2020-2

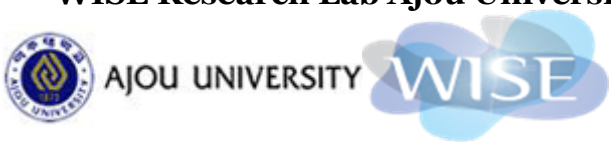
# 웹 시스템 설계

# 학습과제

*Week 13*

React.js

**Composed by:**
**WISE Research Lab Ajou University**



## 학습목표

1. React 의 Redux 활용하여 웹 어플리케이션을 구축할 수 있다.

# React.JS

## 1. [1] Redux 란

- Redux는 'actions'이라는 이벤트를 사용하여 애플리케이션 상태를 관리하고 업데이트하기 위한 패턴 및 라이브러리이다.
  - ✓ 전체 애플리케이션에서 사용해야 하는 상태에 대한 중앙 저장소 역할을 하며, 상태를 업데이트하는 방식을 일관되게 만든다.
- Redux는 애플리케이션의 여러 부분에서 필요한 상태인 '전역' 상태를 관리하는 데 도움이 된다.
- Redux에서 제공하는 패턴과 라이브러리를 사용하면 애플리케이션의 상태 변경이 발생할 때 애플리케이션 로직이 어떻게 작동하는지 쉽게 이해할 수 있다.
- Redux는 예측 가능하고 테스트 가능한 코드를 작성하도록 안내한다.
  - ✓ 애플리케이션이 예상대로 작동하게 해준다.

## 1.1 [2]Redux Actions

- Redux Action은 type field 가 있는 일반 JavaScript 객체이다.
  - ✓ type field는 이 작업에 설명이 포함된 이름을 제공하는 문자열이어야 한다.
  - ✓ type field 외 다른 field(예: payload)를 가질 수 있다.
- Redux Action은 애플리케이션에서 발생한 일을 설명하는 이벤트로 생각할 수 있다.
- 아래의 예시는 Redux Action의 예시이다.

```
const addTodoAction = {
  type: 'todos/todoAdded',
  payload: 'Buy milk'
}
```

[1] https://redux.js.org/

[2] https://redux.js.org/tutorials/fundamentals/part-2-concepts-data-flow#actions

## 1.2 [3]Redux Reducers

- Redux Reducers는 현재 상태와 Acton 객체를 받아 상태를 업데이트하는 방법을 결정하는 함수이다.
  - ✓ Redux Reducers는 수신된 Action (이벤트) 유형을 기반으로 이벤트를 처리하는 이벤트 리스너로 생각할 수 있다.
- Redux Reducers 사용시 다음 규칙들을 지켜야 한다.
  - ✓ 상태 및 작업 인수를 기반으로 새 상태 값만 계산해야 한다.
  - ✓ 기존 상태를 수정할 수 없다.
    - ◆ 대신 기존 상태를 복사하고 복사된 값을 변경하여 변경 불가능한 업데이트를 수행해야한다.
  - ✓ 비동기 논리를 수행하거나 임의의 값을 계산하지 않아야 한다.
- 아래의 예시는 Redux Reducers의 예시이다.

```
const initialState = { value: 0 }

function counterReducer(state = initialState, action) {

  // Check to see if the reducer cares about this action
  if (action.type === 'counter/incremented') {
    // If so, make a copy of `state`
    return {
      ...state,
      // and update the copy with the new value
      value: state.value + 1
    }
  }

  // otherwise return the existing state unchanged
  return state
}
```

[3] https://redux.js.org/tutorials/fundamentals/part-2-concepts-data-flow#reducers

## 1.3 [4]Redux Store

- Redux Store 객체는 현재 애플리케이션의 상태를 가진다.
- Redux Store는 Reducer를 전달하여 생성된다.
- 현재 상태 값을 반환하는 'getState' 메소드가 있다.
- 아래의 예시는 Redux store의 예시이다.

```
import { configureStore } from '@reduxjs/toolkit'

const store = configureStore({ reducer: counterReducer })

console.log(store.getState())
// {value: 0}
```

## 1.4 [5]Redux Dispatch

- Redux Dispatch는 Redux Store 객체의 메소드이다.
- State를 업데이트하는 유일한 방법은 Dispatch 메소드를 호출하고 액션 객체를 전달하는 것입니다.
  - ✓ Store 객체는 Reducer 함수를 실행하고 내부에 새 상태 값을 저장한다.
  - ✓ 'getState'를 호출하여 업데이트 된 값을 검색할 수 있다.
- Action을 Dispatch하는 것은 애플리케이션에서 '이벤트 트리거'로 생각할 수 있다.
- 아래의 예시는 Redux Dispatch의 예시이다.

```
store.dispatch({ type: 'counter/incremented' })

console.log(store.getState())
// {value: 1}
```

[4] https://redux.js.org/tutorials/fundamentals/part-2-concepts-data-flow#store

[5] https://redux.js.org/tutorials/fundamentals/part-2-concepts-data-flow#dispatch

## 1.5 [6]Redux Selectors

- Redux Selectors는 Redux Store의 state 값을 추출하는 함수이다.
- 애플리케이션이 커짐에 따라 앱의 다른 부분이 동일한 데이터를 읽어야하기 때문에 반복되는 논리를 방지할 수 있다.
- 아래의 예시는 Redux Selectors의 예시이다.

```
const selectCounterValue = state => state.value

const currentValue = selectCounterValue(store.getState())

console.log(currentValue)
// 2
```

## 1.6 [7]Connected Components

- connect 함수는 React 컴포넌트를 Redux Store에 연결한다.
- 연결된 컴포넌트에 Redux Store에서 필요한 데이터와 Redux Store에 Dispatch를 전달하는 데 사용할 수 있는 기능을 제공한다.
- 아래의 예시는 connect 함수의 예시이다.

```
import { connect } from 'react-redux'

const App = props => {
  return <div>{props.user}</div>
}

const mapStateToProps = state => {
  return state
}

export default connect(mapStateToProps)(App)
```

- 아래의 명령어를 실행하여 React Redux 프로젝트를 생성할 수 있다.

```
npx create-react-app my-redux-app --template redux
```

---

[6] https://redux.js.org/tutorials/fundamentals/part-2-concepts-data-flow#selectors

[7] https://react-redux.js.org/api/connect

# 학습과제

## 1. 과제 개요

- 추가, 완료, 삭제가 가능한 TodoList 를 구현한다
- Redux 를 통해 TodoList 를 관리해야 하며, 전체 Todo 수와 완료된 Todo 수를 출력할 수 있어야 한다.

※ **Hint:** redux.js.org의 [8]**todo example**을 참고하여 학습과제를 준비할 수 있습니다.

## 2. 학습과제 파일 구성 및 과제 제출 양식

- 학습과제는 'npx create-react-app [프로젝트 이름]' 명령어로 통해 만들어진 스켈레톤 프로젝트 구성되며, 추가되는 파일 구성은 아래와 같다.

| 상위 폴더 명 | 추가 파일 이름 |
|---|---|
| **actions** | todo.js |
| **reducers** | index.js |
| | todo.js |
| **components** | Form.js |
| | Header.js |
| | List.js |
| | TodoList.js |

## 3. 제출방법 및 주의사항

- 압축 파일의 이름을 "Lab13_자신의학번.zip"으로 지정하여 폴더(프로젝트)를 압축한다.
- 설치된 node package 들을 제외한 프로젝트 폴더를 압축해서 제출
  - ✓ 즉, node_module 폴더는 압축파일에 포함시키지 않는다.
- 압축된 프로젝트 폴더와 함께, 제출한 학습보고서 1~4 을 작성하여 함께 제출한다.

[8] https://redux.js.org/introduction/examples#todos

- 지각 제출 시, 0 점으로 처리한다.
- 채점 시 완성도의 평가는 Google 의 Chrome 브라우저에서 렌더링 된 화면을 기준으로 함

※ **"4. 프로그램 설계", "5. 실행 예시"**를 참고하여 프로그램이 올바르게 동작하도록 구현한다.

## 4. 프로그램 설계

### 4.1 React Components

#### 4.1.1 Header.js

- Header 클래스 컴포넌트를 아래와 같이 작성한다.
- render 함수는 다음의 엘리먼트를 반환한다.
  - ✓ <h1>: Lab13_자신의 학번

#### 4.1.2 Form.js

- Form 클래스 컴포넌트는 사용자의 Todo를 입력 받아 추가하는 Component이다.
- Form 클래스 컴포넌트를 아래와 같이 작성한다.
- render 함수는 다음의 엘리먼트를 반환한다.
  - ✓ <input>: 값을 입력 받는 역할을 한다.
  - ✓ <button>: 입력 받은 값을 TodoList 에 추가하는 역할을 한다.
    - ◆ 버튼 클릭 시: Dispatch 메소드를 호출하여 creatTodo Action을 전달한다.

#### 4.1.3 List.js

- Todo들의 List 역할을 하는 Component이다.
- List 클래스 컴포넌트를 아래와 같이 작성한다.
- render 함수는 다음의 엘리먼트를 반환한다.
  - ✓ <ul>: 추가된 Todo들을 가지는 리스트
  - ✓ <li>: Todo

- ◆ : 추가된 Todo의 text를 가진다.
- ◆ <button>: 완료 버튼
  - 버튼 클릭 시: Dispatch 메소드를 호출하여 toggleTodo Action을 전달한다.
- ◆ <button>: 삭제 버튼
  - 버튼 클릭 시: Dispatch 메소드를 호출하여 deleteTodo Action을 전달한다.

### 4.1.4 TodoList.js

- TodoList 클래스 컴포넌트를 아래와 같이 작성한다.
- render 함수는 다음의 엘리먼트를 반환한다.
  - ✓ : 전체 Todo 의 개수와 완료된 Todo 의 개수를 나타낸다.
  - ✓ <Form>: '4.1.2'에서 작성한 Component
  - ✓ <List>: '4.1.3'에서 작성한 Component

### 4.1.5 App.js

- TodoList 클래스 엘리먼트를 아래와 같이 작성한다.
- render 함수는 다음의 컴포넌트를 반환한다.
  - ✓ <Header>: '4.1.1'에서 작성한 Component
  - ✓ <TodoList>: '4.1.4'에서 작성한 Component

### 4.1.6 index.js

- 다음과 같이 [9]Provider에 store를 생성한다.

```
<Provider store={createStore(reducers)}>
        <App />
</Provider>
```

[9] https://redux.js.org/tutorials/fundamentals/part-5-ui-react#passing-the-store-with-provider

## 4.2 Redux Actions & Reducers

### 4.2.1 todo.js (Actions)

- createTodo: Todo 생성
  - ✓ 다음과 같은 필드를 가진다.
    - ◆ type: CREATE_TODO
    - ◆ id: todoID++
    - ◆ text
- deleteTodo: Todo 삭제
  - ✓ 다음과 같은 필드를 가진다
    - ◆ type: DELETE_TODO
    - ◆ text
- toggleTodo: Todo 완료
  - ✓ 다음과 같은 필드를 가진다
    - ◆ type: TOGGLE_TODO
    - ◆ id:

### 4.2.2 todo.js (Reducers)

- CREATE_TODO
  - ✓ 입력 받은 Todo를 생성한다.
  - ✓ TodoList 컴포넌트의 Total 상태가 증가한다.
- DELETE_TODO
  - ✓ state.filter 메소드를 사용하여 Todo 삭제한다.
  - ✓ TodoList 컴포넌트의 Total 상태가 감소한다.

- TOGGLE_TODO
  - ✓ state.map 메소드를 사용하여 다음과 같이 상태를 변경한다.
    - ◆ 완료하지 않은 경우
      - ● TodoList 컴포넌트의 DONE 상태가 증가한다.
      - ● <li>(Todo)를 초록색으로 변경한다.
    - ◆ 이미 완료한 경우
      - ● 다시 진행중인 상태로 바꾼다.
      - ● TodoList 컴포넌트의 DONE 상태가 감소한다.

### 4.2.2 index.js (Reducers)

- 아래의 코드를 작성하여 todo 폴더의 모든 Reducer를 하나의 todo 객체로 병합한다.
  - ✓ [10]combineReducers 함수를 사용한다.

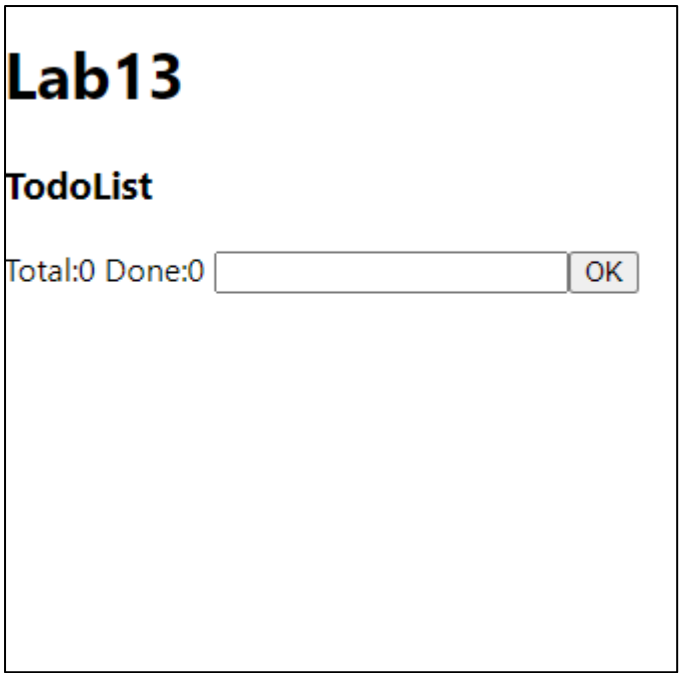
```
import { combineReducers } from 'redux';
import todo from './todo';

// Merge multiple reducers in single reducer object (root reducer)
export default combineReducers({
    todo
})
```

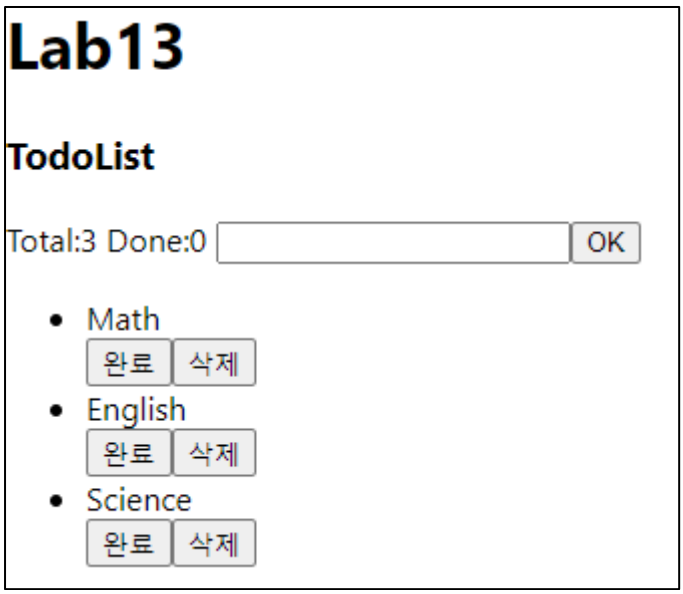
[10] https://redux.js.org/api/combinereducers

## 5. 실행 예시

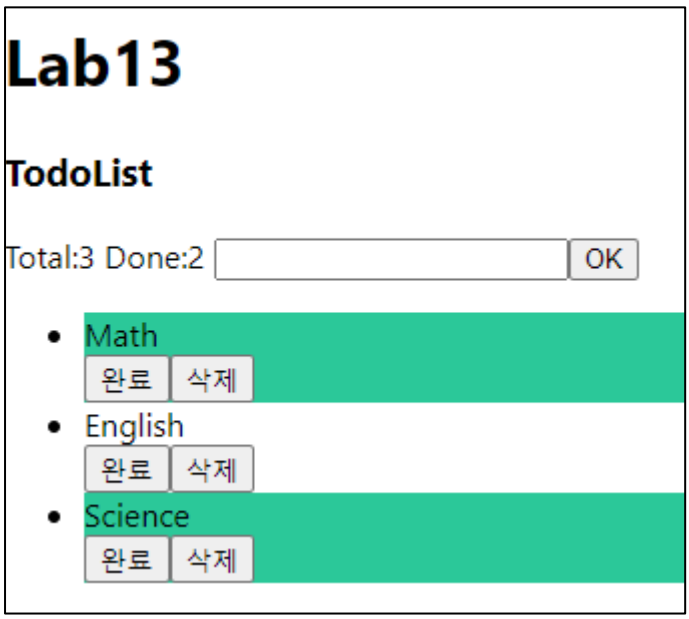
### 1. 시작화면




### 2. Todo 3 가지를 추가한 경우



**3. Todo 2 가지를 완료한 경우**




**4. 3 에서 Math 를 삭제한 경우**