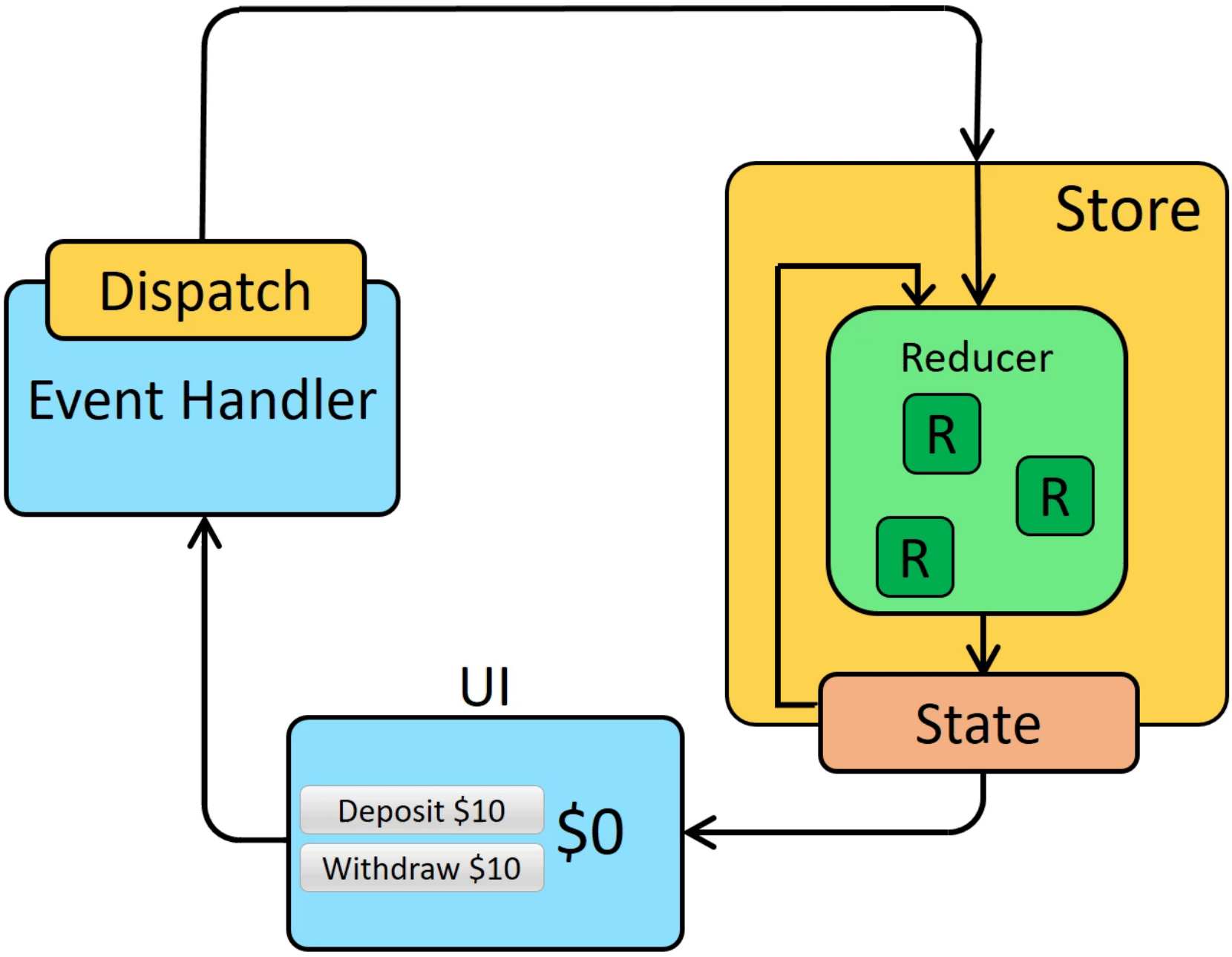
Redux

Status	Not started
Assign	React.js

💎 리덕스 란?_redux

액션이라는 이벤트를 사용하여 애플리케이션 상태를 관리하고 업데이트하기 위한 상태관리 라이브러리로써,
리덕스 라이브러리는 예측가능한 방식으로만 업데이트될 수 있도록 보장하는 규칙과 함께 전체 애플리케이션에서 사용해야 하는 상태에 대한 ***중앙 저장소 같은 역할***을 한다.

💎 리덕스의 데이터 흐름_redux-data-flow



Redux의 데이터 흐름은

동일한 단방향으로 컴포넌트에서 ***dispatch***(store의 내장함수; store에서 주는 state바꾸는 함수)라는 함수를 통해 ***action***(디스페함수이름)이 발동되고 ***reducer***에 정의된 로직에 따라 store의 state가 변화하고 그 state를 쓰는 컴포넌트가 변하는 흐름을 따른다.

💎 리덕스의 주요 용어 정리 _redux-pricipal-terms

Action(액션)

```
{ type : GET_RECIPES_SUCCESS,
payload: recipes }
```

상태에 변화가 필요하다면 액션을 일으키며, 액션은 객체로 표현이 되고 type필드를 반드시 가지고 있어야하는 특징을 가진다.
실생활에서 예를 든다면, 식당에서 음식 주문서와 같은 역할을 하는 거라고 보면 됩니다.

Action Creator(액션생성함수)

```
function action creator () {

 //액션을 반환
 return {
   type: GET_RECIPES_SUCCESS,
   payload: recipes
 }
}
```

액션 객체를 만들어주는 함수입니다.
식당에서 음식 주문을 넣는 자판기와 같은 역할을 하는 거라고 보면 됩니다.

Reducer

```
const initialState = { recipes: [] }
function reducer (state = initialState, action) {
  switch(action.type) {
    case GET_RECIPES_SUCCESS:
      return {
        ...state,
        recipes: action.payload
      }
    default:
      return state
  }
}
```

현재 상태와 액션 객체를 받아, 필요하다면 새로운 상태를 리턴하는 함수
액션 유형을 기반으로 이벤트를 처리하는 이벤트 리스너라고 생각하면 됩니다.
식당에선 음식주문에 따라 음식을 만들어서 음식을 내보내는 역할을 하는 거라고 보면 됩니다.

Store

```
import { createStore } from 'redux'

const initialState = { recipes: [] }
function reducer (state = initialState, action) {
  switch(action.type) {
    case GET_RECIPES_SUCCESS:
      return {
        ...state,
        recipes: action.payload
      }
    default:
      return state
  }
}

const store = createStore(reducer)
```

스토어에는 상태가 들어있습니다.
모든 애플리케이션에 있는 상태를 저장한다면 하나의 스토어에서 모든 상태를 관리할 수 있게 되어지는 것이다.

Dispatch

```
store.dispatch({ type: GET_RECIPES_SUCCESS })
```

스토어의 내장 함수 중 하나이며, 액션 객체를 넘겨줘서 상태를 업데이트하는 유일한 방법으로, 이벤트 트리거(event trigger)이라고 보면 됩니다.

react-hooks useDispatch를 사용하여 dispatch하는 것도 가능합니다.
식당에서 직원이 음식 주문을 받아 주방에 주문서를 전달하는 거라고 보면 됩니다.

Selector

이 또한 스토어의 내장함수인 getState를 사용하여 상태 값을 가져올 때 사용합니다.
react-hooks useSelector를 사용하여 가져오는 것도 가능합니다.

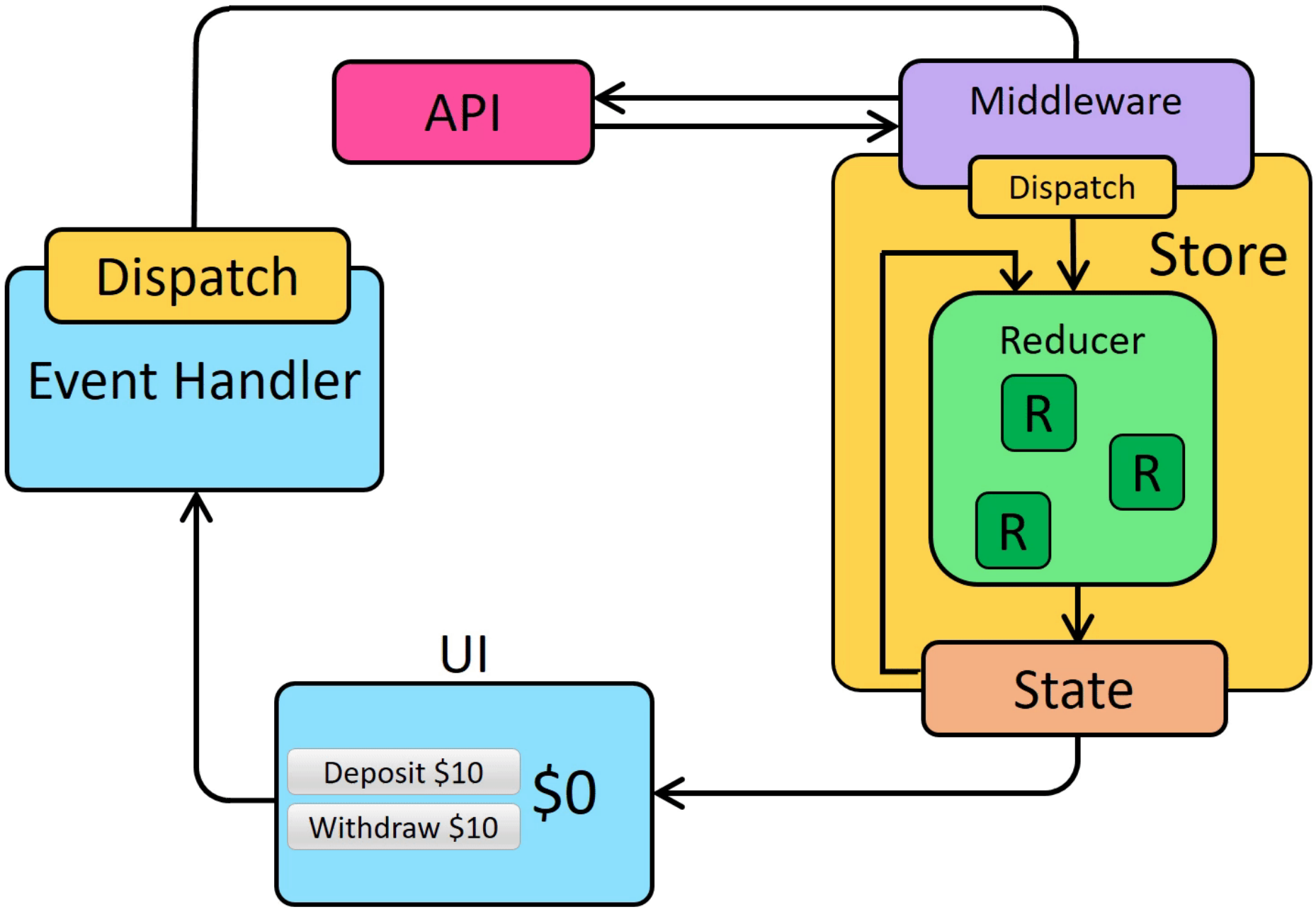
Redux-thunk(middleware)

```
import thunk from 'redux-thunk';
import { createStore, applyMiddleware } from 'redux';

const initialState = { recipes: [] }
function reducer (state = initialState, action) {
  switch(action.type) {
    case GET_RECIPES_SUCCESS:
      return {
        ...state,
        recipes: action.payload
      }
    default:
      return state
  }
}

const middleware = [thunk]
const store = createStore(reducer, applyMiddleware(...middleware))
```

액션 안에 api를 호출하게 되면 비동기적으로 일어날 수 있도록 도와주는 Middleware함수입니다.
보통 리덕스에서 미들웨어를 사용하는 주된 사용 용도는 비동기 작업을 처리할 때 사용하는데, 리액트 앱에서 우리가 만약 백앤드 api를 연동해야 된다면, 리덕스 미들웨어를 사용하여 처리해야 하곤 합니다.
redux의 액션들은 모두 동기적으로 이러나기 때문에 중간에 비동기적 처리해야하는 처리가 있다면 에러가 나게 됩니다. 이런 경우를 대비하여 redux middleware를 사용하여 처리해 줍니다. 중간과정에서 비동기를 처리를 해준 다음 액션을 리듀서로 전달해주는 역할을 합니다.



💎 리덕스를 사용시 지켜야하는 필수규칙_redux-required-rules

- 하나의 애플리케이션 안에는 하나의 스토어만 사용하자는 원칙
- 상태를 변화시키는 방법은 오직 액션을 일으키는 것, 상태를 변화시키는 의도를 정확하게 표현할 수 있고, 상태 변경에 대한 추적이 용이해지게 됩니다.
- 변화를 일으키는 리듀서 함수는 순수한 함수여야 합니다.
  : 이전 상태와 액션 객체를 파라미터로 받아야 합니다
  : 파라미터 외의 값에는 의존하면 안된다. 영향을 받지 않아야 합니다.
  : 이전 상태는 절대로 건드리지 않고, 변화를 준 새로운 상태 객체를 만들어서 반환해야 합니다.
  : 똑같은 파라미터로 호출된 리듀서 함수는 언제나 똑같은 결과 값을 반환해야 합니다. 즉, 기본값 상태는 변하지 않아야 합니다.

💎 리덕스의 특징_redux-features

- 하나의 상태를 낳는다, 즉 하나의 객체 안에 애플리케이션에서 필요한 모든 데이터를 우깨워 넣으며 저렇게 한 곳에 데이터를 중앙 집중적으로 관리하면 아무리 여러 곳에 흩어져 있는 것보다 훨씬 더 데이터를 관리 하기가 쉬워집니다. 단 하나의 스테이트를 유지하는 걸 통해서 애플리케이션의 복잡성을 낮춥니다
- 데이터의 상태변화를 직접 하지못합니다. 디스펙쳐라는 걸 통해서 또 리듀서를 통해서만 데이터의 수정을 가할 수 있습니다. 데이터를 가져올 때 직접 가져오지 못합니다. getState라는 함수를 통해서 가져올 수 있습니다. 데이터를 외부에서 직접적으로 제어할 수 없도록 하는 걸 통해서 의도하지 않게 얘기치 않게 스테이트값이 바뀌는 문제를 사전에 차단하는 걸 통해서 우리 애플리케이션을 보다 예측 가능하게 만드는 것 입니다.
- 그리고 각가의 스테이트값들을 생성할 때 철저하게 통제하고 데이터를 바꿀 대 원본을 바꾸는 것이 아니라 복제하고 복제한 데이터를 수정해서 그것을 새로운 원본으로 만드는 이런 방법을 채택하고 있습니다. 각각의 상태의 변화가 서로에게 영향을 전혀 주지 않는 독립된 형태를 유지할 수 있고 이러한 특징을 잘 이용하면 언도와 리드와 같이 애플로케이션의 상태를 바꾸는 것은 매우 쉽게 처리할 수 있습니다.
- 리덕스를 이용하게 되면 현재 상태 뿐만 아니라 이전의 상태까지 꼼꼼하게 코딩하는 걸 통해서 과거에 어느 시점으로 돌아가서 그 시점에서 이 애플리케이션의 상태가 무엇인가를 찾아내는 걸 통해서 문제해결을 훨씬 더 쉽게 할 수 있도록 도와줍니다.

💎 리덕스의 장 * 단점_redux-pros-and-cons

🌝 장점

- 우리의 코드가 어떤 결과를 가져올지 예측 가능하게 만들어주며 한눈에 보기 쉽게 만들어줍니다.
- 전역으로 사용이 가능합니다.
- 상태끌어올리기보다 효율적이고 짧은 코드를 작성하여 복잡한 코드를 좀더 간략하고 효율적으로 작성이 가능하게 해줍니다. 즉, A,B,C,D 컴포넌트가 있다고 가정했을 때 A에서 D컴포넌트에 접근해 무언가 하려고 한다면 A,B,C,D순서로 접근 후 다시 D,C,B,A루트를 통해 돌아와야 합니다. 이때 하나의 스토어 (Store)라는 매체를 두면 위의 순서가 아닌 A>store>D식의 효율적인 접근이 가능하게 해줍니다.

🌚 단점

- 반복적은 코드를 써야합니다.
- 반복적은 코드를 써야함에 코드양이 많아 집니다.

💎 리덕스를 사용해야 하는 이유?_Why Use Redux?

- 상태 예측 가능하게 만들수 있다.
- 유지 보수가 비교적 쉬워진다
- 의도하지 않게 스테이트 값이 바뀌는것을 사전에 차단할 수 있다.