
# Front End React Development
# sesi 16

# State Management with Redux

INTRO

# 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

INTRO

# 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.

INTRO

# 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.

INTRO

# 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

**INTRO**

# … 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.

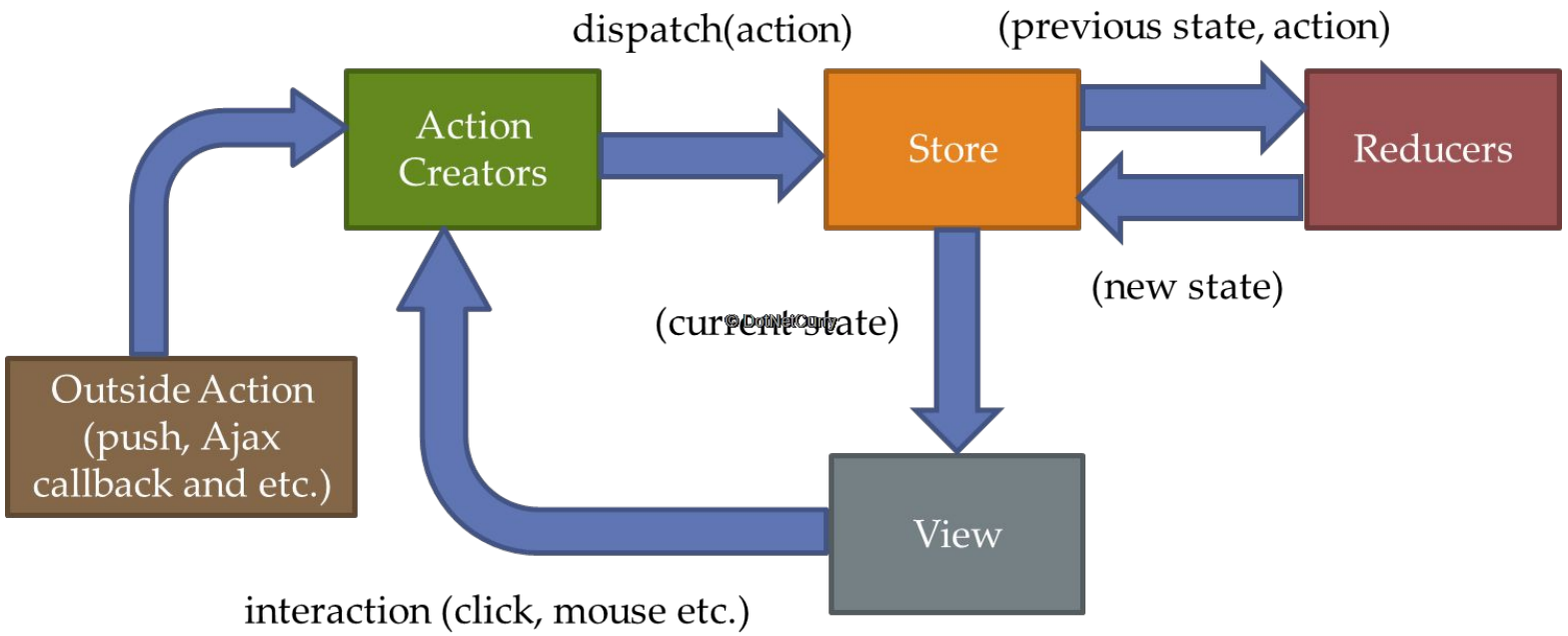
## Reducer update the state

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

# Sesi 16 | State Management with redux

## INTRO

Redux Flow




HACKTIV8

# Sesi 16 | State Management with redux

## INTRO

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
- 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

# Sesi 16 | State Management with redux

## 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  }
```

HACKTIV8

# Sesi 16 | State Management with redux

## REDUX IN ACTION - CLI

### 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' })
```

# Sesi 16 | State Management with redux

## REDUX IN ACTION - CLI

### 6. 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"
}
```

### 7. 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 }
```

HACKTIV8

# Sesi 16 | State Management with redux

## 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>
```

HACKTIV8

## REDUX IN ACTION - WEB

### 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

# Sesi 16 | State Management with redux

## REDUX IN ACTION - WEB

### 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)
```

# Sesi 16 | State Management with redux

## REDUX IN ACTION - WEB

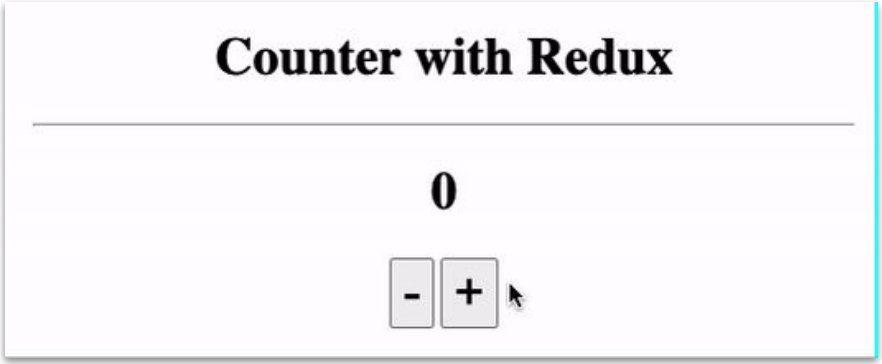
### 5. Implementasi dispatch pada tag-tag HTML

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**

# Connect React with Redux

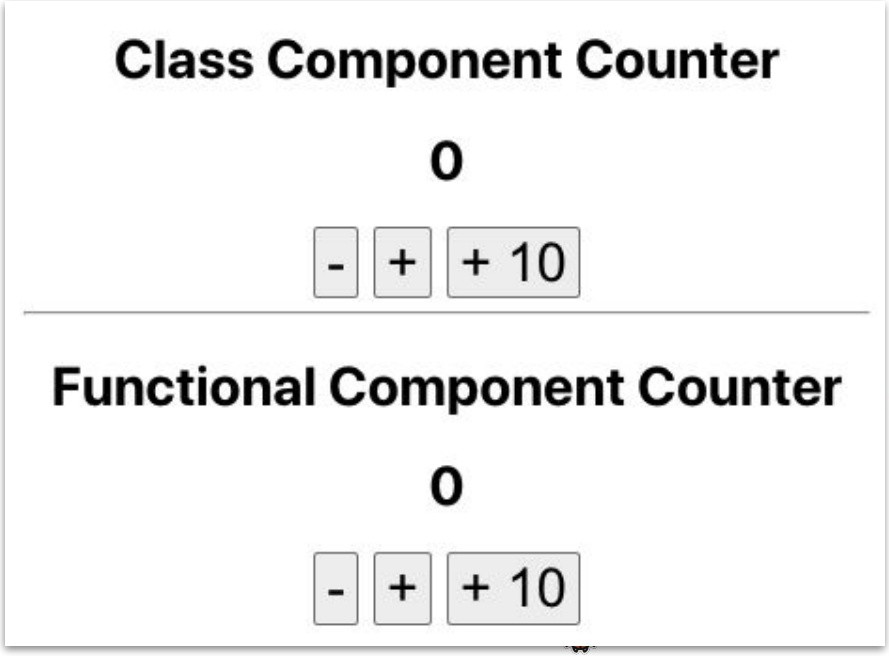
# Sesi 16 | Connect React with Redux

## REACT - REDUX IN ACTION

Untuk “menyambungkan” React dengan Redux kita akan menggunakan library **react-redux**. 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



# Sesi 16 | Connect React with Redux

## REACT - REDUX IN ACTION

### 1. Install library yang diperlukan

Kita akan install **redux** dan **react-redux**

```
> npm install redux
```

```
> npm install 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 src/store
```

```
> touch src/store/index.js
```

### 3. Implementasi kode untuk store

… pada src/store/index.js menjadi seperti di bawah ini

```
1  import { createStore } from 'redux';
2
3  const initialState = {
4    counter: 0
5  }
6
7  const counter = (state = initialState, action) => {
8    switch (action.type) {
9      case "INCREMENT":
10       let substractor = action.payload ? action.payload : 1
11
12       return {
13         counter: state.counter + substractor
14       }
15     case "DECREMENT":
16       return { counter: state.counter - 1 }
17     default:
18       return state
19   }
20 }
21
22 const store = createStore(counter)
23
24 export default store
```

# Sesi 16 | Connect React with Redux

## REACT - REDUX IN ACTION

### 4. src/index.js - import Provider dan store

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

### 5. src/index.js - Implementasi Provider

… untuk membungkus component App dan mengirimkan store sebagai props

```
12 ReactDOM.render(
13   <React.StrictMode>
14     <Provider store={store}>
15       <App />
16     </Provider>
17   </React.StrictMode>,
18   document.getElementById('root')
19 );
```

### 6. src/App.css - Implementasi CSS

```
1  .demo {
2    width: 500px;
3    margin: 0 auto;
4    text-align: center;
5  }
6
7  button {
8    font-size: 2rem;
9  }
```

### 7. Buat folder components dan isinya

```
> mkdir src/components

> touch src/components/CounterClass.js

> touch src/components/CounterFn.js
```

# Sesi 16 | Connect React with Redux

## REACT - REDUX IN ACTION

### 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/components/CounterClass.js untuk implementasi koneksi class component dengan redux. Let’s Go !!

# Sesi 16 | Connect React with Redux

## REACT - REDUX IN ACTION : CLASS COMPONENT ( src/components/CounterClass.js )

### 1. Import libaries

Class component menggunakan library **connect** untuk terhubung dengan redux store

```
1   import { Component } from 'react';
2   import { connect } from 'react-redux';
```

### 2. Buat function increment dan decrement

```
4   class CounterClass extends Component {
5     increment() {
6       this.props.dispatch({
7         type: "INCREMENT"
8       })
9     }
10
11    decrement() {
12      this.props.dispatch({
13        type: "DECREMENT"
14      })
15    }
```

### 3. Buat function customIncrement

… untuk kita melakukan increment menggunakan payload

```
17    customIncrement() {
18      this.props.dispatch({
19        type: "INCREMENT",
20        payload: 10
21      })
22    }
```

HACKTIV8

# Sesi 16 | Connect React with Redux

## REACT - REDUX IN ACTION : CLASS COMPONENT ( src/components/CounterClass.js )

### 4. Update function render()

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

```
24    render() {
25      return (
26        <div class="demo">
27          <h1>Class Component Counter</h1>
28          <h1 id="counter">{this.props.localCounter}</h1>
29          <button id="decrement" onClick={() => this.decrement()}>-</button>
30            
31          <button id="increment" onClick={() => this.increment()}>+</button>
32            
33          <button id="decrement" onClick={() => this.customIncrement()}>+ 10</button>
34        </div>
35      )
36    }
```

# Sesi 16 | Connect React with Redux

## REACT - REDUX IN ACTION : CLASS COMPONENT ( src/components/CounterClass.js )

### 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

```
39  const mapStateToProps = (state) => {
40    return {
41      localCounter: state.counter
42    }
43  }
44
45  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/components/CounterFn.js untuk implementasi koneksi functional component dengan redux.

# Sesi 16 | Connect React with Redux

## REACT - REDUX IN ACTION : FUNCTIONAL COMPONENT ( src/components/CounterFn.js )

### 1. Import libaries

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

```
1  import {
2    useSelector,
3    useDispatch
4  } from 'react-redux';
```

### 2. Implementasi hooks

```
6  const CounterFn = () => {
7    const state = useSelector((state) => state)
8    const dispatch = useDispatch()
```

### 3. Buat semua function yang dibutuhkan

```
10   const increment = () => {
11     dispatch({
12       type: "INCREMENT"
13     })
14   }
15
16   const decrement = () => {
17     dispatch({
18       type: "DECREMENT"
19     })
20   }
21
22   const customIncrement = () => {
23     dispatch({
24       type: "INCREMENT",
25       payload: 10
26     })
27   }
```

# Sesi 16 | Connect React with Redux

## REACT - REDUX IN ACTION : FUNCTIONAL COMPONENT ( src/components/CounterFn.js )

### 4. Update function return

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

```
29    return(
30      <div class="demo">
31        <hr />
32        <h1>Functional Component Counter</h1>
33        <h1 id="counter">{state.counter}</h1>
34        <button id="decrement" onClick={decrement}>-</button>
35          
36        <button id="increment" onClick={increment}>+</button>
37          
38        <button id="decrement" onClick={customIncrement}>+ 10</button>
39      </div>
40    )
```

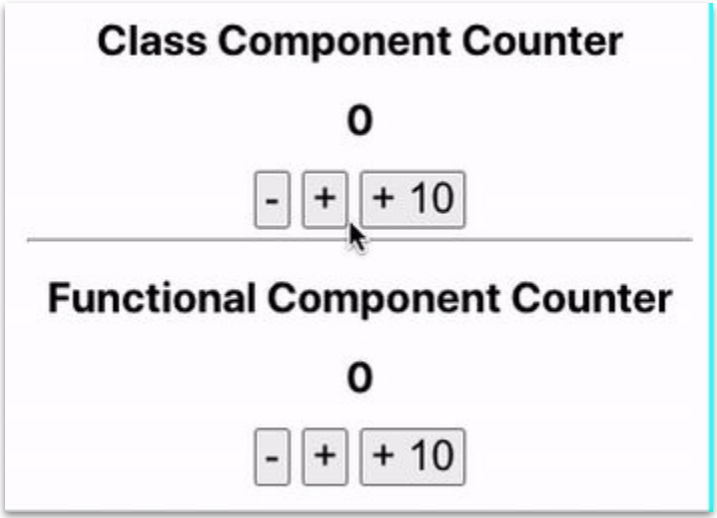
That’s it guys. FYI, untuk bagian export, lakukan export default seperti biasanya.
Dan sekarang saatnya untuk menjalankan aplikasi dan mencoba semua functionalnya

# Sesi 16 | Connect React with Redux

## REACT - REDUX IN ACTION : RUN THE APPLICATION

**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 meng-akses store yang sama. Maka update yang sama haruslah terjadi untuk aksi pada komponen manapun




**Untuk melihat simulasi lebih lengkap, silakan buka Lampiran 2**

HACKTIV8

# Redux Middleware

# Sesi 16 | Redux Middleware

## INTRO + REDUX LOGGER

Kita bisa menambahkan fungsi tambahan seperti extensions atau plugin diantara proses dispatch sebuah action hingga pada saat mencapai reducer. Kita bisa menggunakan Redux middleware untuk logging, crash reporting, proses async ke API, routing dan lain sebagainya.

### Redux Logger

Sudah cukup jelas ya, sebuah logger untuk Redux. Setiap terjadi perubahan state, action di panggil dan berbagai hal yang berhubungan denga Redux, akan muncul informasi di console browser.


### 1. Instalasi

```
> npm install redux-logger
```

# Sesi 16 | Redux Middleware

## REDUX LOGGER

### 2. Import

Lakukan import ini di file tempat kita melakukan createStore. Selain import logger, jangan lupa import applyMiddleware dari redux

```
2  import { createStore, applyMiddleware } from 'redux';
3  import logger from 'redux-logger';
```

### 3. Implementasikan

Panggil applyMiddleware dengan logger nya sebagai parameter kedua dari createStore

```
26  const store = createStore(counter, applyMiddleware(logger))
```

### 4. Run the application

Silakan langsung jalankan aplikasinya dan saksikan aksi dari redux-logger nya pada developer console di browser kita. Recommended untuk menggunakan Chrome atau Firefox

# Sesi 16 | Redux Middleware

## REDUX THUNK

Redux sejatinya tidak memiliki mekanisme untuk menangani action yang sifatnya async. Artinya kita bisa menangani semua yang sifatnya async di React. Ini opsi yang kurang bagus kalau dipandang dari sisi modularitas kode.

Atau opsi kedua, kita bisa gunakan middleware. Ada beberapa opsi middleware untuk menangani async di Redux. Ada Redux Thunk, Redux Saga. Dari segi *learning curve* thunk lebih mudah dipelajari karena itu mari kita fokus ke redux thunk. Redux saga memiliki keunggulan terutama untuk project dengan skala besar dengan fleksibilitas dan boilerplate code yang lebih sedikit dibandingkan Thunk. Tapi untuk sesi kali ini kita akan belajar Thunk dan Redux Thunk

**THUNK**

- Nama ini sebenarnya secara humor merupakan bentuk pas-tense dari “think”
- Adalah sebuah fungsi yang membungkus 1 atau beberapa expresi yang akan di evaluasi ( baca: dikerjakan ) nanti

```
// calculation of 1 + 2 is immediate
// x === 3
let x = 1 + 2;

// calculation of 1 + 2 is delayed
// foo can be called later to perform the calculation
// foo is a thunk!
let foo = () => 1 + 2;
```

# Sesi 16 | Redux Middleware

## REDUX THUNK

### REDUX THUNK

- Mengadopsi konsep thunk ke dalam redux
- Middleware ini membuat kita jadi bisa membuat action creators yang mengembalikan sebuah function daripada sebuah action. Juga bisa untuk melakukan delay terhadap eksekusi dispatch dari sebuah aksi ( contoh kiri ), atau untuk melakukan dispatch hanya jika ada kondisi tertentu yang terpenuhi ( contoh kanan )
- Fungsi yang di return ( inner function ) akan menerima methods **dispatch** dan **getState** dari store sebagai parameters

```
const INCREMENT_COUNTER = 'INCREMENT_COUNTER';

function increment() {
  return {
    type: INCREMENT_COUNTER,
  };
}

function incrementAsync() {
  return (dispatch) => {
    setTimeout(() => {
      // Yay! Can invoke sync or async actions with `dispatch`
      dispatch(increment());
    }, 1000);
  };
}
```

```
function incrementIfOdd() {
  return (dispatch, getState) => {
    const { counter } = getState();

    if (counter % 2 === 0) {
      return;
    }

    dispatch(increment());
  };
}
```

HACKTIV8

# Sesi 16 | Redux Middleware

## REDUX THUNK IN ACTION : FUNCTIONAL COMPONENT ( src/components/CounterFn.js )

Kita langsung saja manfaatkan aplikasi counter kita untuk implementasi fetch data user dari API. Pertama-tama, kita akan coba lakukan aksi kita tanpa menggunakan redux-thunk dan kita lihat apa yang akan terjadi

### 1. Update function return

Tambahkan kode di samping ini ke dalam function return pada functional component yang sudah kita buat, CounterFn.js

```
<hr />
<button onClick={handleFetchUsers}>Get Users</button>
<br /><br />
<table border="1">
  <thead>
    <tr>
      <th>Name</th>
      <th>Username</th>
      <th>Email</th>
    </tr>
  </thead>
  <tbody>
  {
    state.users.map((user, idx) => (
      <tr key={idx}>
        <td>{ user.name }</td>
        <td>{ user.username }</td>
        <td>{ user.email }</td>
      </tr>
    ))
  }
  </tbody>
</table>
```

### 2. Tambahkan function handleFetchUsers

Silakan diikuti dulu, kita akan tambahkan fetchUsers() nya pada langkah berikutnya

```
const handleFetchUsers = () => {
  dispatch(fetchUsers())
}
```

HACKTIV8

# Sesi 16 | Redux Middleware

## REDUX THUNK IN ACTION : FUNCTIONAL COMPONENT ( src/components/CounterFn.js )

### 3. Update src/App.css

Tambahkan style seperti di kanan ini

```css
table {
  border-collapse: collapse;
}

th {
  padding: 5px;
  background-color: #3C5186;
  color: #FFF5DE;
}

td {
  padding: 5px;
}
```

### 4. Buat file src/store/action.js

```js
export const fetchUsers = () => {
  return async (dispatch) => {
    const res = await fetch('https://jsonplaceholder.typicode.com/users')
    const data = await res.json()
    dispatch({
      type: "SET_USERS",
      payload: data
    })
  }
}
```

# Sesi 16 | Redux Middleware

## REDUX THUNK IN ACTION : FUNCTIONAL COMPONENT ( src/components/CounterFn.js )

### 3. Import src/store/action.js

```
2  import { fetchUsers } from '../store/action';
```

### 5. Jalankan aplikasi

… dan klik tombol Get Users, maka akan menghasilkan error seperti di bawah ini :

Pesan error yang diberikan sudah sangat jelas dan begitu pula anjurannya. Kita membuktikan bahwa redux hanya menerima object untuk dispatch nya

Mari kita implementasi redux-thunk untuk mengatasi hal ini

# Sesi 16 | Redux Middleware

## REDUX THUNK IN ACTION : FUNCTIONAL COMPONENT ( src/components/CounterFn.js )

### 1. Install react-redux

```
> npm install react-redux
```

### 2. Import redux-thunk pada store

```
2  import { createStore, applyMiddleware } from 'redux';
3  import logger from 'redux-logger';
4  import thunk from 'redux-thunk';
```

### 3. Apply thunk pada middleware, pada store

```
26  const store = createStore(counter, applyMiddleware(thunk, logger))
```

### 4. Jalankan aplikasi dan klik tombol Get Users

… ekspektasi kita adalah seperti tampilan di bawah ini


| Name | Username | Email |
|---|---|---|
| Leanne Graham | Bret | Sincere@april.biz |
| Ervin Howell | Antonette | Shanna@melissa.tv |
| Clementine Bauch | Samantha | Nathan@yesenia.net |
| Patricia Lebsack | Karianne | Julianne.OConner@kory.org |
| Chelsey Dietrich | Kamren | Lucio_Hettinger@annie.ca |
| Mrs. Dennis Schulist | Leopoldo_Corkery | Karley_Dach@jasper.info |
| Kurtis Weissnat | Elwyn.Skiles | Telly.Hoeger@billy.biz |
| Nicholas Runolfsdottir V | Maxime_Nienow | Sherwood@rosamond.me |
| Glenna Reichert | Delphine | Chaim_McDermott@dana.io |
| Clementina DuBuque | Moriah.Stanton | Rey.Padberg@karina.biz |