);

})

}

deleteTask(cardId, taskId, taskIndex){

let prevState = this.state;

let cardIndex = this.state.cards.findIndex((card)=>card.id == cardId);

let nextState = update(this.state.cards, {

[cardIndex]: {

tasks: {$splice: [[taskIndex,1]]}

}

});

this.setState({cards:nextState});

fetch(‘${API\_URL}/cards/${cardId}/tasks/${taskId}’,{

method: ‘delete’,

headers: API\_HEADERS

})

.then((response) => {

if(!response.ok){

throw new Error(“Server response wasn’t OK”)

}

})

.catch((error)=>{

console.error(“Fetch error:”, error)

this.setState(prevState);

});

}

toggleTask(cardId, taskId, taskIndex){

let prevState = this.state;

let cardIndex = this.state.cards.findIndex((card)=>card.id == cardId);

let newDoneValue;

let nextState = update(this.state.cards, {

[cardIndex]:{

tasks: {

[taskIndex]:{

done: {

$apply: (done) =>{

newDoneValue = !done;

return newDoneValue;

}

}

}

}

}

});

this.setState({cards:nextState});

fetch(‘${API\_URL}/cards/${cardId}/tasks/${taskId}’,{

method: ‘PUT’,

headers: API\_HEADERS,

body: JSON.stringify({done:newDoneValue})

});

}

updateCardStatus(cardId, listId){

let cardIndex = this.state.cards.findIndex((card)=>card.id == cardId);

let card = this.state.cards[cardIndex]

if(card.status !== listId){

this.setState(update(this.state, {

cards: {

[cardIndex]: {

status: {$set: listId }

}

}

}

);

}

}

updateCardPosition(cardId, afterId){

if(cardId !== afterId){

let cardIndex = this.state.cards.findIndex((card)=>card.id == cardId);

let card = this.state.cards[cardIndex];

let afterIndex = this.state.cards.findIndex((card)=>card.id == afterId);

this.setState(update(

this.state, {

cards: {

$splice: [

[cardIndex, 1],

[afterIndex, 0, card]

]

}

}

));

}

}

persistCardDrag(cardId, status){

let cardIndex = this.state.cards.findIndex((card)=>card.id == cardId);

let card = this.state.cards[cardIndex];

fetch(‘${API\_URL}/cards/${cardId}’, {

method: ‘put’,

headers: API\_HEADERS,

body: JSON.stringify({status: card.status, row\_order\_position: cardIndex})

})

.then((response)=>{

if(!response.ok){

throw new Error(“Server response wasn’t OK”);

}

})

.catch((error)=>{

console.error(“Fetch error:”, error);

this.setState(

update(this.state, {

cards: {

[cardIndex]: {

status: { $set: status }

}

}

})

);

});

}

render(){

return <KanbanBoard cards={this.state.cards}

taskCallbacks={{

toggle: this.toggleTask.bind(this),

delete: this.deleteTask.bind(this),

add: this.addTask.bind(this)

}}

cardCallbacks ={{

updateStatus: this.updateCardStatus,

updatePosition: this.updateCardPosition,

persistCardDrag: this.persistCardDrag.bind(this)

}}

/>

}

}

export default KanbanBoardContainer;

**새로운 카드 위치와 상태 저장**

카드 드래그시 원래 카드 ID와 상태 등록 드래그가 끝나면 서버 호출. 실패시 원래 카드 상태로 저장

import React, { Component, PropTypes } from ‘react’;

import ReactCSSTransitionGroup from ‘react-addons-css-transition-group’;

import marked from ‘marked’;

import CheckList from ’./CheckList’;

import { DragSource, DropTarget } from ‘react-dnd’

import constants from ‘./constants’;

let titlePropType =m(props, propName, componentName) =>{

if(props[propName]){

let value = props[propName];

if(typeof value !== ‘string’ || value.length > 80){

return new Error(

‘${propName} in ${componentName} is longer than 80 characters’

);

}

}

};

const cardDragSpec = {

beginDrag(props){

return {

id: props.id,

status: props.status

}

},

endDrag(props){

props.cardCallbacks.persistCardDrag(props.id, props.status);

}

}

const cardDropSpec = {

hover(props, monitor){

const draggedId = monitor.getItem().id;

props.cardCallbacks.updatePosition(draggedId, props.id);

}

}

let collectDrag = (connect, monitor) => {

return {

connectDragSource: connect.dragSource()

};

}

let collectDrop = (connect, monitor) => {

return {

connectDropTarget: connect.dropTarget(),

};

}

class Card extends Component {

constructor(){

super(…arguments);

this.state = {

showDetails: false

}

}

toggleDetails(){

this.setState({showDetails: !this.state.showDetails});

}

render(){

const { connectDragSource, connectDropTarget } = this.props;

let cardDetails;

if(this.state.showDetails){

cardDetails = (<div className=”card\_\_details”>

<span dangerouslySetInnerHTML={{\_\_html: marked(this.props.description)}}

<CheckList cardId={this.props.id}

tasks={this.props.tasks}

taskCallbacks={this.props.taskCallbacks} />

<div>

);

};

let sideColor = {

position: ‘absolute’,

zIndex: -1,

top: 0,

bottom: 0,

left: 0,

width: 7,

backgroundColor: this.props.color

};

return connectDropTarget(connectDragSource(

<div className=”card”>

<div style={sideColor} />

<div className={this.state.showDetails? ”card\_\_title card\_\_title--is-open” : “card\_\_title”} onClick={this.toggleDetails.bind(this)}>{this.props.title}</div>

<ReactCSSTransitionGroup transitionName=”toggle”

transitionEnterTimeout={250}

transitionLeaveTimeout={250}>

{cardDetails}

</div>

</ReactCSSTransitionGroup>

));

}

}

Card.propTypes = {

id: PropTypes.number,

title: titlePropType,

description: PropTypes.string,

color: PropTypes.string,

tasks: PropTypes.arrayOf(PropTypes.object),

taskCallbacks: PropTypes.object,

cardCallbacks: PropTypes.object,

connectDragSource: PropTypes.func.isRequired,

connectDropTarget: PropTypes.func.isRequired

};

const dragHighOrderCard = DragSource(constants.CARD, cardDragSpec, collectDrag)(Card);

const dragDropHighOrderCard = DropTarget(constants.CARD, cardDropSpec, collectDrop)(dragHighOrderCard)

export default dragDropHighOrderCard;

1. **라우팅**
   1. **라우팅을 구현하는 단순한 방식**

**App.js**

import React, { Component } from ‘react’;

import { render } from ‘react-dom’;

import About from ‘./About’;

import Home from ‘./Home’;

import Repos from ‘./Repos’;

class App extends Component {

constructor(){

super(…arguments);

this.state = {

route: window.location.hash.substr(1)

};

}

componentDidMount(){

window.addEventListener(‘hashchange’, ()=>{

this.setState({

route: window.location.hash.substr(1)

});

});

}

render(){

var Child;

switch(this.state.route){

case ‘/about’: Child = About; break;

case ‘/repos’: Child = Repos; break;

default: Child = Home;

}

return (

<div>

<header>App</header>

<menu>

<ul>

<li><a href=”#/about”>About</a></li>

<li><a href=”#/repos”>Repos</a></li>

</ul>

</menu>

<Child />

</div>

)

}

}

render(<App />, document.getElementById(‘root’));

**Home.js**

import React, { Component } from ‘react’;

class Home extends Component {

render(){

return (

<h1>HOME</h1>

);

}

}

export default Home;

**About.js**

import React, { Component } from ‘react’;

class About extends Component {

render(){

return (

<h1>ABOUT</h1>

);

}

}

export default About;

**Repos.js**

import React, { Component } from ‘react’;

class Repos extends Component {

render(){

return (

<h1>Github Repos</h1>

);

}

}

export default Repos;

**styles.css**

body{

margin: 0;

font: 16px/1 sans-serif

}

menu ul{

margin: 0;

padding: 0;

}

menu li{

display: inline-block;

padding: 5px;

}

1. active{

color: #444;

font-weight: bold;

text-decoration: none;

}

header{

padding: 10px;

background-color: #333;

color: #ccc;

font-size: 20px;

font-weight: bold;

}

menu{

background-color: #ccc;

padding: 5px;

margin-top: 0;

margin-bottom: 10px;

}

* 1. **리액트 라우터**

사용자가 url을 변경하면 자동으로 컴포넌트가 언마운트 및 마운트 된다.

npm install --save [react-router@1.x.x](mailto:react-router@1.x.x) history@1.x.x

리액트 라우터의 구성 요소

* Router, Route:
* Link:

import React, { Component } from ‘react’;

import { render } from ‘react-dom’;

import { Router, Route, Link } from ‘react-router’;

import About from ‘./About’;

import Home from ‘./Home’;

import Repos from ‘./Repos’;

class App extends Component {

render(){

return (

<div>

<header>App</header>

<menu>

<ul>

<li><Link to=”/about”>About</Link></li>

<li><Link to=”/repos”>Repos</Link></li>

</ul>

</menu>

{this.props.children}

</div>

);

}

}

render((

<Router>

<Route path=”/” component={App}>

<Route path=”about” component={About}/>

<Route path=”repos” component={Repos}/>

</Route>

</Router>

),document.getElementById(‘root’));

아래는 라우터에서 두개 이상의 컴포넌트를 받는 법

React.render((

<Router>

<Route path=”/” component={App}>

<Route path=”groups” components={{content: Groups, sidebar: GroupsSidebar}}/>

<Route path=”users” components={{content: Users, sidebar: UsersSidebar}}/>

</Route>

</Router>

),element);

render(){

return (

<div>

{this.props.children.sidebar}-{this.props.children.content}

</div>

);

}

**인덱스 라우트**

import React, { Component } from ‘react’;

import { render } from ‘react-dom’;

import { Router, Route, Link, IndexRoute } from ‘react-router’;

import About from ‘./About’;

import Home from ‘./Home’;

import Repos from ‘./Repos’;

class App extends Component {

render(){

return (

<div>

<header>App</header>

<menu>

<ul>

<li><Link to=”/about”>About</Link></li>

<li><Link to=”/repos”>Repos</Link></li>

</ul>

</menu>

{this.props.children}

</div>

);

}

}

render((

<Router>

<Route path=”/” component={App}>

<IndexRoute component={Home}/>

<Route path=”about” component={About}/>

<Route path=”repos” component={Repos}/>

</Route>

</Router>

),document.getElementById(‘root’));

**매개변수를 이용하는 라우트**

window.fetch사용키 위해 npm install --save whatwg-fetch

import React, { Component } from ‘react’;

import ‘whatwg-fetch’;

import { Link } from ‘react-router’;

class Repos extends Component {

constructor(){

super(…arguments);

this.state = {

repositories: []

}

}

componentDidMount(){

fetch(‘https://api.github.com/users/pro-react/repos’)

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

.then((responseData) => {

this.setState({repositories:responseData});

});

}

render(){

let repos = this.state.repositories.map((repo) => (

<li key={repo.id}>

<Link to={“/repos/details/”+repo.name}>{repo.name}</Link>

</li>

));

return (

<div>

<h1>Github Repos</h1>

<ul>

{repos}

</ul>

{this.props.children}

</div>

);

}

}

export default Repos;

ReposDetails

import React, { Component } from ‘react’;

import ‘whatwg-fetch’;

class RepoDetails extends Component {

constructor(){

super(…arguments);

this.state={

repository:{}

}

}

fetchData(repo\_name){

fetch(‘https://api.github.com/repos/pro-react/’+repo\_name)

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

.then((responseData) => {

this.setState({repository:responseData});

});

}

componentDidMount(){

//라우터가 매개변수 속성에 키 “repo\_name”을 주입

let repo\_name = this.props.params.repo\_name;

this.fetchData(repo\_name);

}

componentWillReceiveProps(nextProps){

let repo\_name = nextProps.params.repo\_name;

this.fetchData(repo\_name);

}

render(){

let stars = [];

for(var i=0;i<this.state.repository.stargazers\_count;i++){

stars.push('★');

}

return (

<div>

<h2>{this.state.repository.name}</h2>

<p>{this.state.repository.description}</p>

<span>{stars}</p>

</div>

);

}

}

export default RepoDetails;

App.js

import React, { Component } from ‘react’;

import { render } from ‘react-dom’;

import { Router, Route, Link, IndexRoute } from ‘react-router’;

import About from ‘./About’;

import Home from ‘./Home’;

import Repos from ‘./Repos’;

import ReposDetails from ‘./ReposDetails’;

class App extends Component {

render(){

return (

<div>

<header>App</header>

<menu>

<ul>

<li><Link to=”/about”>About</Link></li>

<li><Link to=”/repos”>Repos</Link></li>

</ul>

</menu>

{this.props.children}

</div>

);

}

}

render((

<Router>

<Route path=”/” component={App}>

<IndexRoute component={Home}/>

<Route path=”about” component={About}/>

<Route path=”repos” component={Repos}>

{/\* UI를 중첩하려는 위치에 라우트를 중첩해 추가한다. \*/}

<Route path=”details/:repo\_name” component={ReposDetails} />

</Route>

</Route>

</Router>

),document.getElementById(‘root’));

**활성 링크 설정**

App.js

import React, { Component } from ‘react’;

import { render } from ‘react-dom’;

import { Router, Route, Link, IndexRoute } from ‘react-router’;

import About from ‘./About’;

import Home from ‘./Home’;

import Repos from ‘./Repos’;

import ReposDetails from ‘./ReposDetails’;

class App extends Component {

render(){

return (

<div>

<header>App</header>

<menu>

<ul>

<li><Link to=”/about” activeClassName=”active”>About</Link></li>

<li><Link to=”/repos” activeClassName=”active”>Repos</Link></li>

</ul>

</menu>

{this.props.children}

</div>

);

}

}

render((

<Router>

<Route path=”/” component={App}>

<IndexRoute component={Home}/>

<Route path=”about” component={About}/>

<Route path=”repos” component={Repos}>

{/\* UI를 중첩하려는 위치에 라우트를 중첩해 추가한다. \*/}

<Route path=”details/:repo\_name” component={ReposDetails} />

</Route>

</Route>

</Router>

),document.getElementById(‘root’));

**속성 전달하기**

**라우트의 속성**

요소가 활성화될 때 컴포넌트를 렌더링하며 Route에 정의한 속성은 컴포넌트의 속성으로 전달된다.

App.js

import React, { Component } from ‘react’;

import { render } from ‘react-dom’;

import { Router, Route, Link, IndexRoute } from ‘react-router’;

import About from ‘./About’;

import Home from ‘./Home’;

import Repos from ‘./Repos’;

import ReposDetails from ‘./ReposDetails’;

class App extends Component {

render(){

return (

<div>

<header>App</header>

<menu>

<ul>

<li><Link to=”/about” activeClassName=”active”>About</Link></li>

<li><Link to=”/repos” activeClassName=”active”>Repos</Link></li>

</ul>

</menu>

{this.props.children}

</div>

);

}

}

render((

<Router>

<Route path=”/” component={App}>

<IndexRoute component={Home}/>

<Route path=”about” component={About} title=”About Us” />

<Route path=”repos” component={Repos}>

{/\* UI를 중첩하려는 위치에 라우트를 중첩해 추가한다. \*/}

<Route path=”details/:repo\_name” component={ReposDetails} />

</Route>

</Route>

</Router>

),document.getElementById(‘root’));

About.js

import React, { Component } from ‘react’;

class About extends Component {

render(){

return (

<h1>{this.props.route.title}</h1>

);

}

}

export default About;

**자식 복제와 속성 주입**

Repos.js

import React, { Component } from ‘react’;

import ‘whatwg-fetch’;

import { Link } from ‘react-router’;

class Repos extends Component {

constructor(){

super(…arguments);

this.state = {

repositories: []

}

}

componentDidMount(){

fetch(‘https://api.github.com/users/pro-react/repos’)

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

.then((responseData) => {

this.setState({repositories:responseData});

});

}

render(){

let repos = this.state.repositories.map((repo) => (

<li key={repo.id}>

<Link to={“/repos/details/”+repo.name}>{repo.name}</Link>

</li>

));

let child = this.props.children && React.cloneElement(this.props.children, {repositories: this.state.repositories});

return (

<div>

<h1>Github Repos</h1>

<ul>

{repos}

</ul>

{child}

</div>

);

}

}

export default Repos;

npm install --save babel-polyfill

Array.prototype.find 사용키 위해

ReposDetails

import React, { Component } from ‘react’;

import ‘babel-polyfill’;

class RepoDetails extends Component {

constructor(){

super(…arguments);

this.state={

repository:{}

}

}

renderRepository(){

let repository = this.props.repositories.find((repo)=>repo.name === this.props.params.repo\_name);

let stars = [];

for(var i=0;i<repository.stargazers\_count;i++){

stars.push('★');

}

return(

<div>

<h2>{repository.name}</h2>

<p>{repository.description}</p>

<span>{stars}</p>

</div>

);

}

render(){

if(this.props.repositories.length > 0){

return this.renderRepository();

}else{

return <h4>Loading…</h4>;

}

}

}

export default RepoDetails;

**UI와 URL의 연결 분리**

상대경로대신 절대경로로 변경

App.js

import React, { Component } from ‘react’;

import { render } from ‘react-dom’;

import { Router, Route, Link, IndexRoute } from ‘react-router’;

import About from ‘./About’;

import Home from ‘./Home’;

import Repos from ‘./Repos’;

import ReposDetails from ‘./ReposDetails’;

class App extends Component {

render(){

return (

<div>

<header>App</header>

<menu>

<ul>

<li><Link to=”/about” activeClassName=”active”>About</Link></li>

<li><Link to=”/repos” activeClassName=”active”>Repos</Link></li>

</ul>

</menu>

{this.props.children}

</div>

);

}

}

render((

<Router>

<Route path=”/” component={App}>

<IndexRoute component={Home}/>

<Route path=”about” component={About} title=”About Us” />

<Route path=”repos” component={Repos}>

{/\* UI를 중첩하려는 위치에 라우트를 중첩해 추가한다. \*/}

<Route path=”**/**repo/:repo\_name” component={ReposDetails} />

</Route>

</Route>

</Router>

),document.getElementById(‘root’));

Repos.js

import React, { Component } from ‘react’;

import ‘whatwg-fetch’;

import { Link } from ‘react-router’;

class Repos extends Component {

constructor(){

super(…arguments);

this.state = {

repositories: []

}

}

componentDidMount(){

fetch(‘https://api.github.com/users/pro-react/repos’)

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

.then((responseData) => {

this.setState({repositories:responseData});

});

}

render(){

let repos = this.state.repositories.map((repo) => (

<li key={repo.id}>

<Link to={“/repo/”+repo.name}>{repo.name}</Link>

</li>

));

let child = this.props.children && React.cloneElement(this.props.children, {repositories: this.state.repositories});

return (

<div>

<h1>Github Repos</h1>

<ul>

{repos}

</ul>

{child}

</div>

);

}

}

**프로그래밍 방식으로 라우트 변경**

라우터가 모든 컴포넌트에 history 객체 주입

history 객체의 탐색 메서드

|  |  |
| --- | --- |
| 메서드 | 설명 |
| pushState | 새로운 URL로 이동 사용예  history.pushState(null, ‘/users/123’)  history.pushState({showGrades: true}, ‘/users/123’) |
| replaceState | 위와 같은 구문 사용. 리다이렉션과 비슷 |
| goBack |  |
| goForward |  |
| Go | n또는 -n만큼 앞, 뒤로 이동 |
| createHref |  |

ServerError.js

import React, { Component } from ‘react’;

const styles = {

root:{

textAlign:’center’

},

alert:{

fontSize:80,

fontWeight:’bold’,

color:’#e9ab2d’

}

};

class ServerError extends Component {

render(){

return (

<div style={styles.root}>

<div style={styles.alert}>&#9888; </div>

{/\* &#9888;은 경고 기호 ⚠를 나타내며 html 엔터티 코드이다. \*/}

<h1>Ops, we have a problem</h1>

<p>Sorry, we couldn’t access the repositories.

please try again in a few moments.</p>

</div>

);

}

}

export default ServerError;

App.js

import React, { Component } from ‘react’;

import { render } from ‘react-dom’;

import { Router, Route, Link, IndexRoute } from ‘react-router’;

import About from ‘./About’;

import Home from ‘./Home’;

import Repos from ‘./Repos’;

import ReposDetails from ‘./ReposDetails’;

import ServerError from ‘./ServerError’;

class App extends Component {

render(){

return (

<div>

<header>App</header>

<menu>

<ul>

<li><Link to=”/about” activeClassName=”active”>About</Link></li>

<li><Link to=”/repos” activeClassName=”active”>Repos</Link></li>

</ul>

</menu>

{this.props.children}

</div>

);

}

}

render((

<Router>

<Route path=”/” component={App}>

<IndexRoute component={Home}/>

<Route path=”about” component={About} title=”About Us” />

<Route path=”repos” component={Repos}>

{/\* UI를 중첩하려는 위치에 라우트를 중첩해 추가한다. \*/}

<Route path=”/repo/:repo\_name” component={ReposDetails} />

</Route>

<Route path=”error” component={ServerError} />

</Route>

</Router>

),document.getElementById(‘root’));

Repos.js

import React, { Component } from ‘react’;

import ‘whatwg-fetch’;

import { Link } from ‘react-router’;

class Repos extends Component {

constructor(){

super(…arguments);

this.state = {

repositories: []

}

}

componentDidMount(){

fetch(‘https://api.github.com/users/pro-react/repos’)

.then((response) => {

if(response.ok){

return response.json();

}else{

throw new Error(“Server response wasn’t OK”);

}

})

.then((responseData) => {

this.setState({repositories:responseData});

})

.catch(error){

this.props.history.pushState(null, ‘/error’);

});//테스트하려면 이터넷 연결을 끊고 반응은 3차례정도 시도

}

render(){

let repos = this.state.repositories.map((repo) => (

<li key={repo.id}>

<Link to={“/repo/”+repo.name}>{repo.name}</Link>

</li>

));

let child = this.props.children && React.cloneElement(this.props.children, {repositories: this.state.repositories});

return (

<div>

<h1>Github Repos</h1>

<ul>

{repos}

</ul>

{child}

</div>

);

}

}

**히스토리**

리액트 라우터는 히스토리 라이브러리에 기반을 둔다.

#히스토리에 기반

example.com/#/path

브라우저의 history를 리액트 라우터의 history로 설정

example.com/path

App.js

import React, { Component } from ‘react’;

import { render } from ‘react-dom’;

import { Router, Route, Link, IndexRoute } from ‘react-router’;

import createBrowserHistory from ‘history/lib/createBrowserHistory’;

import About from ‘./About’;

import Home from ‘./Home’;

import Repos from ‘./Repos’;

import ReposDetails from ‘./ReposDetails’;

import ServerError from ‘./ServerError’;

class App extends Component {

render(){

return (

<div>

<header>App</header>

<menu>

<ul>

<li><Link to=”/about” activeClassName=”active”>About</Link></li>

<li><Link to=”/repos” activeClassName=”active”>Repos</Link></li>

</ul>

</menu>

{this.props.children}

</div>

);

}

}

render((

<Router history={createBrowserHistory()}>

<Route path=”/” component={App}>

<IndexRoute component={Home}/>

<Route path=”about” component={About} title=”About Us” />

<Route path=”repos” component={Repos}>

{/\* UI를 중첩하려는 위치에 라우트를 중첩해 추가한다. \*/}

<Route path=”/repo/:repo\_name” component={ReposDetails} />

</Route>

<Route path=”error” component={ServerError} />

</Route>

</Router>

),document.getElementById(‘root’));

**칸반 앱: 라우팅**

npm install --save [react-router@1.x.x](mailto:react-router@1.x.x) [history@1.x.x](mailto:history@1.x.x)

**CardForm 컴포넌트**

CardForm Component는 NewCard와 EditCard의 공유 사항을 담고 있는다. NewCard는 /new, EditCard는 /edit/:card\_id

CardForm.js

import React, {Component, PropTypes} from ‘react’;

class CardForm extends Component {

handleChange(field, e){

this.props.handleChange(field, e.target.value);

}

handleClose(e){

e.preventDefault();

this.props.handleClose();

}

render(){

return (

<div>

<div className=”card big”>

<form onSubmit={this.props.handleSubmit.bind(this)}>

<input type=’text’

value={this.props.draftCard.title}

onChange={this.handleChange.bind(this, ‘title’)}

placeholder=”Title”

required={true}

autoFocus={true} />

<textarea value={this.props.draftCard.description}

onChange={this.handleChange.bind(this, ‘description’)}

placeholder=”Description”

required={true}

/>

<label htmlFor=”status”>Status</label>

<select id=”status” value={ this.props.draftCard.status} onChange={this.handleChange.bind(this, ‘status’)}>

<option value=”todo”>To Do</option>

<option value=”in-progress”>In Progress</option>

<option value=”done”>Done</option>

</select>

<br />

<label htmlFor=”color”>Color</label>

<input id=”color’

value={ this.props.draftCard.color}

onChange={this.handleChange.bind(this, ‘color’)}

type=”color”

defaultValue=”#ff0000”/>

<div className=’actions’>

<button type=”submit”>{this.props.buttonLabel</button>

</div>

</form>

</div>

<div className=”overlay” onClick={this.handleClose.bind(this)}>

</div>

</div>

);

}

}

CardForm.propTypes {

buttonLabel: PropTypes.string.isRequired,

draftCard: PropTypes.shape({

title: PropTypes.string,

description: PropTypes.string,

status: PropTypes.string,

color: PropTypes.string

}).isRequired,

handleChange: PropTypes.func.isRequired,

handleSubmit: PropTypes.func.isRequired,

handleClose: PropTypes.func.isRequired

}

export default CardForm;

styles.css

.overlay {

position: absolute;

width: 100%;

height: 100%;

top: 0; left: 0; bottom: 0; right: 0;

z-index: 2;

background-color: rgba(0, 0, 0, 0.6);

}

.card.big {

position: absolute;

width: 450px;

height: 200px;

margin: auto;

padding: 15px;

top: 0; left: 0; bottom: 100px; right: 0;

z-index: 3;

}

.card.big input[type=text], textarea {

width : 100%;

margin: 3px 0;

font-size: 13px;

border: none;

}

.card.big input[type=text] {

font-size: 20px;

font-weight: bold;

}

.card.big input[type=text]:focus,

.card.big textarea:focus {

outline: dashed thin #999;

outline-offset: 2px;

}

.card.big label {

margin: 3px 0 7px 3px;

color: #a7a7a7;

display: inline-block;

width: 60px;

}

.actions {

margin-top: 10px;

text-align: right;

}

.card.big button {

font-size:14px;

padding: 8px;

}

**NewCard와 EditCard 컴포넌트**

NewCard.js

import React, {Component, PropTypes} from ‘react’;

import CardForm from ‘./CardForm’;

class NewCard extends Component {

componentWillMount(){

this.setState({

id: Date.now(),

title:’’,

description:’’,

status:’todo’,

color:’#c9c9c9’,

tasks:[]

});

}

handleChange(field, value){

this.setState({[field]: value});

}

handleSubmit(e){

e.preventDefault();

this.props.cardCallbacks.addCard(this.state);

this.props.history.pushState(null, ‘/’);

}

handleClose(e){

this.props.history.pushState(null, ‘/’);

}

render(){

return (

<CardForm draftCard={this.state}

buttonLabel=”Create Card”

handleChange={this.handleChange.bind(this)}

handleSubmit={this.handleSubmit.bind(this)}

handleClose={this.handleClose.bind(this)}

/>

);

}

}

NewCard.propTypes = {

cardCallbacks: PropTypes.object,

};

export default NewCard;

EditCard.js

import React, {Component, PropTypes} from ‘react’;

import CardForm from ‘./CardForm’;

class EditCard extends Component {

componentWillMount(){

let card = this.props.cards.find((card)=>card.id == this.props.params.card\_id);

this.setState({…card});

}

handleChange(field, value){

this.setState({[field]: value});

}

handleSubmit(e){

e.preventDefault();

this.props.cardCallbacks.updateCard(this.state);

this.props.history.pushState(null, ‘/’);

}

handleClose(e){

this.props.history.pushState(null, ‘/’);

}

render(){

return (

<CardForm draftCard={this.state}

buttonLabel=”Edit Card”

handleChange={this.handleChange.bind(this)}

handleSubmit={this.handleSubmit.bind(this)}

handleClose={this.handleClose.bind(this)}

/>

);

}

}

EditCard.propTypes = {

cardCallbacks: PropTypes.object,

};

export default EditCard;

**라우트 설정**

App.js

import React from ‘react’;

import { render } from ‘react-dom’;

import { Router, Route } from ‘react-router’;

import createBrowserHistory from ‘history/lib/createBrowserHistory’;

import KanbanBoardContainer from ‘./KanbanBoardContainer’;

import KanbanBoard from ‘./KanbanBoard’;

import EditCard from ‘./EditCard’;

import NewCard from ‘./NewCard’;

render((

<Router history={createBrowserHistory()}>

<Route component={KanbanBoardContainer}>

<Route path=”/” component={KanbanBoard}>

<Route path=”new” component={NewCard} />

<Route path=”edit/:card\_id” component={EditCard} />

</Route>

</Route>

</Router>

), document.getElementById(‘root’));

**KanbanBoardContainer에서 콜백 호출 및 자식 렌더링**

KanbanBoardContainer.js

import React, { Component } from ‘react’;

import update from ‘react-addons-update’;

import {throttle} from ‘./utils’;

import KanbanBoard from ‘./KanbanBoard’;

import ‘whatwg-fetch’;

import ‘babel-polyfill’;

const API\_URL = ‘http://kanbanapi.pro-react.com’;

const API\_HEADERS = {

‘Content-Type’: ‘application/json’,

‘Authorization’: ‘any-string-you-like’

};

class KanbanBoardContainer extends Component {

constructor(){

super(…arguments);

this.state = {

cards: [],

};

//인수가 변경된 경우에만 호출

this.updateCardStatus = throttle(this.updateCardStatus.bind(this));

this.updateCardPosition = throttle(this.updateCardPosition.bind(this), 500);

}

componentDidMount(){

fetch(API\_URL+’/cards’, {headers: API\_HEADERS})

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

.then((responseData) => {

this.setState({cards: reponseData});

})

.catch((error) => {

console.log(‘Error fetching and parsing data’, error);

});

}

addTask(cardId, taskName){

let prevState = this.state;

let cardIndex = this.state.cards.findIndex((card)=>card.id==cardId);

let newTask = {id:Date.now(), name:taskName, done: false};

let nextState = update(this.state.cards, {

[cardIndex]: {

tasks: {$push: [newTask]}

}

});

this.setState({cards:nextState});

fetch(‘${API\_URL}/cards/${cardId}/tasks’,{

method: ‘post’,

headers: API\_HEADERSM

body: JSON.stringify(newTask)

})

.then((response)=>{

if(response.ok){

return response.json();

}else{

thow new Error(“Server response wasn’t OK”)

}

})

.then((responseData)=>{

newTask.id = responseData.id;

this.setState({cards:nextState});

})

.catch((error)=>{

this.setState(prevState);

})

}

deleteTask(cardId, taskId, taskIndex){

let prevState = this.state;

let cardIndex = this.state.cards.findIndex((card)=>card.id == cardId);

let nextState = update(this.state.cards, {

[cardIndex]: {

tasks: {$splice: [[taskIndex,1]]}

}

});

this.setState({cards:nextState});

fetch(‘${API\_URL}/cards/${cardId}/tasks/${taskId}’,{

method: ‘delete’,

headers: API\_HEADERS

})

.then((response) => {

if(!response.ok){

throw new Error(“Server response wasn’t OK”)

}

})

.catch((error)=>{

console.error(“Fetch error:”, error)

this.setState(prevState);

});

}

toggleTask(cardId, taskId, taskIndex){

let prevState = this.state;

let cardIndex = this.state.cards.findIndex((card)=>card.id == cardId);

let newDoneValue;

let nextState = update(this.state.cards, {

[cardIndex]:{

tasks: {

[taskIndex]:{

done: {

$apply: (done) =>{

newDoneValue = !done;

return newDoneValue;

}

}

}

}

}

});

this.setState({cards:nextState});

fetch(‘${API\_URL}/cards/${cardId}/tasks/${taskId}’,{

method: ‘PUT’,

headers: API\_HEADERS,

body: JSON.stringify({done:newDoneValue})

});

}

updateCardStatus(cardId, listId){

let cardIndex = this.state.cards.findIndex((card)=>card.id == cardId);

let card = this.state.cards[cardIndex]

if(card.status !== listId){

this.setState(update(this.state, {

cards: {

[cardIndex]: {

status: {$set: listId }

}

}

}

);

}

}

updateCardPosition(cardId, afterId){

if(cardId !== afterId){

let cardIndex = this.state.cards.findIndex((card)=>card.id == cardId);

let card = this.state.cards[cardIndex];

let afterIndex = this.state.cards.findIndex((card)=>card.id == afterId);

this.setState(update(

this.state, {

cards: {

$splice: [

[cardIndex, 1],

[afterIndex, 0, card]

]

}

}

));

}

}

persistCardDrag(cardId, status){

let cardIndex = this.state.cards.findIndex((card)=>card.id == cardId);

let card = this.state.cards[cardIndex];

fetch(‘${API\_URL}/cards/${cardId}’, {

method: ‘put’,

headers: API\_HEADERS,

body: JSON.stringify({status: card.status, row\_order\_position: cardIndex})

})

.then((response)=>{

if(!response.ok){

throw new Error(“Server response wasn’t OK”);

}

})

.catch((error)=>{

console.error(“Fetch error:”, error);

this.setState(

update(this.state, {

cards: {

[cardIndex]: {

status: { $set: status }

}

}

})

);

});

}

addCard(card){

let prevState = this.state;

if(card.id ===null){

let card = Object.assign({}, card, {id:Date.now()});

}

let nextState = update(this.state.cards, { $push: [card] });

this.setState({cards:nextState});

fetch(`${API\_URL}/cards`, {

method: ‘post’,

headers: API\_HEADERS,

body: JSON.stringify(card)

})

.then((response) => {

if(response.ok){

return response.json();

}else{

throw new Error(“Server response wasn’t OK”);

}

})

.then((responseData)=>{

card.id = responseData.id;

this.setState({cards:nextState});

})

.catch((error) => {

this.setState(prevState);

});

}

updateCard(card){

let prevState = this.state;

let cardIndex = this.state.cards.findIndex((c)=>c.id == card.id);

let nextState = update(this.state.cards, { [cardIndex]: {$set: card}

});

this.setState({cards:nextState});

fetch(`${API\_URL}/cards/${card.id}`, {

method: ‘put’,

headers: API\_HEADERS,

body: JSON.stringify(card)

})

.then((response) => {

if(response.ok){

}else{

throw new Error(“Server response wasn’t OK”);

}

})

.catch((error) => {

console.log(“Fetch error:”, error);

this.setState(prevState);

});

}

render(){

let kanbanBoard = this.props.children && React.cloneElement(this.props.children, {

cards: this.state.cards,

taskCallbacks:{

toggle: this.toggleTask.bind(this),

delete: this.deleteTask.bind(this),

add: this.addTask.bind(this)

},

cardCallbacks:{

addCard: this.addCard.bind(this),

updateCard: this.updateCard.bind(this),

updateStatus: this.updateCardStatus,

updatePosition: this.updateCardPosition,

persistCardDrag: this.persistCardDrag.bind(this)

}

});

return kanbanBoard;

}

}

export default KanbanBoardContainer;

1. **플럭스를 이용한 리액트 애플리케이션 설계**
   1. **플럭스란?**

단방향 데이터 지원

**스토어**

애플리케이션의 모든 상태(데이터와 ui 상태 포함)를 유지하며 상태가 변경되면 이벤트를 발송한다.

스토어는 스토어자체만 데이터를 변경할 수 있다.

**액션**

스토어의 데이터를 변경

주로 사용자 액션 발생시 액션 발생, ajax요청이나 타이머 웹소켓 이벤트등에서도 발생, 액션은 타입(고유한 이름)과 페이로드를 포함한다.

**디스패처**

액션을 스토어로 전달

* 1. **실용성이 없는 최소 플럭스 앱**

**은행계좌 애플리케이션**

constants.js - 상수 저장(액션 이름)

AppDispatcher.js 파일

BankActions.js 파일

BankBalanceStore.js

App.js

npm install --save flux

**애플리케이션의 상수**

export default {

CREATED\_ACCOUNT: ‘created account’,

WITHDRAW\_FROM\_ACCOUNT: ‘withdraw from account’,

DEPOSITED\_INTO\_ACCOUNT: ‘deposited into account’

}

**디스패처**

import {Dispatcher} from ‘flux’;

class AppDispatcher extends Dispatcher{

dispatch(action = {}){

console.log(“Dispatched”, action);

super.dispatch(action);

}

}

export default new AppDispatcher();

**액션 생성자**

import AppDispatcher from ‘./AppDispatcher’;

import bankConstants from ‘./constants’;

let BankActions = {

createAccount(){

AppDispatcher.dispatch({

type: bankConstants.CREATED\_ACCOUNT,

amount: 0

});

},

depositIntoAccount(amount){

AppDispatcher.dispatch({

type: bankConstants.DEPOSITED\_INTO\_ACCOUNT,

amount: amount

});

},

withdrawFromAccount(amount){

AppDispatcher.dispatch({

type: bankConstants.WITHDRAW\_FROM\_ACCOUNT,

amount: amount

});

}

};

export default BankActions;

**스토어**

npm install --save fbemitter

import {EventEmitter} from ‘fbemitter’;

import AppDispatcher from ‘./AppDispatcher’;

import bankConstants from ‘./constants’;

const CHANGE\_EVENT = ‘change’;

let \_\_emitter = new EventEmitter();

let balance = 0;

let BankBalanceStore = {

getState(){

return balance;

}

addListener: (callback) => {

return \_\_emitter.addListener(CHANGE\_EVENT, callback);

},

};

//스토어의 업데이트 순서를 조율

BankBalanceStore.dispatchToken = AppDispatcher.register((action)=>{

switch(action.type){

case bankConstants.CREATED\_ACCOUNT:

balance = 0;

\_\_emitter.emit(CHANGE\_EVENT);

break;

case bankConstants.DEPOSITED\_INTO\_ACCOUNT:

balance = balance + action.amount;

\_\_emitter.emit(CHANGE\_EVENT);

break;

case bankConstants.WITHDRAW\_FROM\_ACCOUNT:

balance = balance - action.amount;

\_\_emitter.emit(CHANGE\_EVENT);

break;

}

});

export default BankBalanceStore;

**UI 컴포넌트**

App.js

import React, { Component } from 'react';

import { render } from 'react-dom';

import BankBalanceStore from './BankBalanceStore';

import BankActions from './BankActions';

class App extends Component {

constructor(){

super(…arguments);

BankActions.createAccount();

this.state = {

balance: BankBalanceStore.getState()

}

}

componentDidMount(){

this.storeSubscription = BankBalanceStore.addListener(data => this.handleStoreChange(data));

}

componentWillUnmount(){

this.storeSubscription.remove();

}

handleStoreChange(){

this.setState({balance: BankBalanceStore.getState()});

}

deposit(){

BankActions.depositIntoAccount(Number(this.refs.amount.value));

this.refs.amount.value = ‘’;

}

withdraw(){

BankActions.withdrawFromAccount(Number(this.refs.amount.value));

this.refs.amount.value = ‘’;

}

render(){

return (

<div>

<header>FluxTrust Bank</header>

<h1>Your balance is ${(this.state.balance).toFixed(2)}</h1>

<div className=”atm”>

<input type=”text” placeholder=”Enter Amount” ref=”amount” />

<br />

<button onClick={this.withdraw.bind(this)}>Withdraw</button>

<button onClick={this.deposit.bind(this)}>Deposit</button>

</div>

</div>

);

}

}

render(<App />, document.getElementById(‘root’));

styles.css

body {

margin: 0;

font: 16px/1 sans-serif;

background-color: #318435;

color: #fff;

text-align: center;

}

header{

width:100%;

padding: 15px;

text-align: center;

background-color: #000;

}

h1{

font-size: 18px;

}

h2{

font-size: 16px;

}

.atm{

width: 200px;

height: 100px;

border-radius: 10px;

background-color: #000;

text-align: center;

margin: 10px auto 0 auto;

padding: 20px;

}

.atm input{

font-size:25px;

width: 180px

}

.atm button{

margin: 5px;

padding: 5px;

width: 80px;

}

* 1. **플럭스 유틸**

버전 2.1 이후

* + 1. **플럭스 유틸 스토어**

수동으로 이벤트 방출 필요가 없다. 자동으로 된다

Store 사용

import AppDispatcher from ‘./AppDispatcher’;

import {Store} from ‘flux/utils’;

import bankConstants from ‘./constants’;

let balance = 0;

class BankBalanceStore extends Store {

getState(){

return balance;

}

\_\_onDispatch(action){

switch(action.type){

case bankConstants.CREATED\_ACCOUNT:

balance = 0;

this.\_\_emitChange();

break;

case bankConstants.DEPOSITED\_INTO\_ACCOUNT:

balance = balance + action.amount;

this.\_\_emitChange();

break;

case bankConstants.WITHDRAW\_FROM\_ACCOUNT:

balance = balance - action.amount;

this.\_\_emitChange();

break;

}

}

};

export default new BankBalanceStore(AppDispatcher);

ReduceStore 사용

import AppDispatcher from ‘./AppDispatcher’;

import {ReduceStore} from ‘flux/utils’;

import bankConstants from ‘./constants’;

class BankBalanceStore extends ReduceStore {

getInitailState(){

return 0;

}

reduce(state, action){

switch(action.type){

case bankConstants.CREATED\_ACCOUNT:

return 0;

case bankConstants.DEPOSITED\_INTO\_ACCOUNT:

return state + action.amount;

case bankConstants.WITHDRAW\_FROM\_ACCOUNT:

return state - action.amount;

default:

return state;

}

}

};

export default new BankBalanceStore(AppDispatcher);

* Store: 기본 스토어의 래퍼
* ReduceStore: reduce 함수를 사용하여 자체 상태 수정, 저장 데이터는 단일 기본형 값(문자열, 부울 값, 숫자), 기본형 값의 배열([1,2,3,4],기본형 값의 객체({name:’cassio’, age: 35}, 리액트의 불변성 도우미를 이용해 조작할 중첩 객체를 포함하는 객체
* MapStore: ReduceStore의 변형, 단일값이 아닌 키 값 쌍을 저장하는 추가 도우미 메서드 포함
  + 1. **컨테이너 컴포넌트 고차 함수**

App.js

import React, { Component } from 'react';

import { render } from 'react-dom';

import { Container } from ‘flux/utils’;

import BankBalanceStore from './BankBalanceStore';

import BankActions from './BankActions';

class App extends Component {

constructor(){

super(…arguments);

BankActions.createAccount();

}

deposit(){

BankActions.depositIntoAccount(Number(this.refs.amount.value));

this.refs.amount.value = ‘’;

}

withdraw(){

BankActions.withdrawFromAccount(Number(this.refs.amount.value));

this.refs.amount.value = ‘’;

}

render(){

return (

<div>

<header>FluxTrust Bank</header>

<h1>Your balance is ${(this.state.balance).toFixed(2)}</h1>

<div className=”atm”>

<input type=”text” placeholder=”Enter Amount” ref=”amount” />

<br />

<button onClick={this.withdraw.bind(this)}>Withdraw</button>

<button onClick={this.deposit.bind(this)}>Deposit</button>

</div>

</div>

);

}

}

App.getStores = () => ([BankBalanceStore]);//[BankBalanceStore]반환

App.calculateState = (prevState) => ({balance: BankBalanceStore.getState()});

const AppContainer = Container.create(App);

render(<AppContainer />, document.getElementById(‘root’));

* 1. **비동기 플럭스**
     1. **waitFor: 스토어 업데이트 순서**

여러 스토어가 있을 때 기다려라

BankRewardsStore는 고객 잔고에 따른 고객 등급 유지

import AppDispatcher from ‘./AppDispatcher’;

import BankBalanceStore from ‘./BankBalanceStore’;

import bankConstants from ‘./constants’;

import {ReduceStore} from ‘flux/utils’;

class BankRewardsStore extends ReduceStore {

getInitialState(){

return ‘Basic’;

}

reduce(state, action){

this.getDispatcher().waitFor([

BankBalanceStore.getDispatchToken()

]);

if(action.type === bankConstants.DEPOSITED\_INTO\_ACCOUNT || action.type === bankConstants.WITHDRAW\_FROM\_ACCOUNT ){

let balance = BankBalanceStore.getState();

if(balance < 5000)

return ‘Basic’;

else if (balance < 10000)

return ‘Silver’;

else if (balance < 50000)

return ‘Gold’;

else

return ‘Platinum’;

}

return state;

}

}

export default new BankRewardsStore(AppDispatcher);

App.js

import React, { Component } from 'react';

import { render } from 'react-dom';

import { Container } from ‘flux/utils’;

import BankBalanceStore from './BankBalanceStore';

import BankRewardsStore from ‘./ BankRewardsStore’;

import BankActions from './BankActions';

class App extends Component {

constructor(){

super(…arguments);

BankActions.createAccount();

}

deposit(){

BankActions.depositIntoAccount(Number(this.refs.amount.value));

this.refs.amount.value = ‘’;

}

withdraw(){

BankActions.withdrawFromAccount(Number(this.refs.amount.value));

this.refs.amount.value = ‘’;

}

render(){

return (

<div>

<header>FluxTrust Bank</header>

<h1>Your balance is ${(this.state.balance).toFixed(2)}</h1>

<h2>Your Points Rewards Tier is {this.state.rewardsTier}</h2>

<div className=”atm”>

<input type=”text” placeholder=”Enter Amount” ref=”amount” />

<br />

<button onClick={this.withdraw.bind(this)}>Withdraw</button>

<button onClick={this.deposit.bind(this)}>Deposit</button>

</div>

</div>

);

}

}

App.getStores = () => ([BankBalanceStore, BankRewardsStore]);//[BankBalanceStore]반환

App.calculateState = (prevState) => (

{

balance: BankBalanceStore.getState(),

rewardsTier: BankRewardsStore.getState()

});

const AppContainer = Container.create(App);

render(<AppContainer />, document.getElementById(‘root’));

* + 1. **비동기 데이터 가져오기**

api호출시 스토어에 알리는 액션,

api정상완료시 스토어에 알리는 액션,

api실패시 스토어에 알리는 액션

* 1. **에어치프 애플리케이션**

출발 공항과 도착 공항 입력시 API 호출 후 항공 요금을 가져온다

* + 1. **설정: 프로젝트 구성과 기본 파일**

app

actions

api

components

stores

----

App.js

플럭스 애플리케이션은 항상 디스패처가 하나다

AppDispatcher.js

import { Dispatcher } from ‘flux’;

export default new Dispatcher();

constants.js

export default {

FETCH\_AIRPORT: ‘fetch airports’,//리스트

FETCH\_AIRPORT\_SUCCESS: ‘fetch airports success’,

FETCH\_AIRPORT\_ERROR: ‘fetch airports error’,

CHOOSE\_AIRPORT: ‘choose airport’,

FETCH\_TICKETS: ‘fetch tickets’,//티켓

FETCH\_TICKETS\_SUCCESS: ‘fetch tickets success’,

FETCH\_TICKETS\_ERROR: ‘fetch tickets error’,

}

* + 1. **공항을 가져오는 액션 생성자와 API 도우미 작성**

airports.json

[

{ “code”: “ATL”, “city”: “Atlanta GA”, “country”: “US” },

{ “code”: “LHR”, “city”: “LONDON”, “country”: “GB” },

{ “code”: “JFK”, “city”: “New York NY”, “country”: “US” },

{ “code”: “ORD”, “city”: “Chicago IL”, “country”: “US” },

{ “code”: “HND”, “city”: “Tokyo”, “country”: “JP” },

{ “code”: “LAX”, “city”: “Log Angeles CA”, “country”: “US” },

{ “code”: “CDG”, “city”: “Paris”, “country”: “FR” },

]

api/AirCheapAPI.js

import ‘whatwg-fetch’;

import AirportActionCreators from ‘../actions/ AirportActionCreators’;

let AirCheapAPI = {

fetchAirports(){

fetch(‘airports.json’)

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

.then((responseData) => {

//성공 액션 호출

AirportActionCreators.fetchAirportsSuccess(responseData);

})

.catch((error) => {

//실패 액션 호출

AirportActionCreators.fetchAirportsError(error);

});

}

};

export default AirCheapAPI;

actions/AirportActionCreators.js

import AppDispatcher from ‘../AppDispatcher’;

import constants from ‘../constants’;

import AirCheapAPI from ‘../api/AirCheapAPI’;

let AirportActionCreators = {

fetchAirports(){

AirCheapAPI.fetchAirports();

AppDispatcher.dispatch({

type: constants.FETCH\_AIRPORTS,

});

},

fetchAirportsSuccess(response){

AppDispatcher.dispatch({

type: constants.FETCH\_AIRPORTS\_SUCCESS,

payload: {reponse}

})

},

fetchAirportsError(error){

AppDispatcher.dispatch({

type: constants.FETCH\_AIRPORTS\_ERROR,

payload: {error}

})

},

};

export default AirportActionCreators;

* + 1. **AirportStore**

import AppDispatcher from ‘../AppDispatcher’;

import constants from ‘../constants’;

import {ReduceStore} from ‘flux/utils’;

class AirportStore extends ReduceStore {

getInitialState(){

return [];

}

reduce(state, action){

switch(action.type){

case constants.FETCH\_AIRPORTS\_SUCCESS:

return action.payload.reponse;

default:

return state;

}

}

}

export default new AirportStore(AppDispatcher);

* + 1. **App 컴포넌트**

npm install --save react-auto-suggest

import React, { Component } from ‘react’;

import { render } from ‘react-dom’;

import { Container } from ‘flux/utils’;

import Autosuggest from ‘react-auto-suggest’;

import AirportStore from ‘./stores/AirportStore’;

import AirportActionCreators from ‘./actions/AirportActionCreators’;

class App extends Component {

componentDidMount(){

AirportActionCreators.fetchAirports();

}

getSuggestions(input, callback){

const escapedInput = input.trim().toLowerCase();

const airportMatchRegex = new RegExp(‘\\b’ + escapedInput, ‘i’);

const suggestions = this.state.airports.filter(airport => airportMatchRegex.test(airport.city))

.sort((airport1, airport2) => {

airport1.city.toLowerCase().indexOf(escapedInput) -

airport2.city.toLowerCase().indexOf(escapedInput)

})

.slice(0, 7)

.map(airport => `${airport.city} - ${airport.country} (${airport.code})`);

callback(null, suggestions);

}

render(){

return (

<div>

<header>

<div className=”header-brand”>

<img src=”logo.png” height=”35” />

<p>Check discount ticket prices and pay using your AirCheap points</p>

</div>

<div className=”header-route”>

<Autosuggest id=’origin’

suggestions={this.getSuggestions.bind(this)}

inputAttributes={{placeholder:’From’}} />

<Autosuggest id=’destination’

suggestions={this.getSuggestions.bind(this)}

inputAttributes={{placeholder:’To’}} />

</div>

</header>

</div>

);

}

}

App.getStores = () => ([AirportStore]);

App.calculateState = (prevState) => ({

airports: AirportStore.getState()

});

const AppContainer = Container.create(App);

render(<AppContainer />, document.getElementById(‘root’));

styles.css

\* {

box-sizing: border-box;// 해당 엘리먼트의 패딩과 테두리가 더는 너비를 늘리지 않습니다.

}

body {

margin: 0;

font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;

}

header {

padding-top: 10px;

border-bottom: 1px solid #ccc;

border-top: 4px solid #08516E;

height: 115px;

background-color: #f6f6f6;

}

p {

margin:0;

font-size: 10px;

}

.header-brand {

text-align: center;

}

.header-route {

margin-top: 10px;

margin-left: calc(50% - 205px)

}

.react-autosuggest {

position: relative;

float: left;

margin-right: 5px;

}

.react-autosuggest input {

width: 200px;

height: 30px;

padding: 14px 10px;

font-size: 13px;

border: 1px solid #aaaaaa;

border-radius: 4px;

}

.react-autosuggest input[aria-expanded="true"] {//collapse 를 연 상태로

border-bottom-left-radius: 0;

border-bottom-right-radius: 0;

}

.react-autosuggest input:focus {

outline: none;

}

.react-autosuggest\_\_suggestions {

position: absolute;

top: 29px;

width: 200px;

margin: 0;

padding: 0;

list-style-type: none;

border: 1px solid #aaaaaa;

background-color: #fff;

font-size: 13px;

border-bottom-left-radius: 4px;

border-bottom-right-radius: 4px;

z-index: 2;

}

.react-autosuggest\_\_suggestions-section-suggestions {

margin: 0;

padding: 0;

list-style-type: none;

}

.react-autosuggest\_\_suggestion {

cursor: pointer;

padding: 10px 10px;

}

.react-autosuggest\_\_suggestion--focused {

background-color: #ddd;

}

.ticket {

padding: 20px 10px;

background-color: #fafafa;

margin: 5px;

border: 1px solid #e5e5df;

border-radius: 3px;

box-shadow: 0 1px 0 rgba(0, 0, 0, 0.25);

}

.ticket span {

display: inline-block;

}

.ticket-company {

font-weight: bold;

font-style: italic;

width: 13%;

}

.ticket-location {

text-align: center;

width: 29%;

}

.ticket-separator {

text-align: center;

width: 6%;

}

.ticket-connection {

text-align: center;

width: 10%;

}

.ticket-points {

width: 13%;

text-align: right;

}

* + 1. **에어치프 애플리케이션 완성: 항공권 로딩**

**API 도우미**

flights.json

[

{

“id”: “fc704c16fd79”,

“company”: “US Airlines”,

“points”: 25000,

“duration”: 590,

“segment”:[

{

“duration”: 590,

“departureTime”: “2016-10-10T21:30-03:00”,

“arrivalTime”: “2016-10-11T06:20-04:00”,

“origin”: “GRU”,

“destination”: “JFK”

}

]

},

{

“id”: “3fe21e46fd78”,

“company”: “Dalta”,

“points”: 20000,

“duration”: 862,

“segment”:[

{

“duration”: 635,

“departureTime”: “2016-10-16T20:25-03:00”,

“arrivalTime”: “2016-10-17T06:00-04:00”,

“origin”: “GRU”,

“destination”: “YYZ”,

“connectionDuration”:125

},

{

“duration”: 102,

“departureTime”: “2016-10-17T08:05-04:00”,

“arrivalTime”: “2016-10-17T09:47-04:00”,

“origin”: “YYZ”,

“destination”: “JFK”

}

]

},

]

api/AirCheapAPI.js

import ‘whatwg-fetch’;

import AirportActionCreators from ‘../actions/ AirportActionCreators’;

let AirCheapAPI = {

fetchAirports(){

fetch(‘airports.json’)

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

.then((responseData) => {

//성공 액션 호출

AirportActionCreators.fetchAirportsSuccess(responseData);

});

.catch((error) => {

//실패 액션 호출

AirportActionCreators.fetchAirportsError(error);

})

},

fetchTickets(origin, destination){

fetch(‘flights.json’)

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

.then((responseData) => {

AirportActionCreators.fetchTicketsSuccess(responseData)

})

.catch((error) => {

AirportActionCreators.fetchTicketsError(error)

});

}

};

export default AirCheapAPI;

**액션 생성자**

actions/AirportActionCreators.js

import AppDispatcher from ‘../AppDispatcher’;

import constants from ‘../constants’;

import AirCheapAPI from ‘../api/AirCheapAPI’;

let AirportActionCreators = {

fetchAirports(){

AirCheapAPI.fetchAirports();

AppDispatcher.dispatch({

type: constants.FETCH\_AIRPORTS,

});

},

fetchAirportsSuccess(response){

AppDispatcher.dispatch({

type: constants.FETCH\_AIRPORTS\_SUCCESS,

payload: {reponse}

})

},

fetchAirportsError(error){

AppDispatcher.dispatch({

type: constants.FETCH\_AIRPORTS\_ERROR,

payload: {error}

})

},

chooseAirport(target, code){

AppDispatcher.dispatch({

type: constants.CHOOSE\_AIRPORT,

target,

code

});

},

fetchTickets(){

AirCheapAPI.fetchTickets();

AppDispatcher.dispatch({

type: constants.FETCH\_TICKETS,

})

},

fetchTicketsSuccess(response){

AppDispatcher.dispatch({

type: constants.FETCH\_TICKETS\_SUCCESS,

payload: {response}

});

},

fetchTicketsError(error){

AppDispatcher.dispatch({

type: constants.FETCH\_TICKETS\_ERROR,

payload: {error}

});

},

};

export default AirportActionCreators;

**스토어**

stores/RouteStore.js

import AppDispatcher from ‘../AppDispatcher’;

import constants from ‘../constants’;

import {MapStore} from ‘flux/utils’;

class RouteStore extends MapStore {

reduce(state, action){

switch(action.type){

case constants.CHOOSE\_AIRPORT:

//action.target은 “origin”이나 “destination”일 수 있다

//action.code는 선택된 공항 코드를 포함한다.

return state.set(action.target, action.code);

default:

return state;

}

}

}

export default new RouteStore(AppDispatcher);

stores/TicketStore.js

import AppDispatcher from ‘../AppDispatcher’;

import constants from ‘../constants’;

import {ReduceStore} from ‘flux/utils’;

class TicketStore extends ReduceStore {

getInitialState(){

return [];

}

reduce(state, action){

switch(action.type){

case constants.FETCH\_TICKETS:

return [];//검색시마다 비운다

case constants.FETCH\_TICKETS\_SUCCESS:

return action.payload.response;

default:

return state;

}

}

}

export default new TicketStore(AppDispatcher);

**인터페이스 컴포넌트**

항공권 정보 한행을 표시(속성을 통해 유전)

components/TicketItem.js

import React, { Component, PropTypes } from ‘react’;

const dateConfig = {

weekday: “short”,

year: “numeric”,

month: “short”,

day: “numeric”,

hour: “2-digit”,

minute: “2-digit”

};

class TicketItem extends Component {

render(){

let {ticket} = this.props;

let departureTime = new Date(ticket.segment[0].departureTime).toLocaleDateString(“en-US”, dateConfig);

let arrivalTime = new Date(ticket.segment[ticket.segment.length-1].arrivalTime).toLocaleDateString(“en-US”, dateConfig);

let stops;

if(ticket.segment.length === 2){

stops = ‘1 stop’;

}else if(ticket.segment.length-1 > 1){

stops = ticket.segment.length-1 + ‘ stops’;

}

return (

<div className=’ticket’>

<span className=”ticket-company”>{ticket.company}</span>

<span className=”ticket-location”><strong>{ticket.segment[0].origin}</strong>{‘ ‘}<small>{departureTime}</span>

<span className=”ticket-seperator”>

</span>

<span className=”ticket-loaction”>

<strong>{ticket.segement[ticket.segment.length-1].destination</strong>{‘ ‘}

<small>arrivalTime</small>

</span>

<span className=”ticket-connection”>

{stops}

</span>

<span className=”ticket-points”>

<button>{ticket.points} points</button>

</span>

</div>

);

}

}

TicketItem.propTypes = {

ticket: PropTypes.shape({

id: PropTypes.string,

company: PropTypes.string,

points: PropTypes.number,

duration: PropTypes.number,

segment: PropTypes.array

}),

};

export default TicketItem;

app.js

import React, { Component } from ‘react’;

import { render } from ‘react-dom’;

import { Container } from ‘flux/utils’;

import Autosuggest from ‘react-auto-suggest’;

import AirportStore from ‘./stores/AirportStore’;

import RouteStore from ‘./stores/RouteStore’;

import TicketStore from ‘./stores/TicketStore’;

import TicketItem from ‘./components/TicketItem’;

import AirportActionCreators from ‘./actions/AirportActionCreators’;

class App extends Component {

componentDidMount(){

AirportActionCreators.fetchAirports();

}

getSuggestions(input, callback){

const escapedInput = input.trim().toLowerCase();

const airportMatchRegex = new RegExp(‘\\b’ + escapedInput, ‘i’);

const suggestions = this.state.airports.filter(airport => airportMatchRegex.test(airport.city))

.sort((airport1, airport2) => {

airport1.city.toLowerCase().indexOf(escapedInput) -

airport2.city.toLowerCase().indexOf(escapedInput)

})

.slice(0, 7)

.map(airport => `${airport.city} - ${airport.country} (${airport.code})`);

callback(null, suggestions);

}

handleSelect(target, suggestion, event){

const airportCodeRegex = /\(([^)]+)\)/;

let airportCode = airportCodeRegex.exec(suggestion)[1];

AirportActionCreators.chooseAirport(target, airportCode);

}

componentWillUpdate(nextProps, nextState){

let originAndDestinationSelected = nextState.origin && nextState.destination;

let selectionHasChangedSinceLastUpdate = nextState.origin !== this.state.origin || nextState.destination !== this.state.destination;

if(originAndDestinationSelected && selectionHasChangedSinceLastUpdate){

AirportActionCreators.fetchTickets(nextState.origin,

nextState.destination);

}

}

render(){

let ticketList = this.state.tickets.map((ticket)=>(

<TicketItem key={ticket.id} ticket={ticket} />

));

return (

<div>

<header>

<div className=”header-brand”>

<img src=”logo.png” height=”35” />

<p>Check discount ticket prices and pay using your AirCheap points</p>

</div>

<div className=”header-route”>

<Autosuggest id=’origin’

suggestions={this.getSuggestions.bind(this)}

onSuggestionSelected={this.handleSelect.bind(this, ‘origin’)}

inputAttributes={{placeholder:’From’}} />

<Autosuggest id=’destination’

suggestions={this.getSuggestions.bind(this)}

onSuggestionSelected={this.handleSelect.bind(this, ‘destination’)}

inputAttributes={{placeholder:’To’}} />

</div>

</header>

<div>

{ticketList}

</div>

</div>

);

}

}

App.getStores = () => ([AirportStore, RouteStore, TicketStore]);

App.calculateState = (prevState) => ({

airports: AirportStore.getState(),

origin: RouteStore.get(‘origin’),

destination: RouteStore.get(‘destination’),

tickets: TicketStore.getState()

});

const AppContainer = Container.create(App);

render(<AppContainer />, document.getElementById(‘root’));

* 1. **비동기 데이터를 가져오는 구현 개선하기**
     1. **AppDispatcher의 dispatchAsync 메서드**

npm install --save babel-polyfill

Object.assign이 작동하도록

AppDispatcher.js

import { Dispatcher } from ‘flux’;

import ‘babel-polyfill’;

class AppDispatcher extends Dispatcher{

dispatch(action ={}){

console.log(“Dispatched”, action.type);

super.dispatch(action);

}

dispatchAsync(promise, types, payload){

const { request, success, failure } = types;

this.dispatch({ type: request, payload: Object.assin({}, payload)});

promise.then(

response => this.dispatch({

type: success,

payload: Object.assign({}, payload, { response })

}),

error => this.dispatch({

type: failure,

payload: Object.assign({}, payload, { error })

})

);

}

}

export default new AppDispatcher();

actions/AirportActionCreators.js

import AppDispatcher from ‘../AppDispatcher’;

import constants from ‘../constants’;

import AirCheapAPI from ‘../api/AirCheapAPI’;

let AirportActionCreators = {

fetchAirports(origin, destination){

AppDispatcher.dispatchAsync(

AirCheapAPI.fetchAirports(), {

request: constants.FETCH\_AIRPORTS,

success: constants.FETCH\_AIRPORTS\_SUCCESS,

failure:constants.FETCH\_AIRPORTS\_ERROR

})

},

chooseAirport(target, code){

AppDispatcher.dispatch({

type: constants.CHOOSE\_AIRPORT,

target,

code

});

},

fetchTickets(origin, destination){

AppDispatcher.dispatchAsync(

AirCheapAPI.fetchTickets(origin, destination),

{

request: constants.FETCH\_TICKETS,

success: constants.FETCH\_TICKETS\_SUCCESS,

failure:constants.FETCH\_TICKETS\_ERROR

}

);

}

};

export default AirportActionCreators;

api/AirCheapAPI.js

import ‘whatwg-fetch’;

let AirCheapAPI = {

fetchAirports(){

return fetch(‘airports.json’)

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

},

fetchTickets(origin, destination){

return fetch(‘flights.json’)

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

}

};

export default AirCheapAPI;

* 1. **칸반 앱: 플럭스 아키텍처로 전환**
     1. **리팩터: 플럭스 기본 구조 제작 및 파일 이동**

npm install --save flux

kanbanapp

app

actions

api

components

----

Card.js

----

CardForm.js

----

CheckList.js

----

EditCard.js

----

KanbanBoard.js

----

KanbanBoardContainer.js

----

List.js

----

NewCard.js

stores

----

App.js

----

constants.js

----

utils.js

**임포트 수정**

App.js

import React from 'react';

import { render } from 'react-dom';

import { Router, Route } from 'react-router';

import createBrowserHistory from 'history/lib/createBrowserHistory'

import KanbanBoardContainer from './components/KanbanBoardContainer';

import KanbanBoard from './components/KanbanBoard';

import EditCard from './components/EditCard';

import NewCard from './components/NewCard';

render((

<Router history={createBrowserHistory()}>

<Route component={KanbanBoardContainer}>

<Route path="/" component={KanbanBoard}>

<Route path="new" component={NewCard} />

<Route path="edit/:card\_id" component={EditCard} />

</Route>

</Route>

</Router>

), document.getElementById('root'));

KanbanBoardContainer.js

import React, { Component } from 'react';

import update from 'react-addons-update';

import {throttle} from '../utils';

import KanbanBoard from './KanbanBoard';

// Polyfills

import 'whatwg-fetch';

import 'babel-polyfill';

const API\_URL = 'http://kanbanapi.pro-react.com';

const API\_HEADERS = {

'Content-Type': 'application/json',

/\*

\* Change the Authorization to any string you like. It can be your pet's name,

\* your middle name, your favorite animal, your superpower of choice...

\* An unique authorization will allow you to have your own environment for cards and tasks

\*/

Authorization: 'CHANGE THIS VALUE'

};

class KanbanBoardContainer extends Component {

constructor(){

super(...arguments);

this.state = {

cards:[],

};

this.updateCardStatus = throttle(this.updateCardStatus.bind(this));

this.updateCardPosition = throttle(this.updateCardPosition.bind(this),500);

}

componentDidMount(){

fetch(API\_URL+'/cards', {headers: API\_HEADERS})

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

.then((responseData) => {

this.setState({cards: responseData});

})

.catch((error) => {

console.log('Error fetching and parsing data', error);

});

}

addTask(cardId, taskName){

// Keep a reference to the original state prior to the mutations

// in case we need to revert the optimistic changes in the UI

let prevState = this.state;

// Find the index of the card

let cardIndex = this.state.cards.findIndex((card)=>card.id == cardId);

// Create a new task with the given name and a temporary ID

let newTask = {id:Date.now(), name:taskName, done:false};

// Create a new object and push the new task to the array of tasks

let nextState = update(this.state.cards, {

[cardIndex]: {

tasks: {$push: [newTask] }

}

});

// set the component state to the mutated object

this.setState({cards:nextState});

// Call the API to add the task on the server

fetch(`${API\_URL}/cards/${cardId}/tasks`, {

method: 'post',

headers: API\_HEADERS,

body: JSON.stringify(newTask)

})

.then((response) => {

if(response.ok){

return response.json()

} else {

// Throw an error if server response wasn't 'ok'

// so we can revert back the optimistic changes

// made to the UI.

throw new Error("Server response wasn't OK")

}

})

.then((responseData) => {

// When the server returns the definitive ID

// used for the new Task on the server, update it on React

newTask.id=responseData.id

this.setState({cards:nextState});

})

.catch((error) => {

this.setState(prevState);

});

}

deleteTask(cardId, taskId, taskIndex){

// Keep a reference to the original state prior to the mutations

// in case we need to revert the optimistic changes in the UI

let prevState = this.state;

// Find the index of the card

let cardIndex = this.state.cards.findIndex((card)=>card.id == cardId);

// Create a new object without the task

let nextState = update(this.state.cards, {

[cardIndex]: {

tasks: {$splice: [[taskIndex,1]] }

}

});

// set the component state to the mutated object

this.setState({cards:nextState});

// Call the API to remove the task on the server

fetch(`${API\_URL}/cards/${cardId}/tasks/${taskId}`, {

method: 'delete',

headers: API\_HEADERS

})

.then((response) => {

if(!response.ok){

// Throw an error if server response wasn't 'ok'

// so we can revert back the optimistic changes

// made to the UI.

throw new Error("Server response wasn't OK")

}

})

.catch((error) => {

console.error("Fetch error:",error)

this.setState(prevState);

});

}

toggleTask(cardId, taskId, taskIndex){

// Keep a reference to the original state prior to the mutations

// in case we need to revert the optimistic changes in the UI

let prevState = this.state;

// Find the index of the card

let cardIndex = this.state.cards.findIndex((card)=>card.id == cardId);

// Save a reference to the task's 'done' value

let newDoneValue;

// Using the $apply command, we will change the done value to its opposite

let nextState = update(

this.state.cards, {

[cardIndex]: {

tasks: {

[taskIndex]: {

done: { $apply: (done) => {

newDoneValue = !done

return newDoneValue;

}

}

}

}

}

});

// set the component state to the mutated object

this.setState({cards:nextState});

// Call the API to toggle the task on the server

fetch(`${API\_URL}/cards/${cardId}/tasks/${taskId}`, {

method: 'put',

headers: API\_HEADERS,

body: JSON.stringify({done:newDoneValue})

})

.then((response) => {

if(!response.ok){

// Throw an error if server response wasn't 'ok'

// so we can revert back the optimistic changes

// made to the UI.

throw new Error("Server response wasn't OK")

}

})

.catch((error) => {

console.error("Fetch error:",error)

this.setState(prevState);

});

}

updateCardStatus(cardId, listId) {

// Find the index of the card

let cardIndex = this.state.cards.findIndex((card)=>card.id == cardId);

// Get the current card

let card = this.state.cards[cardIndex]

// Only proceed if hovering over a different list

if(card.status !== listId){

// set the component state to the mutated object

this.setState(update(this.state, {

cards: {

[cardIndex]: {

status: { $set: listId }

}

}

}));

}

}

updateCardPosition(cardId , afterId){

// Only proceed if hovering over a different card

if(cardId !== afterId) {

// Find the index of the card

let cardIndex = this.state.cards.findIndex((card)=>card.id == cardId);

// Get the current card

let card = this.state.cards[cardIndex]

// Find the index of the card the user is hovering over

let afterIndex = this.state.cards.findIndex((card)=>card.id == afterId);

// Use splice to remove the card and reinsert it a the new index

this.setState(update(this.state, {

cards: {

$splice: [

[cardIndex, 1],

[afterIndex, 0, card]

]

}

}));

}

}

persistCardDrag (cardId, status) {

// Find the index of the card

let cardIndex = this.state.cards.findIndex((card)=>card.id == cardId);

// Get the current card

let card = this.state.cards[cardIndex]

fetch(`${API\_URL}/cards/${cardId}`, {

method: 'put',

headers: API\_HEADERS,

body: JSON.stringify({status: card.status, row\_order\_position: cardIndex})

})

.then((response) => {

if(!response.ok){

// Throw an error if server response wasn't 'ok'

// so we can revert back the optimistic changes

// made to the UI.

throw new Error("Server response wasn't OK")

}

})

.catch((error) => {

console.error("Fetch error:",error);

this.setState(

update(this.state, {

cards: {

[cardIndex]: {

status: { $set: status }

}

}

})

);

});

}

addCard(card){

// Keep a reference to the original state prior to the mutations

// in case we need to revert the optimistic changes in the UI

let prevState = this.state;

// Add a temporary ID to the card

if(card.id===null){

let card = Object.assign({}, card, {id:Date.now()});

}

// Create a new object and push the new card to the array of cards

let nextState = update(this.state.cards, { $push: [card] });

// set the component state to the mutated object

this.setState({cards:nextState});

// Call the API to add the card on the server

fetch(`${API\_URL}/cards`, {

method: 'post',

headers: API\_HEADERS,

body: JSON.stringify(card)

})

.then((response) => {

if(response.ok){

return response.json()

} else {

// Throw an error if server response wasn't 'ok'

// so we can revert back the optimistic changes

// made to the UI.

throw new Error("Server response wasn't OK")

}

})

.then((responseData) => {

// When the server returns the definitive ID

// used for the new Card on the server, update it on React

card.id=responseData.id

this.setState({cards:nextState});

})

.catch((error) => {

this.setState(prevState);

});

}

updateCard(card){

// Keep a reference to the original state prior to the mutations

// in case we need to revert the optimistic changes in the UI

let prevState = this.state;

// Find the index of the card

let cardIndex = this.state.cards.findIndex((c)=>c.id == card.id);

// Using the $set command, we will change the whole card

let nextState = update(

this.state.cards, {

[cardIndex]: { $set: card }

});

// set the component state to the mutated object

this.setState({cards:nextState});

// Call the API to update the card on the server

fetch(`${API\_URL}/cards/${card.id}`, {

method: 'put',

headers: API\_HEADERS,

body: JSON.stringify(card)

})

.then((response) => {

if(!response.ok){

// Throw an error if server response wasn't 'ok'

// so we can revert back the optimistic changes

// made to the UI.

throw new Error("Server response wasn't OK")

}

})

.catch((error) => {

console.error("Fetch error:",error)

this.setState(prevState);

});

}

render() {

let kanbanBoard = this.props.children && React.cloneElement(this.props.children, {

cards: this.state.cards,

taskCallbacks:{

toggle: this.toggleTask.bind(this),

delete: this.deleteTask.bind(this),

add: this.addTask.bind(this)

},

cardCallbacks:{

addCard: this.addCard.bind(this),

updateCard: this.updateCard.bind(this),

updateStatus: this.updateCardStatus,

updatePosition: this.updateCardPosition,

persistCardDrag: this.persistCardDrag.bind(this)

}

});

return kanbanBoard;

}

}

export default KanbanBoardContainer;

Card.js

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

import ReactCSSTransitionGroup from 'react-addons-css-transition-group';

import marked from 'marked';

import { DragSource, DropTarget } from 'react-dnd';

import constants from '../constants';

import CheckList from './CheckList';

import {Link} from 'react-router';

let titlePropType = (props, propName, componentName) => {

if (props[propName]) {

let value = props[propName];

if (typeof value !== 'string' || value.length > 80) {

return new Error(

`${propName} in ${componentName} is longer than 80 characters`

);

}

}

};

const cardDragSpec = {

beginDrag(props) {

return {

id: props.id,

status: props.status

};

},

endDrag(props) {

props.cardCallbacks.persistCardDrag(props.id, props.status);

}

}

const cardDropSpec = {

hover(props, monitor) {

const draggedId = monitor.getItem().id;

props.cardCallbacks.updatePosition(draggedId, props.id);

}

}

let collectDrag = (connect, monitor) => {

return {

connectDragSource: connect.dragSource()

};

}

let collectDrop = (connect, monitor) => {

return {

connectDropTarget: connect.dropTarget(),

};

}

class Card extends Component {

constructor() {

super(...arguments);

this.state = {

showDetails: false

};

}

toggleDetails() {

this.setState({showDetails: !this.state.showDetails});

}

render() {

const { connectDragSource, connectDropTarget } = this.props;

let cardDetails;

if (this.state.showDetails) {

cardDetails = (

<div className="card\_\_details">

<span dangerouslySetInnerHTML={{\_\_html:marked(this.props.description)}} />

<CheckList cardId={this.props.id}

tasks={this.props.tasks}

taskCallbacks={this.props.taskCallbacks} />

</div>

);

}

let sideColor = {

position: 'absolute',

zIndex: -1,

top: 0,

bottom: 0,

left: 0,

width: 7,

backgroundColor: this.props.color

};

return connectDropTarget(connectDragSource(

<div className="card">

<div style={sideColor} />

<div className="card\_\_edit"><Link to={'/edit/'+this.props.id}>✎</Link></div>

<div className={

this.state.showDetails? "card\_\_title card\_\_title--is-open" : "card\_\_title"

} onClick={this.toggleDetails.bind(this)}>

{this.props.title}

</div>

<ReactCSSTransitionGroup transitionName="toggle"

transitionEnterTimeout={250}

transitionLeaveTimeout={250}>

{cardDetails}

</ReactCSSTransitionGroup>

</div>

));

}

}

Card.propTypes = {

id: PropTypes.number,

title: titlePropType,

description: PropTypes.string,

color: PropTypes.string,

tasks: PropTypes.arrayOf(PropTypes.object),

taskCallbacks: PropTypes.object,

cardCallbacks: PropTypes.object,

connectDragSource: PropTypes.func.isRequired,

connectDropTarget: PropTypes.func.isRequired

};

const dragHighOrderCard = DragSource(constants.CARD, cardDragSpec, collectDrag)(Card);

const dragDropHighOrderCard = DropTarget(constants.CARD, cardDropSpec, collectDrop)(dragHighOrderCard);

export default dragDropHighOrderCard

List.js

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

import { DropTarget } from 'react-dnd';

import Card from './Card';

import constants from '../constants';

const listTargetSpec = {

hover(props, monitor) {

const draggedId = monitor.getItem().id;

props.cardCallbacks.updateStatus(draggedId, props.id)

}

};

function collect(connect, monitor) {

return {

connectDropTarget: connect.dropTarget()

};

}

class List extends Component {

render() {

const { connectDropTarget } = this.props;

let cards = this.props.cards.map((card) => {

return <Card key={card.id}

cardCallbacks={this.props.cardCallbacks}

taskCallbacks={this.props.taskCallbacks}

{...card} />

});

return connectDropTarget(

<div className="list">

<h1>{this.props.title}</h1>

{cards}

</div>

);

}

};

List.propTypes = {

title: PropTypes.string.isRequired,

cards: PropTypes.arrayOf(PropTypes.object),

cardCallbacks: PropTypes.object,

taskCallbacks: PropTypes.object,

connectDropTarget: PropTypes.func.isRequired

};

export default DropTarget(constants.CARD, listTargetSpec, collect)(List);

**플럭스 기본 파일 추가**

AppDispatcher.js 추가

stores/CardStore.js 추가

actions/CardActionCreators.js와 TaskActionCreators.js 추가

api/KanbanApi.js 추가

AppDispatcher.js

import {Dispatcher} from ‘flux’;

import ‘babel-polyfill’;

class AppDispatcher extends Dispatcher {

dispatchAsync(promise, types, payload){

const { request, success, failure } = types;

this.dispatch(

{type: request, payload: Object.assign({}, payload)}

);

promise.then(

response => this.dispatch({

type: success,

payload: Object.assign({}, payload, { response })

}),

error => this.dispatch({

type: failure,

payload: Object.assign({}, payload, { error })

})

);

}

}

export default new AppDispatcher();

stores/CardStore.js

import AppDispatcher from ‘../AppDispatcher’;

import constants from ‘../constants’;

import {ReduceStore} from ‘flux/utils’;

class CardStore extends ReduceStore {

getInitialState(){

return [];

}

reduce(state, action){

switch(action.type){

default:

return state;

}

}

}

export default new CardStore(AppDispatcher);

actions/CardActionCreators.js

import AppDispatcher from ‘../AppDispatcher’;

import constants from ‘../constants’;

import KanbanAPI from ‘../api/KanbanApi’;

let CardActionsCreators = {};

actions/TaskActionCreators.js

import AppDispatcher from ‘../AppDispatcher’;

import constants from ‘../constants’;

import KanbanAPI from ‘../api/KanbanApi’;

let TaskActionCreators = {};

app/KanbanApi.js

import ‘whatwg-fetch’;

import ‘babel-polyfil’;

let KanbanAPI = {

};

export default KanbanAPI;

* + 1. **플럭스 아키텍쳐로 데이터 가져오기 전환**

**KanbanBoardContainer 편집**

KanbanBoardContainer.js

import React, { Component } from 'react';

import update from 'react-addons-update';

import {throttle} from '../utils';

import KanbanBoard from './KanbanBoard';

import {Container} from ‘flux/utils’;

import CardActionCreators from ‘../actions/CardActionCreators’;

import CardStore from ‘../stores/CardStore’;

// Polyfills

import 'whatwg-fetch';

import 'babel-polyfill';

const API\_URL = 'http://kanbanapi.pro-react.com';

const API\_HEADERS = {

'Content-Type': 'application/json',

/\*

\* Change the Authorization to any string you like. It can be your pet's name,

\* your middle name, your favorite animal, your superpower of choice...

\* An unique authorization will allow you to have your own environment for cards and tasks

\*/

Authorization: 'CHANGE THIS VALUE'

};

class KanbanBoardContainer extends Component {

constructor(){

super(...arguments);

this.updateCardStatus = throttle(this.updateCardStatus.bind(this));

this.updateCardPosition = throttle(this.updateCardPosition.bind(this),500);

}

componentDidMount(){

CardActionCreators.fetchCards();

}

addTask(cardId, taskName){

// Keep a reference to the original state prior to the mutations

// in case we need to revert the optimistic changes in the UI

let prevState = this.state;

// Find the index of the card

let cardIndex = this.state.cards.findIndex((card)=>card.id == cardId);

// Create a new task with the given name and a temporary ID

let newTask = {id:Date.now(), name:taskName, done:false};

// Create a new object and push the new task to the array of tasks

let nextState = update(this.state.cards, {

[cardIndex]: {

tasks: {$push: [newTask] }

}

});

// set the component state to the mutated object

this.setState({cards:nextState});

// Call the API to add the task on the server

fetch(`${API\_URL}/cards/${cardId}/tasks`, {

method: 'post',

headers: API\_HEADERS,

body: JSON.stringify(newTask)

})

.then((response) => {

if(response.ok){

return response.json()

} else {

// Throw an error if server response wasn't 'ok'

// so we can revert back the optimistic changes

// made to the UI.

throw new Error("Server response wasn't OK")

}

})

.then((responseData) => {

// When the server returns the definitive ID

// used for the new Task on the server, update it on React

newTask.id=responseData.id

this.setState({cards:nextState});

})

.catch((error) => {

this.setState(prevState);

});

}

deleteTask(cardId, taskId, taskIndex){

// Keep a reference to the original state prior to the mutations

// in case we need to revert the optimistic changes in the UI

let prevState = this.state;

// Find the index of the card

let cardIndex = this.state.cards.findIndex((card)=>card.id == cardId);

// Create a new object without the task

let nextState = update(this.state.cards, {

[cardIndex]: {

tasks: {$splice: [[taskIndex,1]] }

}

});

// set the component state to the mutated object

this.setState({cards:nextState});

// Call the API to remove the task on the server

fetch(`${API\_URL}/cards/${cardId}/tasks/${taskId}`, {

method: 'delete',

headers: API\_HEADERS

})

.then((response) => {

if(!response.ok){

// Throw an error if server response wasn't 'ok'

// so we can revert back the optimistic changes

// made to the UI.

throw new Error("Server response wasn't OK")

}

})

.catch((error) => {

console.error("Fetch error:",error)

this.setState(prevState);

});

}

toggleTask(cardId, taskId, taskIndex){

// Keep a reference to the original state prior to the mutations

// in case we need to revert the optimistic changes in the UI

let prevState = this.state;

// Find the index of the card

let cardIndex = this.state.cards.findIndex((card)=>card.id == cardId);

// Save a reference to the task's 'done' value

let newDoneValue;

// Using the $apply command, we will change the done value to its opposite

let nextState = update(

this.state.cards, {

[cardIndex]: {

tasks: {

[taskIndex]: {

done: { $apply: (done) => {

newDoneValue = !done

return newDoneValue;

}

}

}

}

}

});

// set the component state to the mutated object

this.setState({cards:nextState});

// Call the API to toggle the task on the server

fetch(`${API\_URL}/cards/${cardId}/tasks/${taskId}`, {

method: 'put',

headers: API\_HEADERS,

body: JSON.stringify({done:newDoneValue})

})

.then((response) => {

if(!response.ok){

// Throw an error if server response wasn't 'ok'

// so we can revert back the optimistic changes

// made to the UI.

throw new Error("Server response wasn't OK")

}

})

.catch((error) => {

console.error("Fetch error:",error)

this.setState(prevState);

});

}

updateCardStatus(cardId, listId) {

// Find the index of the card

let cardIndex = this.state.cards.findIndex((card)=>card.id == cardId);

// Get the current card

let card = this.state.cards[cardIndex]

// Only proceed if hovering over a different list

if(card.status !== listId){

// set the component state to the mutated object

this.setState(update(this.state, {

cards: {

[cardIndex]: {

status: { $set: listId }

}

}

}));

}

}

updateCardPosition(cardId , afterId){

// Only proceed if hovering over a different card

if(cardId !== afterId) {

// Find the index of the card

let cardIndex = this.state.cards.findIndex((card)=>card.id == cardId);

// Get the current card

let card = this.state.cards[cardIndex]

// Find the index of the card the user is hovering over

let afterIndex = this.state.cards.findIndex((card)=>card.id == afterId);

// Use splice to remove the card and reinsert it a the new index

this.setState(update(this.state, {

cards: {

$splice: [

[cardIndex, 1],

[afterIndex, 0, card]

]

}

}));

}

}

persistCardDrag (cardId, status) {

// Find the index of the card

let cardIndex = this.state.cards.findIndex((card)=>card.id == cardId);

// Get the current card

let card = this.state.cards[cardIndex]

fetch(`${API\_URL}/cards/${cardId}`, {

method: 'put',

headers: API\_HEADERS,

body: JSON.stringify({status: card.status, row\_order\_position: cardIndex})

})

.then((response) => {

if(!response.ok){

// Throw an error if server response wasn't 'ok'

// so we can revert back the optimistic changes

// made to the UI.

throw new Error("Server response wasn't OK")

}

})

.catch((error) => {

console.error("Fetch error:",error);

this.setState(

update(this.state, {

cards: {

[cardIndex]: {

status: { $set: status }

}

}

})

);

});

}

addCard(card){

// Keep a reference to the original state prior to the mutations

// in case we need to revert the optimistic changes in the UI

let prevState = this.state;

// Add a temporary ID to the card

if(card.id===null){

let card = Object.assign({}, card, {id:Date.now()});

}

// Create a new object and push the new card to the array of cards

let nextState = update(this.state.cards, { $push: [card] });

// set the component state to the mutated object

this.setState({cards:nextState});

// Call the API to add the card on the server

fetch(`${API\_URL}/cards`, {

method: 'post',

headers: API\_HEADERS,

body: JSON.stringify(card)

})

.then((response) => {

if(response.ok){

return response.json()

} else {

// Throw an error if server response wasn't 'ok'

// so we can revert back the optimistic changes

// made to the UI.

throw new Error("Server response wasn't OK")

}

})

.then((responseData) => {

// When the server returns the definitive ID

// used for the new Card on the server, update it on React

card.id=responseData.id

this.setState({cards:nextState});

})

.catch((error) => {

this.setState(prevState);

});

}

updateCard(card){

// Keep a reference to the original state prior to the mutations

// in case we need to revert the optimistic changes in the UI

let prevState = this.state;

// Find the index of the card

let cardIndex = this.state.cards.findIndex((c)=>c.id == card.id);

// Using the $set command, we will change the whole card

let nextState = update(

this.state.cards, {

[cardIndex]: { $set: card }

});

// set the component state to the mutated object

this.setState({cards:nextState});

// Call the API to update the card on the server

fetch(`${API\_URL}/cards/${card.id}`, {

method: 'put',

headers: API\_HEADERS,

body: JSON.stringify(card)

})

.then((response) => {

if(!response.ok){

// Throw an error if server response wasn't 'ok'

// so we can revert back the optimistic changes

// made to the UI.

throw new Error("Server response wasn't OK")

}

})

.catch((error) => {

console.error("Fetch error:",error)

this.setState(prevState);

});

}

render() {

let kanbanBoard = this.props.children && React.cloneElement(this.props.children, {

cards: this.state.cards,

taskCallbacks:{

toggle: this.toggleTask.bind(this),

delete: this.deleteTask.bind(this),

add: this.addTask.bind(this)

},

cardCallbacks:{

addCard: this.addCard.bind(this),

updateCard: this.updateCard.bind(this),

updateStatus: this.updateCardStatus,

updatePosition: this.updateCardPosition,

persistCardDrag: this.persistCardDrag.bind(this)

}

});

return kanbanBoard;

}

}

KanbanBoardContainer.getStores = () => ([CardStore]);

KanbanBoardContainer.calculateState = (prevState) => ({

cards: CardStore.getState()

});

export default Container.create(KanbanBoardContainer);

* + 1. **FetchCards 액션, API 메서드 호출, 스토어 콜백 구현**

**카드 가져오기 상수와 액션 생성자**

constants.js

export default {

CARD: ‘card’,

FETCH\_CARDS: ‘fetch cards’,

FETCH\_CARDS\_SUCCESS: ‘fetch cards success’,

FETCH\_CARDS\_ERROR: ‘fetch cards error’

};

actions/CardActionCreators.js

import AppDispatcher from ‘../AppDispatcher’;

import constants from ‘../constants’;

import KanbanAPI from ‘../api/KanbanApi’;

let CardActionsCreators = {

fetchCards(){

AppDispatcher.dispatchAsync(KanbanAPI.fetchCards(), {

request: constants.FETCH\_CARDS,

success: constants.FETCH\_CARDS\_SUCCESS,

failure: constants.FETCH\_CARDS\_ERROR,

});

}

};

export default CardActionCreators;

**fetchCards API 메서드**

app/KanbanApi.js

import ‘whatwg-fetch’;

import ‘babel-polyfil’;

const API\_URL = ‘http://kanbanapi.pro-react.com’;

const API\_HEADERS = {

‘Content-Type’: ‘application/json’,

Authorization: ‘any-string-you-like’

}

let KanbanAPI = {

fetchCards(){

return fetch(`${API\_URL}/cards`, {headers:API\_HEADERS})

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

}

};

export default KanbanAPI;

**CardStore: FETCH\_CARDS\_SUCCESS에 반응**

stores/CardStore.js

import AppDispatcher from ‘../AppDispatcher’;

import constants from ‘../constants’;

import {ReduceStore} from ‘flux/utils’;

class CardStore extends ReduceStore {

getInitialState(){

return [];

}

reduce(state, action){

switch(action.type){

case constants.FETCH\_CARDS\_SUCCESS:

return action.payload.response;

default:

return state;

}

}

}

export default new CardStore(AppDispatcher);

* + 1. **모든 카드와 태스크 조작을 플럭스 아키텍처로 이식하기**

KanbanBoardContainer 컴포넌트에 남아 있는 데이터 조작 메서드

|  |  |
| --- | --- |
| 메서드 | 설명 |
| addCard | 카드 속성을 포함하는 객체를 매개변수로 받고 새로운 카드를 생성한다. |
| updateCard | 업데이트된 카드 속성을 포함하는 객체를 매개변수로 받고 지정된 카드의 속성을 업데이트한다. 리팩터링 후에는 원래 카드 속성과 변경된 카드 속성 의 두 속성을 받느다. |
| updateCardPosition | 현재 카드 ID와 위치를 바꿀 카드 ID를 받는다. 카드 드래그 앤드 드랍 작업 중에 호출되며 지정된 카드의 위치를 바꾼다. |
| updateCardStatus | 현재 카드 ID와 새로운 상태 ID를 받는다. 카드 드래그 앤드 드롭 작업 중에 호출되며 카드 상태를 업데이트한다. |
| persistCardDrag | 지정한 카드 ID와 새로운 카드 상태를 포함하는 객체를 받는다. 카드 드래그 앤드 드롭 작업 후에 호출되며 새 카드의 위치와 상태를 서버에 저장한다. |
| addTask | 카드 ID와 태스크 이름을 받고 지정한 카드에 새로운 태스크를 생성한다. 리팩터링 후에는 태스크 이름이 아니라 태스크 객체 하나를 전달한다. |
| deleteTask | 카드ID, 태스크ID,태스크 인덱스를 받고 태스트를 삭제한다. 리팩터링 후에는 ID가 아니라 해당 카드 객체를 전달한다. |
| toggleTask | 카드ID, 태스크ID,태스크 인덱스를 받고 태스트의 “done”속성을 토글한다.. 리팩터링 후에는 ID가 아니라 해당 카드 객체를 전달한다. |

* + 1. **기능 이식을 위한 준비**

constants.js

export default {

CARD: ‘card’,

FETCH\_CARDS: ‘fetch cards’,

FETCH\_CARDS\_SUCCESS: ‘fetch cards success’,

FETCH\_CARDS\_ERROR: ‘fetch cards error’,

CREATE\_CARD: ‘create card’,

CREATE\_CARD\_SUCCESS: ‘create card success’,

CREATE\_CARD\_ERROR: ‘create card error’,

UPDATE\_CARD: ‘update card’,

UPDATE\_CARD\_SUCCESS: ‘update card success’,

UPDATE\_CARD\_ERROR: ‘update card error’,

UPDATE\_CARD\_STATUS: ‘update card status’,

UPDATE\_CARD\_POSITION: ‘update card position’,

PERSIST\_CARD\_DRAG: ‘persist card drag’,

PERSIST\_CARD\_DRAG\_SUCCESS: ‘persist card drag success’,

PERSIST\_CARD\_DRAG\_ERROR: ‘persist card drag error’,

CREATE\_TASK: ‘create task’,

CREATE\_TASK\_SUCCESS: ‘create task success’,

CREATE\_TASK\_ERROR: ‘create task error’,

DELETE\_TASK: ‘delete task’,

DELETE\_TASK\_SUCCESS: ‘delete task success’,

DELETE\_TASK\_ERROR: ‘delete task error’,

TOGGLE\_TASK: ‘toggle task’,

TOGGLE\_TASK\_SUCCESS: ‘toggle task success’,

TOGGLE\_TASK\_ERROR: ‘toggle task error’,

};

**액션 생성자**

actions/CardActionCreators.js

import AppDispatcher from ‘../AppDispatcher’;

import constants from ‘../constants’;

import KanbanAPI from ‘../api/KanbanApi’;

import {throttle} from ‘../utils’;

import CardStore ‘../stores/CardStore’;

let CardActionsCreators = {

fetchCards(){

AppDispatcher.dispatchAsync(KanbanAPI.fetchCards(), {

request: constants.FETCH\_CARDS,

success: constants.FETCH\_CARDS\_SUCCESS,

failure: constants.FETCH\_CARDS\_ERROR,

});

},

addCard(card){

AppDispatcher.dispatchAsync(KanbanAPI.addCard(card),{

request: constants.CREATE\_CARD,

success: constants.CREATE\_CARD\_SUCCESS,

failure: constants.CREATE\_CARD\_ERROR

}, {card});

},

updateCard(card, draftCard){

AppDispatcher.dispatchAsync(KanbanAPI.updateCard(card, draftCard),{

request: constants.UPDATE\_CARD,

success: constants.UPDATE\_CARD\_SUCCESS,

failure: constants.UPDATE\_CARD\_ERROR

}, {card, draftCard});

},

updateCardStatus: throttle((cardId, listId) => {

AppDispatcher.dispatch({

type: constants.UPDATE\_CARD\_STATUS,

payload: {cardId, listId}

});

}),

updateCardPosition: throttle((cardId, afterId) => {

AppDispatcher.dispatch({

type: constants.UPDATE\_CARD\_POSITION,

payload: {cardId, afterId}

});

}, 500),

persistCardDrag(cardProps){

let card = CardStore.getCard(cardProps.id)

let cardIndex = CardStore.getCardIndex(cardProps.id)

AppDispatcher.dispatchAsync(KanbanAPI.persistCardDrag(card.id, card.status, cardIndex),{

request: constants.PERSIST\_CARD\_DRAG,

success: constants.PERSIST\_CARD\_DRAG\_SUCCESS,

failure: constants.PERSIST\_CARD\_DRAG\_ERROR

}, {cardProps});

},

};

export default CardActionCreators;

TaskActionCreators.js

import AppDispatcher from ‘../AppDispatcher’;

import constants from ‘../constants’;

import KanbanAPI from ‘../api/KanbanApi’;

let TaskActionCreators = {

addTask(cardId, task){

AppDispatcher.dispatchAsync(KanbanAPI.addTask(cardId, task),{

request: constants.CREATE\_TASK,

success: constants.CREATE\_TASK\_SUCCESS,

failure: constants.CREATE\_TASK\_ERROR

}, {cardId, task});

},

deleteTask(cardId, task, taskIndex){

AppDispatcher.dispatchAsync(KanbanAPI.deleteTask(cardId, task),{

request: constants.DELETE\_TASK,

success: constants.DELETE\_TASK\_SUCCESS,

failure: constants.DELETE\_TASK\_ERROR

}, {cardId, task, taskIndex});

},

toggleTask(cardId, task, taskIndex){

AppDispatcher.dispatchAsync(KanbanAPI.toggleTask(cardId, task),{

request: constants.TOGGLE\_TASK,

success: constants.TOGGLE\_TASK\_SUCCESS,

failure: constants.TOGGLE\_TASK\_ERROR

}, {cardId, task, taskIndex});

},

};

export default TaskActionCreators;

* + 1. **컴포넌트**