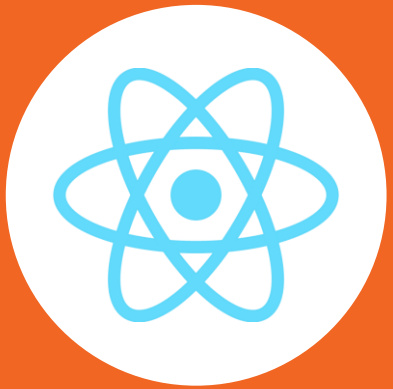
FPT Education

FPT POLYTECHNIC



LẬP TRÌNH ĐA NỀN TẢNG VỚI REACT NATIVE

BÀI 5: GIỚI THIỆU VỀ REDUX VÀ REDUX TOOLKIT

PHẦN 1: GIỚI THIỆU VỀ REDUX

Mục tiêu

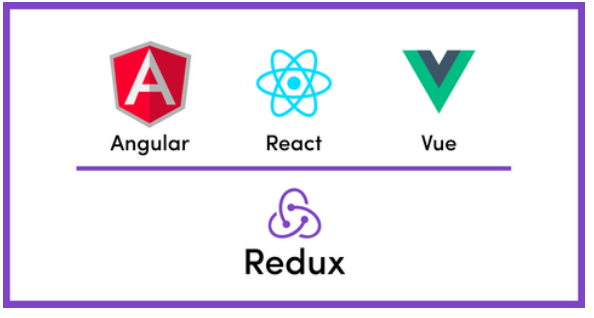
- [ ] Tìm hiểu các khái niệm về Redux
- [ ] Giới thiệu về kiến trúc của Redux
- [ ] Hiểu về store
- [ ] Hiểu cách action hoạt động

Giới thiệu Redux

Redux là gì? Store, Actions, và Reducers hoạt động như thế nào?

Redux là một vùng chứa trạng state có thể dự đoán được cho các ứng dụng JavaScript. Vậy điều đó thực sự có nghĩa là gì?

Nếu chúng ta đào sâu hơn vào câu câu hỏi này, chúng ta thấy rằng Redux là một thư viện quản lý state mà bạn có thể sử dụng với bất kỳ thư viện hoặc khung JS nào như React, Angular hoặc Vue.



Tại sao chúng ta lại phải sử dụng Redux

Một ứng dụng sẽ có các state của nó, có thể là sự kết hợp của các state của các component bên trong của nó.

Hãy lấy một trang web thương mại điện tử làm ví dụ. Một trang web thương mại điện tử sẽ có một số component như component giỏ hàng, component hồ sơ người dùng, component phần đã xem trước đó, v.v.

Chúng tôi sẽ lấy thành phần giỏ hàng hiển thị số lượng mặt hàng trong giỏ hàng của người dùng. State của component giỏ hàng sẽ bao gồm tất cả các mặt hàng mà người dùng đã thêm vào giỏ hàng và tổng số mặt hàng đó. Tại mọi thời điểm ứng dụng được thiết lập và chạy, component này phải hiển thị số lượng mặt hàng được cập nhật trong giỏ hàng của người dùng.

Bất cứ khi nào người dùng thêm một mặt hàng vào giỏ hàng, ứng dụng phải xử lý nội bộ hành động đó bằng cách thêm mục đó vào đối tượng giỏ hàng. Nó phải duy trì state của nó trong local và cũng hiển thị cho người dùng tổng số mặt hàng trong giỏ hàng trong giao diện người dùng.

Tương tự, việc xóa một mặt hàng khỏi giỏ hàng sẽ làm giảm số lượng mặt hàng trong giỏ hàng bên trong. Nó sẽ xóa mặt hàng khỏi object giỏ hàng và cũng hiển thị tổng số mặt hàng được cập nhật trong giỏ hàng trong giao diện người dùng.

Chúng ta có thể duy trì rất tốt trạng thái bên trong của các component bên trong chúng, nhưng khi một ứng dụng phát triển lớn hơn, nó có thể phải chia sẻ một số state giữa các component. Điều này không chỉ để hiển thị chúng trong chế độ xem mà còn để quản lý hoặc cập nhật chúng hoặc thực hiện một số logic dựa trên giá trị của chúng.

Giới thiệu Redux

Nhiệm vụ xử lý nhiều state từ nhiều component một cách hiệu quả này có thể trở nên khó khăn khi ứng dụng tăng kích thước.

Vì vậy Redux ra đời, nó là một thư viện quản lý trạng thái, Redux về cơ bản sẽ lưu trữ và quản lý tất cả các state của ứng dụng.

Nó cũng cung cấp cho chúng ta một số API quan trọng mà chúng ta có thể thực hiện các thay đổi đối với state hiện tại cũng như tìm nạp state hiện tại của ứng dụng.

- Điều gì làm cho Redux có thể dự đoán được?

State là Read-only trong Redux. Điều làm cho Redux có thể dự đoán được là để thực hiện thay đổi state của ứng dụng, chúng ta cần dispatch một action mô tả những thay đổi chúng ta muốn thực hiện trong state.

Những action này sau đó được thực hiện bởi một thứ được gọi là bộ reducer, công việc duy nhất của nó là chấp nhận hai thứ (action và state hiện tại của ứng dụng) và trả về một phiên bản cập nhật mới của state.

Chúng ta sẽ nói nhiều hơn về các action và reducer trong các phần sau.

Lưu ý rằng reducer không thay đổi bất kỳ phần nào của state. Thay vào đó, một reducer tạo ra một phiên bản mới của state với tất cả các cập nhật cần thiết.

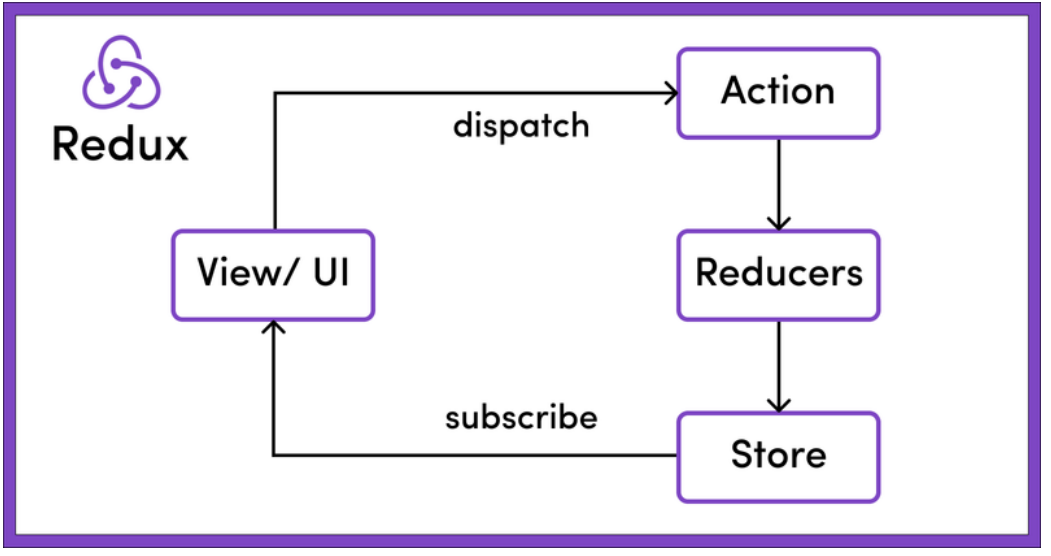
Tiếp tục với ví dụ trên của chúng ta về một trang web thương mại điện tử, nếu state ban đầu của giỏ hàng là nó có 0 mặt hàng, thì một action thêm một mặt hàng vào giỏ hàng sẽ tăng số lượng mặt hàng trong giỏ hàng lên 1. Và dispatch action thêm một mặt hàng vào giỏ hàng một lần nữa sẽ tăng số lượng mặt hàng trong giỏ hàng lên 2.

Với state ban đầu, với một danh sách action cụ thể theo một thứ tự cụ thể, nó sẽ luôn cung cấp cho chúng ta cùng một state cuối cùng của thực thể. Đây là cách Redux làm cho quản lý có thể dự đoán được.

Trong phần sau, chúng ta sẽ đi sâu vào các khái niệm cốt lõi của Redux – store, actions and reducers.

Nguyên lý hoạt động

- Kiến trúc của Redux



Redux Store là gì?

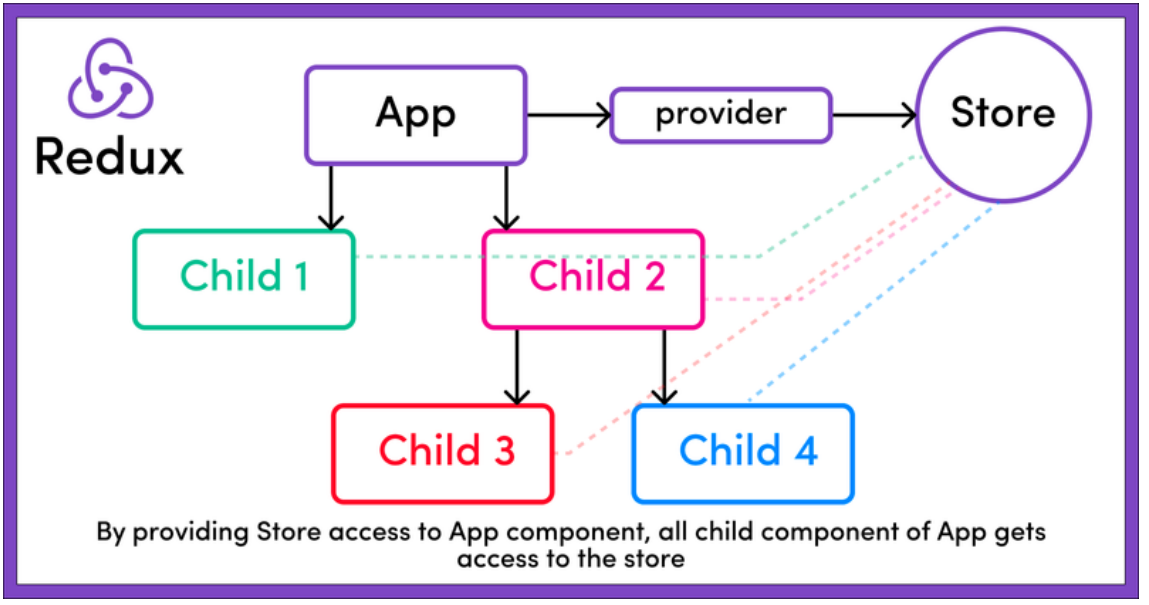
Redux store là vùng lưu trữ chính, trung tâm lưu trữ tất cả các state của một ứng dụng. Nó được xem xét và duy trì như một single source of truth cho state của ứng dụng.

Nếu store được cung cấp cho App.tsx (bằng cách gói component App trong thẻ <Provider> </Provider>) như được hiển thị trong đoạn mã bên dưới, thì tất cả các thành phần con của nó (các thành phần con của App.js) cũng có thể truy cập state của ứng dụng từ store. Điều này làm cho nó hoạt động như một state toàn cục.

Redux store

```
// src/index.js

import React from 'react'
import ReactDOM from 'react-dom'
import { Provider } from 'react-redux'

import { App } from './App'
import createStore from './createReduxStore'

const store = createStore()

// As of React 18
const root = ReactDOM.createRoot(document.getElementById('root'))
root.render(
  <Provider store={store}>
    <App />
  </Provider>
)
```

Redux store


Redux
App
provider
Store
Child 1
Child 2
Child 3
Child 4
By providing Store access to App component, all child component of App gets access to the store

Redux store

- State của toàn bộ ứng dụng được lưu trữ dưới dạng cây đối tượng JS trong một store duy nhất như dưới đây:

```
{
   noOfItemInCart: 2,
   cart: [
   {
      bookName: "Harry Potter and the Chamber of Secrets",
      noOfItem: 1,
   },
   {
      bookName: "Harry Potter and the Prisoner of Azkaban",
      noOfItem: 1
   }]
}
```

Action trong Redux

Action trong Redux là gì?

Cách duy nhất để thay đổi state là phát ra một action, đó là một đối tượng mô tả những gì đã xảy ra. State trong Redux là chỉ đọc. Điều này giúp bạn hạn chế bất kỳ phần nào của chế độ xem hoặc bất kỳ cuộc gọi network nào để ghi/cập nhật state trực tiếp.

Thay vào đó, nếu bất cứ ai muốn thay đổi state của ứng dụng, thì họ sẽ cần thể hiện ý định làm như vậy bằng cách phát ra hoặc gửi một action.

Thay vào đó, nếu bất cứ ai muốn thay đổi state của ứng dụng, thì họ sẽ cần thể hiện ý định làm như vậy bằng cách phát ra hoặc dispatch một action.

Action trong Redux

Hãy lấy ví dụ về ví dụ cửa hàng ở trên, nơi chúng ta có 2 cuốn sách trong cửa hàng: 'Harry Potter và the Chamber of Secrets' và 'Harry Potter và the Prisoner of Azkaban'. Chỉ có một bản sao của mỗi cái.

Bây giờ nếu người dùng muốn thêm một mặt hàng khác vào giỏ hàng, thì họ sẽ phải nhấp vào nút "Add to Cart" bên cạnh mặt hàng.

Khi nhấp vào nút 'Thêm vào giỏ hàng', một action sẽ được gửi đi. Action này không gì khác ngoài một đối tượng JS mô tả những thay đổi cần được thực hiện trong cửa hàng. Một cái gì đó như thế này:

Action trong Redux

```
const dispatch = useDispatch()

const addItemToCart = () => {
return {
    type: "ADD_ITEM_TO_CART"
    payload: {
        bookName: "Harry Potter and the Goblet of Fire",
        noOfItem: 1,
        }
    }
}

<button onClick = {() => dispatch(addItemToCart())}>Add to cart</button>
```

Action trong Redux

Lưu ý cách trong ví dụ trên, chúng tôi dispatch một action khi nhấp vào nút. Hay đúng hơn, để cụ thể hơn, chúng tôi dispatch một cái gì đó được gọi là trình tạo action - đó là hàm addItemToCart (). Điều này lần lượt trả về một action là một đối tượng JS thuần túy mô tả mục đích của action được biểu thị bằng khóa type cùng với bất kỳ dữ liệu nào khác cần thiết cho sự thay đổi state. Trong trường hợp này, đó là tên của cuốn sách sẽ được thêm vào giỏ hàng được biểu thị bằng khóa payload.

Mỗi action phải có ít nhất một type liên quan đến nó. Bất kỳ chi tiết nào khác cần được thông qua là tùy chọn và sẽ phụ thuộc vào type action chúng ta dispatch.

Action trong Redux

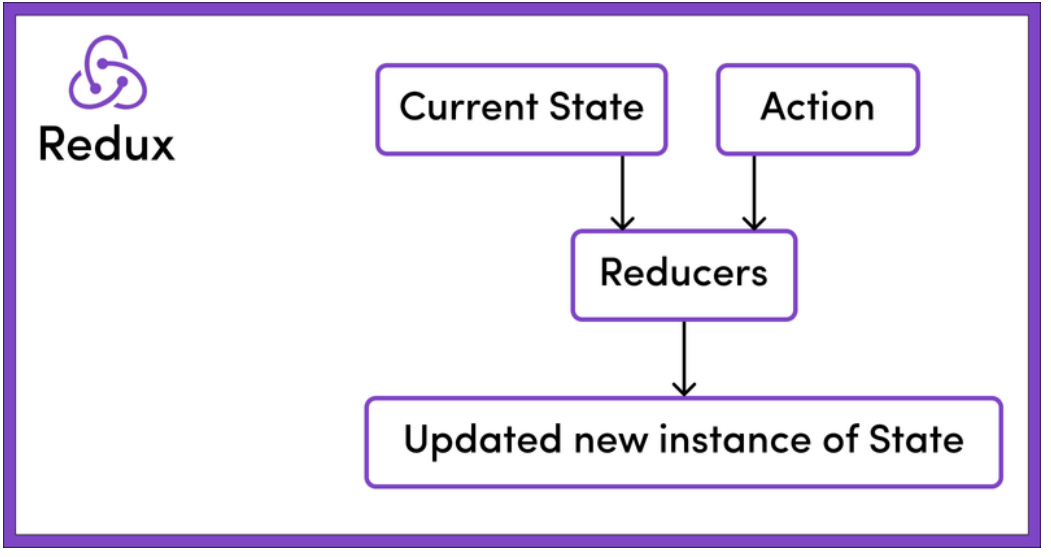
Ví dụ: đoạn mã trên gửi action sau:

```
//Hành động được tạo bởi người tạo hành động addItemToCart()
{
    type: "ADD_ITEM_TO_CART" // Note: Mỗi action phải có một key
    payload:
    {
        bookName: "Harry Potter and the Goblet of Fire",
        noOfItem: 1
    }
}
```

Reducer

Reducer trong Redux là gì?

Để chỉ định cách cây state được biến đổi bằng các action, chúng ta viết các bộ pure reducers.



Reducer

Reducer, như tên cho thấy, có hai điều: state trước đó và một action. Sau đó, họ giảm nó (đọc nó trở lại) thành một thực thể: phiên bản state cập nhật mới.

Vì vậy, reducer về cơ bản là các hàm JS thuần túy nhận state trước đó và một action và trả về state mới được cập nhật.

Có thể có một reducer nếu đó là một ứng dụng đơn giản hoặc nhiều reducer thuộc các phần hoặc slide khác nhau của state toàn cục trong một ứng dụng lớn hơn.

Ví dụ: có thể có một bộ reducer xử lý trạng thái của giỏ hàng trong ứng dụng mua sắm, sau đó có thể có bộ reducer xử lý phần chi tiết người dùng của ứng dụng, v.v.

Reducer

Bất cứ khi nào một action được gửi đi, tất cả các bộ reducer đều được kích hoạt. Mỗi bộ reducer lọc ra action bằng cách sử dụng một câu lệnh lọc dựa trên action type. Bất cứ khi nào câu lệnh switch khớp với hành động được thông qua, các bộ reducer tương ứng sẽ thực hiện hành động cần thiết để thực hiện cập nhật và trả về một phiên bản mới của state toàn cục.

Tiếp tục với ví dụ trên, chúng ta có thể có một bộ reducer như sau:

```
const initialCartState = {
    noOfItemInCart: 0,
    cart: []
}
```

Reducer

```
const cartReducer = (state = initialCartState, action) => {
   switch (action.type) {
      case "ADD_ITEM_TO_CART":
         return {
            ...state,
            noOfItemInCart: state.noOfItemInCart + 1,
            cart : [
               ...state.cart,
               action.payload
            ]
         }
      case "DELETE_ITEM_FROM_CART":
         return {
            // Các login còn lại
         }
      default:
         return state
   }
}
```

Reducer

Bạn cần phải khai báo giá trị ban đầu cho reducer, như ví dụ trên là initialCartState, để tránh xử lý trường hợp gọi các bộ reducer lần đầu tiên khi state có thể undefine.

Trong đoạn code trên, chúng ta đã tạo ra một reducer gọi là cartReducer là một hàm JS thuần túy. Hàm này chấp nhận hai tham số: state và action.

Trong ví dụ trên, khi nhấp vào nút, chúng ta đã dispatch một action tên là addItemToCart (). Trình tạo action này đã dispatch một action type ADD_ITEM_TO_CART.

Reducer

Tiếp theo, chúng ta đã tạo một reducer gọi là cartReducer lấy state (với state ban đầu mặc định) và action làm tham số. Nó lựa chọn hàm thực thi trong reducer dựa trên action type và sau đó bất kỳ case nào khớp với action type được gửi đi, nó sẽ thực hiện cập nhật cần thiết và trả về phiên bản mới của state cập nhật.

Lưu ý ở đây rằng state trong redux là bất biến. Vì vậy, các bộ reducer tạo một bản sao của toàn bộ state hiện tại trước, thực hiện các thay đổi cần thiết và sau đó trả về một phiên bản mới của state - với tất cả các thay đổi/cập nhật cần thiết.

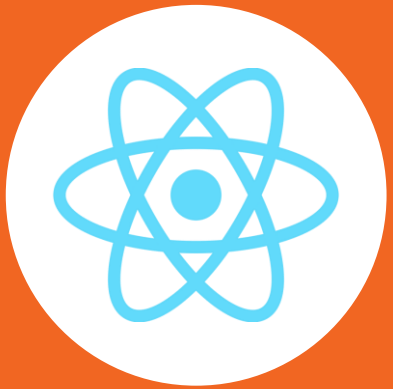
Tổng kết

Nói tóm lại, ba nguyên tắc sau đây là cách Redux hoạt động:

- State toàn cục của một ứng dụng được lưu trữ trong một object tree trong một store duy nhất
- Cách duy nhất để thay đổi state là emit một action, đó là một đối tượng mô tả những gì đã xảy ra
- Để chỉ định cách state tree được biến đổi bằng các action, chúng ta viết các bộ pure reducers.

FPT Education

FPT POLYTECHNIC



LẬP TRÌNH ĐA NỀN TẢNG VỚI REACT NATIVE

BÀI 5: GIỚI THIỆU VỀ REDUX VÀ REDUX TOOLKIT

PHẦN 2: GIỚI THIỆU VỀ REDUX TOOLKIT

Mục tiêu

- [ ] Giới thiệu về Redux Tookit
- [ ] Tìm hiểu các API của Redux Toolkit và Redux Query
- [ ] Cài đặt Redux Toolkit và Redux persist.

Redux toolkit là gì?

Package Redux Toolkit được dự định là cách tiêu chuẩn để viết logic Redux. Ban đầu nó được tạo ra để giúp giải quyết ba mối quan tâm phổ biến về Redux:

- “Cấu hình store Redux quá phức tạp”
- “Tôi phải thêm rất nhiều gói để Redux làm bất cứ điều gì hữu ích”
- “Redux yêu cầu quá nhiều mã soạn sẵn”

Redux Toolkit cũng bao gồm khả năng fetching dữ liệu và lưu vào bộ nhớ đệm mạnh mẽ mà chúng tôi đã đặt tên là 'RTK Query'. Nó được bao gồm trong package dưới dạng một tập hợp các điểm vào riêng biệt. Nó là tùy chọn, nhưng có thể loại bỏ sự cần thiết phải tự viết tay logic fetching dữ liệu.

Redux Toolkit dựa trên kiến trúc của Redux, phần giới thiệu về Redux các bạn đã được giới thiệu từ phần bài ở trên.

- Redux Toolkit bao gồm các API sau:
  - configureStore(): createStore để cung cấp các tùy chọn cấu hình đơn giản hóa và mặc định tốt. Nó có thể tự động kết hợp các slice reducers của bạn, thêm bất kỳ Redux middleware nào bạn cung cấp, bao gồm redux-thunk theo mặc định và cho phép sử dụng Tiện ích mở rộng Redux DevTools.
  - createReducer(): Điều đó cho phép bạn cung cấp bảng tra cứu các action type cho các hàm reducer, thay vì viết câu lệnh switch. Ngoài ra, nó tự động sử dụng thư viện immer để cho phép bạn viết các bản cập nhật immutable đơn giản hơn với mutative code thông thường, như state.todos[3].completed = true.

- createAction(): tạo ra một hàm tạo action cho chuỗi type acton đã cho. Bản thân hàm có toString() được định nghĩa, để nó có thể được sử dụng thay cho hằng số kiểu.
- createSlice(): chấp nhận một đối tượng gồm các hàm reducer, tên slice và giá trị state ban đầu và tự động tạo slice reducer action và action type tương ứng.
- createAsyncThunk: chấp nhận một chuỗi action type và một hàm trả về promise và tạo ra một thunk dispatche các action type đang chờ xử lý/ thực hiện/ bị từ chối dựa trên promise đó
- createAsyncThunk: chấp nhận một chuỗi action type và một hàm trả về promise và tạo ra một thunk dispatche các action type pending/fulfilled/rejected dựa trên promise đó

Giới thiệu

- createEntityAdapter: tạo một tập hợp các reducer và selectors có thể tái sử dụng để quản lý dữ liệu chuẩn hóa trong store
- createSelector từ thư viện Reselect, được xuất lại để dễ sử dụng.

RTK Query là gì?

RTK Query được cung cấp dưới dạng thêm vào tùy chọn trong gói @reduxjs/toolkit. Nó được xây dựng nhằm mục đích giải quyết trường hợp sử dụng fetching dữ liệu và lưu vào bộ nhớ đệm, cung cấp bộ công cụ nhỏ gọn nhưng mạnh mẽ để xác định lớp API interface cho ứng dụng của bạn. Nó nhằm đơn giản hóa các trường hợp phổ biến để tải dữ liệu trong ứng dụng web, loại bỏ nhu cầu tự viết tay logic fetching dữ liệu & bộ nhớ đệm.

RTK Query được xây dựng trên lõi Redux Toolkit để triển khai, sử dụng Redux nội bộ cho kiến trúc của nó. Mặc dù kiến thức về Redux và RTK không bắt buộc để sử dụng RTK Query, bạn nên khám phá tất cả các khả năng quản lý store toàn cầu bổ sung mà nó cung cấp, cũng như cài đặt tiện ích mở rộng trình duyệt Redux DevTools, hoạt động hoàn hảo với RTK Query để

duyệt qua và phát lại dòng thời gian của yêu cầu &; hành vi bộ nhớ cache của bạn.

RTK Query được bao gồm trong quá trình cài đặt gói Redux Toolkit cốt lõi. Nó có sẵn thông qua một trong hai điểm dưới đây:

```
import { createApi } from '@reduxjs/toolkit/query'

/* Điểm vào dành riêng cho react tự động tạo các hook tương ứng với các điểm cuối đã xác định */
import { createApi } from '@reduxjs/toolkit/query/react'
```

- RTK Query bao gồm các API sau:
  - createApi(): Chức năng cốt lõi của RTK Query. Nó cho phép bạn xác định một tập hợp các endpoints và mô tả cách truy xuất dữ liệu từ một loạt các endpoints, bao gồm cấu hình về cách fetch và chuyển đổi dữ liệu đó. Trong hầu hết các trường hợp, bạn nên sử dụng tùy chọn này một lần cho mỗi ứng dụng, với "một slice API cho mỗi URL cơ sở" làm quy tắc chung.
  - fetchBaseQuery(): Một wrapper nhỏ xung quanh fetch nhằm mục đích đơn giản hóa các yêu cầu. Dự định là baseQuery được đề xuất sẽ được sử dụng trong createApi cho phần lớn người dùng.

- <ApiProvider />: Có thể được sử dụng làm Provider nếu bạn chưa có cửa hàng Redux.
- setupListeners(): Một tiện ích được sử dụng để kích hoạt các hành vi refetchOnMount và refetchOnReconnect.

Cài đặt thư viện

- Để sử dụng RTK, các bạn cài đặt các thư viện theo câu lệnh sau:

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

- Cài thư viện thành công, sau đó bạn cần thực hiện vài bước setup redux và RTK trong project của mình

Setup store

- Tạo tệp có tên src/redux/store.js. Nhập configureStore API từ Redux Toolkit. Chúng ta sẽ bắt đầu bằng cách tạo một store Redux trống và xuất nó:

DaNenTang2 - store.ts

```ts
import {configureStore} from '@reduxjs/toolkit';

export const store = configureStore({
  reducer: {},
});

// Suy ra các loại 'RootState' và 'AppDispatch' từ chính store
export type RootState = ReturnType<typeof store.getState>;
// Loại suy luận: {posts: PostsState, comments: CommentsState, users: UsersState}
export type AppDispatch = typeof store.dispatch;
```

Điều này tạo ra một Redux store và cũng tự động định cấu hình tiện ích mở rộng Redux DevTools để bạn có thể kiểm tra store trong khi phát triển.

Setup store

- Khi store được tạo, chúng ta có thể làm cho nó có sẵn cho các React component của chúng ta bằng cách đặt một React-Redux <Provider> bọc ứng dụng của chúng ta trong App.tsx. Nhập Redux store mà chúng ta vừa tạo vào <Provider>.

DaNenTang2 - App.tsx

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

function App(): JSX.Element {
  return (
    <Provider store={store}>
      <SafeAreaProvider>
        <MusicScreen />
      </SafeAreaProvider>
    </Provider>
  );
}
```

Redux persist

- Tới đây cơ bản bạn đã setup store cơ bản xong. Nhưng trong quá trình phát triển ứng dụng. Bạn sẽ không muốn các dữ liệu trong store mất đi khi tắt ứng dụng. Ví dụ, trước kia bạn lưu thông tin cá nhân, token, ... trong AsyncStorage. Thì bây giờ bạn có thể sử dụng thư viện redux-persits để lưu thông tin của Redux store vào AsyncStorage.
- Redux persist là gì?

  Redux Persist là một công cụ được sử dụng để lưu liền mạch object state Redux của ứng dụng vào AsyncStorage. Khi khởi chạy ứng dụng, Redux Persist truy xuất state cũ và lưu nó trở lại Redux.

Redux persist

Thông thường, Redux set state ban đầu của ứng dụng khi khởi chạy. Ngay sau đó, Redux Persist tìm nạp state cũ của bạn, ghi đè lên bất kỳ state ban đầu nào trong một quá trình mà nhóm Redux Persist gọi là bù nước.

Redux là một thư viện JavaScript mã nguồn mở để quản lý và tập trung trạng thái ứng dụng. Nó duy trì state của toàn bộ ứng dụng trong một object state bất biến duy nhất, không thể thay đổi trực tiếp. Khi một cái gì đó thay đổi, một object mới được tạo bằng cách sử dụng các action và reducer.

Setup Redux persist

- Để sử dụng redux persist bạn phải chọn một package dùng để lưu lại các state của store, ở bài này chúng ta sẽ sử dụng thư viện @react-native-async-storage/async-storage.

```
npm install @react-native-async-storage/async-storage
```

- Cài thêm thư viện redux persits

```
npm i redux-persist
```

Setup Redux persist

- Trở lại file store.js chúng ta sẽ config thêm redux persists. Import thêm các package sau:

DaNenTang2 - store.ts

```
1  import {combineReducers} from '@reduxjs/toolkit';
2
3  import AsyncStorage from '@react-native-async-storage/async-storage';
4  import {persistStore, persistReducer} from 'redux-persist';
5  import autoMergeLevel2 from 'redux-persist/es/stateReconciler/autoMergeLevel2';
```

- Gọi combineReducers để chứa tất cả các reducer trong ứng dụng. Phần này chúng ta sẽ tìm hiểu ở bài sau:

Setup Redux persist

DaNenTang2 - store.ts

```
const rootReducer = combineReducers({
  // ...Các reducer thêm vào đây
});
export type RootState = ReturnType<typeof rootReducer>;
```

- Tạo object persistConfig, chứa các cấu hình của persist, gồm key, store là vùng chứa state của store, whitelist là tên reducer bạn muốn lưu state lại, stateReconciler quyết định cách thức xử lý sự khác biệt giữa state mới và state cũ khi chúng được lấy từ Storage. autoMergeLevel2 sẽ so sánh hợp nhất object level 2 của object.

Setup Redux persist

DaNenTang2 - store.ts

```
const persistConfig = {
  key: 'root',
  storage: AsyncStorage,
  whitelist: ['Tên reducer bạn muốn lưu lại'],
  stateReconciler: autoMergeLevel2,
};
```

Setup Redux persist

- Đưa object persistConfig và rootReducer vào persistReducer. Sau đó thêm persistReducer kết hợp lại các reducer.

DaNenTang2 - store.ts

```
1  const persistedReducer = persistReducer<RootState>(persistConfig, rootReducer);
2
3  export const store = configureStore({
4    reducer: persistedReducer,
5  });
6
7  export const persistor = persistStore(store);
8
```

Setup Redux persist

- Bọc PersistGate lại toàn bộ app bên dưới Provider, sau đó thêm persistor chúng ta đã config ở store vào.

DaNenTang2 - App.tsx

```tsx
import {PersistGate} from 'redux-persist/integration/react';

function App(): JSX.Element {
  return (
    <Provider store={store}>
      <PersistGate loading={null} persistor={persistor}>
        <SafeAreaProvider>
          <MusicScreen />
        </SafeAreaProvider>
      </PersistGate>
    </Provider>
  );
}
```

Kết luận

- Ở bài học này, các bạn đã được giới thiệu về Redux, Redux Toolkit các api của RTK, cách cài đặt thư viện, và thư viện bổ trợ lưu trữ dữ liệu Redux persist. Ở bài học sau, các bạn sẽ được học cách áp dụng RTK vào ứng dụng thực tế.

Tổng kết

- Tìm hiểu các khái niệm về Redux
- Giới thiệu về kiến trúc của Redux
- Hiểu về store
- Hiểu cách action hoạt động
- Giới thiệu về Redux Tookit
- Tìm hiểu các API của Redux Toolkit và Redux Query
- Cài đặt Redux Toolkit và Redux persist.

Kết thúc