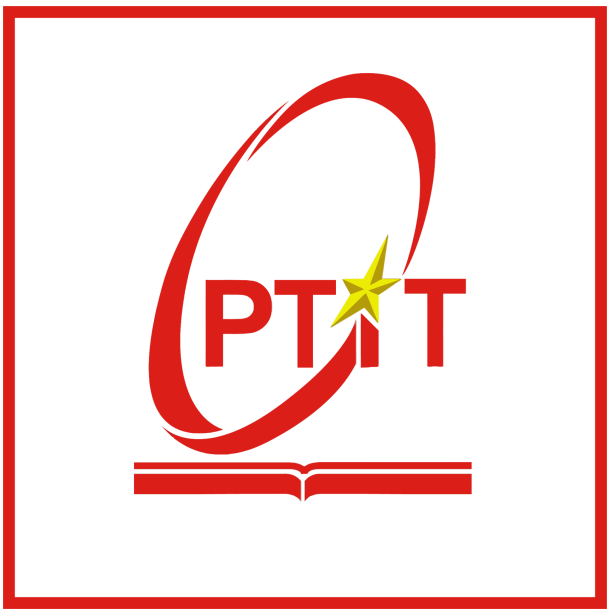
BỘ THÔNG TIN VÀ TRUYỀN THÔNG
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



BÁO CÁO THỰC TẬP CƠ SỞ TUẦN 6

TÌM HIỂU VỀ REDUX

Giảng viên hướng dẫn: TS. Kim Ngọc Bách

Sinh viên thực hiện:

- Nguyễn Quang Trung - B22DCDT321

MỤC LỤC

I. REDUX

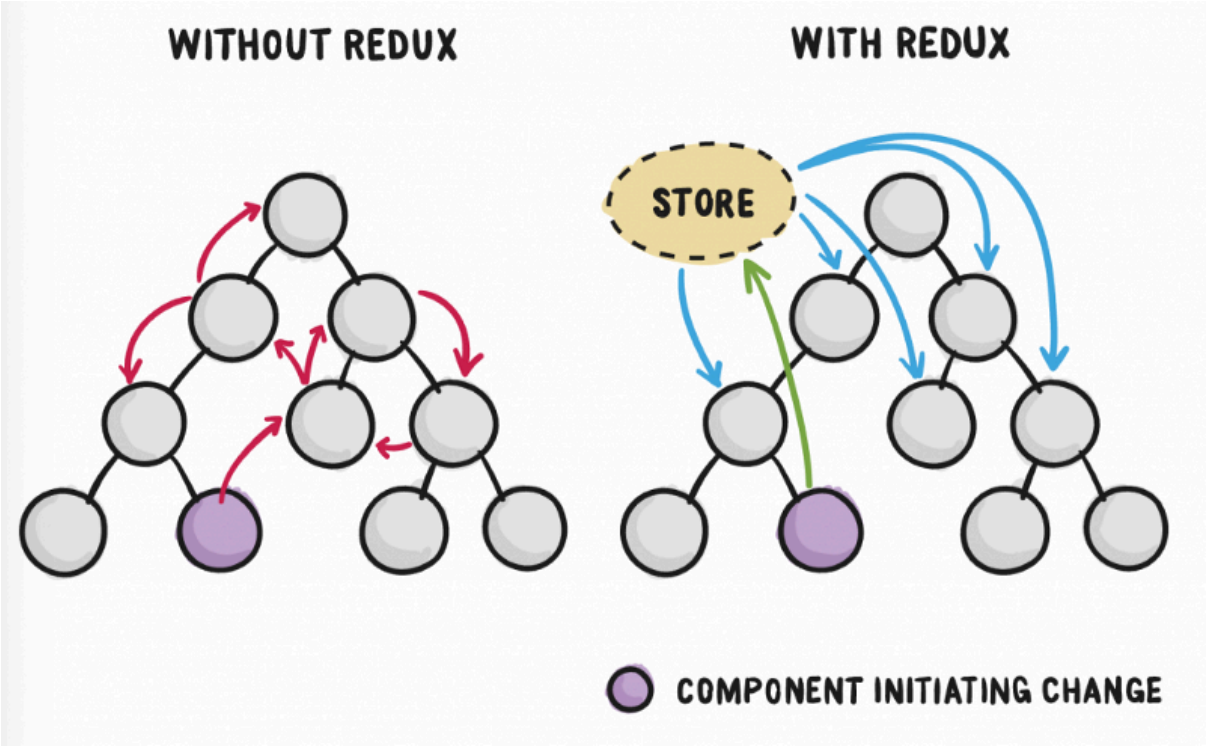
Redux là gì?

Redux là một predictable state management tool cho các ứng dụng Javascript. Nó giúp bạn viết các ứng dụng hoạt động một cách nhất quán, chạy trong các môi trường khác nhau (client, server, and native) và dễ dàng để test. Redux ra đời lấy cảm hứng từ tư tưởng của ngôn ngữ Elm và kiến trúc Flux của Facebook. Do vậy Redux thường dùng kết hợp với React.

Redux là một thư viện JavaScript để quản lý trạng thái ứng dụng. Nó được sử dụng phổ biến với các thư viện front-end như React để quản lý trạng thái của ứng dụng một cách nhất quán và có thể dự đoán được. Redux giúp quản lý toàn bộ trạng thái ứng dụng trong một cửa hàng (store) duy nhất, cho phép truy cập và thay đổi trạng thái một cách rõ ràng và có tổ chức.

Các khái niệm cốt lõi:

- State: Là đối tượng chứa toàn bộ dữ liệu của ứng dụng. State được quản lý tại một nơi duy nhất (store) trong ứng dụng Redux.
- Action: Là một đối tượng mô tả một sự kiện hoặc một ý định thay đổi state. Mỗi action phải có một thuộc tính type để định nghĩa loại hành động và có thể kèm theo dữ liệu cần thiết cho sự thay đổi.
- Reducer: Là một hàm thuần túy (pure function) nhận vào state hiện tại và action, sau đó trả về state mới. Reducer xác định cách state sẽ thay đổi để đáp ứng một action cụ thể.



Lý do ra đời Redux

Do yêu cầu cho các ứng dụng single-page sử dụng Javascript ngày càng trở lên phức tạp thì code của chúng ta phải quản lý nhiều state hơn.

Với Redux, state của ứng dụng được giữ trong một nơi gọi là store và mỗi component đều có thể access bất kỳ state nào mà chúng muốn từ chúng store này.

Tại sao ta lại cần state management tool

Hầu hết các lib như React, Angular, etc được built theo một cách sao cho các components đến việc quản lý nội bộ các state của chúng mà không cần bất kỳ một thư viện or tool nào từ bên ngoài.

Nó sẽ hoạt động tốt với các ứng dụng có ít components nhưng khi ứng dụng trở lên lớn hơn thì việc quản lý states được chia sẻ qua các components sẽ biến thành các công việc lặt nhặt.

Trong một app nơi data được chia sẻ thông qua các components, rất dễ nhầm lẫn để chúng ta có thể thực sự biết nơi mà một state đang live. Một sự lý tưởng là data trong một component nên live trong chỉ một component. Vì vậy việc share data thông qua các components anh em sẽ trở nên khó khăn hơn.

Ví dụ, trong react để share data thông qua các components anh em, một state phải live trong component cha. Một method để update chính state này sẽ được cung cấp bởi chính component cha này và pass như props đến các components con.

Tại sao nên sử dụng Redux React Native?

Khi phát triển ứng dụng di động với React Native, việc quản lý trạng thái trở thành một thách thức lớn khi ứng dụng ngày càng phức tạp. Redux giúp giải quyết vấn đề này bằng cách cung cấp một cơ chế quản lý trạng thái tập trung, giúp việc theo dõi và kiểm soát các thay đổi trạng thái trở nên dễ dàng hơn.

Việc sử dụng Redux React Native mang lại lợi ích:

- Dễ dàng theo dõi trạng thái: Tất cả trạng thái của ứng dụng được lưu trữ trong một cửa hàng duy nhất, giúp dễ dàng theo dõi và gỡ lỗi.
- Khả năng mở rộng: Với Redux, ứng dụng có thể dễ dàng mở rộng mà không lo lắng về việc quản lý trạng thái trở nên phức tạp.
- Tính dự đoán: Bằng cách sử dụng Redux, mọi thay đổi trong ứng dụng đều có thể dự đoán được dựa trên hành động và reducer tương ứng, giúp giảm thiểu lỗi trong quá trình phát triển.

Kết hợp React Native và Redux không chỉ giúp quản lý trạng thái hiệu quả mà còn mang lại cấu trúc rõ ràng và dễ bảo trì cho các ứng dụng di động, đặc biệt là khi ứng dụng trở nên lớn hơn và phức tạp hơn.

Các cấu trúc cơ bản của Redux

Store: Nơi lưu trữ toàn bộ trạng thái của ứng dụng

Khái niệm: Store là nơi lưu trữ toàn bộ trạng thái của ứng dụng trong một ứng dụng Redux. Mỗi ứng dụng chỉ có một store duy nhất, và store này chứa toàn bộ state của ứng dụng dưới dạng một cây đối tượng (object tree).

Vai trò:

- Quản lý state: Store lưu trữ trạng thái hiện tại của ứng dụng. Mọi thay đổi về state đều phải được thực hiện thông qua store.
- Cung cấp phương thức truy cập state: Store cho phép các phần khác của ứng dụng truy cập và đọc state hiện tại.
- Cho phép đăng ký lắng nghe state: Store cho phép các thành phần trong ứng dụng đăng ký lắng nghe các thay đổi của state để thực hiện cập nhật giao diện người dùng hoặc các hành động khác.
- Xử lý các action: Khi một action được dispatch, store sẽ chuyển action đó đến reducer để xử lý và cập nhật state.

Action: Các đối tượng mô tả những gì đã xảy ra

Khái niệm: Action là các đối tượng JavaScript mô tả một sự kiện hoặc một hành động đã xảy ra trong ứng dụng. Mỗi action bắt buộc phải có một thuộc tính type, thường là một chuỗi (string) xác định loại hành động, và có thể chứa các dữ liệu bổ sung khác cần thiết để thực hiện thay đổi state.

Vai trò:

- Mô tả sự thay đổi: Action không thực hiện thay đổi state trực tiếp mà chỉ mô tả các thay đổi cần thực hiện. Ví dụ, một action có thể mô tả việc thêm một sản phẩm vào giỏ hàng hoặc cập nhật thông tin người dùng.
- Độc lập và rõ ràng: Mỗi action là một đối tượng đơn giản và thuần túy, giúp việc theo dõi và gỡ lỗi các thay đổi trong ứng dụng trở nên dễ dàng hơn.

Ví dụ về Action:

```
{
  type: 'ADD_TODO',
  payload: {
    id: 1,
    text: 'Learn Redux'
  }
}
```

Trong ví dụ này, action mô tả việc thêm một nhiệm vụ mới là "Learn Redux" vào danh sách công việc cần làm.

Reducer: Các hàm thuần túy cập nhật state dựa trên action

Khái niệm: Reducer là các hàm thuần túy (pure function) nhận vào state hiện tại và action, sau đó trả về một state mới đã được cập nhật dựa trên action đó. Reducer không thay đổi trực tiếp state cũ mà tạo ra một bản sao mới của state với các thay đổi được áp dụng.

Vai trò:

- Cập nhật state: Reducer chịu trách nhiệm xác định cách state sẽ thay đổi để phản ứng với một action. Mỗi loại action sẽ được xử lý bởi một reducer tương ứng.
- Giữ nguyên tính chất của hàm thuần túy: Reducer luôn trả về một state mới mà không thay đổi trực tiếp state cũ và không gây ra tác dụng phụ nào (side effects).

Ví dụ về một reducer:

```
function todoReducer(state = [], action) {
  switch (action.type) {
    case 'ADD_TODO':
      return [...state, action.payload];
    case 'REMOVE_TODO':
      return state.filter(todo => todo.id !== action.payload.id);
    default:
      return state;
  }
}
```

Trong ví dụ trên, reducer todoReducer cập nhật danh sách công việc cần làm dựa trên các action ADD_TODO và REMOVE_TODO.

Dispatch: Hàm để gửi action đến store

Khái niệm: Dispatch là một hàm có sẵn trong store, cho phép bạn gửi (dispatch) một action đến store để yêu cầu thực hiện thay đổi state.

Vai trò:

- Khởi tạo quy trình cập nhật state: Khi một action được dispatch, store sẽ truyền action đó đến các reducer để xử lý. Reducer sẽ quyết định cách cập nhật state dựa trên action nhận được.
- Giao tiếp với store: Dispatch là cách duy nhất để gửi action đến store. Bằng cách này, bạn đảm bảo rằng mọi thay đổi trong state đều được quản lý và xử lý thông qua quy trình chính thức của Redux.

Ví dụ về dispatch một action:

```
store.dispatch({
  type: 'ADD_TODO',
  payload: {
    id: 1,
    text: 'Learn Redux'
  }
});
```

Trong ví dụ trên, action ADD_TODO được gửi đến store bằng cách sử dụng hàm dispatch. Store sau đó sẽ chuyển action này đến reducer để cập nhật state.

Kết hợp các thành phần trên, Redux cung cấp một cách tiếp cận rõ ràng và có tổ chức để quản lý trạng thái của ứng dụng, giúp duy trì tính nhất quán và dễ dự đoán trong các ứng dụng phức tạp.

Tích hợp Redux vào React Native

Redux là một thư viện quản lý trạng thái mạnh mẽ, phổ biến trong việc xây dựng các ứng dụng React Native. Việc tích hợp Redux vào React Native giúp quản lý trạng thái của ứng dụng một cách hiệu quả và dễ dàng hơn. Dưới đây là hướng dẫn chi tiết về cách tích hợp Redux vào một dự án React Native.

Cài đặt: Các thư viện cần thiết

Để sử dụng Redux trong React Native, bạn cần cài đặt các thư viện sau:

- redux: Thư viện chính của Redux, giúp quản lý trạng thái toàn cục.

- react-redux: Thư viện kết nối Redux với React, cung cấp các công cụ như Provider và connect.

Cài đặt các thư viện này thông qua npm hoặc yarn:

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

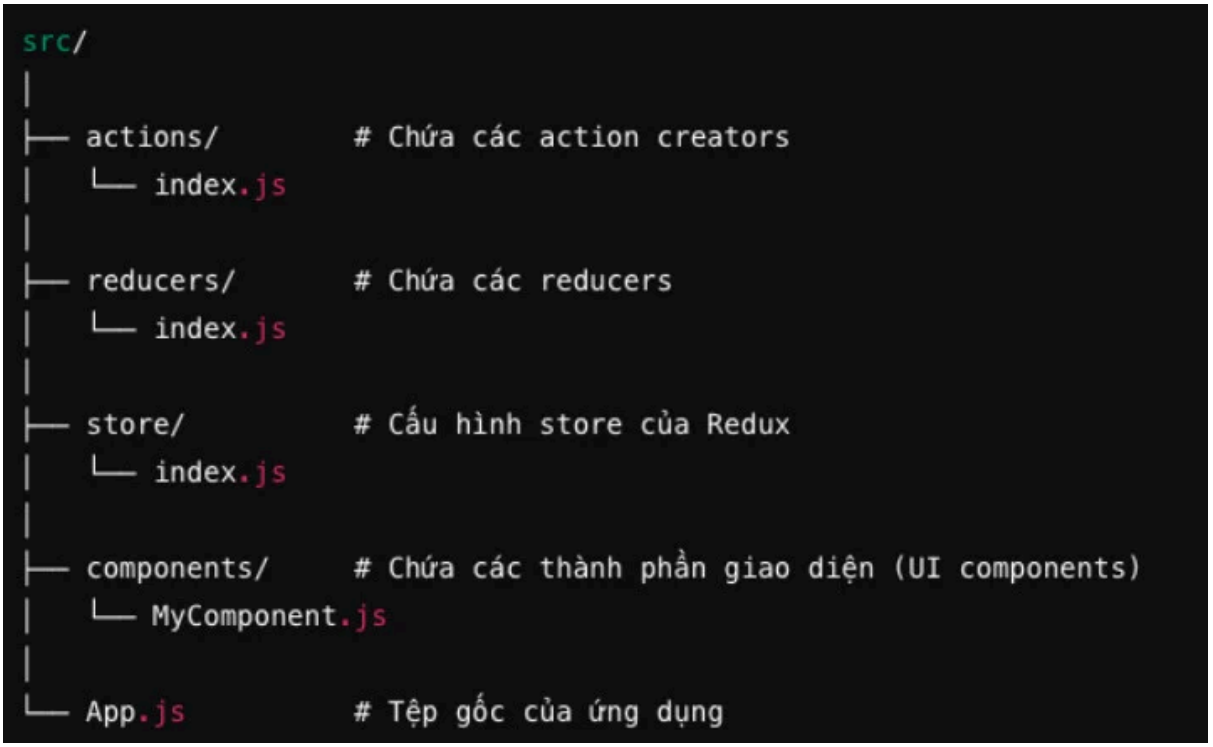
hoặc:

```
yarn add redux react-redux
```

Cấu trúc dự án

Khi sử dụng Redux trong React Native, tổ chức các file và thư mục một cách hợp lý là rất quan trọng để dễ bảo trì và mở rộng.

Dưới đây là một cấu trúc dự án phổ biến:



- actions/: Chứa các action creators, nơi định nghĩa các hành động (action) mà bạn muốn thực hiện.
- reducers/: Chứa các reducers, các hàm thuần túy nhận state hiện tại và action, sau đó trả về state mới.

- store/: Chứa cấu hình store của Redux, nơi tập hợp các reducers và middleware (nếu có).
- components/: Chứa các thành phần giao diện được kết nối với Redux store.

Provider: Bao bọc ứng dụng với Provider để kết nối với store

Provider là một thành phần từ thư viện react-redux, giúp kết nối Redux store với toàn bộ ứng dụng React Native.

Trước tiên, bạn cần tạo store và kết hợp các reducers trong file store/index.js:

```
import { createStore } from 'redux';
import rootReducer from '../reducers';

const store = createStore (rootReducer);

export default store;
```

Sau đó, trong App.js, bạn bao bọc ứng dụng với Provider:

```
import React from 'react';
import { Provider } from 'react-redux';
import store from './store';
import MyComponent from './components/MyComponent';

const App = () => {
  return (
    <Provider store={store}>
      <MyComponent />
    </Provider>
  );
};

export default App;
```

Connect: Kết nối component với store để lấy dữ liệu và dispatch action

Để một component có thể truy cập vào state hoặc dispatch action, bạn cần sử dụng hàm connect từ react-redux.

Ví dụ, trong một component như MyComponent.js:

```
import React from 'react';
import { connect } from 'react-redux';
import { someAction } from '../actions';

const MyComponent = ({ myState, dispatch }) => {
  const handlePress = () => {
    dispatch(someAction());
  };

  return (
    <View>
      <Text>{myState}</Text>
      <Button title="Do Something" onPress={handlePress} />
    </View>
  );
},

// Hàm này ánh xạ state từ Redux store tới props của component
const mapStateToProps = state => ({
myState: state.someReducer.myState,
});

// Kết nối component với store
export default connect (mapStateToProps) (MyComponent);
```

- mapStateToProps: Là hàm ánh xạ state từ Redux store thành props của component, giúp component truy cập vào state.
- dispatch: Làm cho component có thể gửi các hành động (action) tới store thông qua dispatch.

Việc tích hợp Redux vào React Native không chỉ giúp quản lý trạng thái một cách hiệu quả mà còn giúp cấu trúc ứng dụng rõ ràng, dễ bảo trì hơn. Bằng cách tổ chức dự án hợp lý, sử dụng Provider và connect đúng cách, bạn có thể xây dựng các ứng dụng React Native mạnh mẽ và dễ dàng mở rộng.

II. REDUX TOOLKIT

Mục đích

Gói Redux Toolkit được tạo ra nhằm trở thành cách tiêu chuẩn để viết logic Redux. Nó ra đời để giải quyết 3 vấn đề phổ biến khi dùng Redux:

1. "Cấu hình Redux store quá phức tạp"
2. "Phải cài quá nhiều gói mới dùng được Redux một cách hiệu quả"
3. "Redux yêu cầu viết quá nhiều mã boilerplate (mã mẫu lặp lại)"

Redux Toolkit không thể giải quyết mọi trường hợp, nhưng giống như triết lý của create-react-app, nó cung cấp một số công cụ để đơn giản hóa quy trình thiết lập, xử lý các tình huống phổ biến, và cung cấp một số tiện ích giúp bạn viết mã dễ hơn.

Ngoài ra, Redux Toolkit còn bao gồm một tính năng mạnh mẽ để fetch và cache dữ liệu, gọi là "RTK Query". Tính năng này nằm trong cùng gói nhưng là phần tùy chọn, có thể giúp bạn không cần phải tự viết logic fetch dữ liệu nữa.

Các công cụ này rất có ích cho mọi lập trình viên dùng Redux, từ người mới bắt đầu cho đến người nhiều kinh nghiệm đang muốn đơn giản hóa ứng dụng.

Cài đặt: Tạo một ứng dụng React Redux mới

Cách khuyến nghị để bắt đầu một ứng dụng mới với React và Redux Toolkit là sử dụng các mẫu template chính thức như:

- Redux Toolkit + TypeScript + Vite
- Next.js với Redux

Các template này đã được cấu hình sẵn Redux Toolkit và React-Redux, kèm theo một ví dụ nhỏ minh họa cách sử dụng nhiều tính năng của Redux Toolkit.

Dùng với ứng dụng hiện có

Redux Toolkit có sẵn trên NPM và có thể dùng với các trình bundler như Webpack hoặc trong ứng dụng Node:

```bash
npm install @reduxjs/toolkit
```

Nếu bạn cần kết nối với React:

```bash
npm install react-redux
```

Gói này cũng bao gồm một bản build ESM sẵn sàng sử dụng với thẻ <script type="module"> trong trình duyệt.

Những gì Redux Toolkit bao gồm

- **configureStore()**: Cấu hình Redux store một cách đơn giản, hỗ trợ tự động kết hợp các slice reducer, thêm middleware, hỗ trợ Redux DevTools.
- **createReducer()**: Giúp viết reducer dễ hơn, không cần switch-case, sử dụng thư viện immer để cập nhật state theo cách "mutate" mà vẫn bất biến.
- **createAction()**: Tạo nhanh action creator từ chuỗi tên action.
- **createSlice()**: Tạo slice bao gồm reducer, action, action type chỉ trong một bước.
- **combineSlices()**: Kết hợp nhiều slice lại và hỗ trợ load động.
- **createAsyncThunk**: Dùng cho async logic như gọi API. Tự động tạo action dạng pending/fulfilled/rejected.
- **createEntityAdapter**: Quản lý dữ liệu chuẩn hóa dễ dàng hơn.
- **createSelector**: Từ thư viện Reselect, giúp tạo selector hiệu quả hơn.

RTK Query

RTK Query là một **tiện ích mở rộng đi kèm Redux Toolkit** để **giải quyết việc lấy và cache dữ liệu**. Nó giúp:

- Tự động xử lý gọi API.
- Tự cache và update lại khi cần.
- Không cần tự viết useEffect, fetch, axios, v.v.

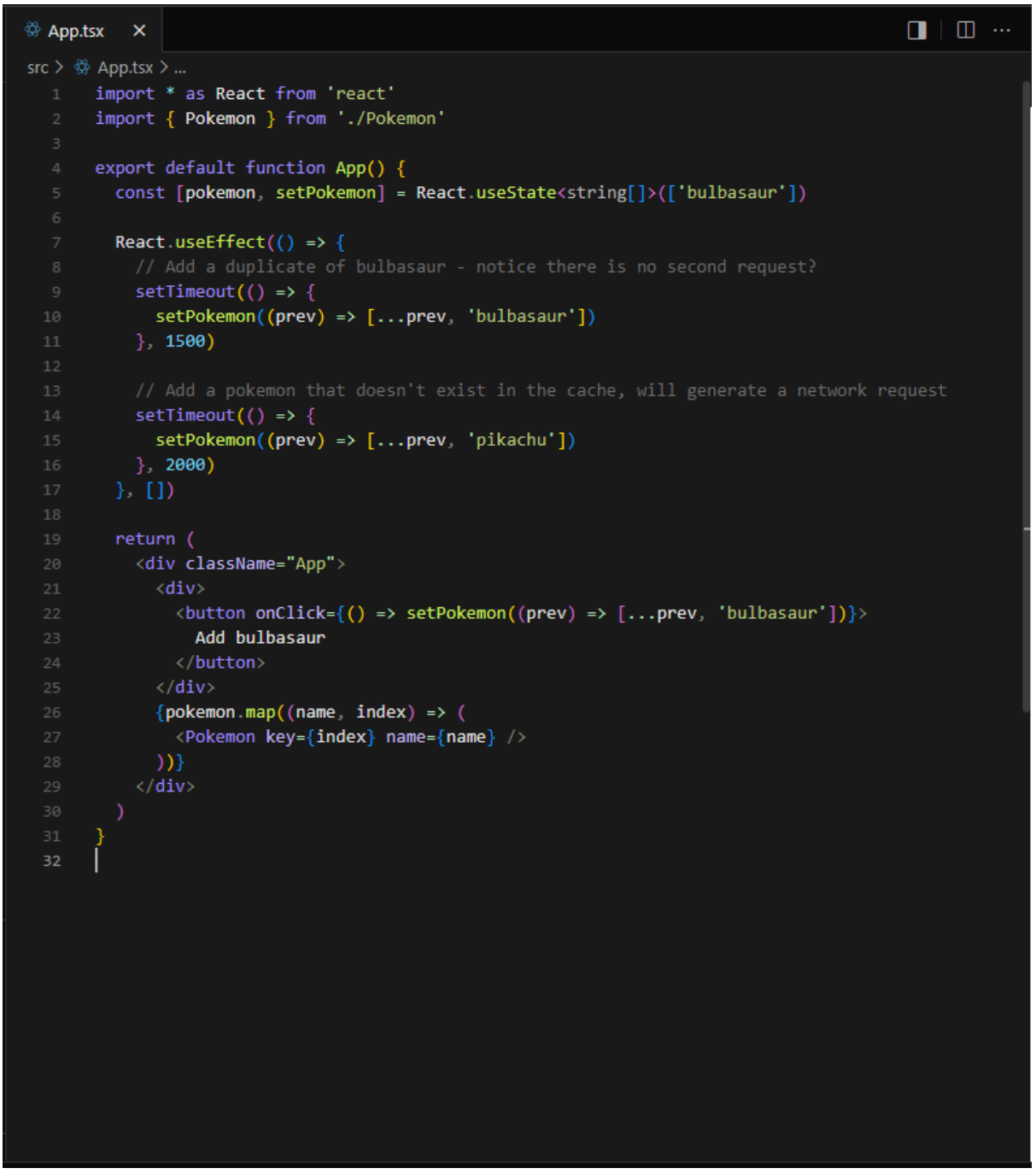
Nó được xây dựng dựa trên Redux Toolkit core. Dù không cần biết nhiều về Redux để dùng RTK Query, bạn nên tìm hiểu để tận dụng tối đa khả năng của nó. Ngoài ra, Redux DevTools hoạt động cực tốt với RTK Query, giúp debug hành vi gọi API dễ dàng.

RTK Query đã được tích hợp trong Redux Toolkit. Bạn có thể sử dụng qua hai điểm nhập chính:

- @reduxjs/toolkit/query – core API
- @reduxjs/toolkit/query/react – dùng cho React, hỗ trợ tạo hook tự động

III. THỰC HÀNH

Xây dựng ứng dụng React sử dụng Redux Toolkit và RTK Query để hiển thị danh sách các Pokémon từ PokeAPI



App.tsx | store.ts

src > store.ts > ...

```ts
import { configureStore } from '@reduxjs/toolkit'
import { pokemonApi } from './services/pokemon'

export const store = configureStore({
  reducer: {
    [pokemonApi.reducerPath]: pokemonApi.reducer,
  },
  // adding the api middleware enables caching, invalidation, polling and other features of `
  middleware: (getDefaultMiddleware) =>
    getDefaultMiddleware().concat(pokemonApi.middleware),
})

```

App.tsx | Pokemon.tsx

src > Pokemon.tsx > ...

```tsx
import { useGetPokemonByNameQuery } from './services/pokemon'

export const Pokemon = ({ name }: { name: string }) => {
  const {
    data,
    error,
    isLoading,
    isFetching,
    refetch,
  } = useGetPokemonByNameQuery(name)

  return (
    <div style={{ float: 'left', textAlign: 'center' }}>
      {error ? (
        <>Oh no, there was an error</>
      ) : isLoading ? (
        <>Loading...</>
      ) : data ? (
        <>
          <h3>{data.species.name}</h3>
          <div>
            <img src={data.sprites.front_shiny} alt={data.species.name} />
          </div>
          <div>
            <button onClick={refetch} disabled={isFetching}>
              {isFetching ? 'Fetching...' : 'Refetch'}
            </button>
          </div>
        </>
      ) : (
        'No Data'
      )}
    </div>
  )
}

```

App.tsx | Pokemon.tsx | index.tsx

src > index.tsx > ...

```tsx
import { render } from 'react-dom'
import { Provider } from 'react-redux'

import App from './App'
import { store } from './store'

const rootElement = document.getElementById('root')
render(
  <Provider store={store}>
    <App />
  </Provider>,
  rootElement
)
```

App.tsx | Pokemon.tsx | pokemon.ts

src > services > pokemon.ts > ...

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

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

// Export hooks for usage in functional components
export const { useGetPokemonByNameQuery } = pokemonApi
```