Chapter 3: React Basics

Redux introduction

Sesi 2

PERHATIAN

Jangan lupa untuk isi form absensi. Bagi instruktur mohon mengisi form absensi yang telah diinfokan dan konfirmasi nomor urut peserta yang hadir.

Untuk student harap mengisi form absensi **di sini** sebelum kelas dimulai. Untuk kode peserta dapat ditanyakan kepada instruktur dan jangan lupa mencantumkan urutan pertemuannya.

Redux

Front-end Architecture Timeline

- 80-an kita punya MVC
- 90-an ada MVP
- 2005 ada MVVM, VIPER
- 2009 ada DCI data, context interaction
- 2013 Flux muncul

Kenapa banyak bermunculan berbagai arsitektur? Dan pasti akan muncul lagi arsitektur-arsitektur lainnya nanti di tahun-tahun mendatang. Yang pasti karena managing state itu adalah sesuatu yang kritikal, dan tidak mudah. Makanya banyak yang menciptakan solusi untuk itu

Flux Architecture Flavors

Kita akan fokus ke flux architecture. Nah Flux ini ada banyak rasa, ada:

- Flux
- Redux
- MobX
- Unstated
- banyak lagi

Kita akan fokus ke Redux, karena untuk saat ini yg jadi de-facto-nya Redux ditambah Dan Abramov yang bikin Redux direkrut Facebook dan bekerja fulltime di React, Redux dan teman-temannya.

When Do I Need Redux?

- Complex data flow
- Many actions
- Same data used in multiple places

Dengan Flux, perubahan menjadi lebih terprediksi.

Dan dengan Redux data jadi terpusat, adanya sudah pasti di Store. Dan sedikit berbeda dengan library lain seperti Flux, dan MobX, Redux ini store nya 1 doang. Semua data ada di 1 tempat.

Redux Principles

Single Source of Truth

State global dari Front End disimpang di sebuah object tree dengan sebuah store (single store)

State is Read Only

Satu-satunya cara untuk merubah state adalah melalui action. Action adalah sebuah object yang menggambarkan apa yang harus terjadi

Changes are made with pure functions

Untuk menjelaskan secara detail bagaimana sebuah state dapat di “ubah” oleh action, kita harus menulis sebuah reducer

… atau dengan kata lain :

One Immutable Store

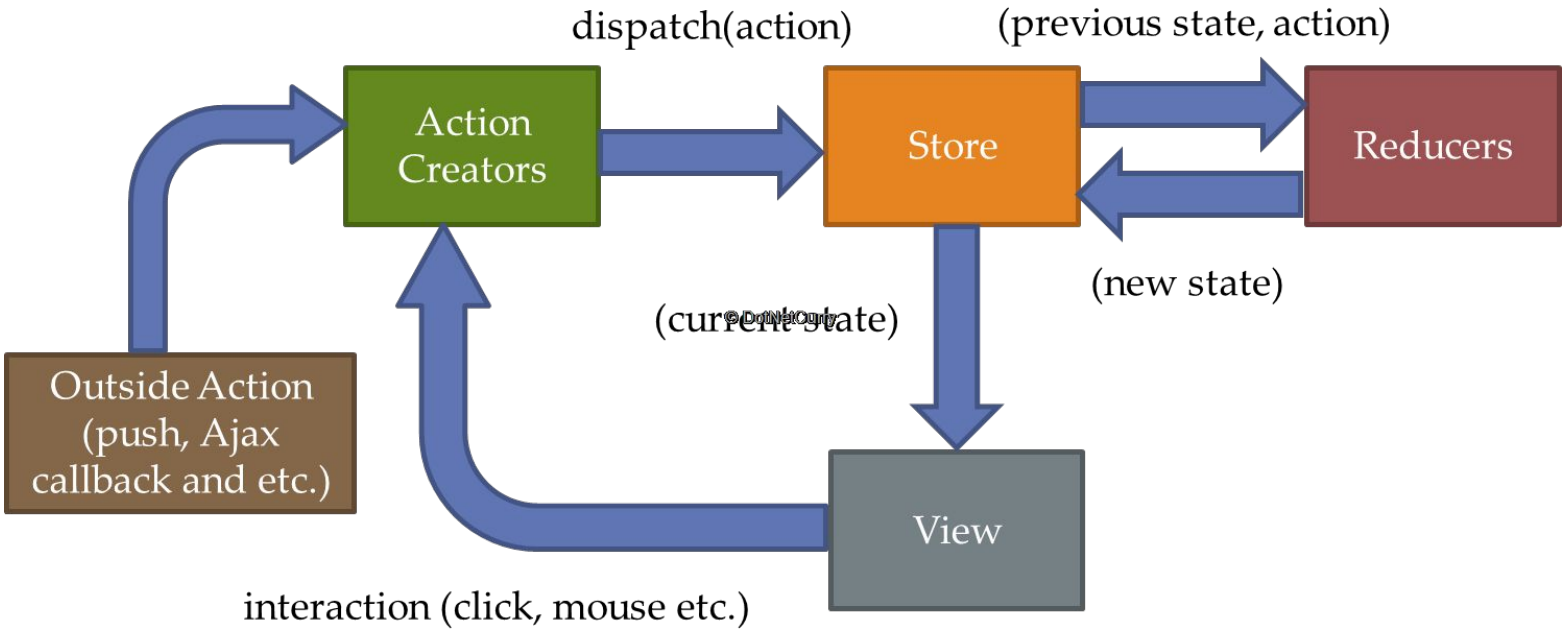
Store bersifat immutable, kita tidak bisa merubahnya secara langsung dari view, harus melalui actions

Actions Trigger Changes

Action adalah plain JS object yang menjelaskan perubahan yang harus dilakukan. Contoh: increment, decrement, add todo, edit todo, delete todo.

Reducer update the state

Apa itu reducer? Reducer itu hanyalah fungsi javascript biasa yang harus kita buat. Parameter-nya ada 2: current state dan action apa yang terjadi. Misalnya kita klik tanda tambah, artinya reducernya akan mengambil state yang sekarang, misalnya 1, terus action nya increment, jadi si Reducer akan me-return state baru yang adalah 1+1 = 2



Introduction

Redux Flow

Apa yang kita butuhkan untuk bermain dengan Redux ?

- **Redux** itu sendiri. Redux bisa didapat melalui instalasi via npm
- **Initial State**. Bisa dalam bentuk tipe data apapun, tapi hampir dapat dipastikan secara practice akan digunakan sebuah object. Di dalamnya akan didaftarkan state apa saja yang kita butuhkan, beserta initial value nya
- **Reducer.** Sebuah function dengan 2 parameter : state dan action. Initial State akan dimasukkan ke dalam parameter sebagai default parameter dari state. Action adalah object dan yang akan sering kita gunakan adalah action.type (untuk memberitahu action apa yang akan dilakukan) dan action.payload (data yang dibawa dan diperlukan untuk melakukan berbagai proses yang hasilnya akan kita gunakan untuk update state

Apa yang kita butuhkan untuk bermain dengan Redux? (Part 2)

- Function **createStore** dengan parameter sebuah reducer untuk meng inisiasi store
- Function **subscribe** (optional) untuk bisa mendaftarkan sebuah aksi yang akan di trigger secara otomatis ketika ada perubahan state
- Function **dispatch** yang menerima parameter berupa object berisi type dan payload (optional) untuk diteruskan dan di konsumsi oleh reducer

Redux in Action (CLI)

Mari kita mencoba untuk bermain dengan REDUX. Kita akan buat counter increment dan decrement seperti pada beberapa sesi sebelum ini TANPA menggunakan React terlebih dahulu. Silakan membuat sebuah folder baru untuk melanjutkan aksi-aksi selanjutnya.

1. Initiate npm

> npm init -y

2. Install redux

> npm install redux

3. Create new js file

Buatlah sebuah file js untuk kita bermain redux. Kali ini kita membuat file dengan nama **redux.js**. Kode-kode selanjutnya akan kita masukkan ke dalam file ini

> touch redux.js

4. Import Redux

```
1  import Redux from 'redux';
2  const { createStore } = Redux;
```

5. Buat initialState

Redux membutuhkan initial state untuk bisa tahu seperti apa kita akan menyimpan state kita, sekaligus untuk memasukkan initial data nya

```
4  const initialState = {
5    counter: 0
6  }
```

6. Buat reducer

initialState akan menjadi default parameter dari state

```
8   const counter = (state = initialState, action) => {
9     switch (action.type) {
10      case "INCREMENT":
11        return { counter: state.counter + 1 }
12      case "DECREMENT":
13        return { counter: state.counter - 1 }
14      default:
15        return state
16    }
17  }
```

7. Panggil function createStore

... dengan reducer tadi sebagai parameter nya

```
19  let store = createStore(counter)
```

8. Panggil function subscribe

Kita akan bind ke sebuah anonymous function untuk melakukan console.log dari state

```
21  store.subscribe(() => console.log(store.getState()))
```

9. Panggil function dispatch

Buatlah beberapa untuk mensimulasikan beberapa aksi yang akan kita lakukan

```
23  store.dispatch({ type: 'INCREMENT' })
24  store.dispatch({ type: 'INCREMENT' })
25  store.dispatch({ type: 'INCREMENT' })
26  store.dispatch({ type: 'INCREMENT' })
27  store.dispatch({ type: 'DECREMENT' })
28  store.dispatch({ type: 'DECREMENT' })
```

10. Update package.json

Pastikan kita update file ini untuk memberitahukan bahwa kita akan melakukan pemanggilan library redux dengan cara “module”

```
{
  "name": "terminal",
  "version": "1.0.0",
  "description": "",
  "main": "redux.js",
  ▷ Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "redux": "^4.1.0"
  },
  "type": "module"
}
```

11. Jalankan dengan node di terminal

Finally, mari kita saksikan apa yang akan dihasilkan dari pemanggilan node terhadap file redux.js yang sudah kita buat

```
~ node redux.js
{ counter: 1 }
{ counter: 2 }
{ counter: 3 }
{ counter: 4 }
{ counter: 3 }
{ counter: 2 }
```

Redux in Action (Web)

Kali ini kita akan buat hal yang sama, tapi versi web. Dan sekali lagi, masih TANPA menggunakan React terlebih dahulu, tapi kita akan menggunakan Redux yang didapat via CDN. Silakan membuat sebuah folder baru untuk melanjutkan aksi-aksi selanjutnya.

1. Buat file-file yang dibutuhkan

> touch index.html script.js

2. Initiate index.html

Silakan initiate dengan standard html starting codes

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Play with Redux</title>
</head>
<body>

</body>
</html>
```

3. Sisipkan sedikit CSS pada bagian <head>

```
  <style type="text/css">
    #demo {
      width: 500px;
      margin: 0 auto;
      text-align: center;
    }

    button {
      font-size: 2rem;
    }
  </style>
</head>
```

Web Version

Redux in Action

HACKTIV8

4. Masukkan tag-tag HTML ke dalam <body>

```
22    <body>
23      <div id="demo">
24        <h1>Counter with Redux</h1>
25        <hr />
26        <h1 id="counter"></h1>
27        <button id="decrement">-</button>
28        <button id="increment">+</button>
29      </div>
```

5. Masukkan script-script yang dibutuhkan sebelum </body>

```
31      <script src="https://unpkg.com/redux@latest/dist/redux.min.js"></script>
32      <script src="./script.js"></script>
33    </body>
```

Selanjutnya, mari kita update file script.js kita

Selanjutnya, mari kita update file **script.js**...

1. Re-do step 5 dan 6 dari experiment CLI

Yep, kita tidak butuh step-step sebelumnya. Langsung ke step 5 dan 6

```
1   const initialState = {
2     counter: 0
3   }
4
5   const counter = (state = initialState, action) => {
6     switch (action.type) {
7       case "INCREMENT":
8         return { counter: state.counter + 1 }
9       case "DECREMENT":
10        return { counter: state.counter - 1 }
11      default:
12        return state
13    }
14  }
```

2. Panggil function createStore

Karena dari CDN, kita akan panggil dengan cara ini

```
17  var store = Redux.createStore(counter)
```

3. Buat function render

Render di sini akan kita buat untuk menulis ke tag HTML

```
20  var el = document.getElementById('counter')
21  const render = () => {
22    el.innerHTML = store.getState().counter.toString()
23  }
```

4. Panggil render dan daftarkan ke subscribe

```
24  render() // tampilkan angka inisiasi
25  store.subscribe(render)
```

Web Version

Redux in Action

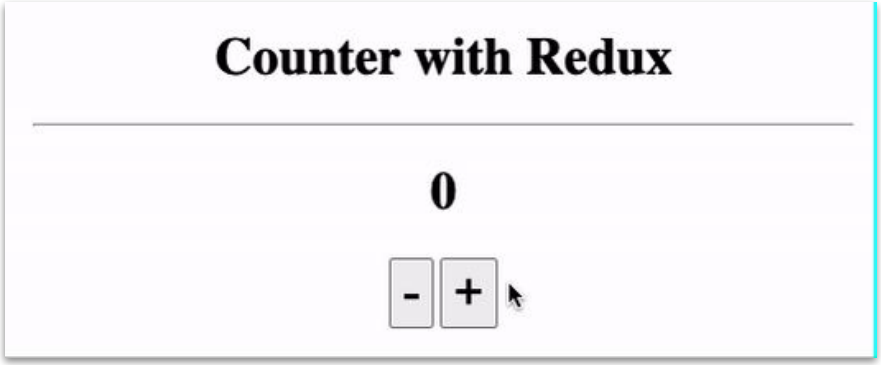
5. Implementasi dispatch pada tag-tag HTML

Kita akan bind event listener ke 2 buah button. Ketika button di click, akan mentrigger dispatch dengan type action yang berbeda

```
29  var incEl = document.getElementById('increment')
30  var decEl = document.getElementById('decrement')
31
32  incEl.addEventListener('click', () => {
33    store.dispatch({ type: 'INCREMENT' })
34  })
35
36  decEl.addEventListener('click', () => {
37    store.dispatch({ type: 'DECREMENT' })
38  })
```

6. Jalankan index.html di browser

Saksikan bagaimana redux kita beraksi



Untuk melihat simulasi lebih lengkap, silakan buka Lampiran 1

Web Version

Redux in Action

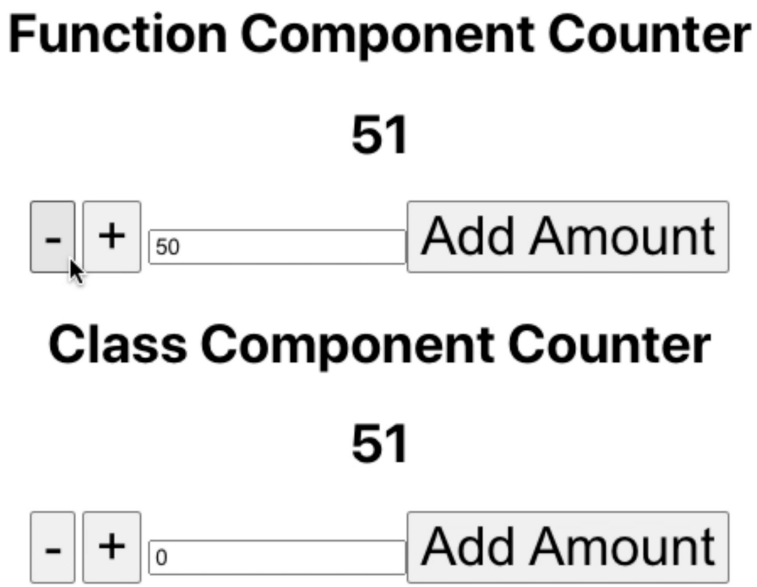
Connecting React with Redux

Untuk “menyambungkan” React dengan Redux kita akan menggunakan library **Redux Toolkit**. Library ini adalah library official yang dikeluarkan oleh pembuat Redux, yang akhirnya di akuisisi oleh React

0. Siapkan sebuah aplikasi React

Initiate sebuah aplikasi React kemudian buatlah 1 buah class component dan 1 buah functional component dengan tampilan yang sama persis. Susunlah kedua component tersebut pada App.js sehingga menjadi tampak seperti pada gambar di samping

Note: React yang digunakan dalam demo ini adalah react versi 18.



Sebelum lanjut, kita harus mengetahui contoh struktur folder yang direkomendasikan dalam dokumentasi redux.

Struktur Folder

- **/src**
  - **index.js**: the starting point for the app
  - **App.js**: the top-level React component
  - **/app**
    - **store.js**: creates the Redux store instance
  - **/features**
    - **/counter**
      - **Counter.js**: a React component that shows the UI for the counter feature
      - **counterSlice.js**: the Redux logic for the counter feature

1. Install library yang diperlukan

Kita akan install **redux toolkit** dan **react redux**

> npm install @reduxjs/toolkit react-redux

2. Buat folder dan file untuk store

For the sake of good practice, kita akan buat sebuah folder baru yang bernama **src**, kemudian kita buat file store kita di dalamnya

> mkdir -p src/features/counter

> touch src/features/counter/counterSlice.js

3. Implementasi kode untuk store

... pada src/features/counter/counterSlice.js menjadi seperti di bawah ini

```
import { createSlice } from "@reduxjs/toolkit";

const initialState = {
    count: 0
}

export const counterSlice = createSlice({
    name: 'counter',
    initialState,
    reducers: {
        increment: (state) => {
            state.count += 1;
        },
        decrement: (state) => {
            state.count -= 1;
        },
        incrementByTen: (state) => {
            state.count += 10;
        }
    }
});

export const { increment, decrement, incrementByAmount } = counterSlice.actions;

export default counterSlice.reducer;
```

4. src/app/store.js - Configure Store

```
import { configureStore } from "@reduxjs/toolkit";
import counterReducer from '../features/counter/counterSlice';

export const store = configureStore({
    reducer: {
        counter: counterReducer,
    }
})
```

5. src/index.js - import Provider dan store

```
import { store } from './app/store';
import { Provider } from 'react-redux';
```

6. src/index.js - Implementasi Provider

... untuk membungkus component App dan mengirimkan store sebagai props

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <Provider store={store}>
      <App />
    </Provider>
  </React.StrictMode>
);
```

7. src/App.js - Import components dan css

```
import './App.css'
import CounterClass from "./features/counter/CounterClassComponent";
import CounterFn from "./features/counter/CounterFnComponent";
```

8. src/App.js - Update component App

```
function App() {
  return (
    <main className="App">
      <CounterFn />
      <CounterClass />
    </main>
  );
}
```

8. src/App.js - Import components dan css

```
1   import './App.css';
2   import CounterClass from './components/CounterClass';
3   import CounterFn from './components/CounterFn';
```

9. src/App.js - Update component App

```
5   function App() {
6     return (
7       <div>
8         <CounterClass />
9         <CounterFn />
10      </div>
11    );
12  }
```

Selanjutnya, kita akan bermain dengan file **src/features/counter/CounterClassComponent.js** untuk **implementasi koneksi class component dengan redux**. Let's Go !!

1. Import libraries dan reducer

Component menggunakan library **connect** untuk terhubung dengan redux store

```
import { Component } from 'react';
import { connect } from 'react-redux';
import {
  increment,
  decrement,
  incrementByAmount
} from './counterSlice';
```

3. Buat state dan method setIncrementAmount

untuk kita melakukan increment menggunakan payload

```
class CounterClass extends Component {

  state = {
    incrementAmount: 0,
  }

  setIncrementAmount = (val) => {
    this.setState({
      incrementAmount: Number(val) || 0
    });
  }
```

React - Class Component

Redux in Action

HACKTIV8

3. Update function render()

Kita akan masukkan function-function yang berisi dispatch sebagai bagian dari event onClick pada setiap button dengan teknik anonymous arrow function

```
render() {
  return(
    <div className="demo">
      <h1>Class Component Counter</h1>
      <h1 id="counter">{this.props.localCount}</h1>
      <button id="decrement" onClick={() => this.props.dispatch(decrement())}>-</button>
      {' '}
      <button id="increment" onClick={() => this.props.dispatch(increment())}>+</button>
      {' '}
      <input
          type="text"
          value={this.state.incrementAmount}
          onChange={(e) => this.setIncrementAmount(e.target.value)}
      />

      <button id="increment" onClick={() => this.props.dispatch(incrementByAmount(this.state.incrementAmount))}>
        Add Amount
      </button>
    </div>
  )
}
```

React - Class Component

Redux in Action

HACKTIV8

5. Buat function mapStateToProps

… yang memberitahukan kepada store nama mapping yang kita inginkan untuk state yang kita butuhkan untuk komponen ini. Juga update bagian export untuk menggunakan connect dengan parameter nya berisi mapStateToProps ini

```
const mapStateToProps = (state) => ({
  localCount: state.counter.count
});

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

Jika kamu mau coba dulu class component nya, silakan comment bagian import dan pemanggilan CounterFn di src/App.js dan jalankan aplikasinya, periksa apakah fungsionalitasnya berjalan baik

Selanjutnya, kita akan bermain dengan file **src/features/CounterFnComponent.js** untuk **implementasi koneksi functional component dengan redux.**

1. Import libraries

Seperti biasa, untuk functional component kita hanya membutuhkan hooks. Hook yang akan kita gunakan adalah **useSelector** untuk connect ke store dan **useDispatch** untuk melakukan dispatch

```
import { useSelector, useDispatch } from "react-redux";
import {
    increment,
    decrement,
    incrementByAmount
} from './counterSlice';
import { useState } from "react";
```

2. Implementasi hooks

```
const CounterFnComponent = () => {
    const count = useSelector((state) => state.counter.count);
    const dispatch = useDispatch();
```

3. Gunakan useState untuk incrementAmount

```
const [incrementAmount, setIncrementAmount] = useState(0);

const addValue = Number(incrementAmount) || 0;
```

React - Functional Component

Redux in Action

HACKTIV8

4. Update function return

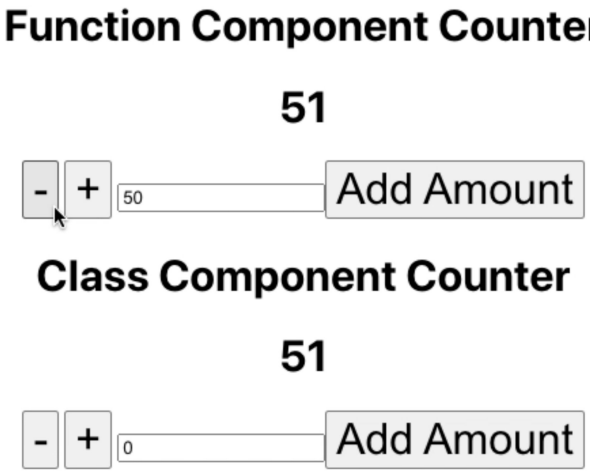
... kembali kita masukkan function-function yang sudah kita buat ke dalam onClick event setiap button, dengan gaya functional component tentunya

```
return (
  <div className="demo">
    <h1>Function Component Counter</h1>
    <h1 id="counter">{count}</h1>
    <button id="decrement" onClick={() => dispatch(decrement())}>-</button>
    {' '}
    <button id="increment" onClick={() => dispatch(increment())}>+</button>
    {' '}
    <input
        type="text"
        value={incrementAmount}
        onChange={(e) => setIncrementAmount(e.target.value)}
    />

    <button id="increment" onClick={() => dispatch(incrementByAmount(addValue))}>Add Amount</button>
  </div>

)
```

That's it guys. FYI, untuk bagian export, lakukan export default seperti biasanya.
Dan sekarang saatnya untuk menjalankan aplikasi dan mencoba semua functionalnya

PERHATIKAN ! Jika implementasi kamu benar, maka aksi apapun pada komponen manapun akan meng-update store dan merubah angka counter **pada kedua komponen**

Kenapa ?
Karena kedua komponen mengakses store yang sama. Maka update yang sama haruslah terjadi untuk aksi pada komponen manapun



Untuk melihat simulasi lebih lengkap, silakan buka Lampiran 2

Thank You

PT Hacktivate Teknologi Indonesia
Gedung Aquarius Pondok Indah
Jalan Sultan Iskandar Muda No.7
Kebayoran Lama, Jakarta Selatan

https://hacktiv8.com

Hacktiv8id

Hacktiv8id

Hacktiv8

Hacktiv8 Indonesia