ReactJS Redux

Eko Kurniawan Khannedy

Eko Kurniawan Khannedy

- Technical architect at one of the biggest ecommerce company in Indonesia
- 14+ years experiences
- www.programmerzamannow.com
- youtube.com/c/ProgrammerZamanNow

Eko Kurniawan Khannedy

- Telegram : @khannedy
- Linkedin : https://www.linkedin.com/company/programmer-zaman-now/
- Facebook : fb.com/ProgrammerZamanNow
- Instagram : instagram.com/programmerzamannow
- Youtube : youtube.com/c/ProgrammerZamanNow
- Telegram Channel : t.me/ProgrammerZamanNow
- Tiktok : https://tiktok.com/@programmerzamannow
- Email : echo.khannedy@gmail.com

Sebelum Belajar

- ReactJS Dasar
- ReactJS Router

Pengenalan

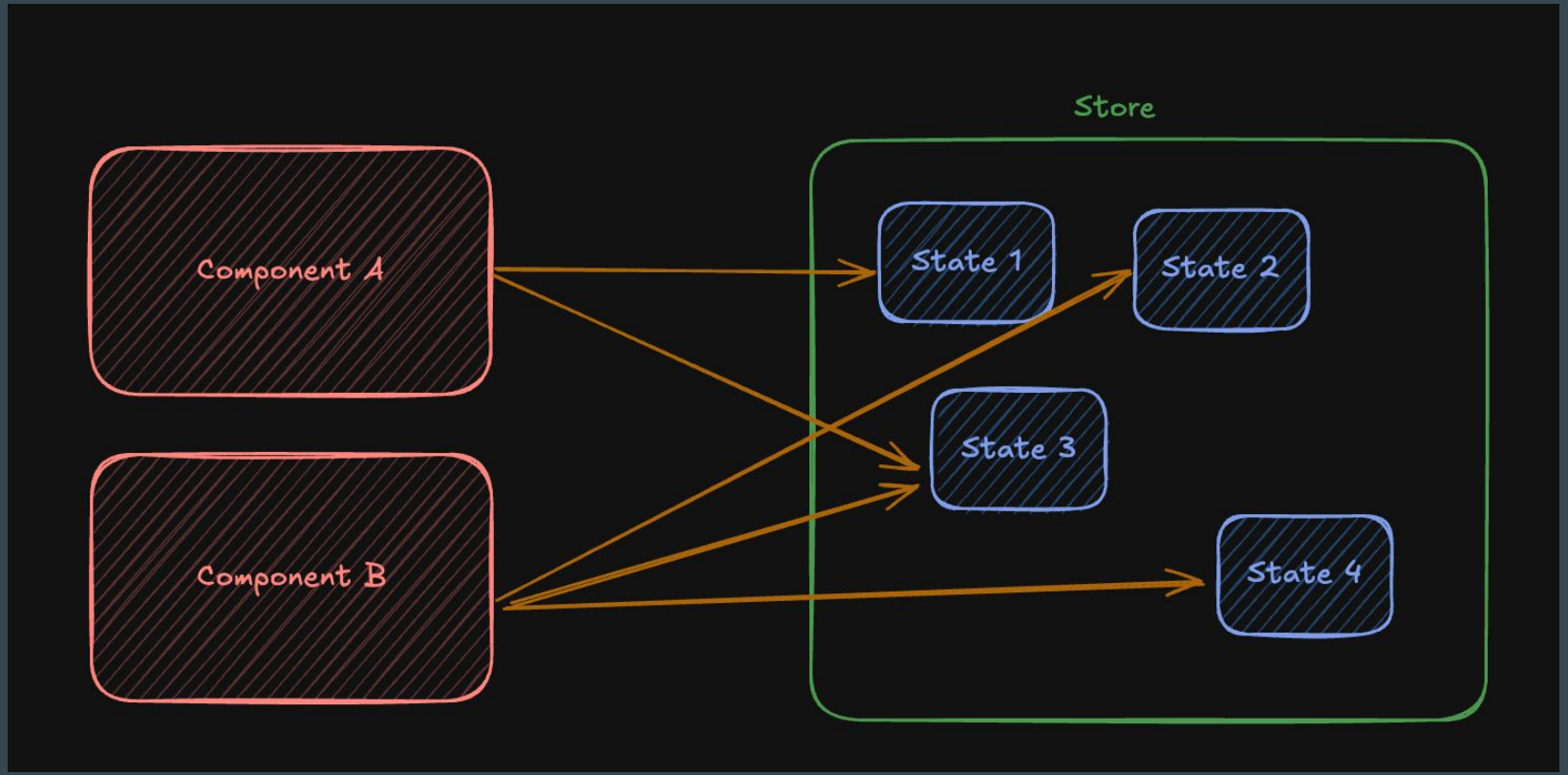
Global State Management

- Sebelumnya, kita sudah belajar tentang State Management di React, untuk manajemen data di dalam komponen
- Kita juga sudah belajar tentang Context, yang digunakan untuk manajemen data dari komponen parent ke child
- Lantas bagaimana jika kita ada kebutuhan membuat data (state) yang memang ingin bisa diakses secara global? Yang artinya bisa diakses dari manapun, komponen manapun, bahkan jika berbeda parent sama sekali. Layaknya Global State Management
- Jawabannya, kita butuh bantuan sebuah tempat untuk menyimpan State tersebut, atau kita sebut Store

Redux

- Redux adalah Store library untuk menyimpan State
- Redux membolehkan kita menyimpan state dan diakses dari komponen atau halaman manapun
- Ini menjadikan manajamen state di multi komponen akan jadi lebih mudah menggunakan Redux
- Redux adalah project opensource yang populer untuk Global State Management
- https://redux.js.org/

Diagram Store



Membuat Project

Membuat Project

- npm create vite@latest belajar-reactjs-redux -- --template react
- Update React ke versi 19
- npm install react-router

Menambah Redux

- npm install @reduxjs/toolkit
- npm install react-redux

Setup

Setup

- Saat ingin menggunakan Redux, kita harus membuat Provider terlebih dahulu
- Caranya aplikasi yang kita buat, bisa kita bungkus dalam komponen Provider milik Redux
- Selanjutnya ada parameter yang wajib kita buat, yaitu store, tapi itu akan kita bahas di materi selanjutnya

Kode : src/main.jsx

main.jsx

```jsx
import {StrictMode} from 'react'
import {createRoot} from 'react-dom/client'
import './index.css'
import App from './App.jsx'
import {Provider} from "react-redux";
import {BrowserRouter, Route, Routes} from "react-router";

createRoot(document.getElementById('root')).render(
  <StrictMode>
    <Provider store={store}>
      <BrowserRouter>
        <Routes>
          <Route path={"/"} element={<App/>}/>
        </Routes>
      </BrowserRouter>
    </Provider>
  </StrictMode>,
)
```

Store

Store

- Store merupakan tempat dimana data State disimpan
- Saat menggunakan Redux, kita perlu membuat Store terlebih dahulu
- Nanti data State yang kita buat, akan kita masukkan kedalam Store tersebut
- Untuk membuat Store, kita bisa menggunakan function configureStore()
- https://redux-toolkit.js.org/api/configureStore

Kode : src/main.jsx

main.jsx

```jsx
import {StrictMode} from 'react'
import {createRoot} from 'react-dom/client'
import './index.css'
import App from './App.jsx'
import {Provider} from "react-redux";
import {BrowserRouter, Route, Routes} from "react-router";
import {configureStore} from "@reduxjs/toolkit";

const store = configureStore({
  reducer: {}
})

createRoot(document.getElementById('root')).render(
  <StrictMode>
    <Provider store={store}>
      <BrowserRouter>
        <Routes>
          <Route path={"/"} element={<App/>}/>
```

State

State

- Setelah kita membuat Store, kita bisa mulai membuat State di Store
- Namun integrasi antara Store dan State itu dilakukan via Reducer
- Redux menggunakan konsep Slice yang merupakan kombinasi dari State dan Reducer
- Untuk membuat Slice, kita bisa menggunakan function createSlice()
- https://redux-toolkit.js.org/api/createSlice
- Saat kita membuat State, kita perlu tentukan initial value (nilai awal nya)

Tugas

- Misal kita akan membuat Slice yang digunakan untuk melakukan Counter
- Kita akan buat dengan nama counterSlice.js

Kode : src/counterSlice.js

counterSlice.js

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

export const counterSlice = createSlice({
  name: 'counter',
  initialState: 0,
  reducers: {
  }
})
```

Meregistrasikan Reducer

- Setelah kita membuat State menggunakan Slice, kita perlu registrasikan Reducer dari Slice tersebut
- Saat ini mungkin kita belum membuat Reducer, nanti kita akan bahas Reducer di materi Action

Kode : src/main.jsx

main.jsx

```jsx
import {StrictMode} from 'react'
import {createRoot} from 'react-dom/client'
import './index.css'
import App from './App.jsx'
import {Provider} from "react-redux";
import {BrowserRouter, Route, Routes} from "react-router";
import {configureStore} from "@reduxjs/toolkit";
import {counterSlice} from "./counterSlice.js";

const store = configureStore({
  reducer: {
    counter: counterSlice.reducer
  }
})
```

Menggunakan State

- Untuk menggunakan State yang ada di Store, kita bisa menggunakan function useSelector()
- https://react-redux.js.org/api/hooks#useselector
- Kita bisa tentukan State mana yang akan kita gunakan dengan menyebutkan nama state nya yang sama dengan nama di Slice

Kode : src/Counter.jsx

Counter.jsx

```jsx
import {useSelector} from "react-redux";

export default function Counter() {

  const counter = useSelector(state => state.counter);

  return (
    <div>
      <h1>Counter : {counter}</h1>
    </div>
  )
}
```

Kode : src/main.jsx

```
16
17  createRoot(document.getElementById('root')).render(
18    <StrictMode>
19      <Provider store={store}>
20        <BrowserRouter>
21          <Routes>
22            <Route path={"/"} element={<App/>}/>
23            <Route path={"/counter"} element={<Counter/>}/>
24          </Routes>
25        </BrowserRouter>
26      </Provider>
27    </StrictMode>,
28  )
```

Immutable State

- Saat kita menggunakan State Management di React, kita diberi function untuk mengubah State nya
- Sedangkan di Redux, hal itu tidak ada
- Kita tidak diberi function untuk mengubah nilai di State
- Lantas bagaimana jika kita ingin mengubah data di State?
- Maka kita harus menggunakan Action, yang akan kita bahas di materi selanjutnya

Action

Action

- Action di Redux sebenarnya adalah Reducer, namun cara membuatnya lebih mudah
- Kita bisa membuat Action dengan cara membuat semua function Reducer di Slice
- Secara otomatis nanti kita bisa menggunakan nama function Reducer sebagai Action di Redux

Immer Library

- Seperti yang kita tahu, bahwa React menyarankan bahwa kita harus membuat data Immutable di State
- Di materi React Dasar kita sudah belajar tentang Immer Library untuk mempermudah
- Saat kita menggunakan Redux, secara otomatis Redux menggunakan Immer Library ketika menggunakan Reducer
- Dengan begitu, kita tidak perlu lagi melakukan copy data lagi secara manual, karena hal itu sudah dilakukan otomatis oleh Immer Library

Tugas

- Kita akan coba tambahkan beberapa Reducer di counterSlice
- increment(state) untuk melakukan increment
- decrement(state) untuk melakukan decrement

Kode : src/counterSlice.js

counterSlice.js

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

export const counterSlice = createSlice({
  name: 'counter',
  initialState: 0,
  reducers: {
    increment: (state) => {
      return state + 1
    },
    decrement: (state) => {
      return state - 1
    }
  }
})

export const {increment, decrement} = counterSlice.actions

```

Memanggil Action

- Setelah kita membuat Action menggunakan Reducer, selanjutnya kita bisa memanggil Action tersebut menggunakan dispatch, sama seperti Reducer biasanya
- Untuk membuat Dispatch, kita bisa menggunakan useDispatch() function
- https://react-redux.js.org/api/hooks#usedispatch
- Selanjutnya kita bisa gunakan Action yang sebelumnya sudah kita buat untuk menentukan Reducer mana yang akan kita panggil

Kode : src/Counter.jsx

Counter.jsx

```jsx
import {useDispatch, useSelector} from "react-redux";
import {decrement, increment} from "./counterSlice.js";

export default function Counter() {

  const counter = useSelector(state => state.counter);
  const dispatch = useDispatch();

  function handleIncrement() {
    dispatch(increment())
  }

  function handleDecrement() {
    dispatch(decrement())
  }

  return (
    <div>
      <h1>Counter : {counter}</h1>
      <button onClick={handleIncrement}>Increment</button>
      <button onClick={handleDecrement}>Decrement</button>
    </div>
```

Action Parameter

- Pada beberapa kasus, mungkin kita ingin mengirimkan parameter di Action
- Semua parameter yang kita kirim di Action, secara otomatis dia akan disimpan di attribute payload di parameter ke 2 di Reducer
- Misal ketika kita kirim dispatch(increment(2)), maka di reducer akan menerima (state, action), dimana action akan berisi type : “counter/increment”, dan action : 2
- Kita bisa menggunakan tipe data apapun pada payload, dari tipe data sederhana seperti number, string, atau tipe data kompleks seperti array, object dan lain-lain

Tugas

- Sekarang kita akan coba tambahkan parameter di Action sehingga bisa increment / decrement dengan value yang dikirim

Kode : src/counterSlice.js

```
export const counterSlice = createSlice({
  name: 'counter',
  initialState: 0,
  reducers: {
    increment: (state, action) => {
      if (action.payload) {
        return state + action.payload
      } else {
        return state + 1
      }
    },
    decrement: (state, action) => {
      if (action.payload) {
        return state - action.payload
      } else {
        return state - 1
      }
```

Kode : src/Counter.jsx

```jsx
  return (
    <div>
      <h1>Counter : {counter}</h1>
      <button onClick={handleIncrement}>Increment</button>
      <button onClick={() => dispatch(increment(2))}>Increment +2</button>
      <button onClick={handleDecrement}>Decrement</button>
      <button onClick={() => dispatch(decrement(2))}>Decrement -2</button>
    </div>
  )
}
```

Global State

Global State

- State yang kita simpan di Store, itu bisa diakses secara global dari komponen manapun
- Untuk membuktikan bahwa State di Store itu Global
- Kita akan coba menampilkan beberapa Counter

Kode : src/main.jsx

```jsx
createRoot(document.getElementById('root')).render(
  <StrictMode>
    <Provider store={store}>
      <BrowserRouter>
        <Routes>
          <Route path={"/"} element={<App/>}/>
          <Route path={"/counter"} element={
            <>
              <Counter/>
              <Counter/>
            </>
          }/>
        </Routes>
      </BrowserRouter>
    </Provider>
```

Selector

Selector

- Sebelumnya, kita sudah tahu untuk mendapatkan data State, kita bisa menggunakan Selector
- Kadang-kadang, kita ingin membuat Action yang digunakan untuk mendapatkan data di State, namun sayangnya bukan seperti itu cara Reducer
- Reducer bukan digunakan untuk mendapatkan data di State
- Untuk mendapatkan data yang lebih detail misalnya, kita bisa membuat Selector secara manual
- Cara membuatnya sama seperti ketika menggunakan Actions

Tugas

- Misal, kita akan coba tambahkan selector untuk getDoubleCounter()
- Isi selector ini adalah mengambil data counter, namun dikalikan 2

Kode : src/counterSlice.js

```
  },
  selectors: {
    getDoubleCounter(state) {
      return state * 2
    }
  }
})

export const {increment, decrement} = counterSlice.actions

export const {getDoubleCounter} = counterSlice.selectors

```

Kode : src/Counter.jsx

```jsx
}

const doubleCounter = useSelector(getDoubleCounter);

return (
  <div>
    <h1>Counter : {counter}</h1>
    <h1>Double Counter : {doubleCounter}</h1>
    <button onClick={handleIncrement}>Increment</button>
    <button onClick={() => dispatch(increment(2))}>Increment +2</button>
    <button onClick={handleDecrement}>Decrement</button>
    <button onClick={() => dispatch(decrement(2))}>Decrement -2</button>
  </div>
)
```

Selector dengan Parameter

- Selector mirip seperti Action, kita bisa tambahkan parameter
- Misal kita akan tambahkan selector baru dengan menambah parameter kali nya yang bisa dikirim oleh pengguna

Kode : src/counterSlice.js

```
22    selectors: {
23      getDoubleCounter(state) {
24        return state * 2
25      },
26      getCounter(state, value) {
27        return state * value
28      }
29    }
30  })
31
32  export const {increment, decrement} = counterSlice.actions
33
34  export const {getDoubleCounter, getCounter} = counterSlice.selectors
35
```

Kode : src/Counter.jsx

```jsx
16
17  const doubleCounter = useSelector(getDoubleCounter);
18  const tripleCounter = useSelector(state => getCounter(state, 3));
19
20  return (
21    <div>
22      <h1>Counter : {counter}</h1>
23      <h1>Double Counter : {doubleCounter}</h1>
24      <h1>Triple Counter : {tripleCounter}</h1>
25      <button onClick={handleIncrement}>Increment</button>
26      <button onClick={() => dispatch(increment(2))}>Increment +2</button>
27      <button onClick={handleDecrement}>Decrement</button>
28      <button onClick={() => dispatch(decrement(2))}>Decrement -2</button>
29    </div>
```

Tugas

Tugas

- Sekarang kita akan membuat TodoList Management yang memiliki multi halaman
- Kita akan manage halaman menggunakan React Router
- Kita akan manage data TodoList nya menggunakan Redux

Buat State untuk TodoList di Store

- Hal pertama yang akan kita buat adalah, State untuk todolist di Store
- TodoList ini adalah Array yang berisikan Object Todo
- Object Todo akan berisi dua attribute, id dan name
- Kita akan buat dalam file src/todoListSlice.js

Kode : src/useTodoList.js

todoListSlice.js

```
1  import {createSlice} from "@reduxjs/toolkit";
2
3  export const todoListSlice = createSlice({
4    name: "todoList",
5    initialState: [],
6    reducers: {
7    }
8  })
```

Buat Action untuk TodoList di Store

- Selanjutnya, kita akan tambahkan Action ke useTodoList
- Kita akan tambahkan beberapa Action ke useTodoList:
- Action addTodo({name}) untuk menambah todo
- Action removeTodo({id}) untuk menghapus todo
- Action updateTodo({id, name}) untuk mengupdate todo

Kode : src/useTodoList.js

```
reducers: {
  addTodo: (state, action) => {
    const {name} = action.payload;
    state.push({name, id: id++});
  },
  removeTodo: (state, action) => {
    const {id} = action.payload;
    const index = state.findIndex(todo => todo.id === id);
    if (index !== -1) {
      state.splice(index, 1);
    }
  },
```

```
  },
  updateTodo(state, action) {
    const {id, name} = action.payload;
    const todo = state.find(todo => todo.id === id);
    if (todo) {
      todo.name = name;
    }
  }
 }
})

export const {addTodo, removeTodo, updateTodo} =
  todoListSlice.actions;
```

Buat Selector untuk TodoList di Store

- Selanjutnya, kita akan tambahkan Selector ke useTodoList
- Selector getTodo(id) untuk mendapatkan todo

Kode : src/useTodoList.js

```
27    },
28    selectors: {
29      getTodo(state, id) {
30        return state.find(todo => todo.id === id);
31      },
32    }
33  })
34
35  export const {addTodo, removeTodo, updateTodo} =
36    todoListSlice.actions;
37
38  export const {getTodo} = todoListSlice.selectors;
39
```

Buat Komponen AddTodo

- Kita akan buat komponen untuk AddTodo
- Komponen ini digunakan untuk menambah data todo ke useTodoList

Kode : src/AddTodo.jsx

AddTodo.jsx

```jsx
import {useState} from "react";
import {useDispatch} from "react-redux";
import {addTodo} from "./todoListSlice.js";
import {useNavigate} from "react-router";

export default function AddTodo() {

  const [name, setName] = useState('');
  const dispatch = useDispatch();
  const navigate = useNavigate();
```

```jsx
function handleAddTodo() {
  dispatch(addTodo({name: name}))
  navigate("/todolist");
}

return (
  <div>
    <h1>Add Todo</h1>
    <input type="text" placeholder="Enter todo name"
           onChange={(e) => setName(e.target.value)}/>
    <button onClick={handleAddTodo}>Add</button>
  </div>
)
```

Buat Komponen UpdateTodo

- Kita akan buat komponen untuk UpdateTodo
- Komponen ini digunakan untuk mengubah data todo ke useTodoList

Kode : src/UpdateTodo.jsx

UpdateTodo.jsx

```jsx
import {useState} from "react";
import {useDispatch, useSelector} from "react-redux";
import {getTodo, updateTodo} from "./todoListSlice.js";
import {useNavigate, useParams} from "react-router";

export default function UpdateTodo() {

  const params = useParams();
  const todo = useSelector((state) =>
    getTodo(state, Number(params.id)));
  const [name, setName] = useState(todo.name);
  const dispatch = useDispatch();
  const navigate = useNavigate();
```

```jsx
function handleUpdateTodo() {
  dispatch(updateTodo({id: todo.id, name: name}))
  navigate("/todolist");
}

return (
  <div>
    <h1>Edit Todo</h1>
    <input type="text" value={name} placeholder="Enter todo name"
           onChange={(e) => setName(e.target.value)}/>
    <button onClick={handleUpdateTodo}>Add</button>
  </div>
)
```

Buat Komponen ListTodo

- Kita akan membuat komponen ListTodo
- Komponen ini digunakan untuk menampilkan daftar todo, link menuju add todo, action untuk edit todo dan hapus todo

Kode : src/ListTodo.jsx

ListTodo.jsx

```jsx
import {useDispatch, useSelector} from "react-redux";
import {Link} from "react-router";
import {removeTodo} from "./todoListSlice.js";

export default function ListTodo() {
  const todos = useSelector((state) => state.todoList);
  const dispatch = useDispatch();

  return (
    <div>
      <h1>List Todo</h1>
      <Link to={"/todolist/add"}>Add Todo</Link>
      <table>
        <thead>
        <tr>
          <th>Id</th>
          <th>Name</th>
          <th>Actions</th>
        </tr>
        </thead>
        <tbody>
        {todos.map((todo) => (
          <tr key={todo.id}>
            <td>{todo.id}</td>
            <td>{todo.name}</td>
            <td>
              <Link to={`/todolist/${todo.id}/edit`}>Update</Link>
              <button onClick={() => dispatch(removeTodo({id: todo.id}))}>
                Delete
              </button>
            </td>
          </tr>
        ))}
        </tbody>
      </table>
```

Tambahkan Routing

- Terakhir, kita tambahkan routing untuk aplikasi TodoList yang sudah kita buat

Kode : src/main.js

```
import ListTodo from "./ListTodo.jsx";
import AddTodo from "./AddTodo.jsx";
import UpdateTodo from "./UpdateTodo.jsx";
import {todoListSlice} from "./todoListSlice.js";

const store = configureStore({
  reducer: {
    counter: counterSlice.reducer,
    todoList: todoListSlice.reducer
  }
})
```

```
createRoot(document.getElementById('root')).render(
  <StrictMode>
    <Provider store={store}>
      <BrowserRouter>
        <Routes>
          <Route path={"/todolist"} element={<ListTodo/>}/>
          <Route path={"/todolist/add"} element={<AddTodo/>}/>
          <Route path={"/todolist/:id/edit"} element={<UpdateTodo/>}/>
          <Route path={"/"} element={<App/>}/>
          <Route path={"/counter"} element={
            <>
              <Counter/>
              <Counter/>
            </>
          }/>
        </Routes>
      </BrowserRouter>
    </Provider>
  </StrictMode>,
```

Materi Selanjutnya

Materi Selanjutnya

- Mulai membuat web frontend menggunakan ReactJS
- Perbanyak studi kasus menggunakan ReactJS