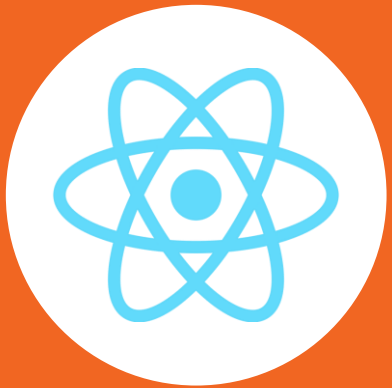
FPT Education

FPT POLYTECHNIC



LẬP TRÌNH ĐA NỀN TẢNG VỚI REACT NATIVE

BÀI 6: QUẢN LÝ STATE VÀ XỬ LÝ NETWORK BẰNG REDUX TOOLKIT

PHẦN 1: QUẢN LÝ STATE BẰNG REDUCER VÀ DISPATCH ACTION TRONG RTK

https://caodang.fpt.edu.vn/

Mục tiêu

- Xây dựng reducer trong **Redux Toolkit**
- Tìm hiểu về **createSlice, createAction**.
- Dispatch action, hiển thị dữ liệu trong reducer với **useDispatch** và **useSelector**

Reducer

- Để tạo reducer trong **Redux Toolkit** có 2 cách:
  - Cách 1: sử dụng **createSlice**
  - Cách 2: sử dụng **createReducer**

createSlice

- **createSlice** là gì?

  Một hàm chấp nhận **state** ban đầu, một đối tượng của các hàm **reducer** và 'tên **slice**' và tự động tạo các action và các **action type** tương ứng với các **reducer** và **state**.

  API này là cách tiếp cận tiêu chuẩn để viết logic **Redux**.

  API này là cách tiếp cận tiêu chuẩn để viết logic **Redux**.

- Tạo mới một file **counter.ts** để chứa một reducer, reducer này có chức năng thay đổi các giá trị liên quan đến biến đếm. Đầu tiên chúng ta import các thư viện **createSlice** và **PayloadAction**.

DaNenTang2 - counter.ts

```
1  import {createSlice} from '@reduxjs/toolkit';
2  import type {PayloadAction} from '@reduxjs/toolkit';
```

- Tiếp theo, khai báo các state ban đầu của reducer. Đây sẽ là các dữ liệu mà **reducer** sẽ chứa. Chúng ta có một **state** tên là **value** giá trị ban đầu là 0. Lưu ý bạn có thể lưu trữ nhiều kiểu dữ liệu khác nhau.

DaNenTang2 - counter.js

```
1  const initialState = {
2    value: 0,
3  };
```

createSlice

- Chúng ta gọi hàm **createSlice** để tạo reducer

```
export const counterSlice = createSlice({
  name: 'counter', initialState,
  reducers: {
    increment: state => {
      state.value += 1;
    },
    decrement: state => {
      state.value -= 1;
    },
    incrementByAmount: (state, action) => {
      state.value += action.payload;
    },
  }});
```

createSlice

- **name:** là tên của reducer, tên này được sử dụng trong action types
- **initialState:** các giá trị state ban đầu **reducer** xử lý.
- **reducers:** các hàm xử lý giá trị state trong reducer. Tên key sẽ được dùng để tạo **action**. Hàm '**builder callback**' được sử dụng để thêm nhiều bộ reducer hoặc một object bổ sung của '**case reducers**', trong đó các key phải là các **action type** khác

Parameters

- Các params của **createSlice:**
  - **initialState**

    Giá trị state ban đầu cho slice state này

    Đây cũng có thể là một hàm **"lazy initializer"**, hàm này sẽ trả về giá trị state ban đầu khi được gọi. Điều này sẽ được sử dụng bất cứ khi nào reducer được gọi với **undefined** là giá trị state của nó và chủ yếu hữu ích cho các trường hợp như đọc state ban đầu từ **localStorage**.

  - **name**

    Tên chuỗi cho slice **state** này. Hằng số **action type** được tạo ra sẽ sử dụng tên này làm tiền tố.

Parameters

- **reducers**

  Một đối tượng chứa các hàm **'case reducer' Redux** (các hàm nhằm xử lý một action type cụ thể, tương đương với một câu lệnh trường hợp duy nhất trong một switch).

- Params **extraReducers**

  Về mặt khái niệm, mỗi slice **reducer** 'sở hữu' slice state của nó. Ngoài ra còn có sự tương ứng tự nhiên giữa logic cập nhật được xác định bên trong **reducer** và các **action type** được tạo dựa trên chúng.

Parameters

- Ký hiệu "builder callback" **extraReducers**

  Cách sử dụng **extraReducers** được khuyến nghị là sử dụng **callback** nhận phiên bản **ActionReducerMapBuilder**.

  Ký hiệu trình tạo này cũng là cách duy nhất để thêm matcher reducers và default case reducers vào slice.

Ví dụ extraReducers

```
import {createAction, createSlice} from '@reduxjs/toolkit';
const incrementBy = createAction('incrementBy');
const decrement = createAction('decrement');

function isRejectedAction(action) {
  return action.type.endsWith('rejected');
}
```

- **incrementBy** và **decrement** là 2 action được tạo từ hàm **createAction. T**rong **createSlice** khi tạo reducer, nó sẽ tự tạo action kiểu như này, tên của action sẽ dựa vào key.
- **isRejectedAcion** là một hàm để bắt type action nếu đó bị **“rejected”**

Ví dụ extraReducers

```
createSlice({ name: 'counter', initialState: 0, reducers: {},
  extraReducers: builder => { builder
      .addCase(incrementBy, (state, action) => {
        // hành động được suy ra chính xác ở đây nếu sử dụng TS
      })
      .addCase(decrement, (state, action) => {})
      .addMatcher(
        isRejectedAction, (state, action) => {},
      )
      // và cung cấp trường hợp mặc định nếu không có case nào khác khớp
      .addDefaultCase((state, action) => {});
  },
});
```

Giá trị trả về của createSlice

- **createSlice** sẽ trả về một object trông như sau:

```
{
   name : string,
   reducer : ReducerFunction,
   actions : Record<string, ActionCreator>,
   caseReducers: Record<string, CaseReducer>.
   getInitialState: () => State
}
```

Mỗi hàm được xác định trong đối số **reducers** sẽ có một trình tạo action tương ứng được tạo bằng **createAction** và bao gồm trong trường **action** của kết quả bằng cách sử dụng cùng một tên hàm.

Ví dụ reducer

- Ví dụ, tạo một ứng dụng counter. Tiếp tục từ phần bài **createSlice.** Hoàn thành ví dụ, chúng ta có ứng dụng như sau:

Ví dụ reducer

- Bổ sung thêm hàm **addMatcher** trong **extraReducer** để bắt **action** được **dispatch.** Nếu user gửi **action RESET_COUNTER**
sẽ reset lai trở về state ban đầu

```
extraReducers: builder => {
  builder.addMatcher(
    action => action.type === RESET_COUNTER.type,
    () => {
      return initialState;
    },
  );
},
```

Ví dụ reducer

- Export các **reducer**, các action type này để dispatch action cho reducer nhận sự kiện.

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

export const CounterReducer = counterSlice.reducer;
```

Ví dụ reducer

- Tiếp theo, viết hook **useAppDispatch** dùng để dispatch các **action**. Hook **useAppSelector** dùng để lấy giá trị từ store

```
import {useDispatch, useSelector} from 'react-redux';

export const useAppDispatch = () => useDispatch();
export const useAppSelector = useSelector;
```

Ví dụ reducer

- Gọi hook **useAppSelector** để gọi state lên **UI, counter** là tên của **reducer**

```
const counter = useAppSelector(state => state.counter);
```

Ví dụ reducer

- Tiếp theo bạn gọi hook **useDispatch,** để dispatch các **action** cho **reducer.**

```
const dispatch = useAppDispatch();

const onIncreaseCounter = () => dispatch(increment());
const onDecrementCounter = () => dispatch(decrement());
const onMultiplyCounter = () => dispatch(multiply(3));
const onResetCounter = () =>
dispatch(RESET_COUNTER());
```

increment, decrement, multiply là các tên action tự động tạo trong reducer của **createSlice**. **RESET_COUNTER** là tên action được tạo từ **createAction**

Ví dụ reducer

- Hiển thị giá trị state, và gắn function vào giao diện.

```
<Text style={styles.counterText}>{counter?.value}</Text>
<Pressable onPress={onIncreaseCounter} style={styles.btn}>
  <Text>Tăng biến đếm</Text>
</Pressable>

<Pressable onPress={onDecrementCounter} style={styles.btn}>
  <Text>Giảm biến đếm</Text>
</Pressable>
```

Kết luận

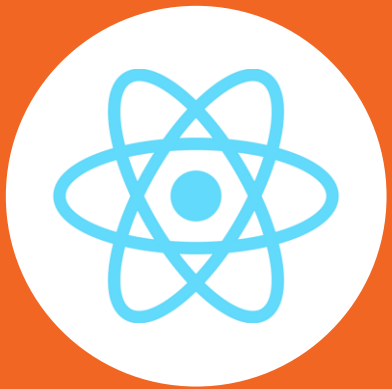
Tổng kết

Ở bài học trên các bạn đã học được cách, tạo reducer, dispatch action và hiển hiển thị dữ liệu từ store, thông qua một số API sau:

- **createSlice**: Tạo reducer và tự động tạo ra action cho reducer
- **createAction**: tạo action
- **useDispatch**: hook dùng để bắn action dựa vào action cung cấp
- **useSelector**: dùng để truy cập dữ liệu từ redux store

FPT Education

FPT POLYTECHNIC



LẬP TRÌNH ĐA NỀN TẢNG VỚI REACT NATIVE

BÀI 6: QUẢN LÝ STATE VÀ XỬ LÝ NETWORK BẰNG REDUX TOOLKIT

PHẦN 2: XỬ LÝ NETWORK BẰNG REDUX TOOLKIT

https://caodang.fpt.edu.vn/

Mục tiêu

- Giới thiệu về **Redux Toolkit query**
- Sử dụng hàm **createApi, fetchBaseQuery** để tạo một request
- Cách sử dụng **queries, mutation** và **transformResponse** trong **endpoints**

- **createApi** là gì?

  createApi tự động tạo các hook React cho từng **query** & **mutation** endpoints.

  Trước kia, để các bạn có thể thao tác với api, bạn phải sử dụng thư viện thứ 3, như **axios,** để gọi api. Đôi khi bạn còn phải sử dụng thêm thư viện **react-query** để xử lý thêm cho các query.

- Để sử dụng Query trong **RTK,** bạn import **createApi, fetchBaseQuery**

```
import {createApi, fetchBaseQuery} from '@reduxjs/toolkit/query/react';
```

  - **fetchBaseQuery():** Một wrapper nhỏ xung quanh **fetch** nhằm mục đích đơn giản hóa các request. Dự định là **baseQuery** được đề xuất sẽ được sử dụng trong **createApi** cho phần lớn người dùng.

- Tiếp theo chúng ta sẽ tạo một hàm xử lý api tên là **pokemonApi** để gọi api

```
export const pokemonApi = createApi({
  reducerPath: 'pokemonApi',
  baseQuery: fetchBaseQuery({
    baseUrl: 'https://pokeapi.co/api/v2/',
  }),
  endpoints: builder => ({
    getPokemonByName: builder.query({
      query: name => `pokemon/${name}`,
    })})});
```

- **reducerPath:** là khóa duy nhất mà service của bạn sẽ được gắn vào store của bạn.

- **baseQuery:** sử dụng kết hợp với **fetchBaseQuery,** chứa api gốc, bạn có thể truyền thêm header vào query và nhiều thứ khác vào query của mình.
- **endpoints:** chứa các function để gọi api, ở ví dụ trên chúng ta có hàm **getPokemonByName. PokemonType** là giá trị mà query sẽ nhận được, **string** là chuỗi tên pokemon, để thêm vào params cho query của chúng ta. Chúng ta sẽ truyền string này khi gọi hàm **getPokemonByName.**

- Từ hàm **getPokemonByName** create api trong endpoint sẽ tự động tạo ra 2 hook **useGetPokemonByNameQuery**, **useLazyGetPokemonByNameQuery**

```
export const {useGetPokemonByNameQuery,
useLazyGetPokemonByNameQuery} = pokemonApi;
```

- **useGetPokemonByNameQuery:** sẽ gọi query ngay tại screen khi mout
- **useLazyGetPokemonByNameQuery:** chỉ gọi query khi chúng ta gọi function trong **useLazyGetPokemonByNameQuery**

- **useGetPokemonByNameQuery:** sẽ tự động query khi screen được mount hoặc khi prop truyền vào payload được thay đổi giá trị. Kết quả trả về cho chúng ta các prop liên quan đến query.

```
const {data, refetch, isLoading} =
useGetPokemonByNameQuery(name);
```

- **data:** dữ liệu trả về khi query api
- **refetch:** function gọi lại api để update dữ liệu
- **isLoading:** trạng thái gọi dữ liệu (boolean)

- **useLazyGetPokemonByNameQuery:** hook cho phép chúng ta gọi query khi muốn

```
const [getPokemonByName, result] =
useLazyGetPokemonByNameQuery();
const {data, isFetching: isLoading} = result || {};
```

  - **getPokemonByName:** function gọi query của endpoint **getPokemonByName.**
  - **result:** kết quả của query, cũng tương tự như kết quả trả về từ **useGetPokemonByNameQuery**. Chúng ta có **data,** dữ liệu trả về, **isFetching** trạng thái gọi query

RTK Mutation

- **Mutations** được sử dụng để gửi cập nhật dữ liệu đến máy chủ và áp dụng các thay đổi cho bộ đệm cục bộ. **Mutations** cũng có thể làm invalidate dữ liệu được lưu trong bộ nhớ cache và buộc tìm nạp lại dữ liệu.
- Nếu muốn request **POST, PUT, DELETE, PATCH,**... sử dụng **mutation** theo hướng dẫn dưới đây:

```
updatePokemon: builder.mutation({
  query: ({name, body}) => ({
    url: `pokemon/${name}`,
    method: 'PATCH',
    body,
  })}),
```

RTK Mutation

- **PokemonResonseType** là type dữ liệu trả về của lệnh query,
- **PokemonDetaiQuerylType** là type dữ liệu body truyền lên query khi gọi hàm **updatePokemon**
- **method** là phương thức truyền lên query, có thể đặt **POST, PUT, GET, DELETE,** ...

- Cách gọi hàm gọi hook lệnh mutation vừa tạo. **updatePokemon** là function bắt đầu gọi lệnh query, **resultUpdatePokemon** là giá trị trả về của lệnh query

```
const [updatePokemon, resultUpdatePokemon] =
useUpdatePokemonMutation();

const onUpdatePokemon = () => {
  updatePokemon({name: 'bulbasaur', body: {name:
'bulbasaur22'}})};
```

Khi gọi function **updatePokemon** chúng ta truyền thêm body vào lệnh query. Tuỳ vào api, mà chúng ta sẽ truyền theo body tương ứng.

Header Query

- Để truyền header vào request, có thể sử dụng **prepareHeaders** trong **fetchBaseQuery**

  Cho phép bạn thêm **headers** vào request. Bạn có thể chỉ định headers ở endpoint cuối, nhưng thông thường bạn sẽ muốn đặt các headers phổ biến như **authorization** tại đây. Đây là một cơ chế tiện lợi, đối số thứ hai cho phép bạn sử dụng **getState** để truy cập store redux của bạn trong trường hợp bạn lưu trữ thông tin bạn sẽ cần ở đó, chẳng hạn như auth token.

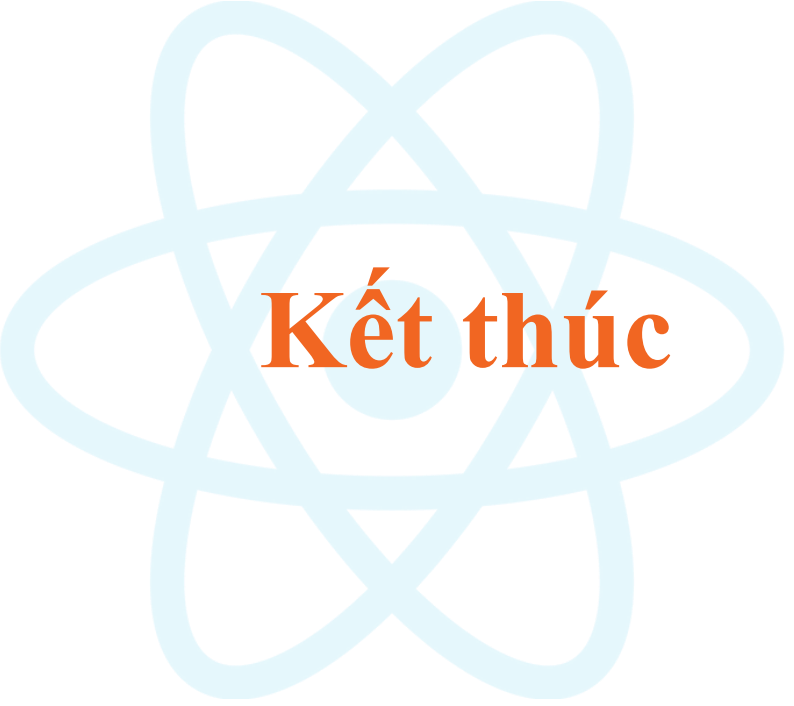
Header Query

- Để thêm token vào header sử dụng prop **headers để set header cho request** cách sau đây:

```
export const pokemonApi = createApi({
  reducerPath: 'pokemonApi',
  baseQuery: fetchBaseQuery({
    baseUrl: 'https://pokeapi.co/api/v2/',
    prepareHeaders: (headers, {getState}) => {
      const token = getState().auth.token;
      if (token) { headers.set('authorization', `Bearer ${token}`)}
      return headers;
    },
  }),
```

Tổng kết

- Ở chương query này, chúng ta đã được học cách gọi api từ **createApi**, cách gọi lệnh **GET** bằng **query**, lệnh **POST, DELETE, PUT, PATCH**, ... qua **mutation**
- Các bạn cũng đã được học cách truyền header thông qua prop **prepareHeaders** của **fetchBaseQuery**



Kết thúc