

# Redux

**BY PANOT WONGKHOT**

**FRONT-END PROGRAMMER BOOTCAMP**

<https://github.com/panotza/frontend-bootcamp>



# Redux

---

คือ JavaScript Framework สำหรับการจัดการ state ของ application โดยมีจุดเด่นในเรื่อง Predictable, Logging, Time travel debugging เนื่องจากการรวม State ของ Application มาไว้ที่เดียว

สร้างโดย **Dan Abramov** และ **Andrew Clark** ปัจจุบันเป็น state management framework ที่ได้รับความนิยมสูงสุดสำหรับ React



# Three Principles

---

Redux มีหลักการอยู่สามข้อคือ

- **Single source of truth** เก็บ state ไว้ใน store เดียวเท่านั้น
- **State is read-only** การแก้ไข state มีได้แค่วิธีเดียวเท่านั้นคือการ dispatch action ดังนั้นส่วนที่เหลื่อใน app จะไม่ได้รับการอนุญาตให้แก้ไข state โดยตรง
- **Changes are made with pure functions** การแก้ไข state จะถูกจัดการด้วย pure function เท่านั้น เราเรียก function นี้ว่า reducer



Dan Abramov

Follow

Working on @reactjs. Co-author of Redux and Create React App. Building tools for humans.

Sep 20, 2016 · 3 min read

# You Might Not Need Redux

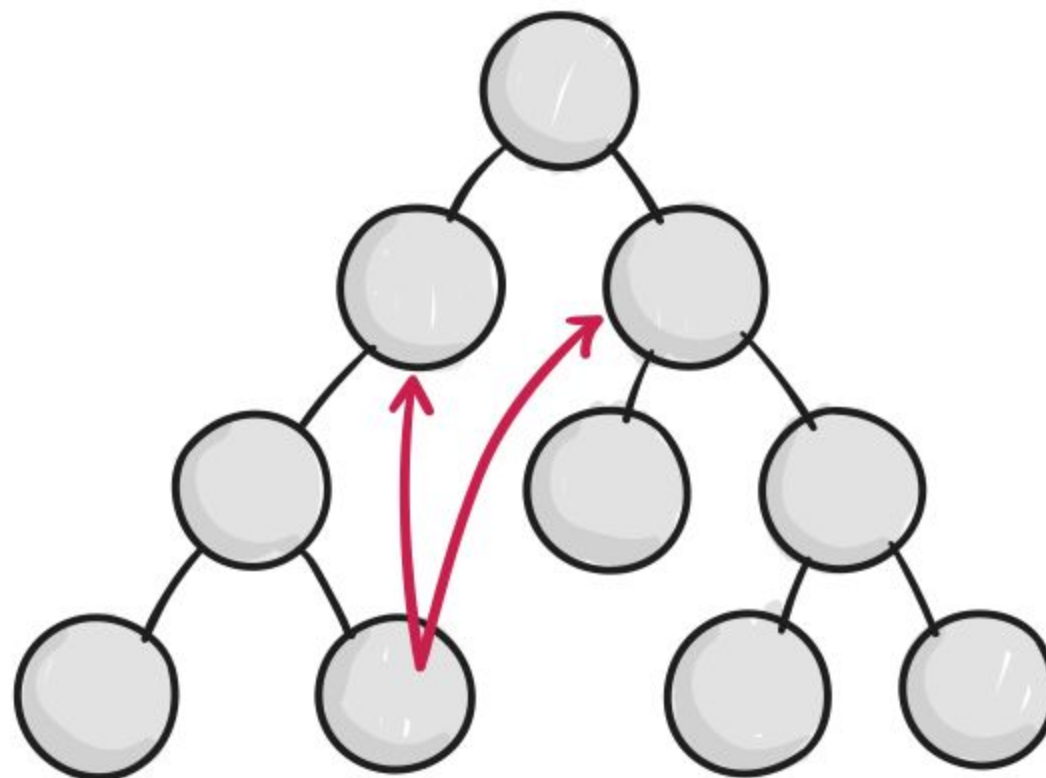
[https://medium.com/@dan\\_abramov/you-might-not-need-redux-be46360cf367](https://medium.com/@dan_abramov/you-might-not-need-redux-be46360cf367)



# Problems

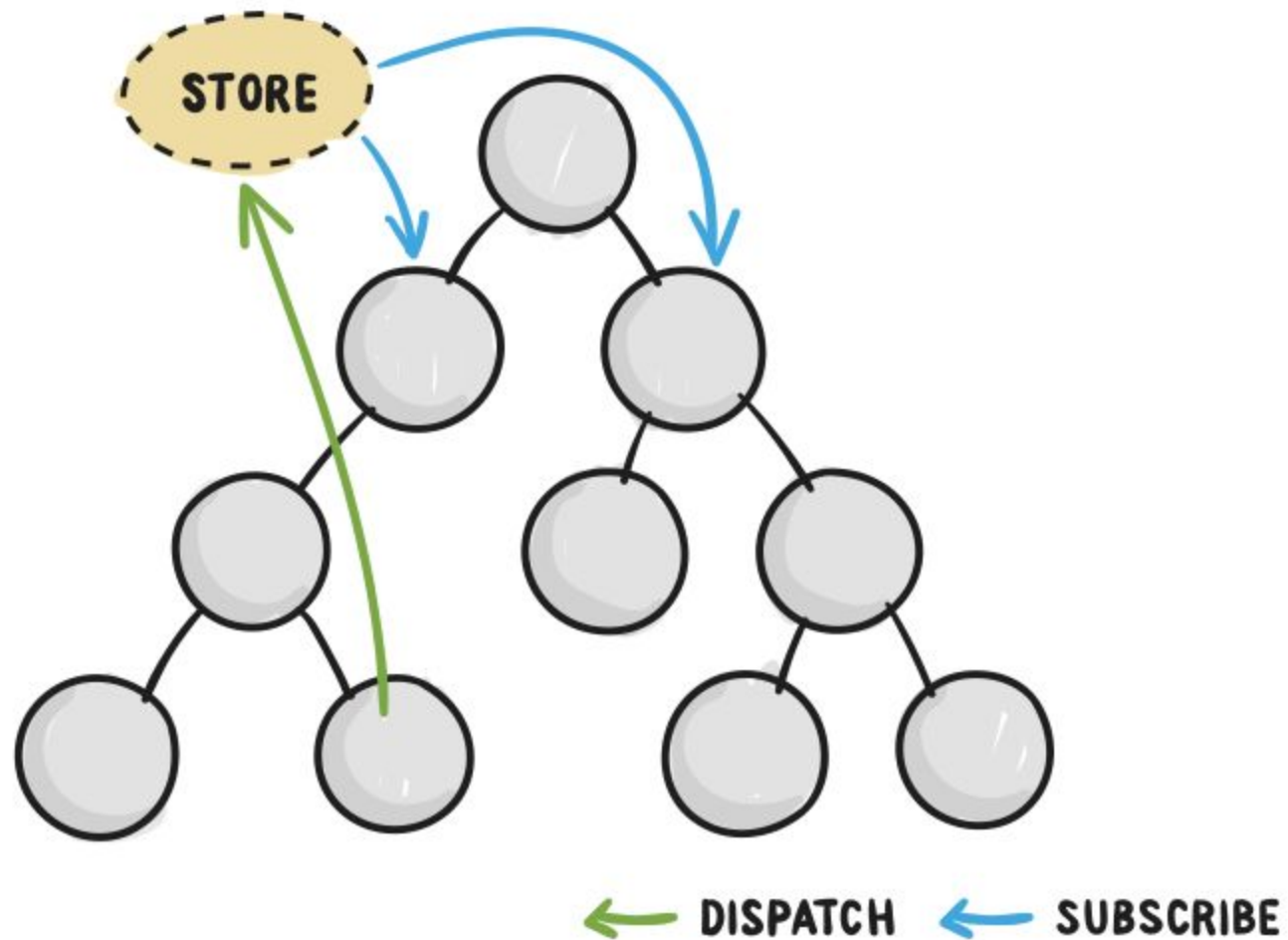






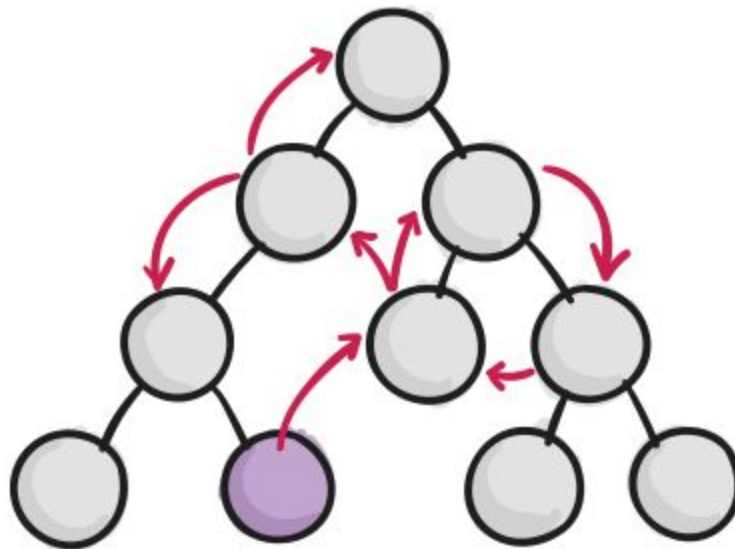
← **POOR PRACTICE WHEN COMPONENTS TRY TO  
COMMUNICATE DIRECTLY**

ภาพจาก <https://css-tricks.com/learning-react-redux/>

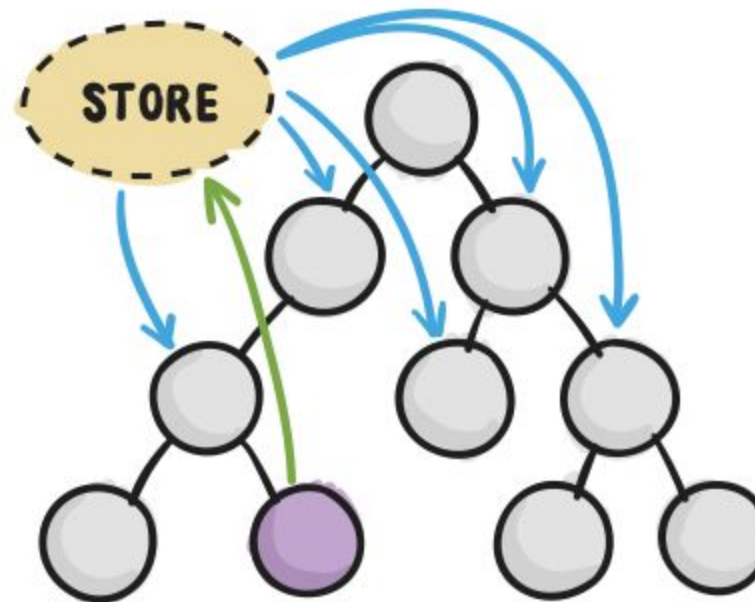


ภาพจาก <https://css-tricks.com/learning-react-redux/>

## WITHOUT REDUX



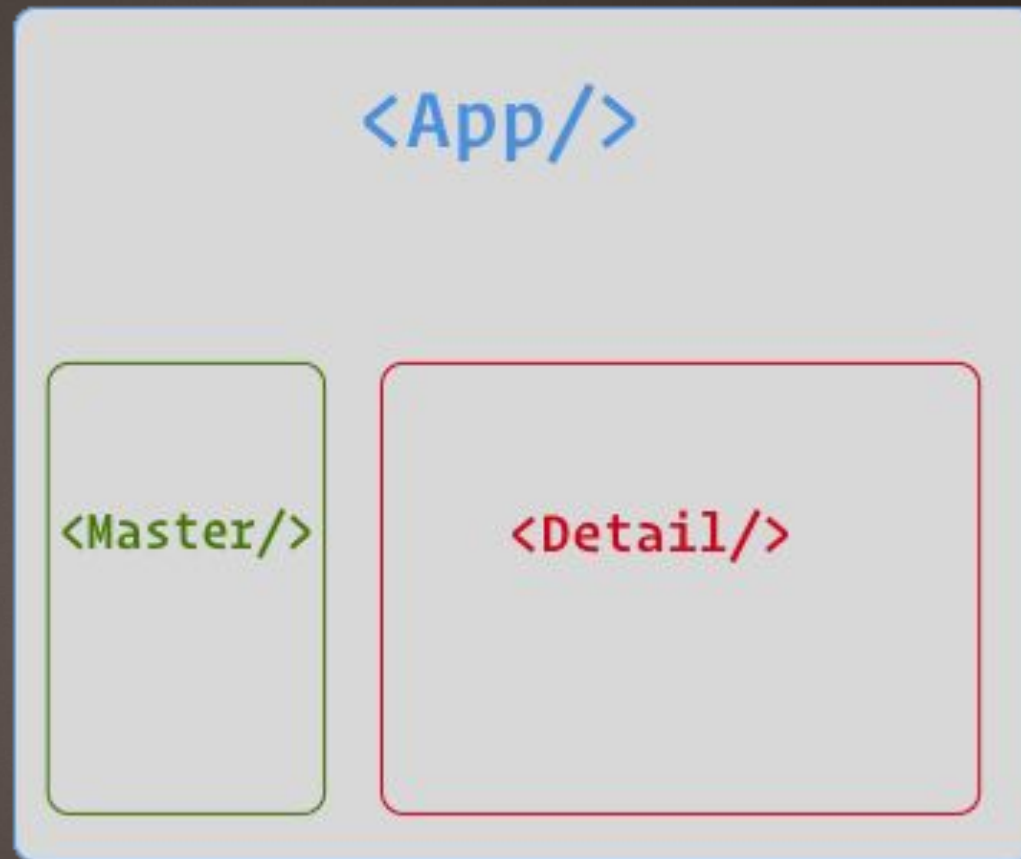
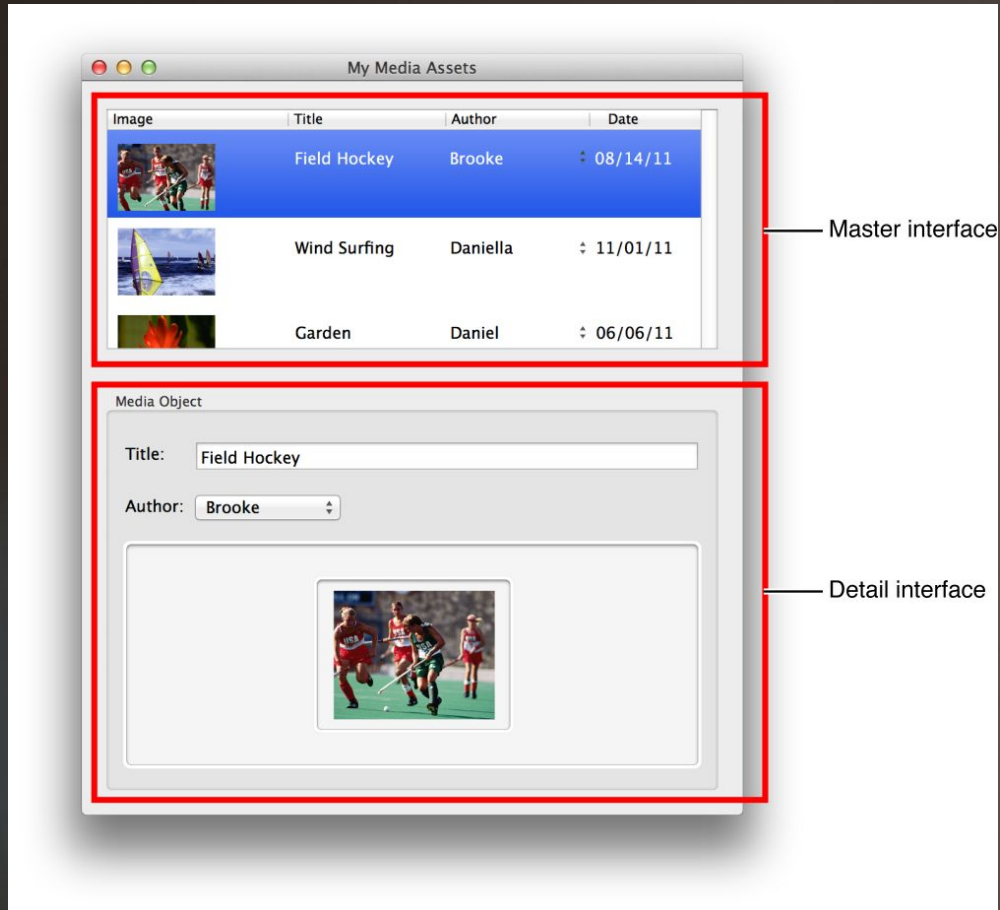
## WITH REDUX



 COMPONENT INITIATING CHANGE

ภาพจาก <https://css-tricks.com/learning-react-redux/>





ภาพจาก <http://blog.isquaredsoftware.com/presentations/2018-03-redux-fundamentals>

# จะใช้ Redux เมื่อไหร่ (Rule of thumb)

---

- มีจำนวนข้อมูลที่เปลี่ยนแปลงไปมาตลอดเวลา
- เราอยากได้ Single source of truth
- เมื่อรู้สึกว่าเก็บ state ไว้ที่ Parent บนสุดมันเริ่มลำบากในการทำงาน



# องค์ประกอบพื้นฐานของ Redux

---

- Store
- Action
- Reducer



# Store

---

Store คือที่สำหรับเก็บ State ของ Application

- อนุญาตให้อ่านค่า state ผ่าน Function `getState()`
- อนุญาตให้แก้ค่า state ผ่าน Function `dispatch(action)`
- ลงทะเบียน listener ผ่าน Function `subscribe(listener)`
- สร้าง Store ได้จาก Function `createStore(reducer)`

# Action

---

คือ **Object { }** ที่ใช้บอกว่าจะแก้ไข State อะไรโดย Action อย่างน้อยจะต้องประกอบด้วย key ชื่อว่า **type** เพื่อกำหนดชื่อให้ Action เช่น

```
{ type: "INCREMENT" }
```

\*โดยปกติแล้วเราสร้าง Action จาก **Action Creator**

\*เรานิยมกำหนดชื่อ type ด้วย snake case ตัวพิมพ์ใหญ่ทั้งหมด เช่น UPPER\_CASE



# Action

---



```
const myAction = { type: "INCREMENT", amount: 5 }
```

# Reducer

---

Reducer คือ **Function** ที่รับเอา State ปัจจุบันกับ Action เข้ามา ทำกระบวนการตาม type ที่กำหนดและ return State ใหม่ไปให้ Store

โดยปกติแล้วเราจะใช้ Switch case ในการช่วยแยก type ของ action

\*Reducer ทำงานแบบ **Sync**

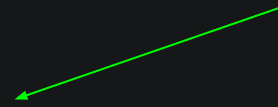
\*Reducer ต้องเขียนให้เป็น **Pure Function**



# Reducer



ต้องกำหนด state เริ่มต้นให้เสมอ



```
function myReducer(state = { count: 0 }, action) {  
  switch (action.type) {  
    case 'INCREMENT':  
      return { ...state, count: state.count + action.amount };  
    default:  
      return state;  
  }  
}
```

# Redux Function



# createStore

---

เป็น Function ที่ใช้สำหรับสร้าง Store โดยเราจะต้องส่ง **Reducer** ให้กับ Function นี้ด้วย เพื่อกำหนดว่า Store จะใช้ Reducer ตัวไหน

```
const store = createStore(myReducer);
```



# Action Creator

---

Action Creator คือ Function ที่ใช้สร้าง Action

```
function IncrementBy(number) {  
    return { type: 'INCREMENT', amount: number };  
}
```



# Dispatch

---

Dispatch คือ Function ที่ส่ง Action ไปหา Reducer

```
store.dispatch({ type: "INCREMENT", amount: 5 })
```



# getState

---

getState เป็น Getter Function ของ Store เราใช้สำหรับอ่านค่า State จาก Store

```
console.log(store.getState());
```



# Subscribe

---

Subscribe คือ Function ที่รับ Listener (**Callback Function**) เข้าไปลงทะเบียนไว้กับ Store เมื่อ State มีการเปลี่ยนแปลง Store จะเรียก Listener ที่ลงทะเบียนไว้

```
store.subscribe(myListener);
```



# Redux Flow





# Redux Flow ฉบับเข้าใจง่าย

เปรียบว่า

Store = ธนาคาร

มีห้องเก็บเงิน (State)



มีพนักงานธนาคาร (Reducer)



มีใบธุรกรรม (Action)



# Redux Flow ฉบับเข้าใจง่าย

---

Flow การทำงานของ Redux ก็เหมือนกับการไปธนาคาร  
เราเปิดบัญชี (Subscribe) เขียนใบธุรกรรม (Action) แล้วก็ส่ง  
(Dispatch) ให้กับพนักงานธนาคาร (Reducer) พนักงานก็ทำการแก้ไข  
ยอดเงิน (State) ให้กับเรา เราสามารถขอดูยอดเงินปัจจุบันได้ (getState)



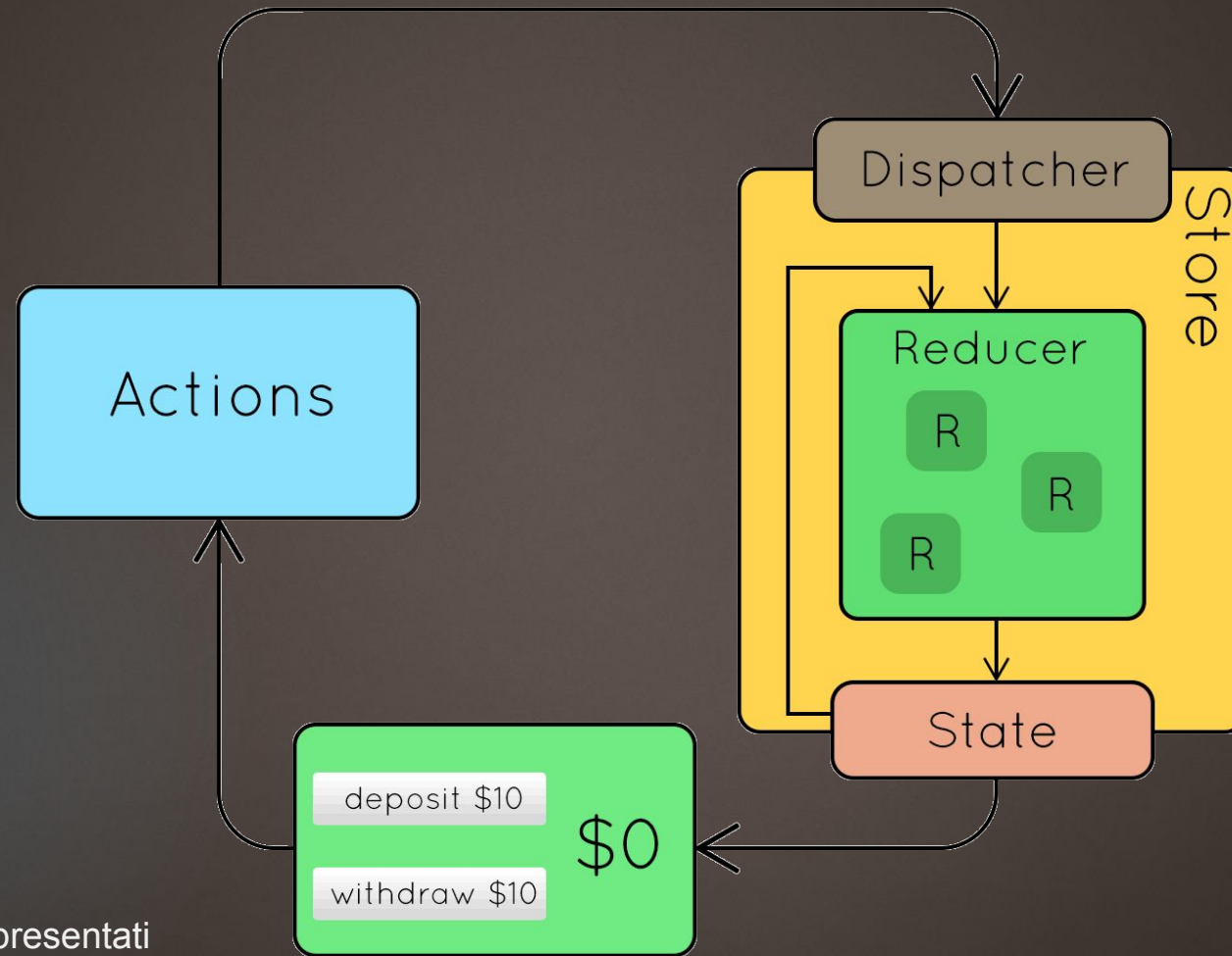
# ສຸບ Flow ບວງ Redux

---

Subscribe => Action => Dispatch => Reducer => getState



# Flow ของ Redux



ภาพจาก  
<http://blog.isquaredsoftware.com/presentations/2018-03-redux-fundamentals>

# Step ๓ การใช้ Redux






# Step ការ​ໃຊ້ Redux

---

1. ສ້າງ Action
2. ສ້າງ Reducer
3. ໃຊ້ createStore ໃນການສ້າງ Store
4. Subscribe Listener
5. Dispatch Action



# สร้าง Action

 ดูได้ใน `redux-example.html`

```
function IncrementBy(number) {  
  return { type: 'INCREMENT', amount: number };  
}
```

# สร้าง Reducer

● ● ● ดูได้ใน redux-example.html ต้องกำหนด state เริ่มต้นให้เสมอ

```
function myReducer(state = { count: 0 }, action) {  
  switch (action.type) {  
    case 'INCREMENT':  
      return { ...state, count: state.count + action.amount };  
    default:  
      return state;  
  }  
}
```

# createStore



ดูได้ใน redux-example.html

```
const store = createStore(myReducer);
```

# Subscribe Listener

   ดูได้ใน `redux-example.html`

```
function myListener() {  
  // อ่านค่า state จาก store  
  console.log(store.getState());  
  console.log('-----');  
}  
  
store.subscribe(myListener);
```

# Dispatch Action



ดูได้ใน redux-example.html

```
store.dispatch(IncrementBy(2));
```



# Lab 1: Redux

---

เปิดไฟล์ lab1.html

- เปิดไฟล์ redux-example.html ขึ้นมาดูประกอบ
- หัดเขียน Action
- หัดเขียน Reducer
- หัด Dispatch Action ไปหา Reducer
- เพิ่ม Action DECREMENT โดยให้ state count - 2



# React-Redux



# React-Redux

---

Redux เเดียวๆไม่สามารถใช้งานกับ React ได้ React-Redux จึงเป็น Library ที่ทำหน้าที่เหมือนกับสะพานที่เชื่อม React กับ Redux ทำให้ทั้งสอง Library ทำงานด้วยกันได้



# React-Redux

---

React-Redux มี API ให้ 2 ตัวคือ

- Component <Provider />
- connect()



# <Provider />

---

Provider เป็น component ใช้สำหรับกำหนด store ให้ application  
วิธีการใช้ก็คือการกำหนด props ชื่อ store = store ที่จะใช้  
โดยปกติแล้วเราจะใช้ Provider เป็น **parent component** ใหญ่ที่สุดใน  
application เพื่อให้ children component ติดต่อกับ Redux ผ่าน  
function connect ได้

# <Provider />

---



```
<Provider store={store}>
```

```
  <App />
```

```
</Provider>
```



# connect

---

```
connect (mapStateToProps, mapDispatchToProps)
```

ใช้เพื่อเชื่อมต่อ Component เข้ากับ Redux store โดยใช้ Pattern HOC

connect มี 2 parameter ที่ใช้บ่อยคือ

- mapStateToProps (ขอ state จาก store)
- mapDispatchToProps (ขอ function dispatch จาก store)

# mapStateToProps

---

เป็น parameter ตำแหน่งที่ 1 ของ connect

- mapStateToProps เป็น **Callback Function** ที่ส่งเข้าไปเพื่อ subscribe Redux store ดังนั้นเราจะได้รับ state จาก function นี้ ใน parameter ตำแหน่งที่ 1
- การ return จาก mapStateToProps หมายถึงการแปลงจาก state ไปเป็น props ให้กับ component

# mapDispatchToProps

เป็น parameter ตำแหน่งที่ 2 ของ connect

- หาก mapDispatchToProps เป็น **Object** ถูกส่งเข้าไปจะถือว่าทุก Function ใน Object เป็น Action Creator ทั้งหมด react-redux จะ implement dispatch และแปลงเป็น props ให้อัตโนมัติ
- หาก mapDispatchToProps เป็น **Callback Function** ถูกส่งเข้าไป เราจะได้รับ dispatch ใน parameter ตำแหน่งที่ 1 เราต้องทำการ **implement** การ dispatch ของ action เอง สามารถใช้ function **bindActionCreators** ช่วยได้ การ return จาก mapDispatchToProps หมายถึงการส่ง action ที่ implement แล้วไปเป็น props ให้กับ component

# mapDispatchToProps

---

- หากไม่ส่งอะไรเข้าไปเราจะได้ function dispatch จาก store มาเป็น props โดย default



# ตัวอย่างการ connect



# connect



```
export default connect() (TodoApp) ;
```

การ connect แบบนี้เท่ากับเราไม่ subscribe กับ store  
แต่เราจะได้ props.dispatch()



# connect



```
function mapStateToProps(state) {  
  return { todos: state.todos };  
}
```

```
export default connect(mapStateToProps)(TodoApp);
```

การ connect แบบนี้เท่ากับเรา subscribe กับ store แล้วเลือกเอาแค่ todos จาก state มาเป็น props พร้อมกับ ได้ props.dispatch()



## ในไฟล์ actionCreators.js

```
export function addTodo(text) {  
  return { type: 'ADD_TODO', text }  
}  
  
export function toggleTodo(index) {  
  return { type: 'TOGGLE_TODO', index }  
}  
  
export function deleteTodo(index) {  
  return { type: 'DELETE_TODO', index }  
}
```

# connect



```
import * as actionCreators from './actionCreators';  
  
export default connect(null, actionCreators)(TodoApp);
```

การ connect แบบนี้เท่ากับเราไม่ subscribe กับ store  
แต่เราจะได้ action creator ที่ถูก implement dispatch() มาแล้วทั้งหมดใส่  
ลงใน props


# connect



```
import { addToDo, deleteToDo } from './actionCreators';  
const mapDispatchToProps = {  
  addToDo,  
  deleteToDo  
}  
  
export default connect(null, mapDispatchToProps)(ToDoApp)
```

เหมือนแบบที่แล้วเพียงแค่ว่าเราเลือก action creator บางตัวมาใส่ใน props เท่านั้น

# connect



```
import * as actionCreators from './actionCreators';  
function mapDispatchToProps(dispatch) {  
  return {  
    actions: bindActionCreators(actionCreators, dispatch)  
  };  
}  
  
export default connect(null, mapDispatchToProps)(TodoApp)
```

เราใช้ Function bindActionCreators ในการ implement Action Creator กับ dispatch แล้วนำมาเก็บไว้ใน key actions

# connect



```
import * as todoActionCreators from './todoActionCreators';
import * as counterActionCreators from './counterActionCreators';
import { bindActionCreators } from 'redux';
function mapDispatchToProps(dispatch) {
  return {
    todoActions: bindActionCreators(todoActionCreators, dispatch),
    counterActions: bindActionCreators(counterActionCreators, dispatch)
  }
}
export default connect(null, mapDispatchToProps)(TodoApp);
```



# connect ภาคต่อ



เราใช้ Function `bindActionCreators` ในการ implement Action Creator กับ `dispatch` จากทั้ง `todoActionCreators` และ `counterActionCreators` แล้วนำมาเก็บไว้ใน key `todoActions`, `counterActions`

# Redux with React



# การใช้งาน Redux กับ React


---

หลังจากสร้าง project ด้วย create-react-app เราต้องลง dependency เพิ่ม 2 ตัวดังนี้

- redux
- react-redux



# การใช้งาน Redux กับ React

 เปิด Terminal ใน folder project

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

หรือ

```
yarn add redux react-redux
```

# Smart & Dumb Component



# Smart & Dumb Component

---

เป็น Pattern ในการแบ่งประเภท Component ออกเป็น 2 แบบคือ  
**Smart** Component และ **Dumb** Component





# Dumb Component

---

- ทำหน้าที่กำหนดรูปร่างหน้าตาของ UI
- จะมี Children Component เป็น Smart หรือ Dumb Component ก็ได้
- มี DOM markup และมี style
- รับ data และ callback จาก props
- จะไม่มี logic ของการแก้ไข data อยู่ในตัวเอง
- ไม่ค่อยมี state เป็นของตัวเอง (ถ้ามีจะเป็น state ที่เกี่ยวข้องกับ UI เช่น tab index)
- เขียน Component จาก Function (ยกเว้นในกรณีที่ต้องการ Life Cycle)
- ปกติแล้วจะอยู่ใน Folder components
- มีอีกชื่อหนึ่งคือ Presentational Component

# Smart Component

---

- ทำหน้าที่จัดการเกี่ยวกับ logic เช่นเชื่อมต่อกับ Redux Store
- จะมี Children Component เป็น Smart หรือ Dumb Component ก็ได้
- แต่ปกติไม่มี DOM markup เป็นของตัวเอง และ ไม่มี style ถ้ามีจะเป็นแค่ div ครอบ children component ที่เรียกใช้
- ทำหน้าที่ส่งข้อมูลให้กับ Dumb Component หรือ Smart Component ตัวอื่น
- ทำหน้าที่ส่ง Action ให้กับ Dumb Component
- ปกติแล้ว Smart Component จะถูกสร้างขึ้นจาก HOC เช่น connect
- ปกติแล้วจะอยู่ใน Folder containers
- มีอีกชื่อหนึ่งคือ Container Component

# Constants



# Constants

---

ใน React เราย้ายเขียนชื่อ Action ด้วย Constant และ export ออกไปใช้ที่อื่นเช่น

```
export const ADD_TODO = 'ADD_TODO';  
export const DELETE_TODO = 'DELETE_TODO';  
export const EDIT_TODO = 'EDIT_TODO';
```



# ตัวอย่างการใช้งาน

ตัวอย่างต่อไปนี้ดูฉบับเต็มได้ที่ Folder  
react-redux-example





```
└─ src
  └─ actions
    JS counter.js
  └─ components
    # App.css
    JS App.js
    JS Button.js
    JS Counter.js
  └─ containers
    JS ButtonContainer.js
    JS CounterContainer.js
  └─ reducers
    JS counter.js
    JS index.js
    JS todos.js
  JS index.js
```





ใน src/actions/counter.js

```
export const COUNTER_INCREMENT = 'INCREMENT';
```

```
export function incrementBy(number) {  
  return { type: COUNTER_INCREMENT, amount: number };  
}
```



ใน src/reducers/index.js

```
import { COUNTER_INCREMENT } from '../actions/counter';

function rootReducer(state = { count: 0 }, action) {
  switch (action.type) {
    case COUNTER_INCREMENT:
      return { ...state, count: state.count + action.amount };
    default:
      return state;
  }
}

export default rootReducer;
```



ใน src/index.js

```
import { createStore } from "redux";
import { Provider } from 'react-redux';
import rootReducer from './reducers';
import App from './components/App';

const store = createStore(rootReducer);
ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>, document.getElementById('root'));
```



ใน src/components/App.js

```
import React from 'react';
import './App.css';
import CounterContainer from '../containers/CounterContainer';
import ButtonContainer from '../containers/ButtonContainer';

const App = () => (
  <div className="App">
    <CounterContainer />
    <ButtonContainer />
  </div>
);

export default App;
```



ใน src/containers/CounterContainers.js

```
import { connect } from 'react-redux';  
import Counter from '../components/Counter';  
  
const mapStateToProps = state => ({  
  count: state.count  
});  
  
export default connect(mapStateToProps)(Counter);
```

 ใน src/components/Counter.js

```
import React from 'react';

const Counter = (props) => (
  <div>{props.count}</div>
);

export default Counter;
```





ใน src/containers/ButtonContainer.js

```
import { connect } from 'react-redux';  
import { incrementBy } from '../actions/counter';  
import Button from '../components/Button';  
  
const mapDispatchToProps = {  
  incrementBy  
};  
  
export default connect(null, mapDispatchToProps)(Button);
```



ใน src/components/Button.js

```
import React from 'react';
```

```
const Button = (props) => (
```

```
  <button onClick={() => props.incrementBy(2)}>increment</button>  
);
```

```
export default Button;
```

# combineReducers



# combineReducers

---

ในกรณีที่ app เราขยายใหญ่ขึ้น เราอาจจะแยก reducer ออกเป็นส่วนย่อยๆ เพื่อจัดการ state ในแต่ละส่วน โดยใช้ Function combineReducers





ใน src/reducers/index.js

```
import { COUNTER_INCREMENT } from '../actions/counter';

function rootReducer(state = { count: 0 }, action) {
  switch (action.type) {
    case COUNTER_INCREMENT:
      return { ...state, count: state.count + action.amount };
    default:
      return state;
  }
}

export default rootReducer;
```

บอเพิ่มเติม



ใน src/reducers/index.js

```
import { combineReducers } from 'redux';  
import todos from './todos';  
import counter from './counter';  
  
export default combineReducers({  
  todos, // มีค่าเท่ากับ todos:  
  count: counter  
});
```





ใน src/reducers/counter.js

```
import { COUNTER_INCREMENT } from '../actions/counter';

function counter(state = 0, action) {
  switch (action.type) {
    case COUNTER_INCREMENT:
      return state + action.amount;
    default:
      return state;
  }
}

export default counter;
```



ใน src/reducers/todos.js

```
export default function todos(state = [], action) {  
  switch (action.type) {  
    case 'ADD_TODO':  
      return [...state, action.text];  
    default:  
      return state;  
  }  
}
```

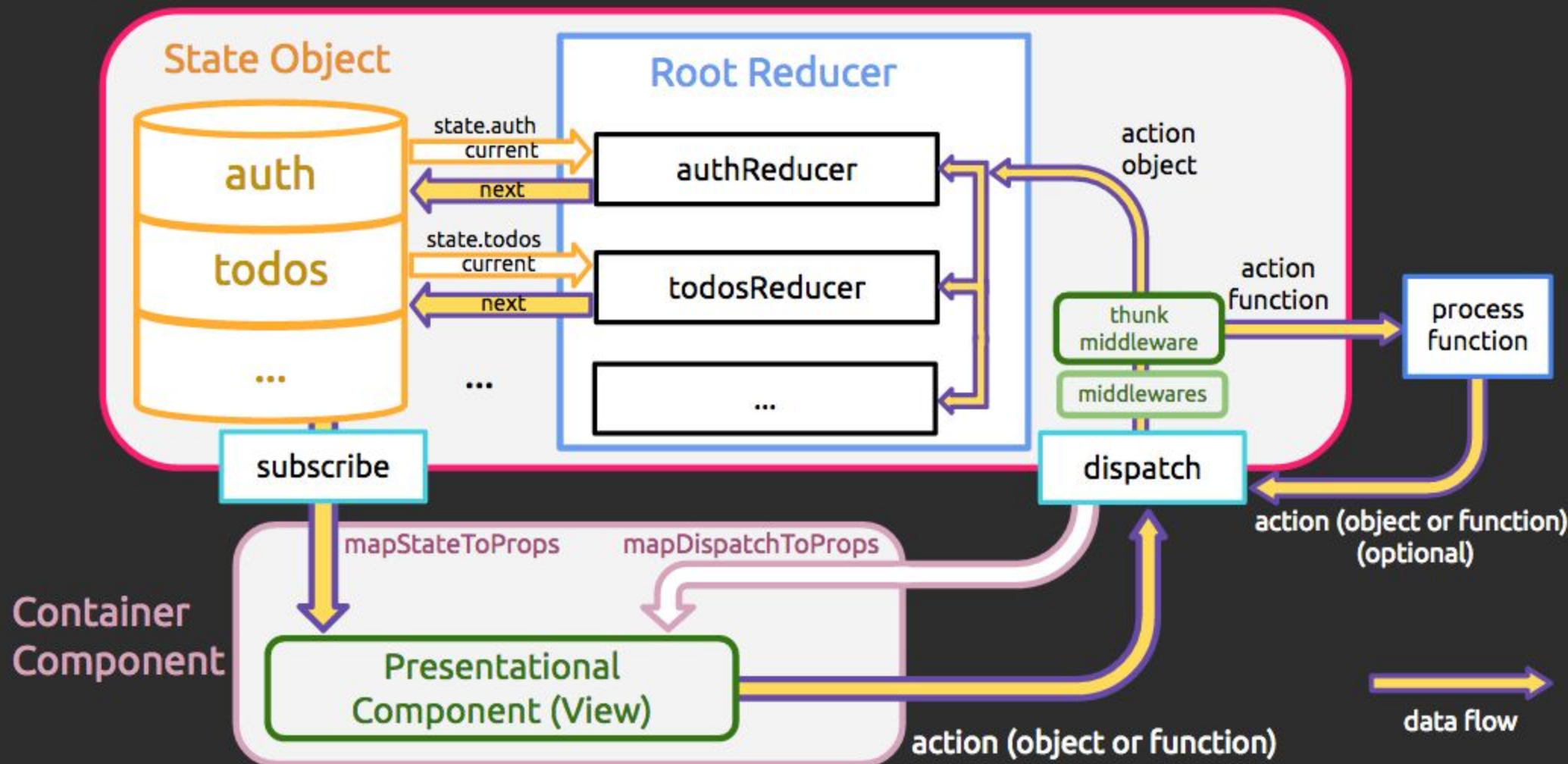


จะให้ผลลัพธ์ state แบบนี้

```
{  
  todos: [] //จัดการโดย todos reducer  
  count: 0 //จัดการโดย counter reducer  
}
```

Store

# Redux Architecture



ภาพจาก <http://codesheep.io/2017/01/06/redux-architecture/>



# createStore



# CreateStore

```
createStore(reducer, preloadedState, enhancer)
```

นอกจากกำหนด Reducer ที่จะกับ Store แล้ว เรายังสามารถกำหนด **State** เริ่มต้นได้ที่ parameter ตำแหน่งที่ 2

และสามารถกำหนด **enhancer** (เช่น middleware logger) ได้ที่ parameter ตำแหน่งที่ 3

\* หากเราอยากกำหนด enhancer แต่ไม่อยากกำหนด State เริ่มต้น เราสามารถใส่ enhancer ในตำแหน่งที่ 2 ได้เลย





```
import { applyMiddleware, createStore } from 'redux';  
import logger from 'redux-logger';  
const store = createStore(  
  reducer,  
  { count: 0 },  
  applyMiddleware(logger)  
);
```

# เมื่อไหร่จะเก็บ ข้อมูลไว้ ใน Redux ?



# การเก็บข้อมูลใน Redux (Rule of thumb)

---

Redux ไม่ได้บังคับว่าจะต้องเก็บ State ของ Component ทั้งหมดไว้ใน Redux store ดังนั้นเราจึงสามารถใช้ State ของ Component ได้ตามปกติ แต่เราก็มี Rule of thumb ในการกำหนดว่า ข้อมูลแบบไหนควรจะเก็บไว้ใน Redux store บ้าง

- เมื่อส่วนอื่นๆของ App อยากจะใช้ข้อมูลนี้
- เมื่ออยากจะใช้ความสามารถ time travel debugging ในการย้อนข้อมูลกลับไปมา
- เมื่ออยากจะ cache ข้อมูลเอาไว้จะได้ไม่ต้อง reload ใหม่



# Lab 2: Redux with React

---

เปิด folder lab2

- ศึกษา code ที่ให้ไป
- เพิ่มปุ่ม decrement ให้ state count - 2



# สำหรับผู้ที่คุณเคยกับ Redux แล้ว



# สิ่งที่ควรรู้หลังจากใช้ Redux เป็นแล้ว

---

- Ducks Modular Redux
- React Router Redux
- Redux Middleware (Redux Logger, Redux Thunk, Redux Saga)
- Redux Form
- Immutable.JS with Redux





# การบ้าน #1 - Todo list

---

- เปิด folder **homework1** ขึ้นมา
- project ที่ให้ยังไม่ได้ลง redux ต้องลงเพิ่มเอง
- ศึกษา code ที่ให้ไป
- ทำการ refactor จากการเก็บ state ที่ **app** มาไว้ที่ **redux store** แทน
- state ของ input box ไม่ต้องเอามาเก็บใน redux store
- สามารถดูตัวอย่างจาก link นี้ในการช่วยทำได้
- <https://codesandbox.io/s/github/reactjs/redux/tree/master/examples/todo>



# การบ้าน #2 - Recipe App with Redux

---

- จากการบ้าน Recipe เวอร์ชันเต็ม
- ให้ทำการ Refactor Recipe App ที่เราได้สร้างไป ด้วยการย้าย State Recipe มาไว้ใน Redux store
- State ของ Input Form Add Recipe ไม่ต้องนำมาเก็บไว้ใน Redux store ก็ได้
- ใครที่ยังทำไม่เสร็จให้ทำของเก่าให้เสร็จก่อนนะจ๊ะ



# การบ้าน #3 - Kaimaidee App

- ใช้ create-react-app ในการสร้าง project
- ให้ใช้ **Redux** และ **Redux Router**
- สร้าง App Kaimaidee e-commerce ให้มี **Feature** ดังนี้
- มี **Navbar** กดที่ My Cart จะเข้าหน้า My Cart กดที่ชื่อ Kaimaidee จะกลับหน้า Home
- ที่ปุ่ม My Cart จะมี **Badge** แสดงจำนวนสินค้าใน cart ตอนนี้ **หากยังไม่มีจะไม่แสดง**
- หน้า Home มีรายการสินค้า มีปุ่มกด **Add cart** กดแล้วจะเพิ่มสินค้าเข้า **cart** ที่ละ 1 ชิ้น
- หน้า My Cart จะแสดงรายการในตะกร้า
- แสดง ID, Title, Price, Quantity, Total, Total Quantity, Total Price
- เก็บ **State** เข้า **Redux Store** ทั้งหมด ห้ามมี **State** ใน **Component**

ได้แรงบันดาลใจมาจาก

<https://codesandbox.io/s/github/reactjs/redux/tree/master/examples/shopping-cart>



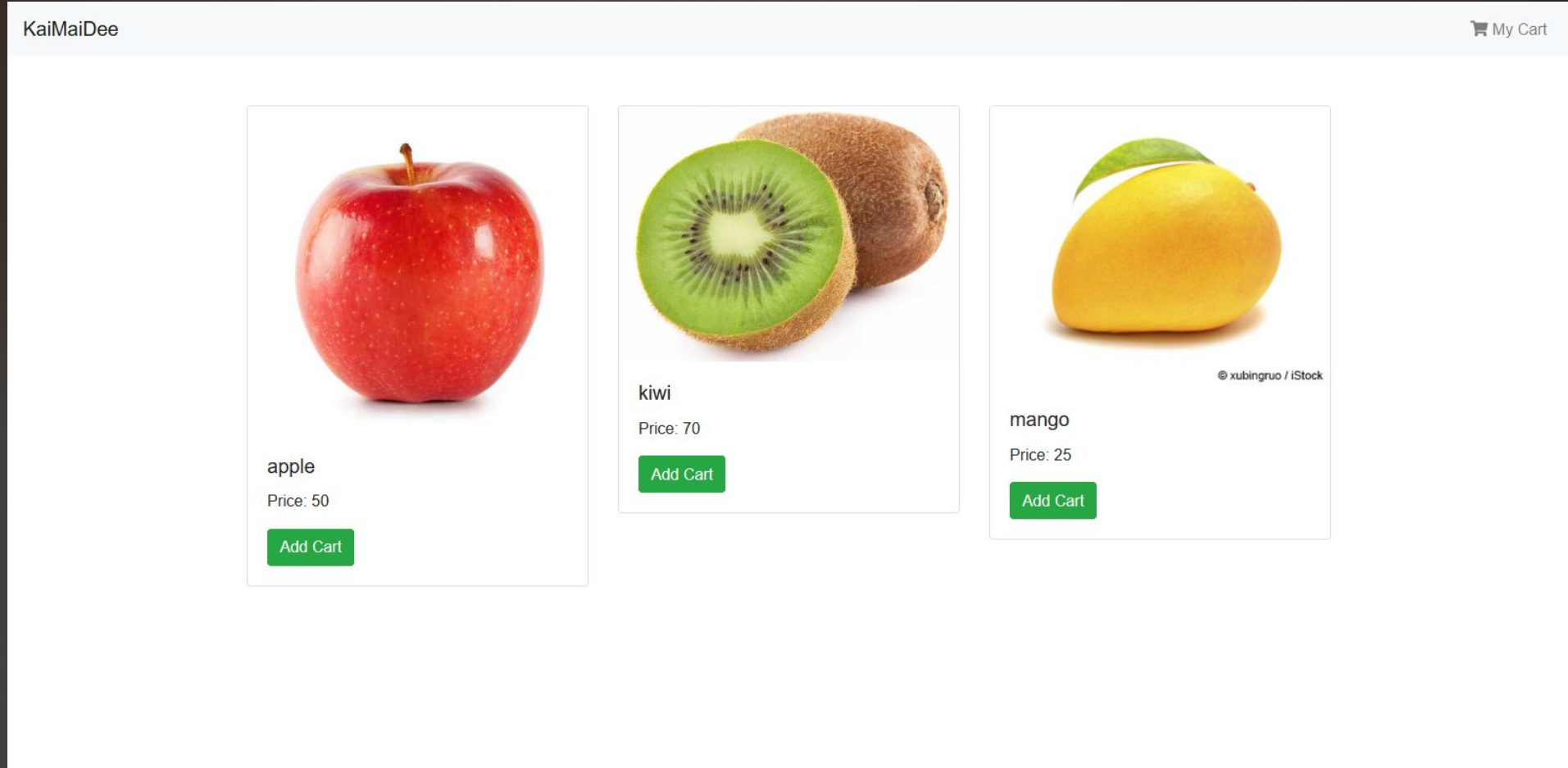
# การบ้าน #3 - Kaimaidee App

state เริ่มต้นของ Product

```
const initialState = [  
  { id: 1, title: 'apple', price: 50, img:  
    'https://www.thealternativedaily.com/pages/images/acv/img-apple1.jpg' },  
  { id: 2, title: 'kiwi', price: 70, img:  
    'https://www.organicfacts.net/wp-content/uploads/2013/07/Kiwi.jpg' },  
  { id: 3, title: 'mango', price: 25, img:  
    'https://media.mercola.com/assets/images/foodfacts/mango-nutrition-facts.jpg' }  
];
```



# การบ้าน #3 - Kaimaidee App



ดูภาพขนาดใหญ่ได้ที่ <https://ibb.co/mic4Rc>



# การบ้าน #3 - Kaimaidee App

KaiMaiDee 13 My Cart

### My Cart

ID	Title	Price	Quantity	Total
1	apple	50	4	200
2	kiwi	70	4	280
3	mango	25	5	125
Total			13	605

ดูภาพขนาดใหญ่ได้ที่ <https://ibb.co/mSNFXH>





# การบ้าน #4 - Snake Game

---

- การบ้านนี้ Optional ทำหรือไม่ทำก็ได้
- ใช้ React ในการสร้างเกมงู แบบที่เราเคยเล่นกันบน Nokia 3310
- งูชนกรอบแล้วตาย ชนหางตัวเองแล้วตาย
- ลองคิด Algorithm เองดูก่อน copy จาก Internet นะครับ
- ไม่บังคับว่าจะต้องใช้ Redux



# สำหรับหาความรู้เพิ่มเติม

redux-architecture โดยพีปิง อดีตนายกสมาคมโปรแกรมเมอร์ไทย

<http://codesheep.io/2017/01/06/redux-architecture/>

รวมกระบวนท่าในการ connect

<https://github.com/reactjs/react-redux/blob/master/docs/api.md#api>

Smart and Dumb component

[https://medium.com/@dan\\_abramov/smart-and-dumb-components-7ca2f9a7c7d0](https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0)

