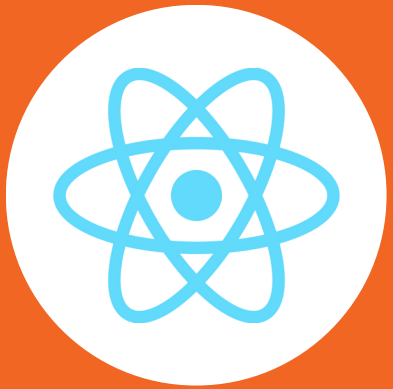
FPT Education

FPT POLYTECHNIC



LẬP TRÌNH ĐA NỀN TẢNG VỚI REACT NATIVE

BÀI 8: TƯƠNG TÁC API TRONG ỨNG DỤNG REACT NATIVE

PHẦN 1: GIỚI THIỆU, THIẾT LẬP AXIOS VÀ TẠO FAKE REST API

Mục tiêu

- [ ] Biết cách xử lý với api
- [ ] Tìm hiểu về các phương thức xử lý api
- [ ] Hiểu cách hoạt động của axios
- [ ] Thiết lập sử dụng axios
- [ ] Cài đặt thư viện tạo fake rest api

Giới thiệu networking

- Trong React Native, **networking** là quá trình giao tiếp và truyền thông dữ liệu giữa ứng dụng của bạn và các máy chủ hoặc dịch vụ bên ngoài. Quá trình này cho phép ứng dụng của bạn truy xuất và gửi yêu cầu đến các API, tải xuống và tải lên dữ liệu, và thực hiện các hoạt động liên quan đến mạng khác.

- Có một số thư viện và công cụ hữu ích trong React Native để thực hiện các tác vụ liên quan đến networking. Dưới đây là một số cách bạn có thể thực hiện networking trong React Native:

  - **Fetch API**: Fetch API là một phương thức xây dựng trong React Native cho phép bạn gửi yêu cầu HTTP và nhận phản hồi từ máy chủ. Bạn có thể sử dụng Fetch API để gửi các yêu cầu GET, POST, PUT, DELETE và các phương thức HTTP khác.

Giới thiệu networking

- **Axios**: Axios là một thư viện HTTP client phổ biến trong React Native. Nó cung cấp các phương thức dễ sử dụng để thực hiện các yêu cầu HTTP và xử lý phản hồi từ máy chủ. Axios hỗ trợ xử lý lỗi, gửi dữ liệu dạng JSON và các tính năng khác liên quan đến mạng.
- **createApi**: là core chức năng trong **RTK** Query. Nó cho phép bạn xác định một tập hợp các 'endpoints' mô tả cách truy xuất dữ liệu từ các API phụ trợ và các nguồn không đồng bộ khác, bao gồm cấu hình về cách tìm nạp và chuyển đổi dữ liệu đó. Nó tạo ra một cấu trúc 'API slice' có chứa logic Redux (và các hook React tùy chọn) đóng gói quá trình tìm nạp dữ liệu và cache dữ liệu cho bạn..

Tìm hiểu về axios

- Trong React Native, Axios là một thư viện HTTP client phổ biến được sử dụng để thực hiện các yêu cầu mạng. Nó cung cấp các phương thức dễ sử dụng để gửi yêu cầu HTTP và xử lý phản hồi từ máy chủ.
- Axios được xây dựng trên cơ sở thư viện Axios gốc dành cho JavaScript, nhưng được tối ưu hóa để hoạt động tốt trong môi trường React Native. Thư viện này hỗ trợ các tính năng như:
  - Hỗ trợ Promise API
  - Chặn yêu cầu và phản hồi
  - Chuyển đổi dữ liệu yêu cầu và phản hồi
  - Huỷ request

Tìm hiểu về axios

- Huỷ request
- Timeouts
- Tuần tự hóa tham số truy vấn với sự hỗ trợ cho các mục nhập lồng nhau
- Tự động yêu cầu tuần tự hóa nội dung tới:
  - JSON **(application/json)**
  - Multipart / FormData **(multipart/form-data)**
  - URL encoded form **(application/x-www-form-urlencoded)**
- Post HTML form dưới dạng JSON

Tìm hiểu về axios

- Để sử dụng **Axios** trong React Native, bạn cần cài đặt thư viện **axios** bằng npm hoặc yarn và sau đó import nó vào mã nguồn của ứng dụng.
- Sau khi cài đặt và import **Axios**, bạn có thể sử dụng các phương thức như **axios.get()**, **axios.post()**, **axios.put()**, **axios.delete()** để gửi yêu cầu mạng và xử lý phản hồi tương ứng.

Cài đặt axios

- **Axios** là một thư viện, thế nên bạn phải cài đặt nó vào project của bạn bằng câu lệnh sau:

```
npm install axios
```

- Tiếp theo chúng ta sẽ bước tới phần thiết lập **axios**, để dễ dàng thao tác dữ liệu và call api với **axios** thì chúng ta nên có một file setting default global **configuaration** cho **axios**. Nó sẽ là file **config header** cho request và chứa tất cả lệnh **request** api sẽ được gửi

Thiết lập axios

- Axios là một thư viện, thế nên bạn phải cài đặt nó vào project của bạn bằng câu lệnh sau:

```
const commonConfigs = {
  baseURL: 'http://192.168.1.100:3000',
  timeout: 10000,
  headers: {
    buildversion: '1.0.0',
    buildnumber: '1',
    platform: Platform.OS,
  },
};

const instance = axios.create(commonConfigs);
```

Thiết lập axios

- **baseURL**: là địa chỉ api chính cho các request của chúng ta
- **timeout**: thời gian tối đa để chờ một request phản hồi
- **headers**: các thông tin của người dùng mà bạn muốn gửi lên, điều này giúp bạn dễ xác định lỗi, do thiết bị hay người dùng nào tạo ra. Để lấy được các thông tin **buildversion** và **buildnumber** bạn có thể cài package **react-native-device-info**

Thiết lập axios

- Nếu như api của bạn yêu cầu thêm các header khác, ví dụ như **Authorization** bằng token thì bạn có thể sử dụng **instance.defaults.headers.common** để truyền token vào **header** khi request được gửi đi. Bạn nên viết riêng ra hàm **setDefaultHeaders** như bên dưới, vì bạn sẽ cần sử dụng chúng nhiều lần.

```
export function setDefaultHeaders(headers: RawAxiosRequestHeaders) {
  Object.keys(headers).forEach(key => {
    instance.defaults.headers.common[key] = headers[key];
  });
}
```

Thiết lập axios

- Khi bạn giao tiếp với API thì không thể nào tránh được các lỗi xảy ra. Thế nên khi lỗi xảy ra khi bạn gọi api thì bạn phải bắt được nó, và hiển thị lỗi cho người dùng biết.

- Sử dụng **axios.interceptors.response.use** để nhận phản hồi khi gọi api

Thiết lập axios

- Trong lập trình tương tác với api, sẽ có nhiều kiểu lỗi trả về, tuỳ vào yêu cầu thiết kế ứng dụng của bạn, mà bạn sẽ xử lý lỗi chúng.

  Ví dụ như, nếu ứng dụng gặp phải lỗi code **401** (token hết hạn) thì phải logout tài khoản ra, bắt người dùng đăng nhập lại. Hoặc hiện thông báo lỗi khi gặp lỗi khác.

```
instance.interceptors.response.use(
  res => res,
  (error: AxiosError) => {
    const {data, status} = error.response || {};
    switch (status) {
      case 400:
        console.error(data);
        break;
      case 401:
        // Request chưa có Authorization
        // Thường khi gặp lỗi này sẽ logout tài khoản
        console.error('unauthorised');
        break;
      case 404:
        // API không tồn tại
        console.error('/not-found');
        break;
      case 500:
        // API server đang xảy ra lỗi
        console.error('/server-error');
        break;
    }
    return Promise.reject(error);
  },
);
```

Thiết lập axios

- Để dễ cho việc gọi và xử lý các phương thức giao tiếp với api chúng ta nên thiết lập chung vào một hàm.

```
const responseBody = <R>(response: AxiosResponse<R>) => response.data;
const responseError = (response: AxiosError) => ({
  isError: true,
  message: response,
});
```

- **responseBody** nhận giá trị api trả về, sau đó return ra giá trị cho chúng ta.
- **responseError** được kích hoạt, khi phương thức giao tiếp api gặp lỗi.

Thiết lập axios

- Dưới đây là 4 phương thức giao tiếp api chính mà chúng ta sẽ được thực hành với axios.

```
export const api = {
  get: <R>(url: string, config?: AxiosRequestConfig) =>
    instance.get<R>(url, config).then(responseBody).catch(responseError),

  post: <R>(url: string, body: any, config?: AxiosRequestConfig) =>
    instance.post<R>(url, body, config).then(responseBody).catch(responseError),

  put: <R>(url: string, body: any, config?: AxiosRequestConfig) =>
    instance.put<R>(url, body, config).then(responseBody).catch(responseError),

  delete: <R>(url: string, config?: AxiosRequestConfig) =>
    instance.delete<R>(url, config).then(responseBody).catch(responseError),
};
```

- Ở bài sau, chúng ta sẽ được học cách gọi api từ các hàm này.

Thiết lập fake api

- Thông thường, các api sẽ được viết từ các ngôn ngữ hay framework backend như **Java**, **Php**, **NodeJs**. Nhưng vì phạm vi mộ học của chúng ta chỉ là React Native nên các bạn sẽ được hướng dẫn cách tạo fake api để gọi api cho ứng dụng của mình
- Ở phần này các bạn sẽ được hướng dẫn tạo fake api cho ứng dụng của bạn bằng **json-server** package
  - **json-server** cung cấp nhận fake API REST đầy đủ mà không cần mã hóa trong vòng chưa đầy 30 giây.

Thiết lập fake api

- Cài đặt **json-server** package ngay trong project react native của bạn :

```
npm install  json-server
```

Thiết lập fake api

- Tạo file **db.json** với một số dữ liệu:

```
{
 "users": [
   {
      "id": 1,
      "name": "Nguyen Van A",
      "birthday": "1990-02-21"
   },
   {
       "id": 2,
       "name": "Nguyen Van B",
       "birthday": "1993-02-22"
    },
 ]
}
```

Thiết lập fake api

- Khởi động máy chủ JSON:

```
npm start json-server --host 192.168.1.100 --port 3000 --watch db.json
```

- **192.168.1.100**: là địa chỉ **IP address** của wifi mà máy tính của bạn kết nối

Thiết lập fake api

- Khi bạn kết nối thành công, terminal sẽ hiện giống như sau:

```
\{^_^}/ hi!

Loading db.json
Done

Resources
http://192.168.1.100:3000/users

Home
http://192.168.1.100:3000

Type s + enter at any time to create a snapshot of the database
Watching...
```

Route

- Với dữ liệu từ file **db.json** bạn đã nạp vào **json-server** sẽ tự động cung cấp cho các bạn api sau

  - GET /users (Lấy tất cả dữ liệu trong users)
  - GET /users/:id (Lấy thông tin chi tiết của một user từ id)
  - POST /users (Tạo mới một user)
  - PUT /users/:id (Cập nhật thông tin user từ id)
  - DELETE /users/:id (Xoá một user từ id)

- **Filter:** nếu bạn muốn lọc lấy những dữ liệu bạn muốn, **json-server** cũng cung cấp cho bạn các params để làm điều đó.
  - GET /users?name=Nguyen Van A&birthday=1990-02-21
  - GET /users?id=1&id=2
- **Paginate:** là một phương thức nữa mà **json-server** cung cấp, nó giúp lấy danh sách dữ liệu theo trang được sử dụng rất phổ biến. Ví dụ danh sách users của bạn có khoảng 100.000.000 người, bạn sẽ chẳng muốn lấy hết danh sách này xuống một lần, điều này sẽ khiến việc lấy dữ liệu rất lâu, giảm hiệu suất ứng dụng và có thể gây hại đến server của bạn.
  - GET /users?_page=1
  - GET /users?_page=1&_limit=20

Sử dụng **_page** và tùy chọn **_limit** để phân trang dữ liệu trả về. Trong header **Link** , bạn sẽ nhận được các liên kết **first**, **prev**, **next** và **last**.

- **Sort:** Nếu bạn muốn sắp xếp dữ liệu của mình trước khi dữ liệu được trả về xài thêm **_sort** và **_order** trong params(thứ tự tăng dần theo mặc định)

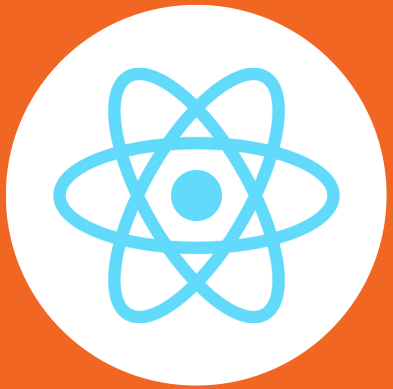
  - GET /users?_sort=id&_order=asc
  - GET /posts/1/comments?_sort=votes&_order=asc

  Đối với nhiều field, hãy sử dụng định dạng sau:

  - GET /posts?_sort=user,views&_order=desc,asc

FPT Education

FPT POLYTECHNIC



LẬP TRÌNH ĐA NỀN TẢNG VỚI REACT NATIVE

BÀI 8: TƯƠNG TÁC API TRONG ỨNG DỤNG REACT NATIVE

PHẦN 2: GET, PUT, POST, DELETE TRONG AXIOS VÀ XỬ LÝ LOADING, ERROR

Mục tiêu

- Chúng ta sẽ thao tác api bằng axios với các phương thức get, put, post và delete
- Hiểu rõ cách gọi api và xử lý lỗi khi gọi api
- Xử lý loading và thông báo lỗi

RESTful API

- **RESTful APIs** (Representational State Transfer APIs) là một kiểu thiết kế và kiến trúc cho các dịch vụ web (web services) dựa trên giao thức HTTP. Nó xác định một tập các quy tắc và hướng dẫn để xây dựng các dịch vụ web có khả năng mở rộng, linh hoạt và dễ dàng tích hợp.
- **RESTful APIs** sử dụng các phương thức HTTP như **GET**, **POST**, **PUT** và **DELETE** để thực hiện các hoạt động CRUD (Create, Read, Update, Delete) trên các tài nguyên (resources). Mỗi tài nguyên được đại diện bằng một URL duy nhất (endpoint), và các phương thức HTTP được sử dụng để thực hiện các hành động tương ứng trên tài nguyên đó.
- Các đặc điểm chính của **RESTful APIs** bao gồm:

- **Kiểu truyền thông dựa trên các tài nguyên (Resource-based)**: Các tài nguyên (resources) được đại diện bằng các URL (endpoint), và các phương thức HTTP được sử dụng để thực hiện các hoạt động trên các tài nguyên đó.
- **Giao tiếp không trạng thái (Stateless)**: Mỗi yêu cầu từ client đến server chứa đủ thông tin cần thiết để server hiểu và xử lý yêu cầu, không cần lưu trạng thái trước đó.
- **Các hoạt động dựa trên các phương thức HTTP**: Các phương thức HTTP như GET, POST, PUT và DELETE được sử dụng để thực hiện các hoạt động tương ứng với Create (tạo mới), Read (đọc), Update (cập nhật) và Delete (xóa) trên các tài nguyên.
- **Sử dụng các định dạng chuẩn**: RESTful APIs sử dụng các định dạng chuẩn như JSON hoặc XML để truyền và nhận dữ liệu.

Các phương thức RESTful API

- Trong RESTful API, các phương thức chính được sử dụng để thực hiện các thao tác trên các tài nguyên (resources) là:
  - **GET**: Phương thức GET được sử dụng để truy vấn và lấy thông tin từ một tài nguyên. Khi gửi một yêu cầu GET, người dùng nhận về dữ liệu từ tài nguyên đã chỉ định trong URL.
  - **POST**: Phương thức POST được sử dụng để tạo mới một tài nguyên. Khi gửi một yêu cầu POST, người dùng gửi dữ liệu lên máy chủ để tạo ra một tài nguyên mới.
  - **PUT**: Phương thức PUT được sử dụng để cập nhật hoặc thay đổi thông tin của một tài nguyên đã tồn tại. Khi gửi một yêu cầu PUT, người dùng cung cấp dữ liệu mới để thay thế toàn bộ tài nguyên hiện có.

Các phương thức RESTful API

- **DELETE**: Phương thức DELETE được sử dụng để xóa một tài nguyên cụ thể từ máy chủ. Phương thức này yêu cầu máy chủ xóa tài nguyên được xác định bởi đường dẫn trong yêu cầu.

- **PATCH**: Phương thức PATCH được sử dụng để cập nhật một phần của một tài nguyên đã tồn tại. Khi gửi một yêu cầu PATCH, người dùng chỉ cung cấp các thông tin cần thiết để cập nhật phần chỉ định của tài nguyên.

- Ngoài ra, có thể sử dụng các phương thức khác như OPTIONS, HEAD, và TRACE trong **RESTful API**, nhưng chúng không được sử dụng phổ biến như các phương thức chính đã nêu trên.

Phương thức GET

- Ở bài trước chúng ta đã viết config cho axios, ở bài này chúng ta sẽ bắt đầu gọi các hàm đã config sẵn ra để sử dụng nhé
- Các hàm gọi api của một đối tượng nào đó, bạn nên để chúng chung tại một file, để ứng dụng của bạn dễ quản lý và bảo trì. Ở đây chúng ta tạo một hàm **getUserList**

```
export const usersAPI = {
  getUserList: (config?: AxiosRequestConfig) =>
    api.get<User[]>('/users', config),
};
```

  - **<User[]>** dùng để khai báo kiểu dữ liệu trả về của hàm **getUserList**

Phương thức GET

- Để lấy danh sách dữ liệu bằng axios, chúng ta chỉ đơn giản gọi hàm **userApi.getUserList()** đã được chúng ta thiết lập sẵn

```
const onGetUserList = async () => {
  const response = await usersAPI.getUserList();
  return response;
};
```

Phương thức POST

- Chúng ta sẽ bổ sung thêm hàm **createUser** trong **userAPI** để tiến hành đẩy dữ liệu lên **json-server** bằng phương thức **POST**

```
export const usersAPI = {
  createUser: (data: any, config?: AxiosRequestConfig) =>
    api.post<User>('/users', data, config),
};
```

Phương thức POST

- Phương thức **POST** là phương thức gửi một dữ liệu vào body lên server, ví dụ ở bên dưới
  - Thêm dữ liệu bằng **axios**

```
const onAddUser = async () => {
  const response = await usersAPI.createUser({
    age: 21,
    name: 'Tran Duy Hai',
  });
  return response;
};
```

Phương thức PUT

- Chúng ta sẽ bổ sung thêm hàm **updateUser** trong **userAPI** để tiến hành chỉnh sửa thông tin dữ liệu lên **json-server** bằng phương thức **PUT**

```
export const usersAPI = {
  updateUser: (id: number, data: any, config?: AxiosRequestConfig) =>
    api.put<User>(`/users/${id}`, data, config),
};
```

Phương thức PUT

- Phương thức **PUT** là phương thức gửi một dữ liệu vào body lên server để server cập nhật dữ liệu bạn gửi lên, ví dụ ở bên dưới
  - Cập nhật dữ liệu bằng **axios**

```
const onUpdateUser = async () => {
  const response = await usersAPI.updateUser(2, {
    age: 17,
    name: 'Tran Duy Ho',
  });
  return response;
};
```

Phương thức DELETE

- Chúng ta sẽ bổ sung thêm hàm **deleteUser** trong **userAPI** để tiến hành xoá thông tin dữ liệu lên **json-server** bằng phương thức **DELETE**

```
export const usersAPI = {
  deleteUser: (id: number, config?: AxiosRequestConfig) =>
    api.delete<User>(`/users/${id}`, config),
};
```

Phương thức DELETE

- Chúng ta sẽ bổ sung thêm hàm **deleteUser** trong **userAPI** để tiến hành xoá thông tin dữ liệu lên **json-server** bằng phương thức **DELETE**
  - Xoá dữ liệu bằng **axios**

```
const onDeleteUser = async () => {
  const response = await usersAPI.deleteUser(2);
  return response;
};
```

Xử lý trạng thái gọi dữ liệu

- Khi bạn giao tiếp với API đôi khi bạn sẽ không muốn người dùng làm gì khác trong khi API đang được xử lý từ server, và chỉ khi API thực hiện xong thì bạn mới cho người dùng sử dụng các tính năng khác của ứng dụng
- Vì vậy sau khi gọi API chúng ta có thể xử lý hiển thị màn hình loading, khi API xử lý xong ta tắt màn hình loading, nếu có lỗi chúng ta sẽ hiện thông báo lỗi

Xử lý trạng thái gọi dữ liệu

- Chúng ta sẽ trở lại hàm **onGeUserList** để xử lý loading, lỗi và thành công khi lấy dữ liệu.

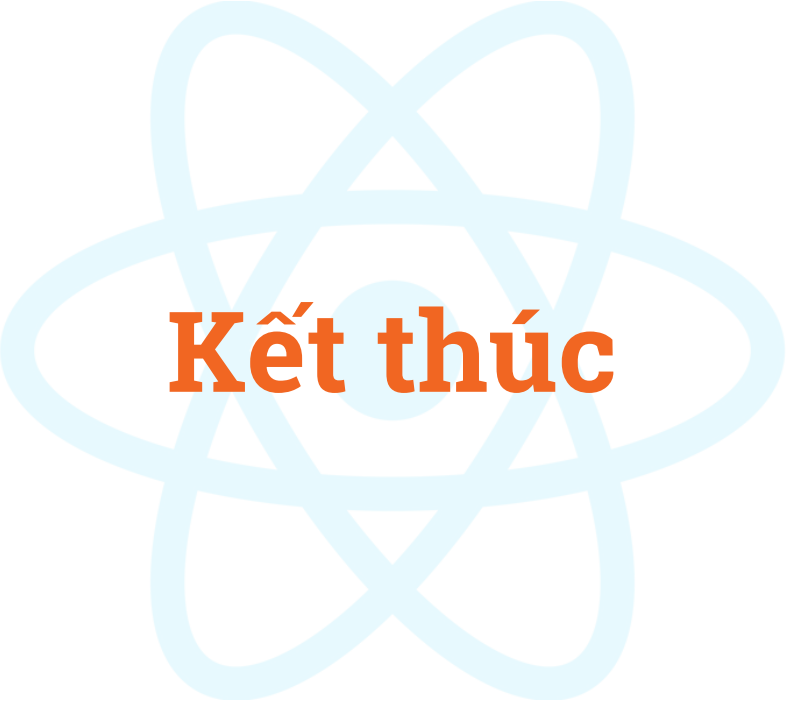
```
const onGetUserList = async () => {
  showLoading();
  const response = await usersAPI.getUserList().finally(() => hideLoading());

  if (response.isError) {
    onError(response);
  } else {
    onSuccess(response);
  }

  return response;
};
```

Xử lý trạng thái gọi dữ liệu

- **showLoading** hàm bật loading ứng dụng lên, khi gọi api hoàn tất **hideLoading** được gọi để tắt loading
- Chúng ta kiểm tra dữ liệu trả về nếu **isError** bằng true thì gọi hàm **onError** để xử lý lỗi theo cách chúng ta muốn, ví dụ như hiện toast thông báo lỗi cho người dùng.

Tổng kết

- [ ] Biết cách xử lý với api
- [ ] Tìm hiểu về các phương thức xử lý api
- [ ] Hiểu cách hoạt động của axios
- [ ] Thiết lập sử dụng axios
- [ ] Cài đặt thư viện tạo fake rest api
- [ ] Chúng ta sẽ thao tác api bằng axios với các phương thức get, put, post và delete
- [ ] Hiểu rõ cách gọi api và xử lý lỗi khi gọi api
- [ ] Xử lý loading và thông báo lỗi



Kết thúc