State Management and Redux

Shan-Hung Wu & DataLab
CS, NTHU

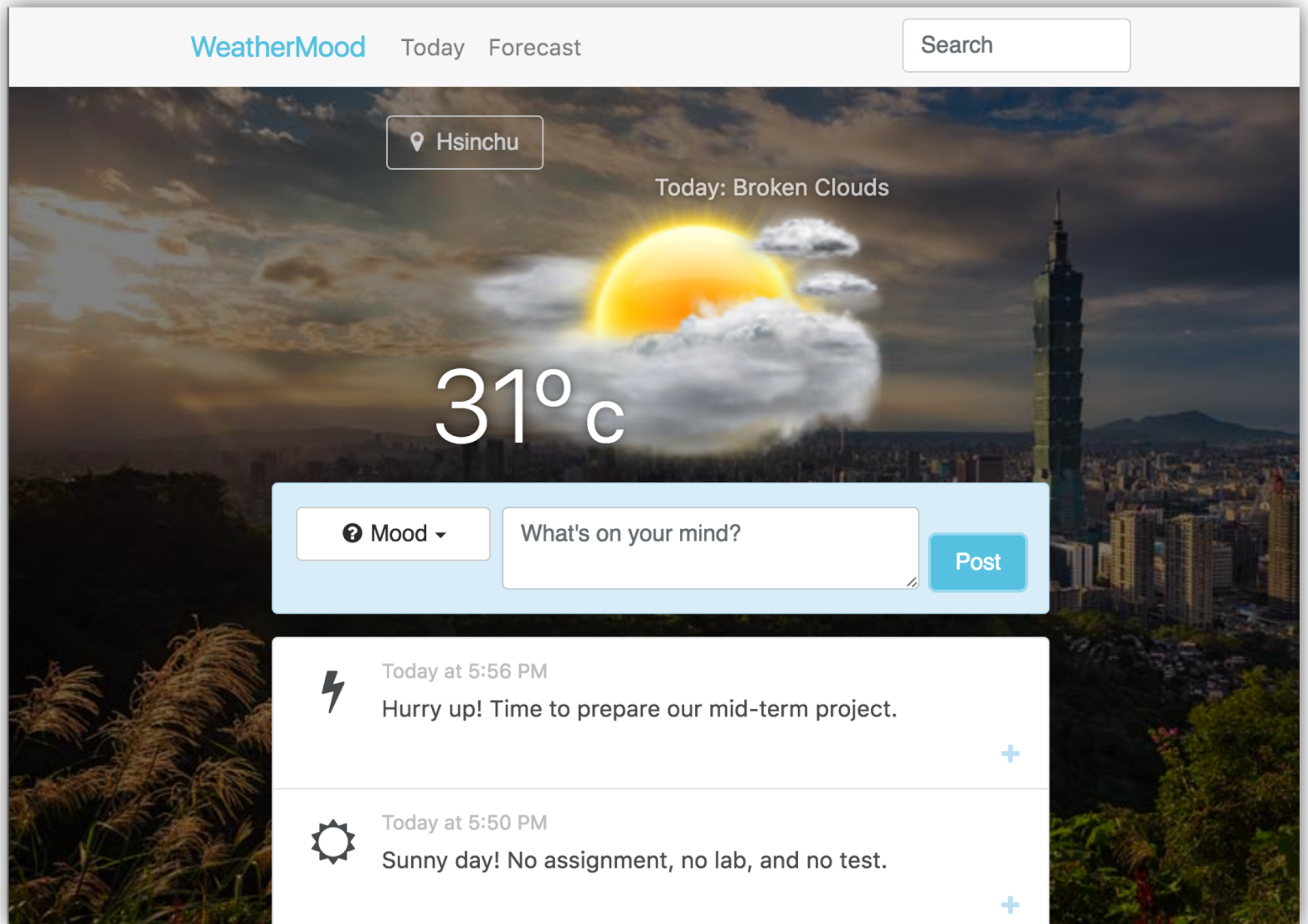
Outline

- WeatherMood: Posts
- Why Redux?
- Actions and Reducers
- Async Actions and Middleware
- Connecting with React Components
- Remarks

Outline

- WeatherMood: Posts
- Why Redux?
- Actions and Reducers
- Async Actions and Middleware
- Connecting with React Components
- Remarks

Clone `weathermood/react-post`



Setup

```
$ npm install --save @babel/polyfill \
  moment uuid
```

- Babel Polyfill
  - Use ES6 Promise to simulation asynchronous post fetching
- Moment
  - For displaying date & time
- UUID
  - Generates unique IDs for new posts

API for Posts

```
// in api/posts.js

listPosts(seatchText).then(posts => {
  ...
});
createPost(mood, text).then(post => {
  ... // post.id
});
createVote(id, mood).then(() => {...});
```

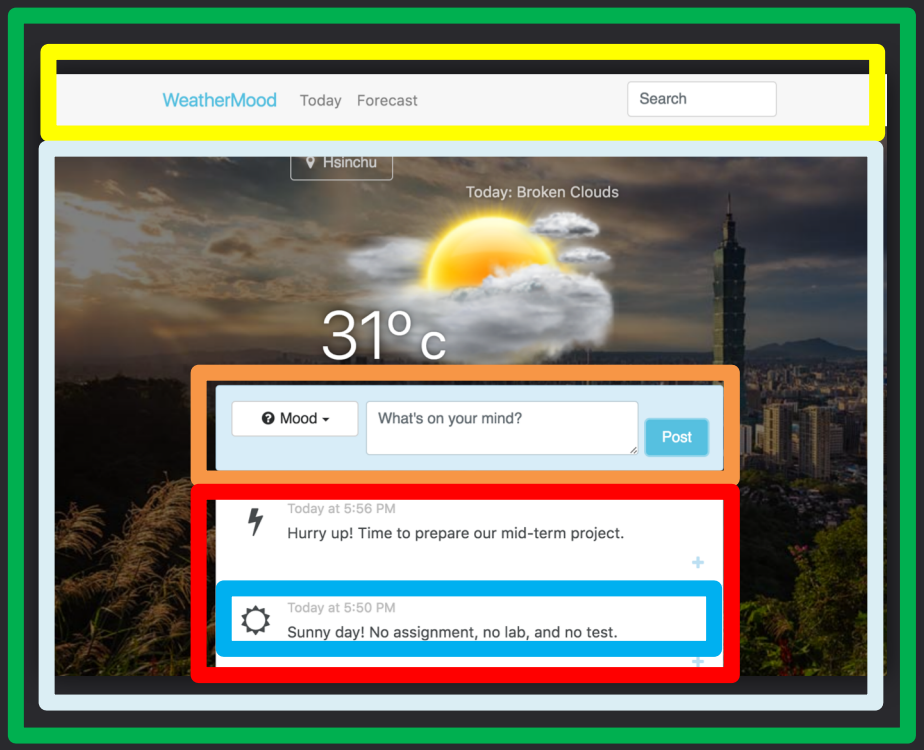
- Asynchronous (ES6 Promise-based)
- Simulated currently

HTML 5 Web Storage

```
localStorage.setItem('key', 'value');
let v = localStorage.getItem('key');
localStorage.removeItem('key');
```

- Specific to domain ***and protocol***
- >5MB
- Values must be ***strings***
  - Use `JSON.stringify()` and `JSON.parse()` for objects
- `sessionStorage` is similar, except data gone when `window` closed

Steps 1 & 2: Components & Props



Navbar

Main

PostForm

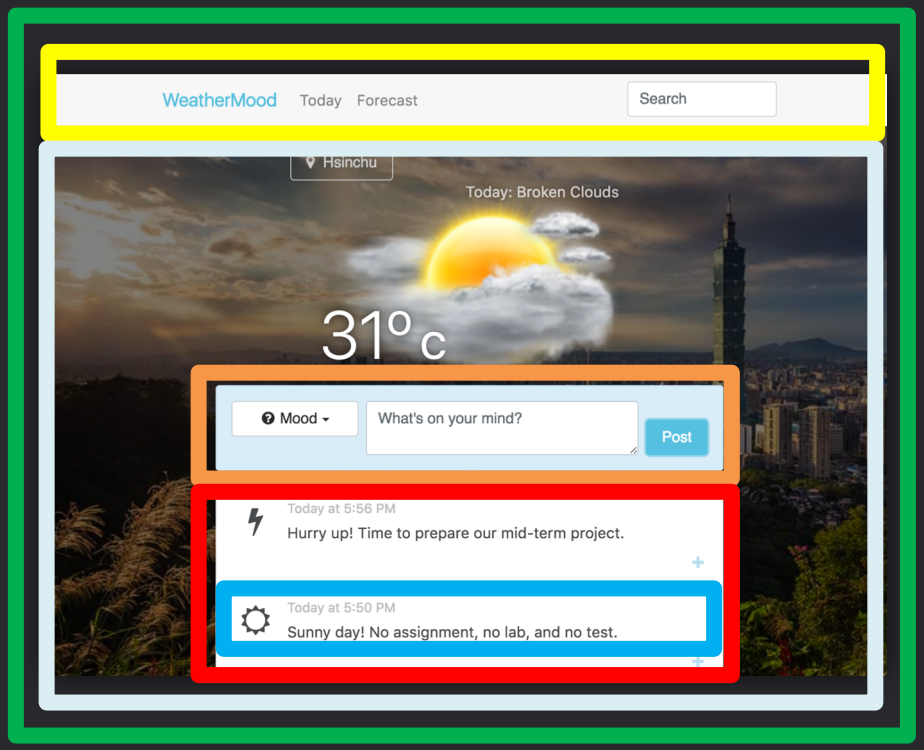
PostList

PostItem

Today

Steps 3 & 4: States

Main {
  searchText
}



Today { posts }

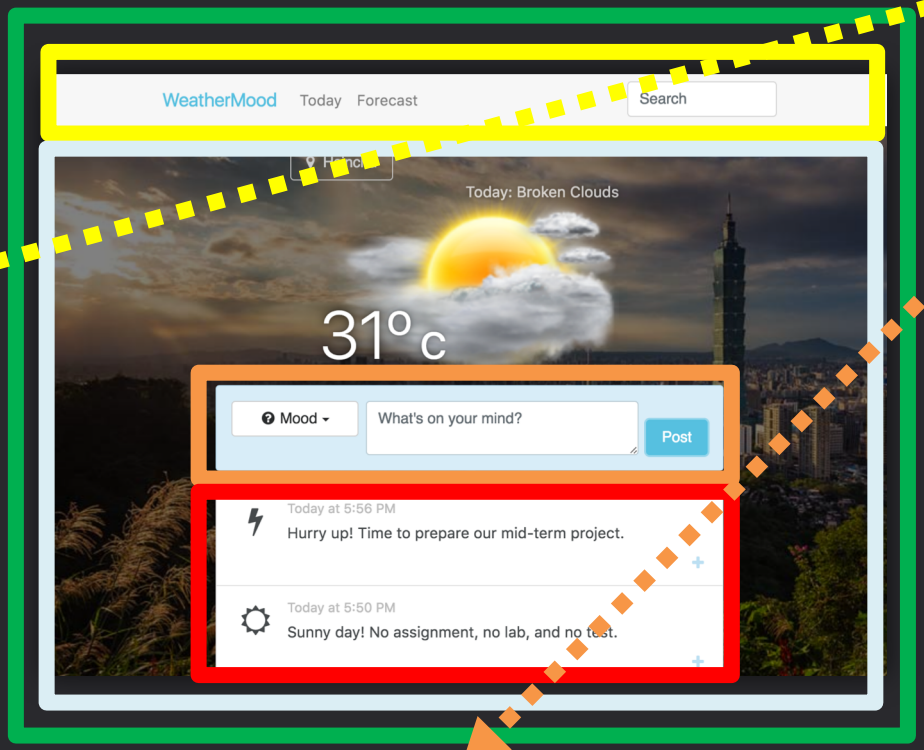
Navbar {
  ~~searchText~~
}

PostForm {
  ~~mood~~, ~~text~~
}

PostList {
  ~~posts~~
}

PostItem {
  ~~votes~~
}

Step 5: Callbacks



Navbar {
~~searchText~~
}

PostForm {
~~mood~~, ~~text~~
}

PostList {
~~posts~~
}

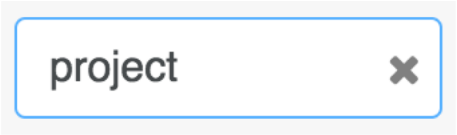
PostItem {
~~votes~~
}

Main {
searchText
}

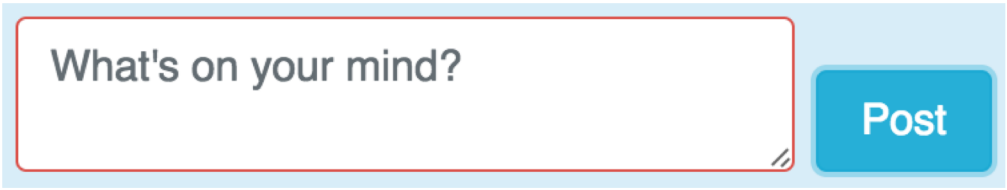
Today { posts }

Details

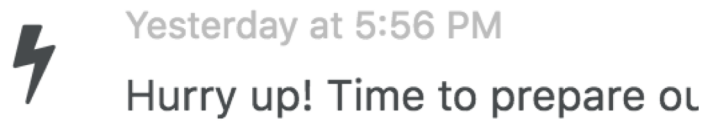
- Search box



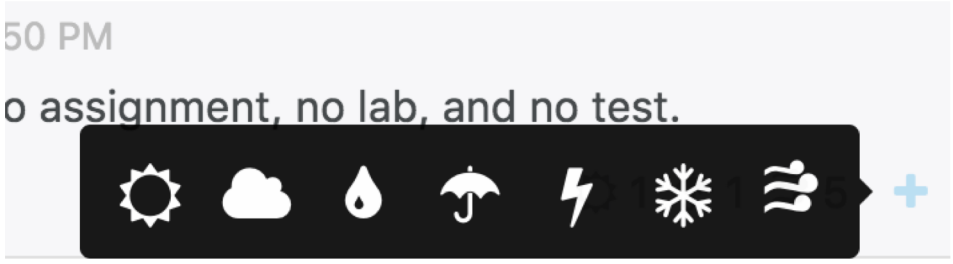
- Form validation



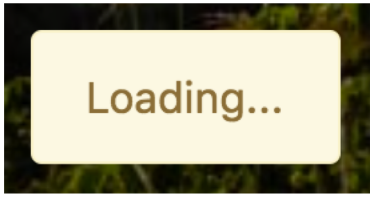
- Timestamp



- Tooltips



- Loading indicators



Outline

- WeatherMood: Posts
- Why Redux?
- Actions and Reducers
- Async Actions and Middleware
- Connecting with React Components
- Remarks

React is Declarative in Terms of ***States***

```
render() {
  return (
    <h1 className={this.state.toggle}>
      Hello {this.props.name}
    </h1>
  );
}
```

- Code for “states,” not “changes of states”
  - Virtual DOM tracks changes automatically
- UI = ***maps from states*** to visual looks
  - Each component is a function of partial states

Limitations I

- States of a component may be controlled outside
  - `Main` and `Today` contain UI code and mixed state logic from their childrens

Main { unit }

WeatherDisplay {
~~temp~~, ~~unit~~
~~weather~~, ~~desc~~
}

WeatherForm {
~~city~~, ~~unit~~
}

Today { weather, temp, desc, city }

Limitations II

- Cannot move/recompose components easily
  - Bad for evolving projects (e.g., startups)

Main { unit }

WeatherDisplay {
~~temp~~, ~~unit~~
~~weather~~, ~~desc~~
}
WeatherForm {
~~city~~, ~~unit~~
}

Today { weather, temp, desc, city }

Limitations III

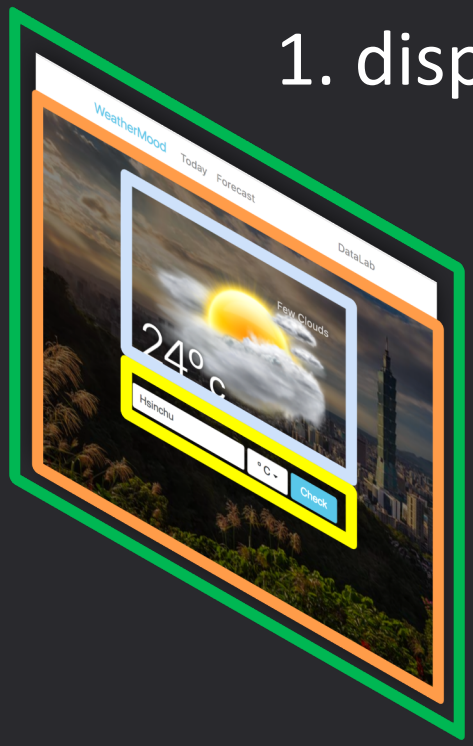
- States are hard to track
  - Spread among multiple components
- State changes are implicit
  - Who made this change? Is change correct?
- Stateful logic cannot be shared between components
  - E.g., stateful logic: “fetch data, render children if 200, otherwise show error ”

Redux

- A state management framework
  - Restricts how you write state management code
- Not tied to, but works well with React

React
(UI)

Redux
(State Store)



1. dispatch(action)

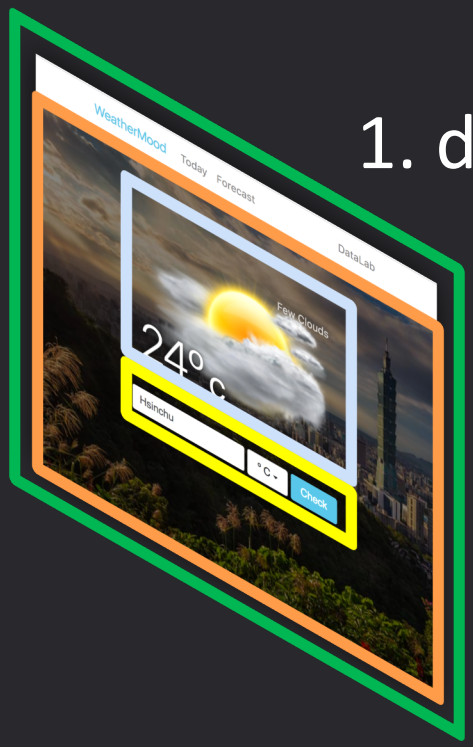
2. reduce(action)

```
Main { unit }
Today {
  weather, temp,
  desc, city
}
```

3. connect(props)

Advantages I

- Separation of concerns
  - Stateful logic apart from (stateless) UI rendering logic
  - States easy to find and inspect



1. dispatch(action)

2. reduce(action)

```
Main { unit }
Today {
  weather, temp,
  desc, city
}
```

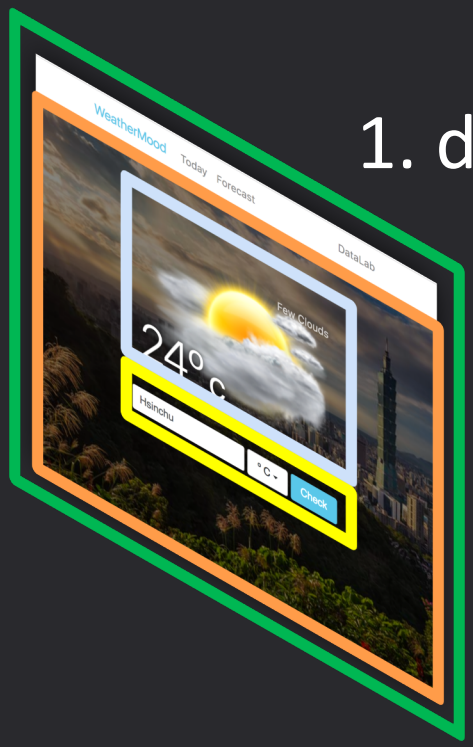
3. connect(props)

Advantages II

- Unidirectional (top-down) data flow in React
  - Loosely coupled components; UI easy to recompose

Advantages III

- "Reducer" makes state changes less buggy
  - Enforce determinism: same (prevState, action), same nextState



1. dispatch(action)

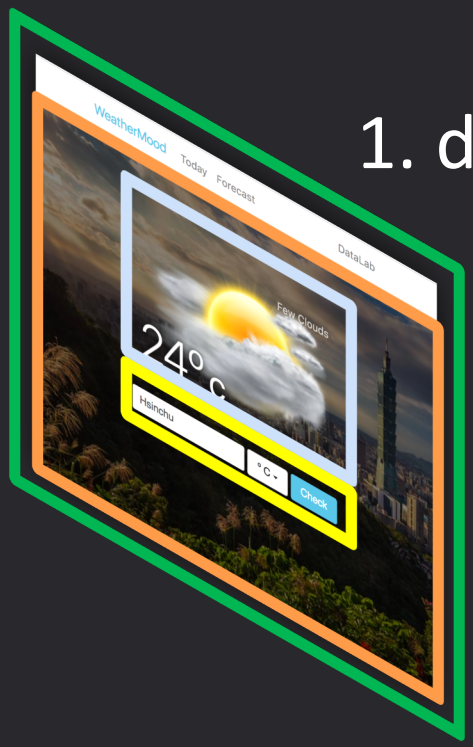
2. reduce(action)

```
Main { unit }
Today {
  weather, temp,
  desc, city
}
```

3. connect(props)

Advantages IV

- “Actions” make stateful logic shareable
  - Action A: “fetch data→ render or show error ”
  - Both comp. X and Y can call dispatch(A)



1. dispatch(action)

2. reduce(action)

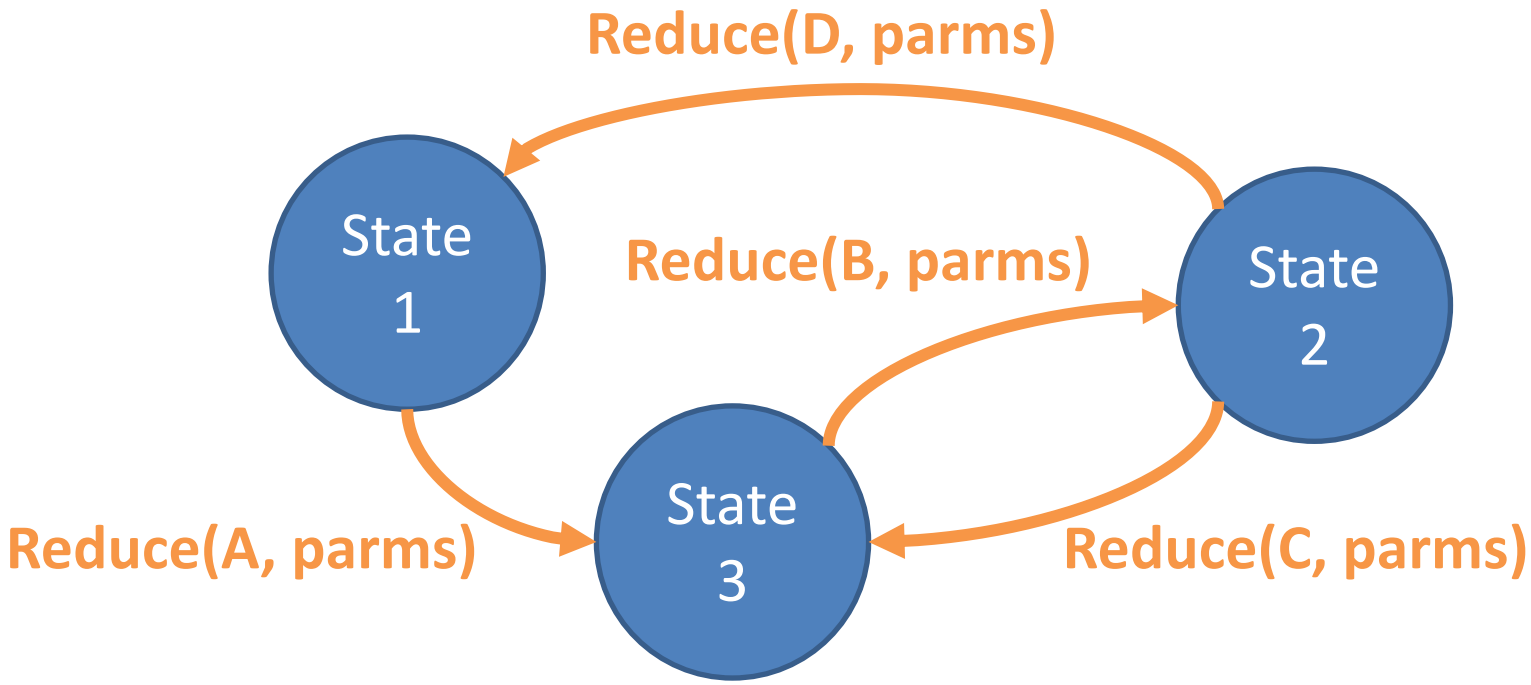
```
Main { unit }
Today {
  weather, temp,
  desc, city
}
```

3. connect(props)

Outline

- WeatherMood: Posts
- Why Redux?
- Actions and Reducers
- Async Actions and Middleware
- Connecting with React Components
- Remarks

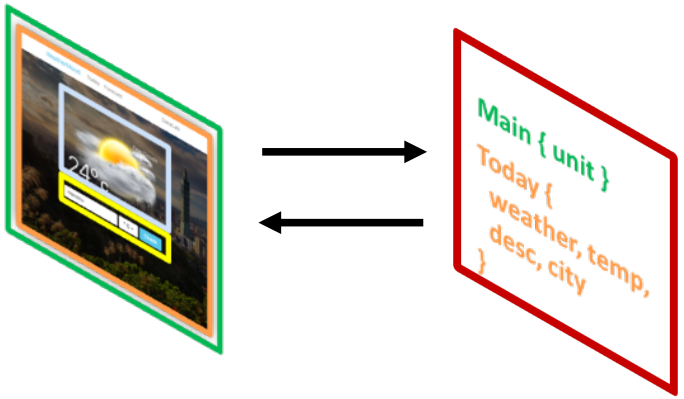
Redux Store Is a State Machine



- State transitions must be ***deterministic***
- I.e., same (prev state, action, params), same next state

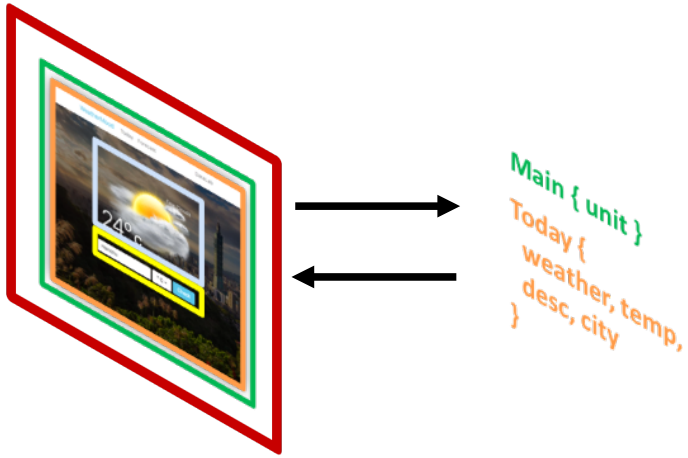
Actions & Reducers

```
// action generator
export function setWeather(code, temp) {
  return { // action and parms
    type: '@WEATHER/SET_WEATHER',
    code,
    temp
  };
}
// reducer
export function weather(state = {...}, action) {
  switch (action.type) {
    case '@WEATHER/SET_WEATHER':
      return {
        ...state,
        code: action.code,
        temp: action.temp
      };
    default:
      return state;
  }
}
```



Using Redux Store

```
// in UI
import {createStore} from 'redux';
import {setWeather, weather} from ...;

const store = createStore(weather);
```




```
// in Component1
store.subscribe(() => {
  console.log(store.getState());
});

// in Component2
store.dispatch(setWeather(800, 21));
```

Reducers Must Be ***Pure*** Functions

- To ensure deterministic state transitions
- Pure functions?
- Same input, same output
  - No `Math.random()` nor `Date.now()`
- No side effect
  - Cannot update variables outside
  - Cannot mutate input
  - Cannot make API calls
- Synchronous

Splitting Reducers

```
export function code(state = -1, action) {
  switch (action.type) {
    case '@CODE/SET_CODE':
      return action.code;
    default:
      return state;
  }
}
export function temp(state = 0, action) {
  switch (action.type) {
    case '@TEMP/SET_TEMP':
      return action.temp;
    default:
      return state;
  }
}

const store = createStore((state, action) => ({ // wrapper
  code: code(state.code, action),
  temp: temp(state.temp, action)
}));
```

- One reducer for independent “state group”

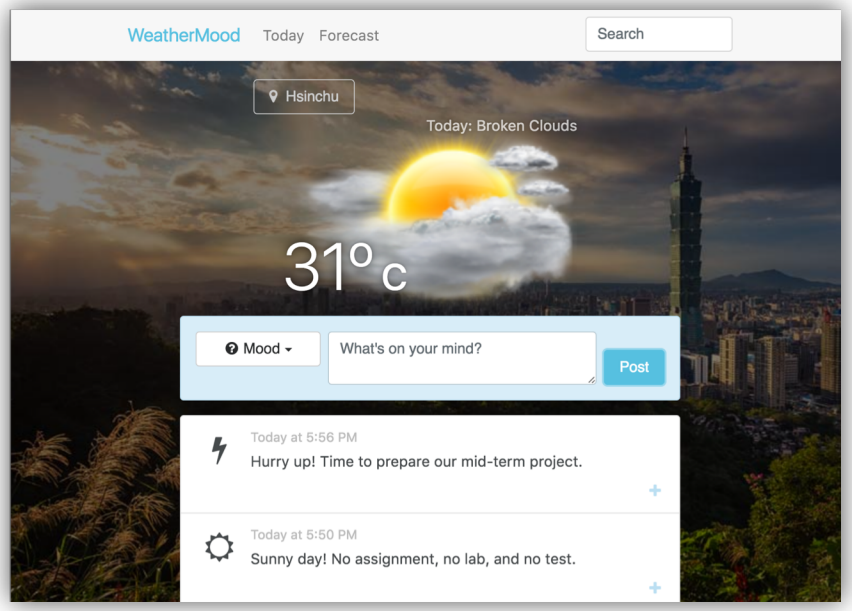
Simplification

```
const store = createStore((state, action) => ({
  code: code(state.code, action),
  temp: temp(state.temp, action)
}));

// same as

import {combineReducers} from 'redux';

const store = createStore(combineReducers({
  code,
  temp
}));
```

Outline

- WeatherMood: Posts
- Why Redux?
- Actions and Reducers
- Async Actions and Middleware
- Connecting with React Components
- Remarks

weathermood/redux-weather



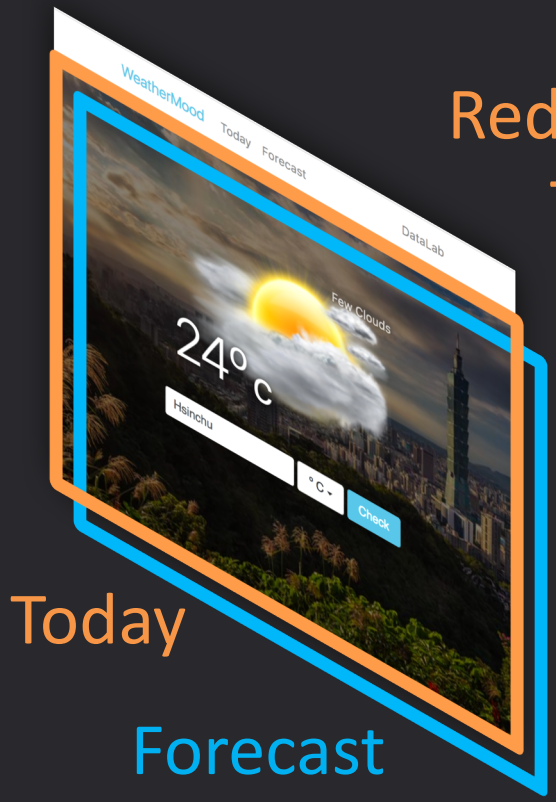
- Looks the same as react-post
- But weather components (`Today`, `Forecast`, etc.) use Redux to manage states

How to Design Reducers?

1. Identify independent “state groups”
   – E.g., weather+forecast vs. posts

How to Design Reducers?

2. Come out lifted state hierarchy as in react
3. Move states of each component to a reducer



Reducers for Today

@weather { city, code, temp. desc }

@unit { unit }

@forecast { city, list }

Reducers for Forecast

Async Actions

- For fetching weather, forecast, posts, etc.
- But reducers must be pure
  - No API call, synchronous
- How?

1. Break async action into ***sequence of steps***
   - State transition for each step is deterministic
2. Dispatch steps in UI following the sequence

```
// action generators
export function startGetWeather() {
  return {type: '@WEATHER/START_GET_WEATHER'};
}
export function endGetWeather(code, temp) {
  return {
    type: '@WEATHER/END_GET_WEATHER',
    code,
    temp
  };
}
// reducers (pure)
...

// in UI
store.dispatch(startGetWeather());
const {code, temp} = ... // AJAX callback
store.dispatch(endGetWeather(code, temp));
```

Problems?

Sate management in UI again

```
$ npm install --save redux-thunk

// high-order action generator
export function getWeather() {
  return (dispatch, state) => {
    dispatch(startGetWeather());
    const {code, temp} = ... // AJAX callback
    dispatch(endGetWeather(code, temp));
  };
}
```

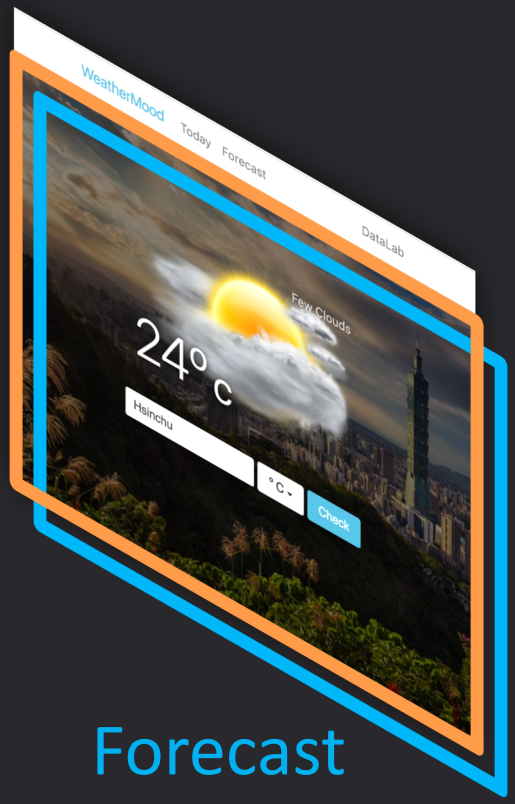
Dispatching Action Sequences

```
// in UI
import {compose, applyMiddleware} from 'redux';
import thunkMiddleware from 'redux-thunk';

const store = createStore(combineReducers({
  ...
}), compose(applyMiddleware(thunkMiddleware)));

store.dispatch(getWeather());
```

Outline

- WeatherMood: Posts
- Why Redux?
- Actions and Reducers
- Async Actions and Middleware
- Connecting with React Components
- Remarks

Today



Forecast

How?

```
@weather {
  city, code, temp. desc
}
@unit { unit }
@forecast { city, list }
```

Tedious Way

1. Create `store` in `Main`, then pass it down to all descendants
   - Lots of repeating props in JSX
2. In each component, call `store.subscribe()` and `dispatch()`
   - No `this.state` and `setState()`
   - Instead, use `this.forceUpdate()` and track when to re-render

React-Redux

```
$ npm install --save react-redux
```

```
// in Main.jsx
import {Provider} from 'react-redux';
render() {
  return (
    <Provider store={...}>...</Provider>
  );
}

// in Today.jsx
import {connect} from 'react-redux';
class Today extends React.Component {
  ... // has this.props.dispatch
}
export default connect(state => ({ // state to props
  ...state.weather,
  unit: state.unit
}))(Today);
```

- Only props in components

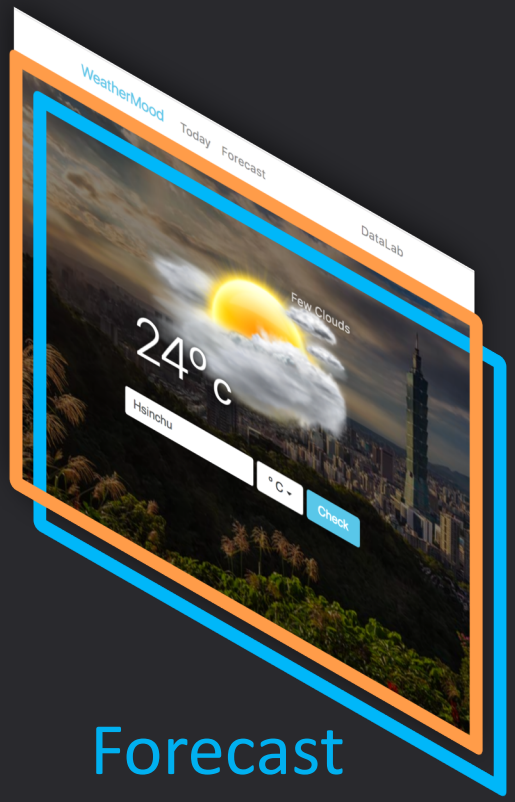
Outline

- WeatherMood: Posts
- Why Redux?
- Actions and Reducers
- Async Actions and Middleware
- Connecting with React Components
- Remarks

Remarks I

- Separation of concerns
- Components can be moved easily

Today

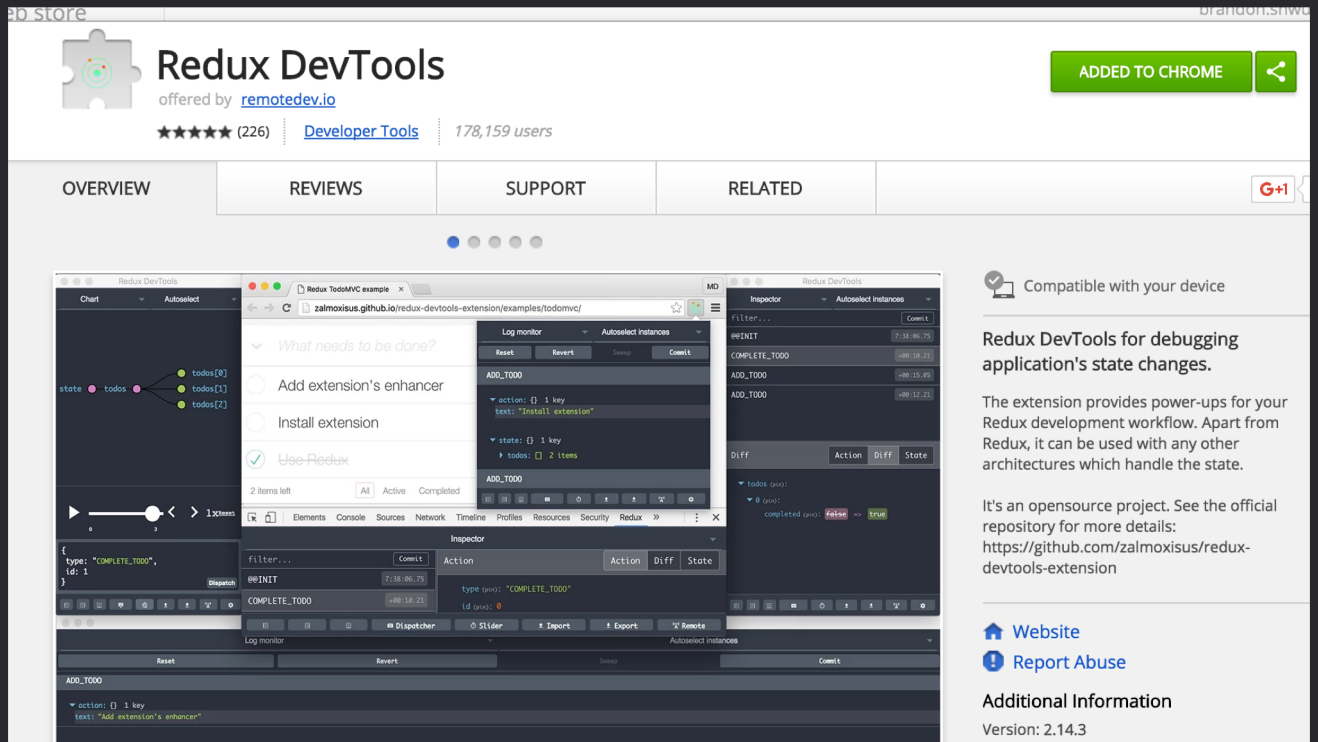


Forecast

```
@weather {
  city, code, temp. desc
}
@unit { unit }
@forecast { city, list }
```

Remarks II

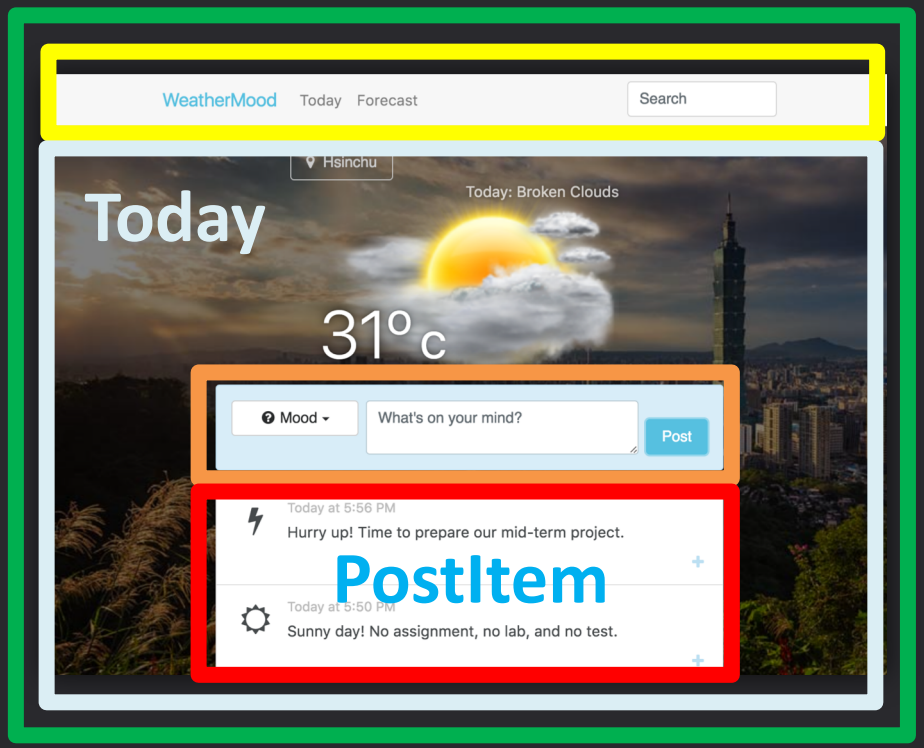
- States easy to inspect
- Explicit, sharable actions (stateful logic)
- Deterministic state transition => time travel



Readings

- Advanced Redux walkthrough (optional)
  - Async actions & flow
  - Middlewares
  - Usage with React Router
  - More examples

Assignment:
Post Components + Redux



Requirements

- Specify reducers and action types (with @'s) in README
- Setup store to allow time travel using Redux DevTools