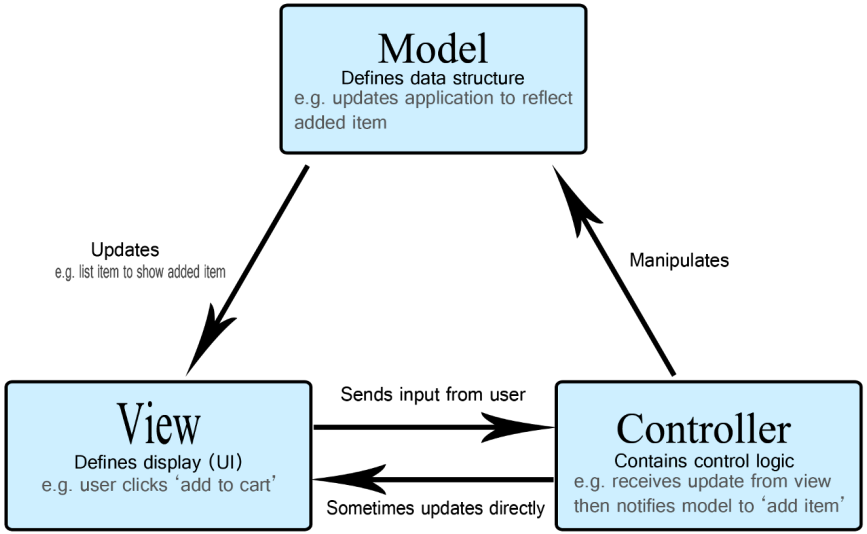
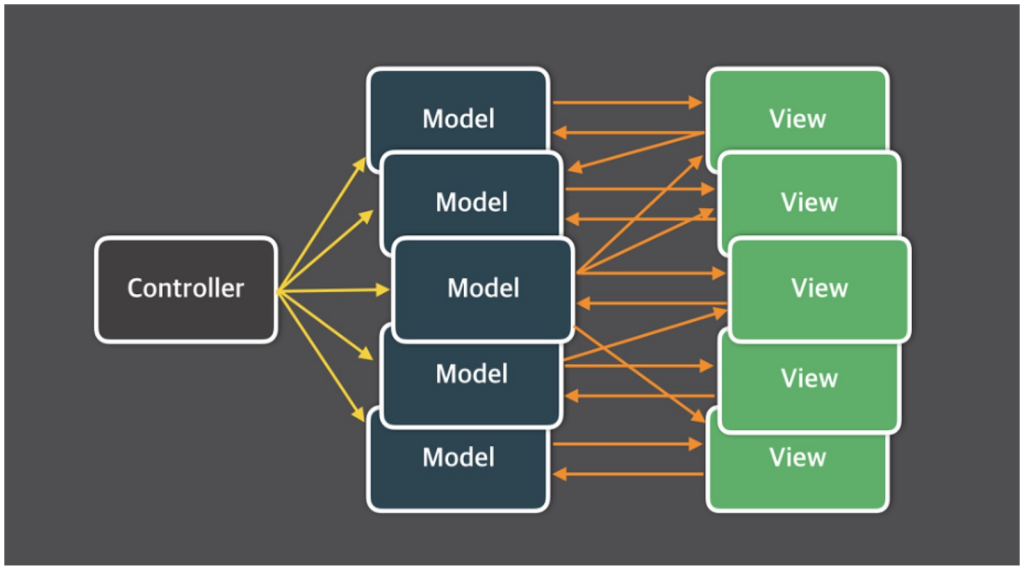
Redux

MVC



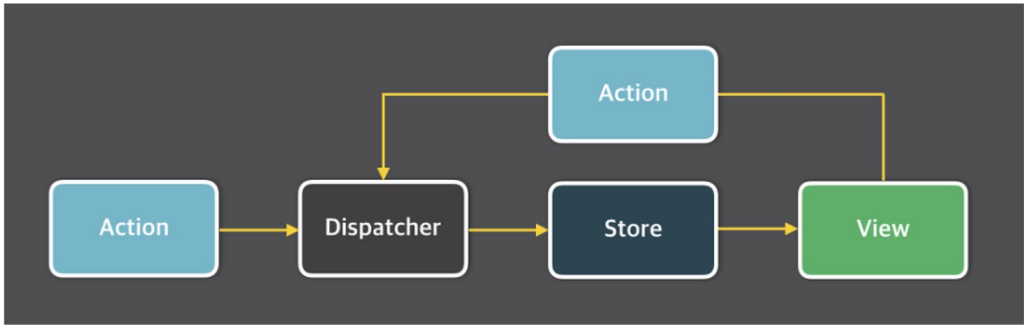
- Model: 데이터 베이스에서 데이터를 가지고 오거나 데이터를 가지고 있다.
- View: 사용자 인터페이스 요소
- Controller: 뷰에서 액션과 이벤트에 대한 값을 받는다. 모델에게 전달해 주기 전 데이터를 가공.

MVC 문제점



- 처음에 MVC방식으로 개발하던 메타(구.페이스북)는 한계를 느낌
- model – view가 복잡하게 얽힘

FLUX



- 그래서 개발된 게 flux패턴!
- redux는 flux 패턴으로 구현된다.

https://redux.js.org/tutorials/essentials/part-1-overview-concepts

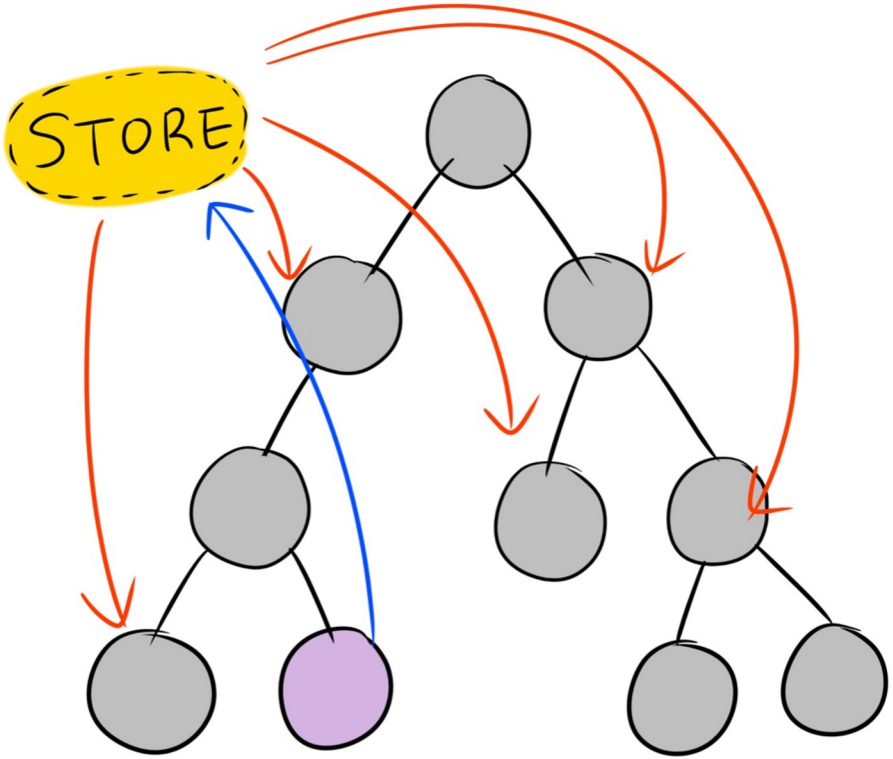
- 왜 상태관리를 해야 하는가?
  리액트를 props로 데이터를 주고 받기 때문에

Redux의 흐름

1. UI:이벤트 발생
2. dispatch(보내다): ui로 부터 이벤트를 받고 action을 스토어에 전달
3. store(저장하다): 상태가 변경 되었음을 UI에게 전달

   *store는 reducer를 실행해 상태는 발생한 일에 따라 업데이트

   *reducer: 이전 상태 + 액션을 기반으로 새로운 상태 값을 계산하는 함수



- store를 통해서 데이터를 주고 받는다, store를 어떻게 잘 설계하느냐가 중요

store

- 단일 스토어
- 단일 스토어 사용 준비
  - 액션정의-> 액션 사용하는 리듀서 만들기 -> 리듀서 합치기 -> 합쳐진 리듀서를 인자로 단일 스토어 만들기
- 준비한 스토어를 리액트 컴포넌트에서 사용하기
  - connection 함수 혹은 hook을 이용해서 연결

action

- 사실 그냥 객체
- 두 가지 형태 존재
  - { type: ‘test’ }
  - { type: ‘test’, params:’hello’ }
- type만이 필수 프로퍼티 이며, type은 문자열
- 액션 생성자(액션을 생성하는 함수, action creator)
  - function 액션 생성자(...args) {return 액션;}
  - 함수를 통해 액션을 생성하고 액션 객체를 리턴햐줌
  - createTes(‘hello’); // { type:’TEST’, params: ‘hello’ }

action은 어떤 일을?

- 액션 생성자를 통해 액션을 만듬
- 만들어낸 액션 객체를 스토어에 보냄
- 스토어는 액션 객체를 받아 스토어의 상태값이 변경
- 변경된 상태 값에 의해 상태를 이용하고 있는 컴포넌트가 변경
- 액션은 스토어에 보내는 일종의 인풋

액션을 준비하기 위해서는?

- 액션의 타입을 정의하여 변수로 빼는 단계
  - 강제는 아님
  - 그냥 타입을 문자열로 넣기보다 미리 정의한 변수 사용이 좋음(스펠링에 주의를 덜 기울여도 됨)
- 액션 객체를 만들어 내어 함수를 만드는 단계
  - 하나의 액션 객체를 만들기 위해 하나의 함수를 만듬
  - 액션의 타입은 미리 정의한 타입 변수로 부터 가져아 사용

reducer

- 액션을 주면 그 액션이 적용되어 달라지거나 혹은 안 달라진 결과를 만들어 줌
- 그냥 함수
  - pure function(같은 인풋을 받으면 같은 결과를 내는 함수)
  - Immutable(오리지널 스테이트와 새로 바뀐 스테이트가 별도의 객체로 만들어 져야함)
  - 왜? 리듀서를 통해 스테이트가 달라졌음을 리더스가 인지하는 방식

reducer

```
function 리듀서(previousState, action) {

        return newState;

}
```

- 액션을 받아서 state 리턴
- 인자로 들어오는 previousState와 리턴되는 newState는 다른 참조를 가지도록 해야 한다.

createStore

const store = createStore(리듀서);

```
createStore<S>(
        reducer,
        preloadedState(초기 스테이트),
        enhancer,
)
```

store 기능

- store.getState();
  - 현재 store의 스테이트를 가져옴
- store.dispatch; store.dispatch(액션 생성자());
  - action을 인자로 넣어서 store의 상태 변경
- store.subscribe(() => {})
  - 스토어에 변경이 생겼을 때 실행
- const unsubscribe = store.subscribe(() => {})
  - subscribe가 제거됨

react-redux

- provider 컴포넌트 제공
- connect 함수를 통해 컨테이너를 만들어줌
  - 컨테이너는 스토어의 state와 disptch를 연결한 컴포넌트에 props로 넣어주는 역할
  - 그렇다면 필요한 것은?
    - 어떤 state를 어떤 props에 연결할 것인지에 대한 정의
    - 어떤 dispatch를 어떤 props에 연결할 것인지에 대한 정의
    - 그 props를 보낼 컴포넌트를 정의

미들웨어(middleware)

- 미들웨어가 디스패치의 앞뒤에 코드를 추가할수 있게 해줌
- 미들웨어가 여러개면 미들웨어는 순차적으로 실행
- 두 단계 존재
  - 스토어를 만들때, 미들웨어를 설정하는 부분
    - {createStore, applyMiddleware} from redux
  - 디스패치가 호출될때 실제로 미들웨어를 통과하는 부분
- dispatch메소드를 통해 store로 가고 있는 액션을 가로채는 코드

redux-thunk

- 리덕스 미들웨어
- 리덕스에서 비동기 처리를 위한 라이브러리
- 액션 생성자를 활용하여 비동기 처리
- 액션 생성자가 액션을 리턴하지 않고, 함수를 리턴함

redux-saga

- 제너레이터 객체를 만들어 내는 제너레이터 생성 함수를 이용
- 리덕스 사가 미들웨어 설정후 내가 만든 사가 함수를 등록한 후 사가 미들웨어 실행
- 등록된 사가 함수를 실행할 액션을 디스패치