Bài 9
API Clients

Mục tiêu

- Trình bày được ý nghĩa giao thức HTTP trong ứng dụng web
- Trình bày được mô hình ứng dụng web cơ bản
- Hiểu được khái niệm API và Web Service (Web API) trong phát triển phần mềm
- Thiết kế được Web API với kiểu kiến trúc RESTful
- Đọc được tài liệu mô tả Web API cơ bản
- Mô phỏng được backend với công cụ xây dựng Mock API
- Sử dụng được thư viện hỗ trợ trong ReactJS
- Sử dụng được Promise để xử lý bất đồng bộ
- Sử dụng async/await để xử lý bất đồng bộ
- Giao tiếp được với back-end thông qua RESTful API

Thảo luận

- Giới thiệu Web API với kiến trúc RESTfull

API là gì ?

- API là viết tắt của Application Programming Interface (giao diện lập trình ứng dụng)
- Đây là phương tiện cho hai hoặc nhiều ứng dụng trao đổi, tương tác với nhau, tạo ra tương tác giữa người dùng với ứng dụng hiệu quả và tiện lợi hơn.
- Với API, các lập trình viên có thể tiếp cận, truy xuất dữ liệu từ máy chủ thể hiện chúng trên ứng dụng phần mềm hoặc website của mình một cách dễ dàng hơn.

API bao gồm

- Web API
- API trên hệ điều hành
- API của thư viện phần mềm hay framework

Những tính năng của Web API

- **Tự động hóa sản phẩm** : Đối với Web API, sẽ giúp người dùng có thể dễ dàng tự động quản lý được công việc.
- **Tích hợp linh động :** API cho phép lấy nội dung ở bất kỳ Website hay ứng dụng nào đó một cách dễ dàng, khiến trải nghiệm người dùng được tăng lên.
- **Cập nhật thông tin theo thời gian thực** :API giúp thay đổi và cập nhật những thông tin mới theo thời gian thực.

  Công nghệ này sẽ giúp những thông tin truyền đi tốt hơn, chính xác hơn và dịch vụ cung cấp cũng được linh hoạt hơn.

Ưu điểm của Web API

- Web API được sử dụng khá rộng rãi ở trên các ứng dụng như: Desktop, mobile và cả ứng dụng ở Website.
- Linh hoạt đối với các dạng dữ liệu trả về Client: Json, XML hay những định dạng khác nữa.
- Dễ dàng xây dựng được HTTP service: URI, URI, request/response headers, caching, versioning, content formats và cả host trong ứng dụng.

Ưu điểm của Web API

- Với mã nguồn mở có thể giúp hỗ trợ những chức năng của Restful một cách đầy đủ.
- Hỗ trợ về thành phần MVC như: routing, controller, action result, filter, model binder, IoC container, dependency injection, unit test.
- Giao tiếp 2 chiều được xác nhận, vì vậy các giao dịch có thể đảm bảo được độ tin cậy cao hơn.

Nhược điểm của Web API

- Web API chưa được gọi là Restful Service bởi nó chỉ mới hỗ trợ mặc định Get, Post.
- Nếu muốn sử dụng tốt nhất bạn cần có kiến thức và am hiểu thật sự về backend.
- Khá mất thời gian cho việc phát triển cũng như nâng cấp, vận hành.
- Hệ thống có thể bị tấn công nếu như không giới hạn chức năng hay điều kiện.

RESTful API là gì?

REST (REpresentational State Transfer) là một dạng chuyển đổi cấu trúc dữ liệu, một kiểu kiến trúc để viết API. Nó sử dụng phương thức HTTP đơn giản để tạo cho giao tiếp giữa các máy.

Vì vậy, thay vì sử dụng một URL cho việc xử lý một số thông tin người dùng, REST gửi một yêu cầu HTTP như GET, POST, DELETE, vv đến một URL để xử lý dữ liệu.

RESTful API là gì?

RESTful API là một tiêu chuẩn dùng trong việc thiết kế các API cho các ứng dụng web để quản lý các resource.

RESTful là một trong những kiểu thiết kế API được sử dụng phổ biến ngày nay để cho các ứng dụng (web, mobile…) khác nhau giao tiếp với nhau.

Ưu điểm của RESTful API

- Giúp cho ứng dụng rõ ràng hơn
- Dữ liệu được trả về với nhiều định dạng khác nhau như: xml, html, json….
- Code đơn giản và ngắn gọn
- Chú trọng vào tài nguyên của hệ thống

Cách hoạt động của RESTful API

- REST hoạt động chủ yếu dựa vào các giao thức HTTP :
- GET (SELECT): Trả về một Resource hoặc một danh sách Resource.
- POST (CREATE): Tạo mới một Resource.
- PUT (UPDATE): Cập nhật thông tin cho Resource.
- DELETE (DELETE): Xoá một Resource.

Những phương thức hay hoạt động này thường được gọi là CRUD tương ứng với Create, Read, Update, Delete

Status code

- Khi chúng ta request một API nào đó thường thì sẽ có vài status code để nhận biết sau:
  - 200 OK – Trả về thành công cho những phương thức GET, PUT, PATCH hoặc DELETE.
  - 201 Created – Trả về khi một Resouce vừa được tạo thành công.
  - 204 No Content – Trả về khi Resource xoá thành công.
  - 304 Not Modified – Client có thể sử dụng dữ liệu cache.
  - 400 Bad Request – Request không hợp lệ
  - 401 Unauthorized – Request cần có auth.
  - 403 Forbidden – bị từ chối không cho phép.

Status code

- 404 Not Found – Không tìm thấy resource từ URI
- 405 Method Not Allowed – Phương thức không cho phép với user hiện tại.
- 410 Gone – Resource không còn tồn tại, Version cũ đã không còn hỗ trợ.
- 415 Unsupported Media Type – Không hỗ trợ kiểu Resource này.
- 422 Unprocessable Entity – Dữ liệu không được xác thực
- 429 Too Many Requests – Request bị từ chối do bị giới hạn

Thảo luận

- Sử dụng công cụ Mock API để mô phỏng Backend

Tại sao sử dụng Mock API?

- Các nhà phát triển frontend và các nhà phát triển phụ trợ có thể làm việc song song, do đó phát triển nhanh chóng.
- Các nhà phát triển UI / UX chỉ có thể bắt đầu các API mô phỏng phụ trợ bắt buộc
- Giao diện người dùng có thể hoạt động như một ứng dụng độc lập trong quá trình phát triển mà không có bất kỳ phụ thuộc API phụ trợ nào.
- Bật phát triển ngoại tuyến
- Dễ dàng demo
- API giả có thể dễ dàng được thay thế bằng API thực khi nó đã sẵn sàng.

Cấu hình Mock API

Thư viện npm sử dụng để mock API là **connect-api-mocker**

connect-api-mocker có thể được sử dụng với rất nhiều Node framework như Connect, Express và BrowserSync, bạn có thể lựa chọn trong các cách trên để cài đặt

Cài đặt

```
npm i --save-dev express connect-api-mocker
```

Cấu hình Mock API

Sử dụng với **Connect**

```
var http = require('http');
var connect = require('connect');
var apiMocker = require('connect-api-mocker');

var app = connect();

app.use('/api', apiMocker('mocks/api'));

http.createServer(app).listen(8080);
```

Cấu hình Mock API

Sử dụng với **Express**

```
var express = require('express');
var apiMocker = require('connect-api-mocker');

var app = express();

app.use('/api', apiMocker('mocks/api'));

app.listen(8080);
```

Cấu hình Mock API

Sử dụng với **BrowserSync**

```
var browserSync = require('browser-sync').create();
var apiMocker = require('connect-api-mocker');
var restMock = apiMocker('/api', 'mocks/api');

browserSync.init({
  server: {
    baseDir: './',
    middleware: [ restMock,
    ],
  },
  port: 8080,
});
```

Demo

- Tạo Mock API

Thảo luận

- Thực hiện HTTP request với Axios

HTTP là gì ?

HTTP (Hyper Text Transfer Protocol) là một giao thức nằm ở tầng ứng dụng (Application layer) của tập giao thức TCP/IP, sử dụng để truyền nhận dữ liệu giữa các hệ thống phân tán thông qua internet

Quá trình làm việc của HTTP

- HTTP client thiết lập một kết nối TCP đến server. Nếu thiết lập thành công, client và server sẽ truyền nhận dữ liệu với nhau thông qua kết nối này, kết nối được thiết lập còn gọi là socket interface
  - Bao gồm các thông tin: địa chỉ IP, loại giao thức giao vận (chính là TCP), và port (mặc định là 80).

- Sau khi kết nối thành công, client gửi một HTTP request đến server thông qua socket interface vừa được thiết lập. Trong gói tin request sẽ chứa đường dẫn yêu cầu

Quá trình làm việc của HTTP

Quá trình làm việc của HTTP sẽ diễn ra như sau:

- Server sẽ nhận và xử lý request từ client thông qua socket, sau đó đóng gói dữ liệu tương ứng và gửi một HTTP response về cho client.

  Dữ liệu trả về sẽ là một file HTML chứa các loại dữ liệu khác nhau như văn bản, hình ảnh,…
- Server đóng kết nối TCP.
- Client nhận được dữ liệu phản hồi từ server và đóng kết nối TCP.

Giới thiệu Axios

Axios là một thư viện HTTP Client dựa trên Promise. Cơ bản thì nó cung cấp một API cho việc xử lý XHR (XMLHttpRequests).

- Tạo XMLHttpRequests từ trình duyệt
- Thực hiện các http request từ node.js
- Hỗ trợ Promise API
- Chặn request và response
- Chuyển đổi dữ liệu request và response
- Hủy requests
- Tự động chuyển đổi về dữ liệu JSON
- Hỗ trợ phía client để chống lại CSRF(tấn công giả mạo)

Demo

- Cài đặt axios

```
npm install axios hoặc yarn add axios
```

- Tạo Request với Axios

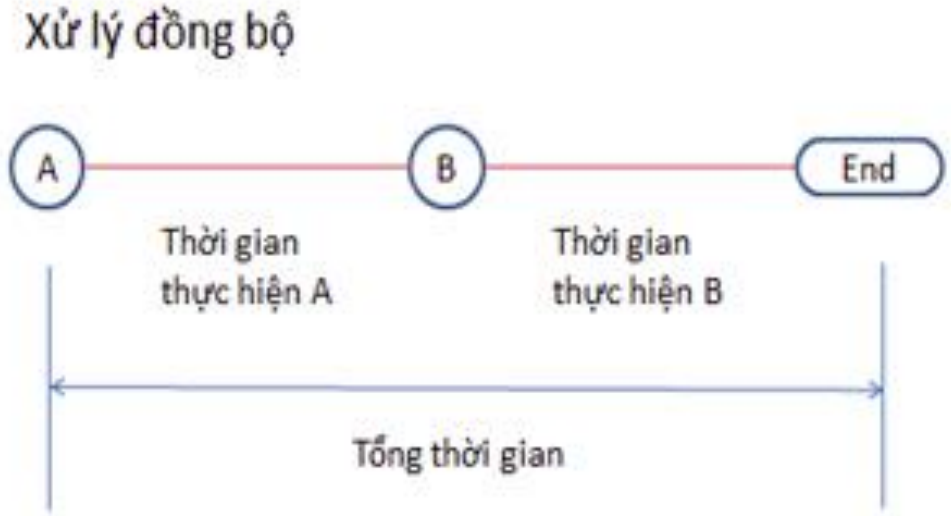
Thảo luận

- Xử lý bất đồng bộ với Promise và Async/await

Sync và Async

Đồng bộ : có nghĩa là thực hiện các công việc một cách tuần tự, công việc này xong thì mới được thực hiện các công việc khác.

Ví dụ có 2 công việc A và B thì khi có nghĩa là A thực hiện xong trước rồi mới tới lượt B. Điều này nó sẽ ảnh hưởng đến hiệu suất của người dùng

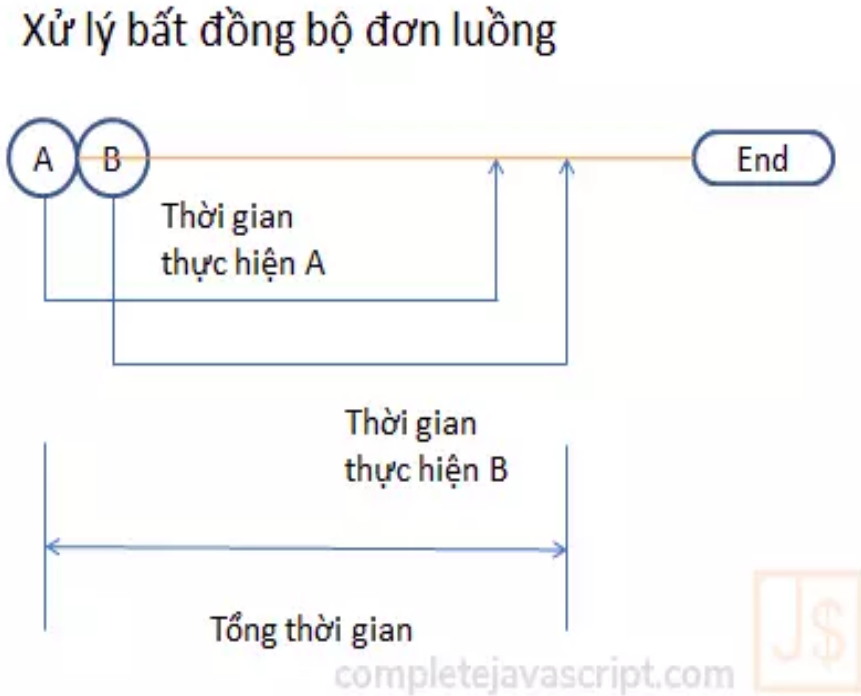


Sync và Async

Bất đồng bộ : Với cách xử lý bất đồng bộ, khi A bắt đầu thực hiện, chương trình tiếp tục thực hiện B mà không đợi A kết thúc.

Việc mà bạn cần làm ở đây là cung cấp một phương thức để chương trình thực hiện khi A hoặc B kết thúc.

Cơ chế giúp bạn thực hiện việc này trong JavaScript là sử dụng Callback, Promise hoặc Async/await.



Sử dụng Promise để xử lý bất đồng bộ

- Promise sinh ra để xử lý kết quả của một hành động cụ thể, kết quả của mỗi hành động sẽ là thành công hoặc thất bại
- Và Promise sẽ giúp chúng ta giải quyết câu hỏi "Nếu thành công thì làm gì? Nếu thất bại thì làm gì?”
- Khi một Promise được khởi tạo thì nó có một trong ba trạng thái sau:
  - Fulfilled Hành động xử lý xong và thành công
  - Rejected Hành động xử lý xong và thất bại
  - Pending Hành động đang chờ xử lý hoặc bị từ chối
- Trong đó hai trạng thái Reject và Fulfilled ta gọi là Settled, tức là đã xử lý xong.

Khái niệm Async/Await

Là một cơ chế giúp bạn thực hiện các thao tác bất đồng bộ một cách tuần tự hơn.

Async/await vẫn sử dụng Promise nhưng mã nguồn của bạn (theo một cách nào đó) sẽ trong sáng và dễ theo dõi.

Để sử dụng, bạn phải khai báo hàm với từ khóa **async,await**

Khái niệm Async/Await

Async - khai báo một hàm bất đồng bộ

- Tự động biến đổi một hàm thông thường thành một Promise.
- Khi gọi tới hàm async nó sẽ xử lý mọi thứ và được trả về kết quả trong hàm của nó.
- Async cho phép sử dụng Await.

Khái niệm Async/Await

Await sẽ được sử dụng ở trước các thao tác cần đồng bộ , tạm dừng việc thực hiện các hàm async

- Khi được đặt trước một Promise, nó sẽ đợi cho đến khi Promise kết thúc và trả về kết quả.
- Await chỉ làm việc với Promises, nó không hoạt động với callbacks.
- Await chỉ có thể được sử dụng bên trong các function async.

Xử lý lỗi trong Async / Await

Một điều tuyệt vời khác về Async / Await là nó cho phép chúng ta bắt các lỗi không mong đợi bằng cách sử dụng try / catch

```
async function doSomethingAsync(){
        try {
            // This async call may fail.
            let result = await someAsyncCall();
        }
        catch(error) {
            // If it does we will catch the error here.
        }
    }
```

Xử lý lỗi trong Async / Await

- Mệnh đề catch sẽ xử lý các lỗi gây ra bởi các hàm bất đồng bộ hoặc bất kỳ lỗi nào chúng ta có thể đã viết bên trong khối try.
- Trong một vài tình huống, chúng ta cũng có thể bắt các lỗi khi đang thực hiện function async.
- Vì tất cả các hàm async đều trả về Promises, chúng ta chỉ cần gọi thêm hàm .catch() khi gọi chúng.

Demo

- Sử dụng Promise để xử lý bất đồng bộ
- Sử dụng Async/Await để xử lý bất đồng bộ

Tổng kết

Qua bài này chúng ta đã tìm hiểu:

- Giới thiệu Web API với kiến trúc RESTfull
- Sử dụng công cụ Mock API để mô phỏng Backend
- Thực hiện HTTP request với Axios
- Xử lý bất đồng bộ với Promise và Async/await