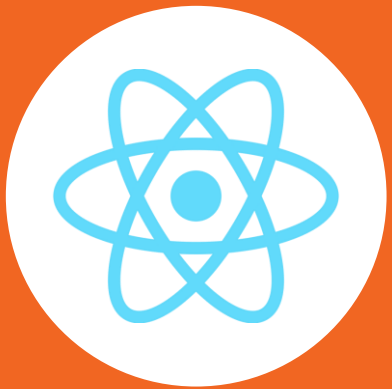
FPT Education

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

FPT Education
FPT POLYTECHNIC

Mục tiêu

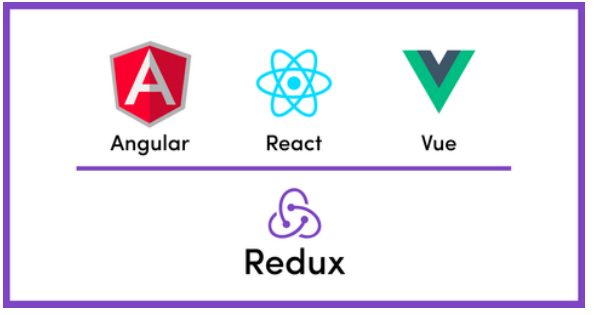
- Tìm hiểu các khái niệm về **Redux**
- Giới thiệu về kiến trúc của **Redux**
- Hiểu về **store**
- Hiểu cách **action** hoạt động

Giới thiệu Redux

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

Redux là một vùng chứa trạng state có thể sử dụng được ở bất cứ đâu trong ứng dụng. Vậy điều đó thực sự có nghĩa là gì?

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.



Giới thiệu Redux

- Tại sao 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.

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.

Giới thiệu Redux

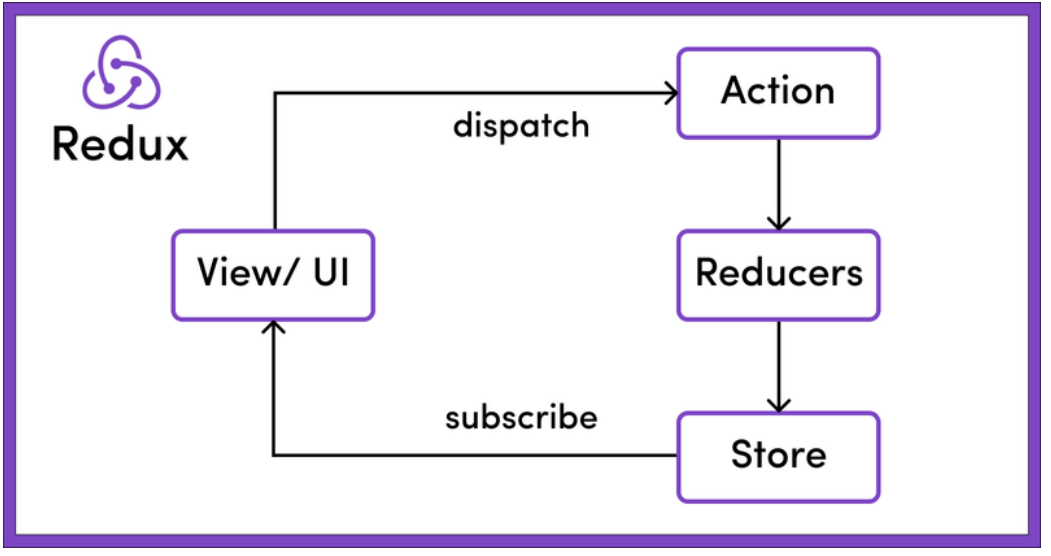
- Điều gì làm cho Redux có thể dự đoán được?

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.**

Nguyên lý hoạt động

- Kiến trúc của Redux



Redux Store là gì?

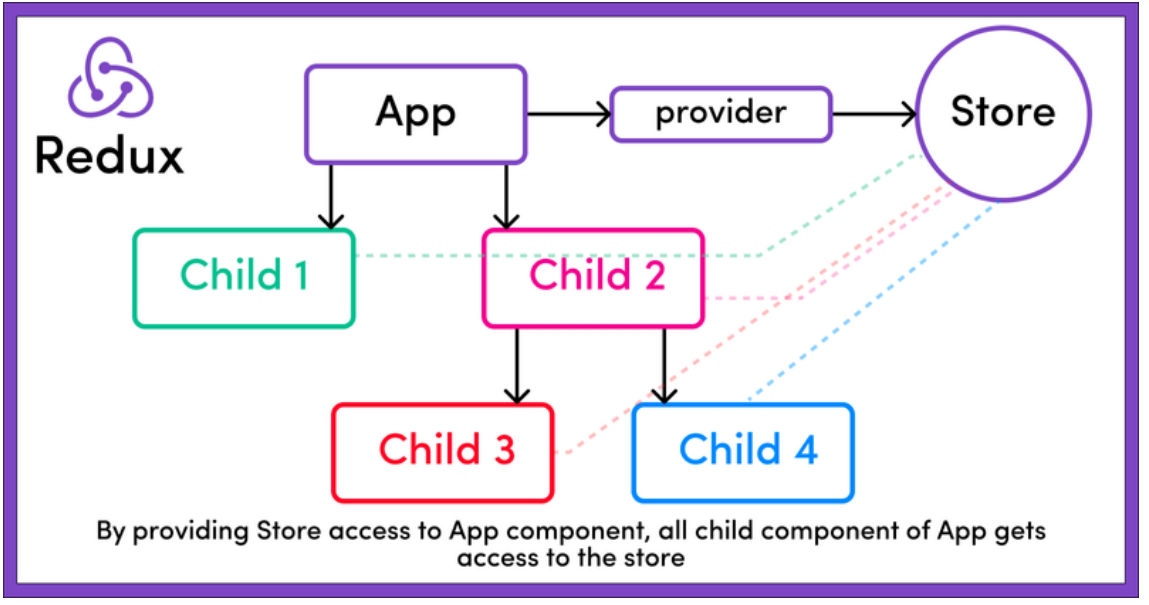
Redux store là vùng lưu trữ chính, trung tâm lưu trữ tất cả các state của một ứng dụng. Nó được xem xét và duy trì như một **single source of truth** cho state của ứng dụng.

Nếu **store** được cung cấp cho **App.tsx** (bằng cách gói component App trong thẻ **<Provider> </Provider>**), 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 object cây 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**.

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

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

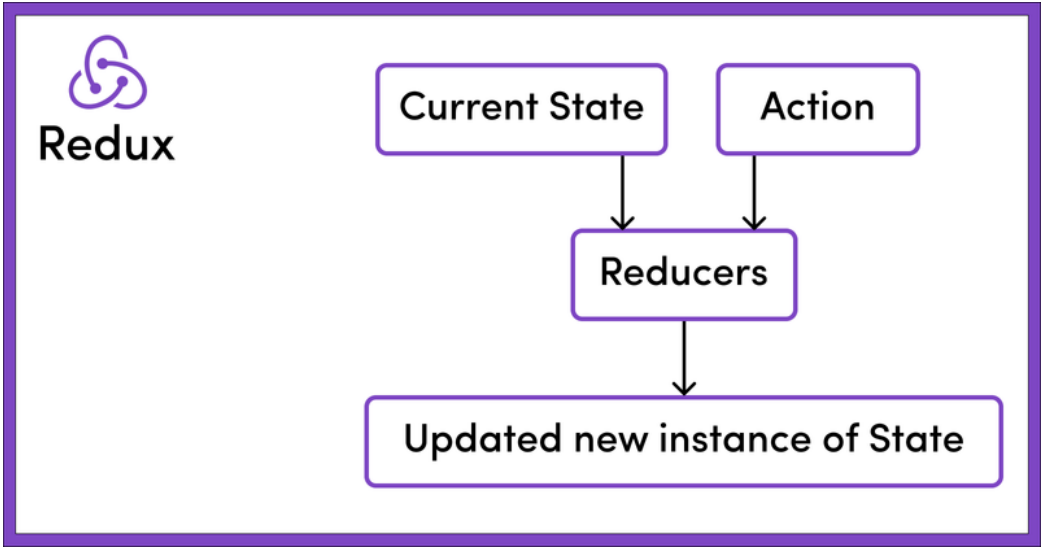
Ví dụ: đoạn mã trên gửi action sau:

```
  //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, cần 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.

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

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.

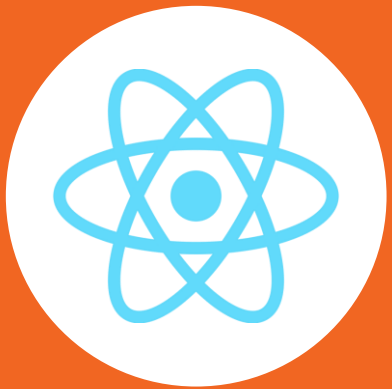
Tổng kết

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.

Giới thiệu

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.

- **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

- **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.

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.

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

```
1  import {configureStore} from '@reduxjs/toolkit';
2
3  export const store = configureStore({
4    reducer: {},
5  });
6
```

Setup store

- Khi **store** được tạo, bọc **<Provider>** vào ứng dụng trong **App.tsx**. Thêm **Redux store** mà chúng ta vừa tạo vào **<Provider>.**

```
import {Provider} from 'react-redux';

function App() {
  return (
    <Provider store={store}>
        <SafeAreaProvider>
          <HomeScreen />
        </SafeAreaProvider>
    </Provider>
  )}
```

Redux persist

- Sử dụng thư viện **redux-persits** để lưu thông tin của **Redux store** vào **AsyncStorage.** Ví dụ, lưu thông tin cá nhân, token, ... trong **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 Persis**t truy xuất state cũ và lưu nó trở lại Redux.

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:

```
import {combineReducers, configureStore} from '@reduxjs/toolkit';
import AsyncStorage from '@react-native-async-storage/async-storage';
import {persistStore, persistReducer} from 'redux-persist';
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
});
```

- 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

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

Setup Redux persist

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

```
const persistedReducer = persistReducer(persistConfig, rootReducer);

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

export const persistor = persistStore(store);
```

Setup Redux persist

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

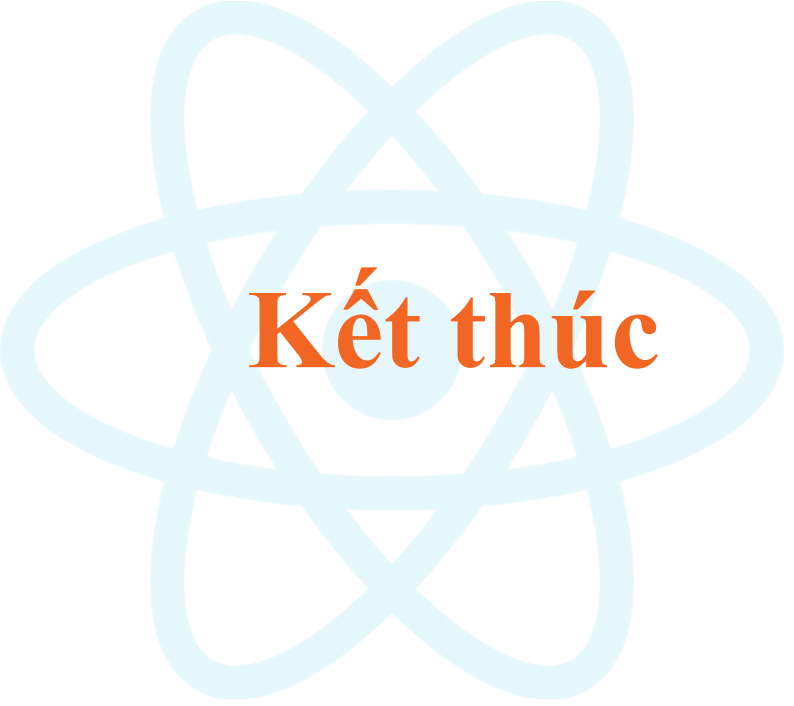
```
import {PersistGate} from 'redux-persist/integration/react';
import {persistor, store} from './screens/slide5_6/redux/store';
function App() {
  return (
    <Provider store={store}>
      <PersistGate loading={null} persistor={persistor}>
      ...
      </PersistGate>
    </Provider>
  )}
```

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