Học phần: Phát triển ứng dụng Web (CT449)

HTTP API

Bùi Võ Quốc Bảo, Nguyễn Minh Trung | Bộ môn CNTT

Tài liệu tham khảo

- http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
- REST API Best Practices
- REST API Tutorial (http://www.restapitutorial.com/)
- The RESTful CookBook (http://restcookbook.com/)

API (Application Programming Interface)

"a set of functions and procedures that allow the creation of applications which access the features or data of an operating system, application, or other service."

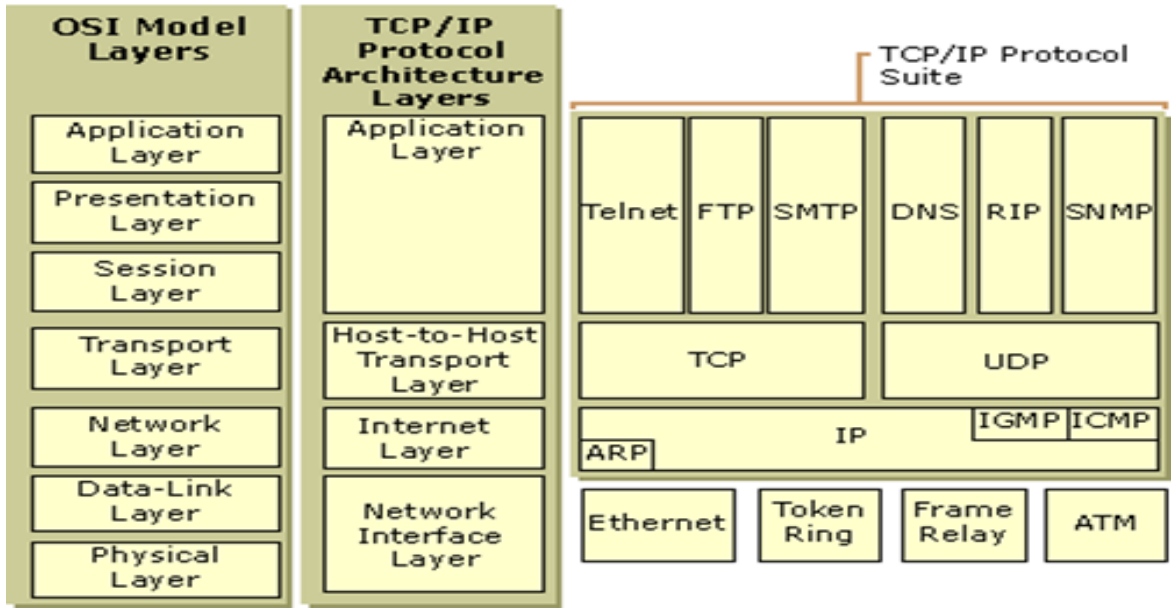
— Oxford English Dictionary

"An interface or go-between that enables a software program to interact with other software."

— Investopedia

Giao thức HTTP (HyperText Transfer Protocal)

- Một giao thức ứng dụng trong bộ các giao thức TCP/IP



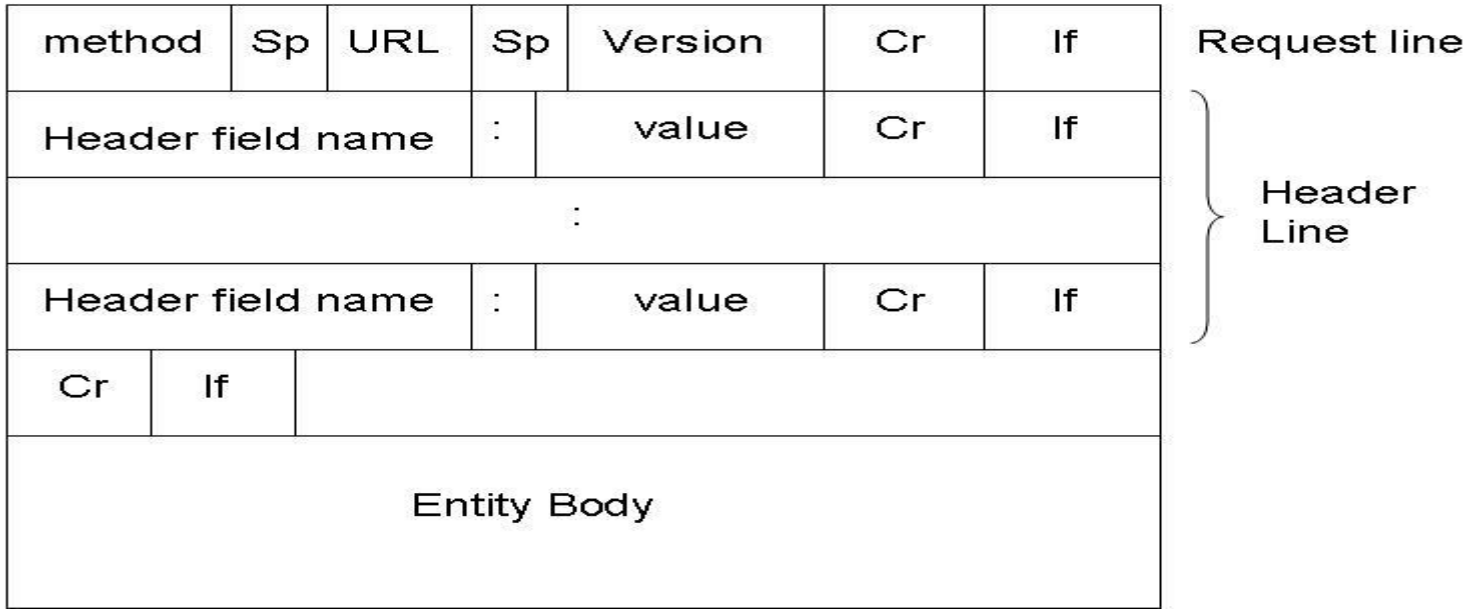
- Dùng để giao tiếp giữa Web server và Web browser
- Dữ liệu trao đổi: các siêu văn bản, hình ảnh, âm thanh, video
- Phiên bản HTTP: 0.9, 1.0, 1.1, /2, /3

Giao thức HTTP (HyperText Transfer Protocal)

- Web server và Web browser trao đổi với nhau thông qua thông điệp HTTP (HTTP message)
  - ✓ Thông điệp yêu cầu
  - ✓ Thông điệp đáp ứng

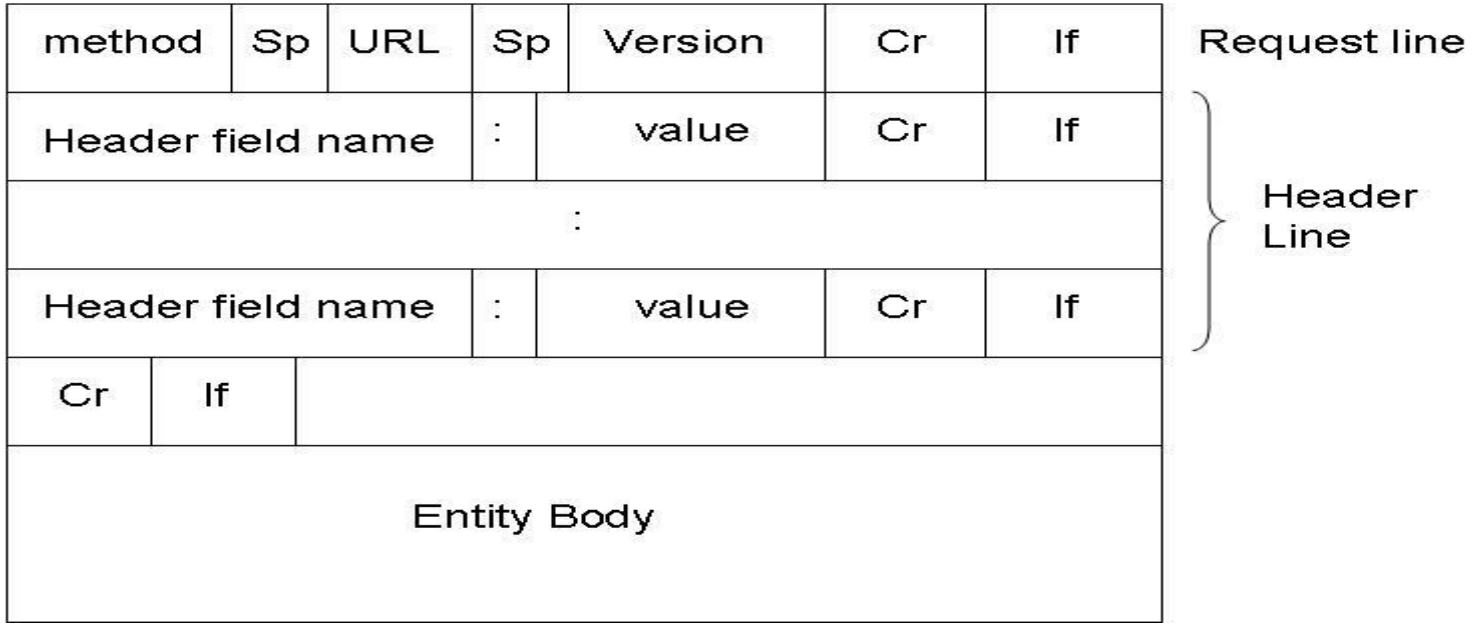
Giao thức HTTP (HyperText Transfer Protocal)

- Thông điệp yêu cầu (HTTP request)
  - ✓ Là thông điệp được gởi từ Web browser tới Web server
  - ✓ Cấu trúc một thông điệp yêu cầu:



Giao thức HTTP (HyperText Transfer Protocal)

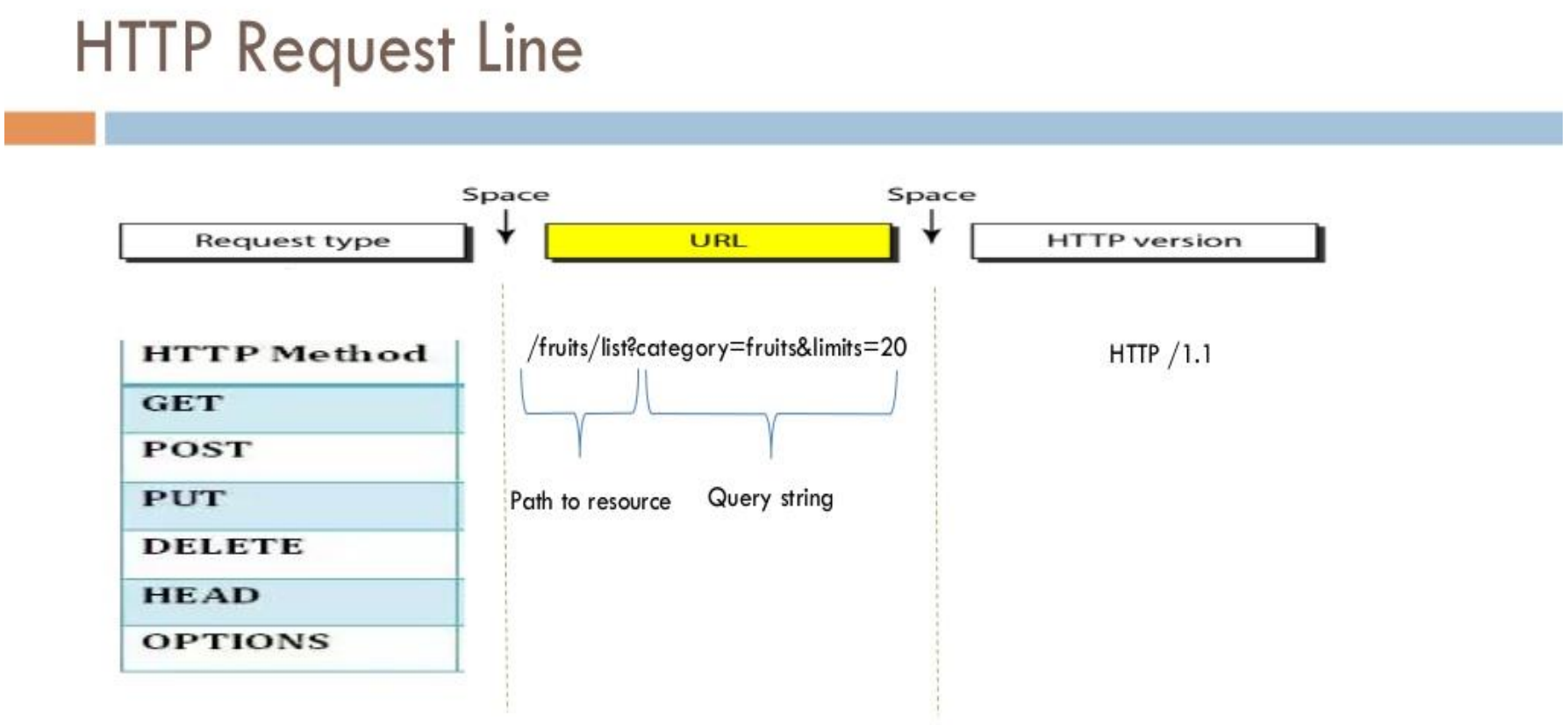
- Cấu trúc một thông điệp yêu cầu:
  - ✓ Gồm 3 phần chính
    - ➢ Dòng yêu cầu (request line)
    - ➢ Các trường tiêu đề (header fields)
    - ➢ Phần thân (Entity body)



Giao thức HTTP (HyperText Transfer Protocal)

- Dòng yêu cầu (request line)
  - ✓ Chứa đựng 3 thông tin
    - ➢ Phương thức yêu cầu (method)
    - ➢ Địa chỉ trang Web liên quan đến yêu cầu (URL)
    - ➢ Phiên bản HTTP (version)
    - ➢ Mỗi thông tin dòng yêu cầu cách nhau bởi 1 khoảng trắng

Giao thức HTTP (HyperText Transfer Protocal)



Giao thức HTTP (HyperText Transfer Protocal)

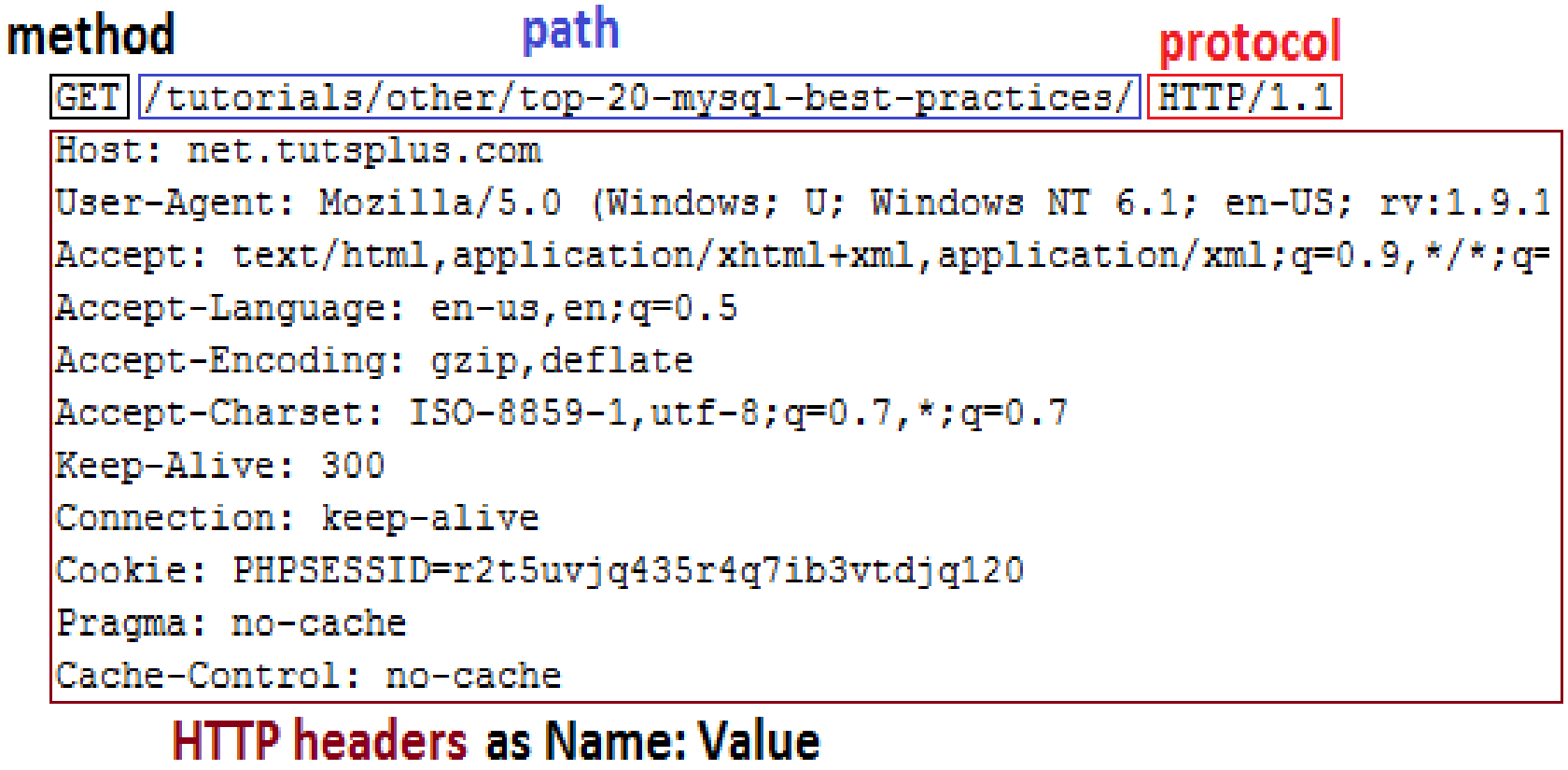
- Phương thức yêu cầu (method)
  - ✓ GET: yêu cầu 1 trang Web từ Web server với địa chỉ được chỉ định trong URL của thông điệp
    - ➢ Các đối số cho trang Web (nếu có) sẽ được đặt bên trong URL
    - ➢ http://www.laptrinhweb.vn/bai-viet?page=2&limit=20
  - ✓ POST: giống như GET
    - ➢ Các đối số cho trang Web (nếu có) sẽ được đặt bên trong thông điệp yêu cầu
  - ✓ PUT: upload một trang Web lên Web server tại địa chỉ được chỉ định bởi URL

Giao thức HTTP (HyperText Transfer Protocal)

- Các trường tiêu đề (header fields)
  - ✓ Cung cấp thông tin kèm theo yêu cầu hay các ràng buộc đối với Web server khi phục vụ yêu cầu từ Web browser
  - ✓ Mỗi trường gồm: <tên trường> : <giá trị trường>
  - ✓ Một số trường cơ bản:
    - ➢ Accept: loại dữ liệu được chấp nhận bởi Web browser (*text/html, text/plain, ...*)
    - ➢ Accept-Charset: bảng mã được chấp nhận bởi Web browser (*utf-8, iso-8859-5, ...*)
    - ➢ Accept-Encoding: cách mã hóa thông tin cho phần thân thông điệp (*gzip, deflate, ...*)
    - ➢ Content-Length: kích thước dữ liệu theo bytes trong phần thân thông điệp
    - ➢ Accept-Language: ngôn ngữ được chấp nhận bởi Web browser

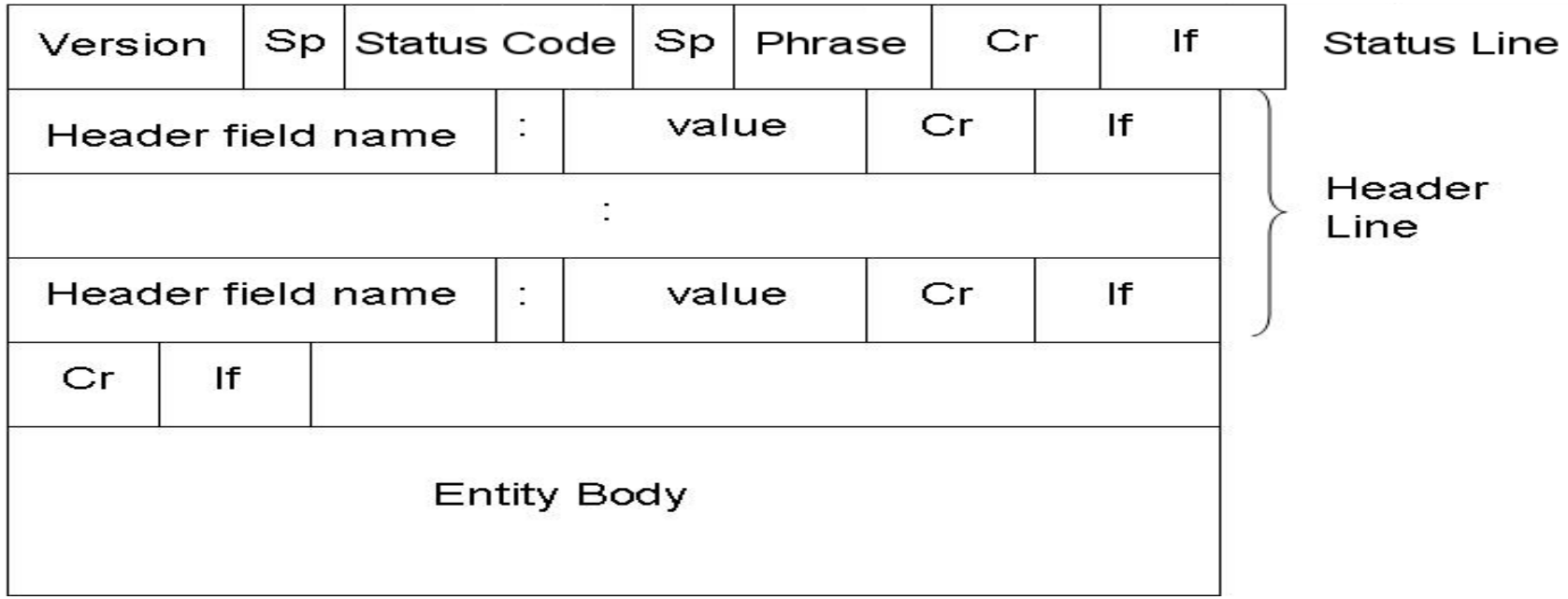
Giao thức HTTP (HyperText Transfer Protocal)

- Các trường tiêu đề (header fields)



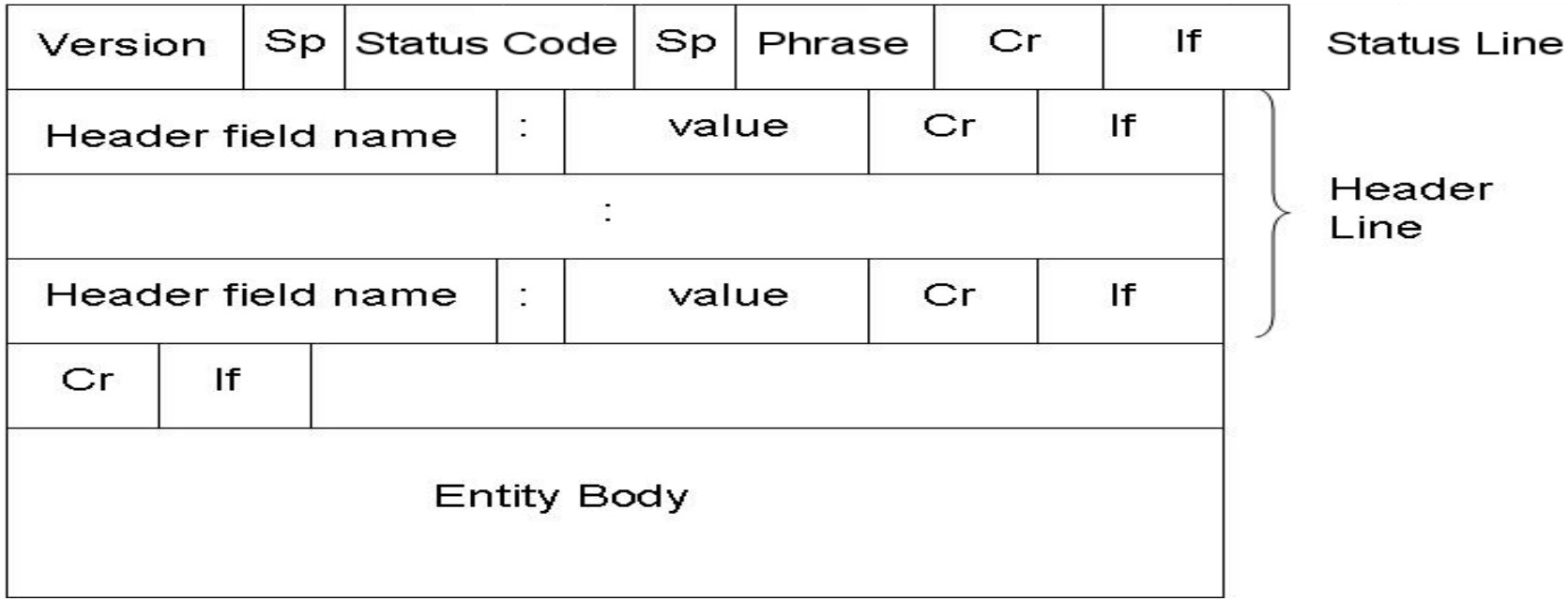
Giao thức HTTP (HyperText Transfer Protocal)

- Thông điệp đáp ứng (HTTP response)
  - ✓ Được gởi từ Web server về Web browser để đáp ứng yêu cầu của Web browser
  - ✓ Cấu trúc thông điệp đáp ứng gồm 3 phần



Giao thức HTTP (HyperText Transfer Protocal)

- Cấu trúc thông điệp đáp ứng gồm 3 phần
  - ✓ Dòng trạng thái (status line)
  - ✓ Các trường tiêu đề (header fields)
  - ✓ Thân thông điệp (message body)



Giao thức HTTP (HyperText Transfer Protocal)

- Dòng trạng thái (status line)
  - ✓ Gồm 3 ký tự số cho biết trạng thái phục vụ yêu cầu của Web server
    - ➢ Ký tự số đầu tiên cho biết trạng thái phục vụ
    - ➢ Hai ký tự cuối cung cấp thông tin chi tiết hơn về trạng thái

Type	Status Codes	Examples
Informational	1xx	100 Continue, 101 Switching Protocols
Success	2xx	200 - OK , 201 - Created, 202 Accepted
Redirection	3xx	300 Multiple Choices, 301 Moved Permanently, 302 - Found
Client Error	4xx	400 Bad Request, 403 - Forbidden , 404 - Not Found , 422 - Unprocessable Entity
Server Error	5xx	500 - Internal Server Error , 503 - Service Unavailable

Giao thức HTTP (HyperText Transfer Protocal)

- Các trường tiêu đề (header fields)
  - ✓ Có 3 loại trường tiêu đề:
    - ➢ Trường tiêu đề thông thường: cung cấp các thông tin cơ bản như Date, Transfer-Encoding.
    - ➢ Trường tiêu đề đáp ứng: truyền thông tin về đáp ứng mà Web server không thể truyền qua dòng trạng thái
      - o Server
      - o Allow
      - o Cache-Control
      - o Location …
    - ➢ Trường tiêu đề thực thể: cung cấp các thông tin về dữ liệu trong phần thân thông điệp.

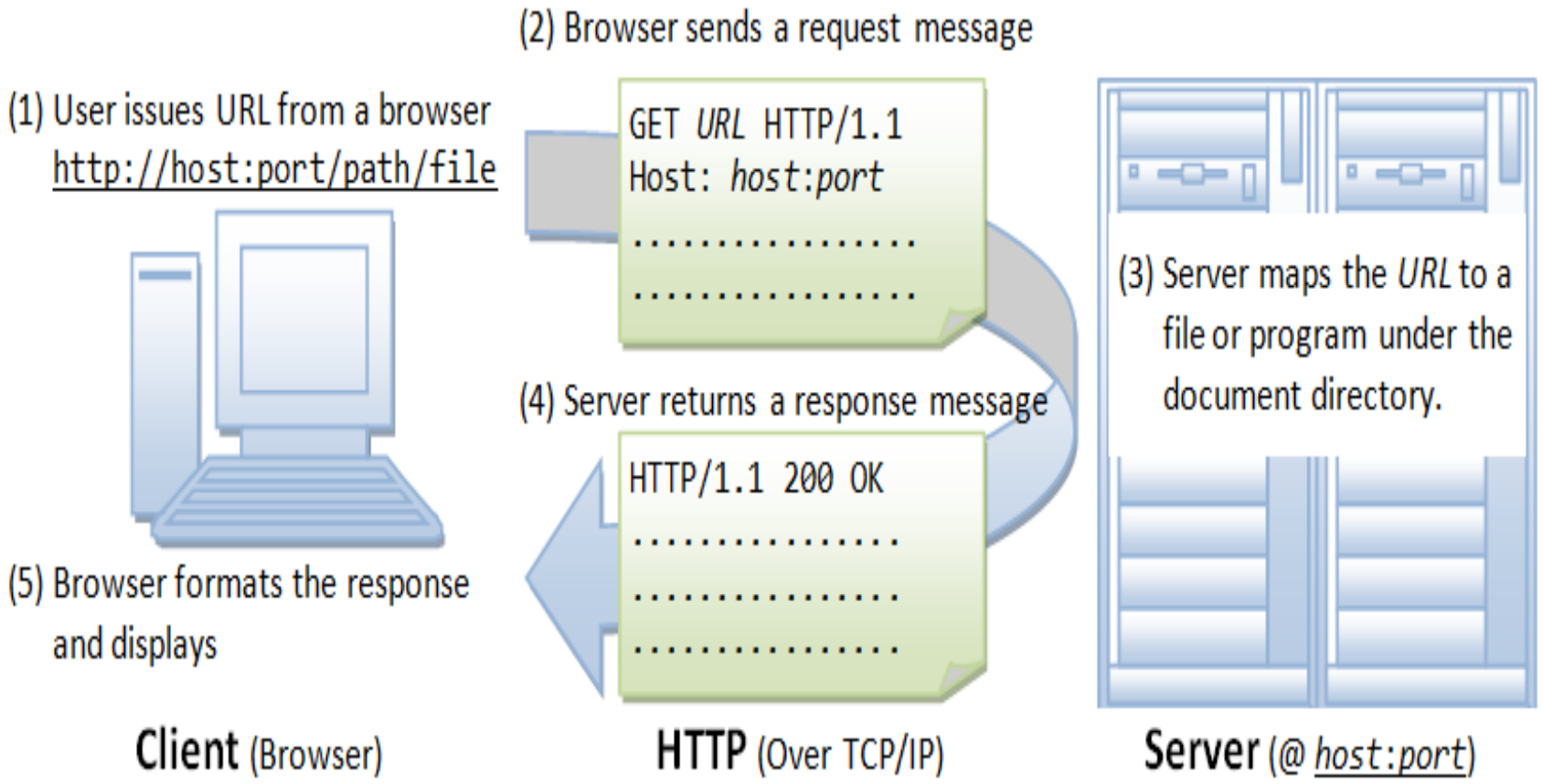
Giao thức HTTP (HyperText Transfer Protocal)

protocol **status code**

```
HTTP/1.x 200 OK
Transfer-Encoding: chunked
Date: Sat, 28 Nov 2009 04:36:25 GMT
Server: LiteSpeed
Connection: close
X-Powered-By: W3 Total Cache/0.8
Pragma: public
Expires: Sat, 28 Nov 2009 05:36:25 GMT
Etag: "pub1259380237;gz"
Cache-Control: max-age=3600, public
Content-Type: text/html; charset=UTF-8
Last-Modified: Sat, 28 Nov 2009 03:50:37 GMT
X-Pingback: http://net.tutsplus.com/xmlrpc.php
Content-Encoding: gzip
Vary: Accept-Encoding, Cookie, User-Agent
```

HTTP headers as Name: Value

Giao thức HTTP (HyperText Transfer Protocal)



HTTP API

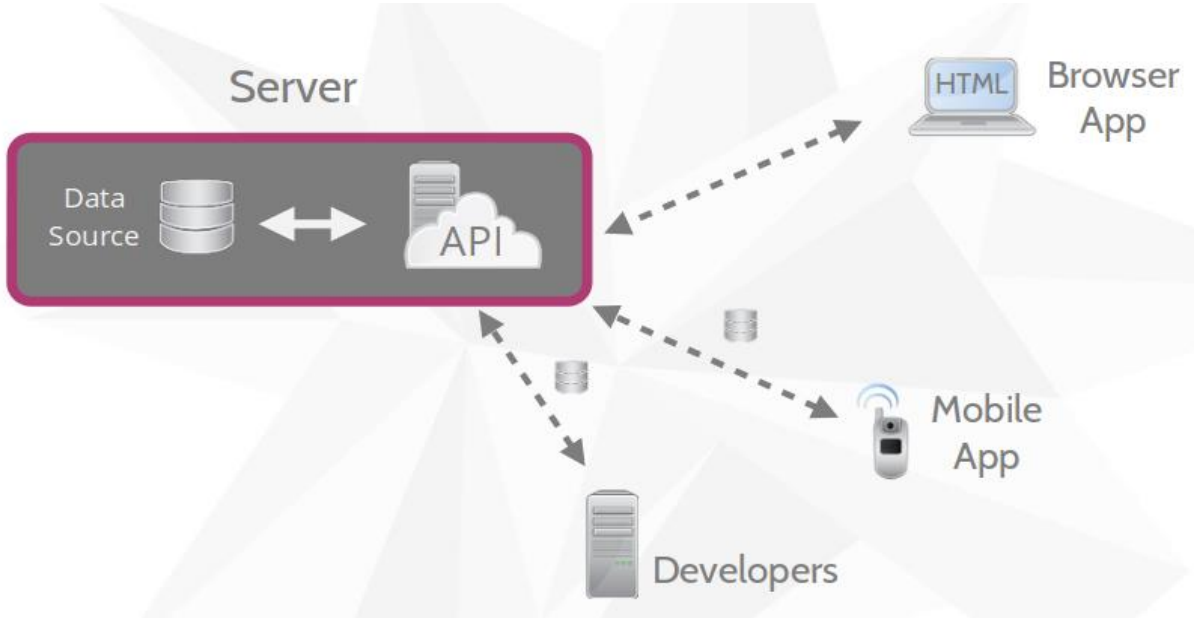
- API được truy xuất thông qua giao thức HTTP
  - Yêu cầu/trả lời trên giao thức HTTP
  - Làm giao thức giao tiếp giữa hai hệ thống.
    - Ví du: sử dụng API HTTP lên lịch cuộc họp Zoom trên lịch Google (Google calendar)

    → API xác định cách Zoom có thể giao tiếp trực tiếp với máy chủ của Google để nhúng cuộc họp Zoom vào sự kiện
  - API HTTP cung cấp các điểm cuối (endpoint) dưới dạng cổng API

    → cho phép các truy vấn HTTP kết nối với máy chủ.
  - Một khái niệm, không phải là một công nghệ → có thể được xây dựng bằng nghiều công nghệ khác nhau: Java, .NET, PHP,...

Xây dựng ứng dụng dùng HTTP API ???

- HTTP API
  - Được phân loại theo các nguyên tắc thiết kế kiến trúc được sử dụng khi chúng được tạo
  - Được sử dụng trong hệ thống thông tin siêu phương tiện hoặc phát triển web



Bên lề: Web Service vs Web API vs WebSocket

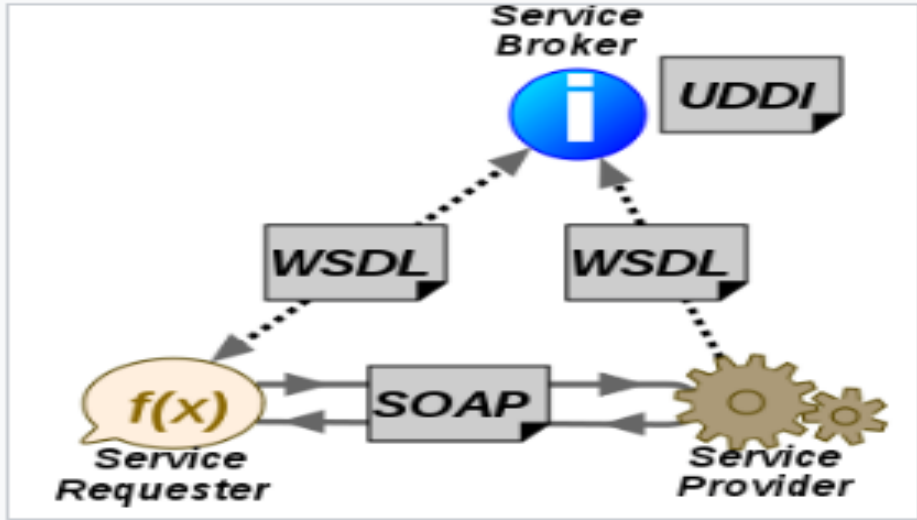
- Web Service:
  - Một thuật ngữ tổng quát nói về một chương trình server lắng nghe các yêu cầu tại một cổng cụ thể trong một mạng, phục vụ các tài liệu web (HTML, JSON, XML, hình ảnh, ...), thường thông qua giao thức HTTP
  - Trước đây thuật ngữ này đồng nghĩa với một giao thức cài đặt web service là SOAP (Simple Object Access Protocol)
  - Ngày nay thuật ngữ này thường đồng nhất với Web/HTTP API

Bên lề: Web Service vs Web API vs WebSocket

- Web Service:
  - XML và HTTP là nền tảng dịch vụ web cơ bản nhất
  - Các thành phần sau được sử dụng bởi tất cả các dịch vụ web điển hình
    - SOAP (Simple Object Access Protocol):
      - giao thức truyền tải dữ liệu, không phụ thuộc vào ngôn ngữ lập trình hay nền tảng
      - dựa trên việc truyền dữ liệu XML dưới dạng một XML document
    - UDDI (Universal Description, Discovery, and Integration)
      - một tiêu chuẩn để chỉ định, xuất bản và khám phá các dịch vụ trực tuyến của nhà cung cấp dịch vụ
    - WSDL (Web Services Description Language)
      - một định dạng XML để mô tả các dịch vụ mạng như một tập hợp các điểm cuối hoạt động và chức năng được cung cấp bởi dịch vụ web

Bên lề: Web Service vs Web API vs WebSocket

- Web Service:



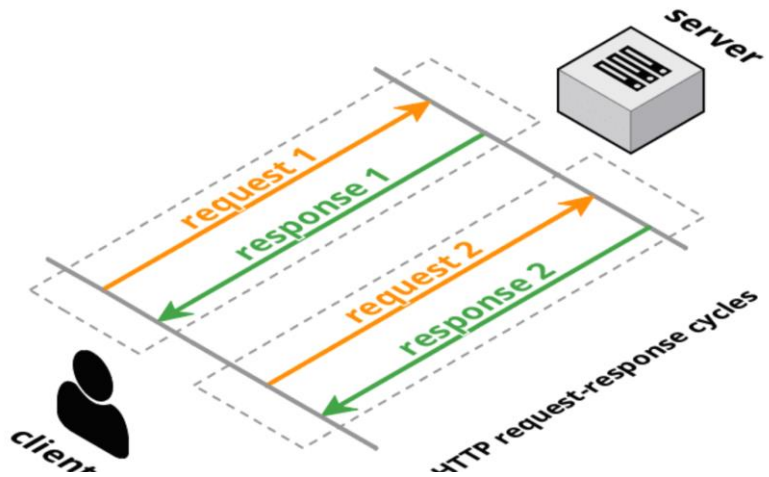
Web services architecture: the service provider sends a WSDL file to UDDI. The service requester contacts UDDI to find out who is the provider for the data it needs, and then it contacts the service provider using the SOAP protocol. The service provider validates the service request and sends structured data in an XML file, using the SOAP protocol. This XML file would be validated again by the service requester using an XSD file.

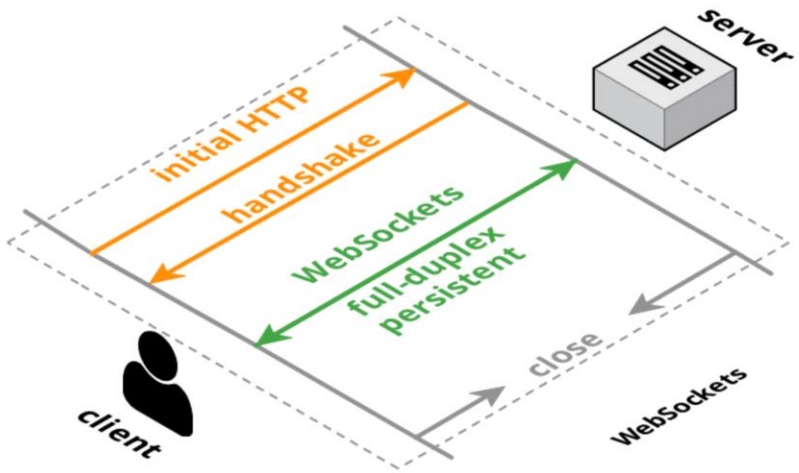
Bên lề: Web Service vs Web API vs WebSocket

- Web API (HTTP API)
  - API được truy xuất thông qua giao thức HTTP
  - Phổ biến nhất là dạng REST (Representational State Transfer) API. Tuy nhiên có thể được cài đặt theo dạng khác như RPC (Remote Procedure Call) API

Bên lề: Web Service vs Web API vs WebSocket

- WebSocket
  - Một giao thức truyền thông cho phép tạo phiên kết nối tương tác hai chiều (full-duplex) giữa client và server bằng cách sử dụng một TCP socket .
    - Server có thể gửi dữ liệu cho client mà không cần client phải thực hiện yêu cầu trước
    - Kết nối giữa máy khách và máy chủ sẽ tiếp tục tồn tại cho đến khi nó bị một bên (máy khách hoặc máy chủ) chấm dứt





REST

- **RE**presentational **S**tate **T**ransfer
  - Một kiểu/dạng kiến trúc phần mềm (software architectural style) dùng trong việc thiết kế các ứng dụng mạng
  - Một kiểu kiến trúc để cung cấp các tiêu chuẩn giữa các hệ thống máy tính trên web, giúp các hệ thống giao tiếp với nhau dễ dàng hơn
  - Mô hình thiết kế phổ biến nhất cho các HTTP API
- Ràng buộc trong kiến trúc REST (6 ràng buộc)
  - Uniform interface, Client-server, stateless, cache, layered system, code-on-demand (optional)

REST

- Ràng buộc trong kiến trúc REST (6 ràng buộc)
  - Uniform interface:
    - Một ràng buộc mô tả giao tiếp giữa máy khách và máy chủ.
      - Để cho phép mỗi phần phát triển không phụ thuộc vào nhau.
    - Giao diện thống nhất được chia thành bốn nhóm chính, được gọi là các nguyên tắc: *resource-based*, *the manipulation of resources using representations*, *self-descriptive messages*, *hypermedia as the Engine of Application State (HATEOAS)*

REST

- Ràng buộc trong kiến trúc REST (6 ràng buộc)
  - Uniform interface :
    - Resource-based
      - Giao diện phải xác định duy nhất từng tài nguyên liên quan đến tương tác giữa máy khách và máy chủ.
      - Các tài nguyên riêng lẻ được xác định trong các yêu cầu (request) sử dụng URI làm định danh tài nguyên
    - The manipulation of resources using representations
      - Khi máy khách đưa ra yêu cầu với máy chủ, máy chủ sẽ phản hồi bằng một tài nguyên đại diện cho trạng thái hiện tại của tài nguyên.
      - Tài nguyên này có thể được thao tác bởi máy khách
      - Client có thể yêu cầu loại mà nó mong muốn cho biểu diễn, chẳng hạn như JSON, XML hoặc văn bản thuần túy.

REST

- Ràng buộc trong kiến trúc REST (6 ràng buộc)
  - Uniform interface :
    - Self-descriptive messages
      - Mỗi biểu diễn tài nguyên phải mang đủ thông tin để mô tả cách xử lý thông điệp (message).
      - Nó cũng phải cung cấp thông tin về các hành động bổ sung mà client có thể thực hiện trên tài nguyên.
    - Hypermedia as the Engine of Application State (HATEOAS)
      - Cách mà client có thể tương tác với phản hồi bằng cách điều hướng bên trong nó thông qua hệ thống phân cấp để nhận được thông tin bổ sung.

REST

- Ràng buộc trong kiến trúc REST (6 ràng buộc)
  - Uniform interface :
    - Hypermedia as the Engine of Application State (HATEOAS)

HATEOAS

HATEOAS is a way that the client can interact with the response by navigating within it through the hierarchy in order to get complementary information.

For example, here the client makes a `GET` call to the `order` URI :

```
GET https://<HOST>/orders/1234
```

The response comes with a navigation link to the items within the `1234` order, as in the following code block:

```
{
    id : 1234,
    any-other-json-fields...,
    links": [
        {
            "href": "1234/items",
            "rel": "items",
            "type" : "GET"
        }
    ]
}
```

What happens here is that the link fields allow the client to navigate until `1234/items` in order to see all the items that belong to the `1234` order.

REST

- Ràng buộc trong kiến trúc REST (6 ràng buộc)
  - Client-server architecture
    - Tách máy Client-server:
      - Các ứng dụng máy khách và ứng dụng máy chủ PHẢI có thể phát triển riêng biệt mà không phụ thuộc vào nhau.
  - Stateless:
    - Trạng thái máy khách không được máy chủ duy trì: nó sẽ coi mọi yêu cầu là mới. không có phiên (session), không có lịch sử (history).
    - Tất cả thông tin cần thiết để xử lý các yêu cầu (requests) của client đều có trong các yêu cầu tới máy chủ.

REST

- Ràng buộc trong kiến trúc REST (6 ràng buộc)
  - Cache:
    - Mục đích của bộ nhớ đệm là không bao giờ phải tạo cùng một phản hồi nhiều lần.
      + Chiến lược này là tăng tốc độ và giảm tốc độ xử lý của máy chủ.
      + Bộ nhớ đệm có thể được thực hiện trên máy chủ hoặc phía máy khách
  - Layered system
    - Mỗi lớp phải hoạt động độc lập và chỉ tương tác với các lớp được kết nối trực tiếp với nó.
    - Hệ thống phân lớp cho phép một kiến trúc bao gồm các lớp phân cấp bằng cách hạn chế hành vi của thành phần.
      + Ví dụ, trong một hệ thống phân lớp, mỗi thành phần không thể nhìn thấy trực tiếp ngoài lớp mà chúng đang tương tác.

REST

- Ràng buộc trong kiến trúc REST (6 ràng buộc)
  - Layered system
    - Thiết kế API phải đảm bảo rằng máy khách và máy chủ không biết liệu chúng có đang giao tiếp trực tiếp với nhau hay không hoặc có các lớp trung gian khác giữa chúng.

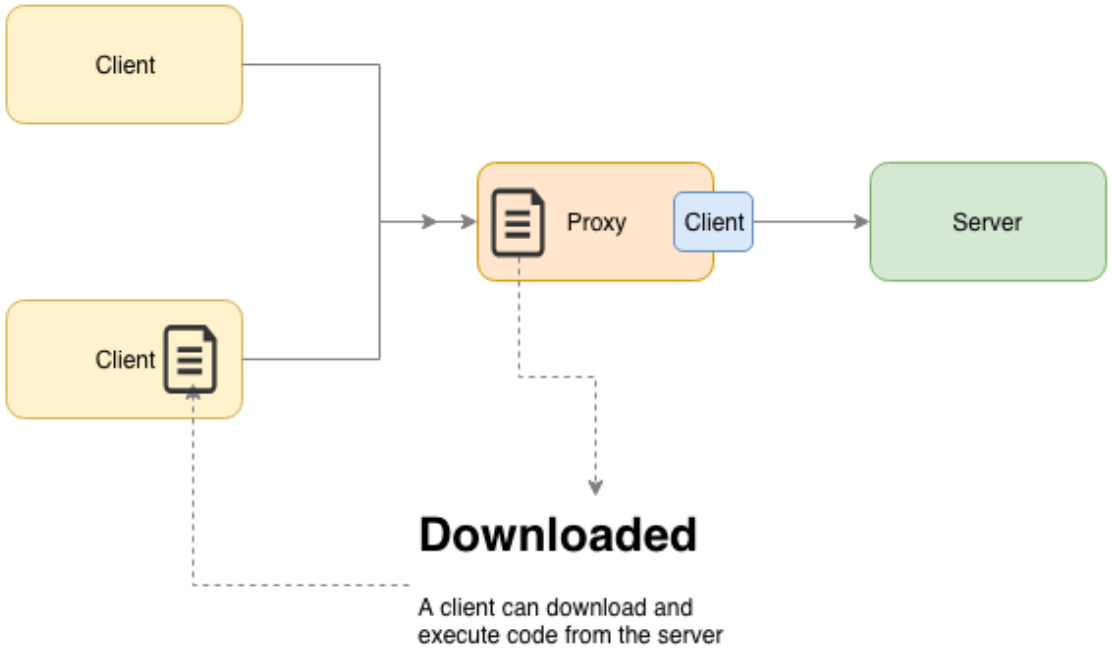
      → Điều này rất hữu ích cho khả năng mở rộng cũng như bảo mật.

      + REST cho phép bạn sử dụng kiến trúc hệ thống phân lớp nơi bạn triển khai các API trên máy chủ A và lưu trữ dữ liệu trên máy chủ B và xác thực các yêu cầu trong Máy chủ C chẳng hạn. Thông thường một clientkhông thể biết liệu nó được kết nối trực tiếp với máy chủ cuối hay một máy trung gian trên đường đi

REST

- Ràng buộc trong kiến trúc REST (6 ràng buộc)
  - Code-on-demand (optional)
    - REST cũng cho phép mở rộng chức năng máy client bằng cách cho phép tải xuống và thực thi mã dưới dạng các applet hoặc script ở máy client.

      → đơn giản hóa client bằng cách giảm số lượng các tính năng bắt buộc phải triển khai trước



REST

- Các khái niệm cơ bản:
  - Tài nguyên (resource) và trạng thái đại diện (representation)
  - Thao tác trên tài nguyên (manipulation of resources)

Tài nguyên

- Một khái niệm trừu tượng
  - Bất kì thông tin/khái niệm nào có thể được đặt tên có thể được xem như một tài nguyên
  - Các "danh từ", không phải các "động từ"
- Xác định bởi các URI (Uniform Resource Identifier)
  - Nhiều URI có thể tham chiếu đến cùng một tài nguyên

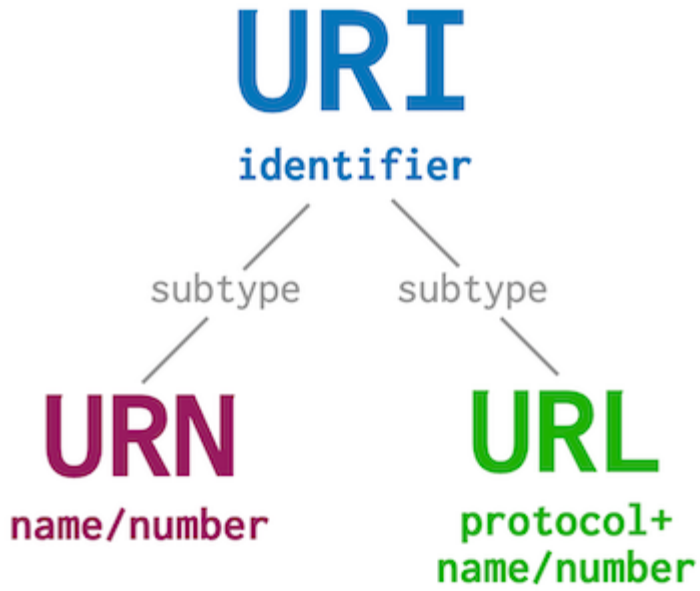
Tài nguyên

Sự khác biệt giữa URL, URI và URN

- Sự khác biệt giữa URI và URL là URI có thể chỉ là một cái tên của chính nó hoặc một cái tên với một giao thức cho bạn biết cáchtiếp cận nó - đó là một URL.

- google.com là một URI vì nó chỉ là tên của một tài nguyên.

- https://google.com là một URL vì nó vừa là tên của tài nguyên (google.com) vừa là cách để đến đó (https: //)



Tài nguyên

Sự khác biệt giữa URL, URI và URN

- A Uniform Resource Identifier (URI) is a string of characters that uniquely identify a name or a resource on the internet. A URI identi-fies a resource by name, location, or both. URIs have two specializations known as Uniform Resource Locator (URL), and Uniform Resource Name (URN).
- A Uniform Resource Locator (URL) is a type of URI that specifies not only a resource, but how to reach it on the internet—like http:// , ftp:// , or mailto:// .
- A Uniform Resource Name (URN) is a type of URI that uses the specific naming scheme of urn: —like urn:isbn:0-486-27557-4 or urn:isbn:0-395-36341-1.
- So a URI or URN is like your name, and a URL is a specific subtype of URI that's like your name combined with your address.

→ All URLs are URIs, but not all URIs are URLs.

Tài nguyên

- Có 2 dạng tài nguyên chính:
  - Tài nguyên dạng tập hợp (collection resource)
  - Collections are simply groups of resources

  Ví dụ: /lecturers
  - Tài nguyên dạng thể hiện (instance resource)

  Ví dụ: /lecturers/002626

Trạng thái đại diện

- Dữ liệu/trạng thái miêu tả/đại diện cho tài nguyên
  - Thông tin trao đổi giữa client và server
- Thông thường ở định dạng JSON hoặc XML
- Ví dụ về tài nguyên và trạng thái đại diện:
  - Tài nguyên: person
  - Trạng thái đại diện:

```
{
    "name": "Peter Jackson",
    "address": "2 rue Charles Camichel 31071 Toulouse cedex 7",
    "phone": "(+33) 5 34 32 21 69"
}
```

Các thao tác trên tài nguyên

- GET – Safe, Idempotent*, Cachable
- POST
- PATCH
- PUT – Idempotent
- DELETE – Idempotent

*Idempotent (lũy đẳng): trạng thái của server không có sự khác biệt dù một yêu cầu được gửi đến server một lần hay nhiều lần

Các thao tác trên tài nguyên

`GET <collection URI>` = Đọc tất cả các tài nguyên

`GET <instance URI>` = Đọc một tài nguyên

`POST <collection URI>` = Tạo mới một tài nguyên

`PATCH <instance URI>` = Cập nhật một phần tài nguyên

`PUT <instance URI>` = Tạo mới hoặc thay thế tài nguyên

`DELETE <instance URI>` = Xóa một tài nguyên

Dùng PUT để tạo mới hoặc cập nhật

Tạo mới: ID của tài nguyên mới được chỉ định khi gửi yêu cầu tạo mới:

```
PUT /lecturers/clientSpecifiedId

{
  "name": "Do Thanh Nghi",
  "department": "IT"
}
```

Dùng PUT để tạo mới hoặc cập nhật

Cập nhật: Thay thế toàn bộ các thuộc tính:

```
PUT /lecturers/existingId

{
  "name": "Bui Vo Quoc Bao",
  "department": "IT"
}
```

Dùng POST để tạo mới

Trên tài nguyên tập hợp (ID của tài nguyên mới tạo do server sinh ra):

```
POST /lecturers

{
   "name": "Nguyen Van A"
  "department": "Science"
}
```

Dùng PATCH để cập nhật

Trên tài nguyên thể hiện (cập nhật các thuộc tính cần thay đổi):

```
PATCH /lecturers/003025

{
  "department": "IT"
}
```

Summary of HTTP Methods

The below table summarises the use of HTTP methods discussed above.

HTTP Method	CRUD	Collection Resource (e.g. /users)	Single Resouce (e.g. /users/123)
POST	Create	201 (Created), 'Location' header with link to /users/{id} containing new ID	Avoid using POST on a single resource
GET	Read	200 (OK), list of users. Use pagination, sorting, and filtering to navigate big lists	200 (OK), single user. 404 (Not Found), if ID not found or invalid
PUT	Update/Replace	405 (Method not allowed), unless you want to update every resource in the entire collection of resource	200 (OK) or 204 (No Content). Use 404 (Not Found), if ID is not found or invalid
PATCH	Partial Update/Modify	405 (Method not allowed), unless you want to modify the collection itself	200 (OK) or 204 (No Content). Use 404 (Not Found), if ID is not found or invalid
DELETE	Delete	405 (Method not allowed), unless you want to delete the whole collection — use with caution	200 (OK). 404 (Not Found), if ID not found or invalid

Thiết kế REST API

- Tài nguyên là các danh từ, không phải là động từ
- Tất cả các tài nguyên nên hỗ trợ trả lời cho các thao tác: GET, PUT, POST, PATCH và DELETE (có thể chỉ gửi thông báo lỗi)
- Tất cả các thông tin trạng thái phải được lưu giữ ở phía client
  - Ví dụ: nếu client thực hiện một xử lý gồm 3 bước thì client phải ghi nhớ là nó đang thực hiện bước nào
- Không yêu cầu client phải tự xây dựng các URI, thay vào đó, server cần gửi đính kèm URI của các tài nguyên trong câu trả lời về cho client (HATEOAS)

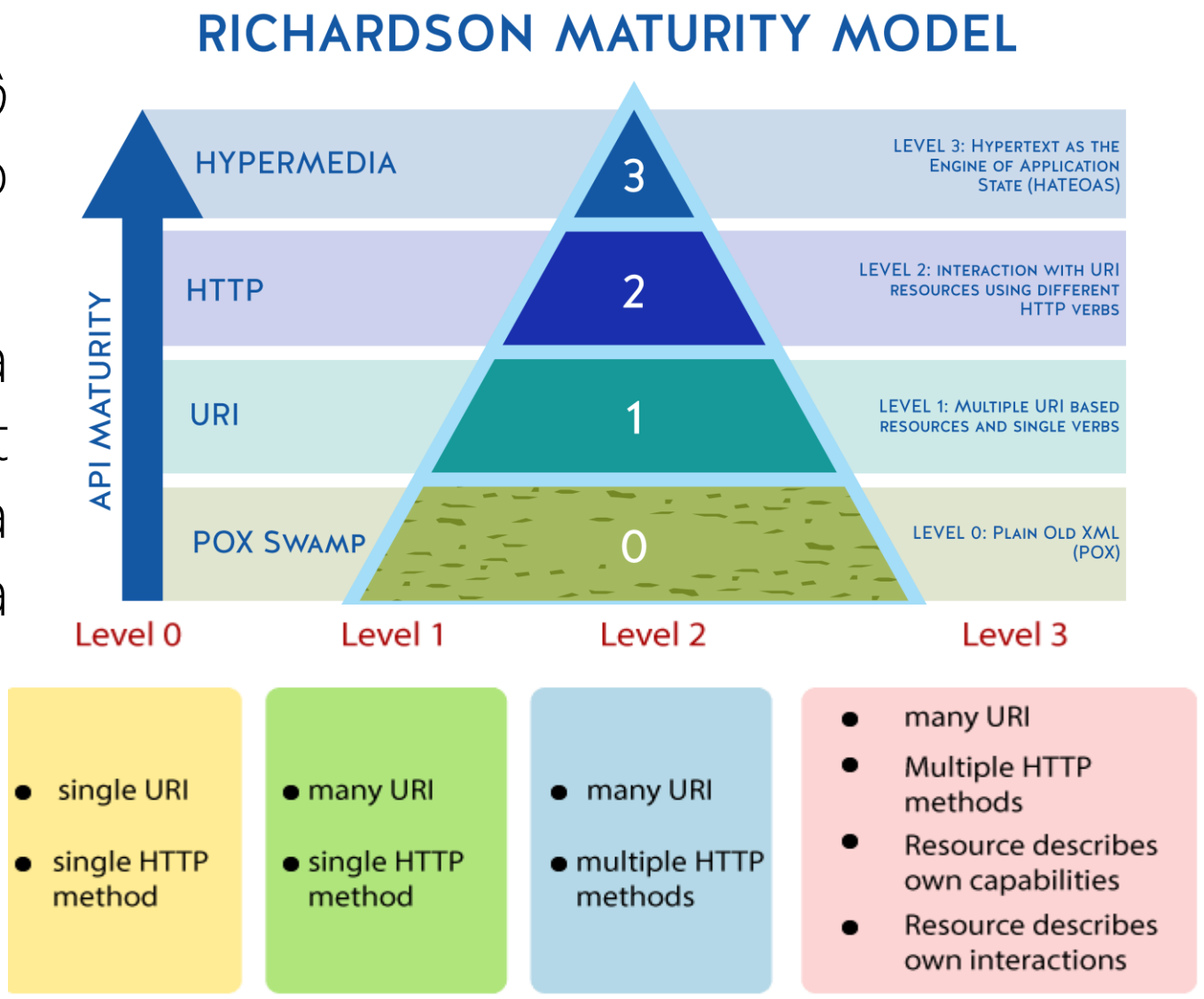
HATEOAS: Hypermedia as the Engine of Application State

Request: **GET /api/v1/cars/711**
Response:

```
{
  "id": 711,
  "manufacturer": "bmw",
  "model": "X5",
  "seats": 5,
  "drivers": [{
      "id": "23",
      "name": "Stefan Jauker",
      "links": [{"rel": "self","href": "/api/v1/drivers/23"}]}]
}
```

Mô hình trưởng thành Richardson

- Mô hình đo lường *mức độ restful* cho HTTP API bao gồm 4 cấp độ
- Richardson đã sử dụng ba yếu tố chính để quyết định sự trưởng thành của mức độ tuân thủ REST của các dịch vụ web
  - URI,
  - HTTP Methods,
  - HATEOAS (Hypermedia)



RPC (Remote Procedure Call)

- Sử dụng mô hình máy khách-máy chủ
- Trong hệ thống phân tán:
  - Được sử dụng để xây dựng các ứng dụng phân tán, dựa trên máy khách-máy chủ
  - Máy khách gọi một chức năng mà việc triển khai nó nằm trong một máy chủ từ xa.

RPC (Remote Procedure Call)

- Một dạng kiến trúc được sử dụng rộng rãi trong phát triển web để xây dựng các API
- RPC là hình thức tương tác API sớm nhất, đơn giản nhất.
  - Một API được xây dựng bằng cách định nghĩa các phương thức công khai (public methods)
    - Các phương thức được gọi với các đối số.
  - RPC chỉ là một loạt các hàm, nhưng trong ngữ cảnh của API HTTP, đòi hỏi phải đưa phương thức vào URL và các đối số trong chuỗi hoặc nội dung truy vấn.
  - Máy khách tạo một yêu cầu đến máy chủ đáp ứng nó bằng dữ liệu ở định dạng JSON hoặc XML

RPC (Remote Procedure Call)

- Ví dụ:
  - RPC

```
POST /sayHello HTTP/1.1
HOST: api.example.com
Content-Type: application/json

{"name": "Racey McRacerson"}
```

  - JavaScript

```
/* Signature */
function sayHello(name) {
  // ...
}

/* Usage */
sayHello("Racey McRacerson");
```

RPC vs REST

- RPC API là dạng *hướng hành động* (các thủ tục, mệnh lệnh) trong khi REST API là dạng *hướng tài nguyên*
- RPC thường chỉ sử dụng GET và POST, URI sẽ thể hiện hành động cụ thể sẽ thực hiện trong khi REST API hỗ trợ GET, POST, PUT, PATCH, and DELETE.

RPC vs REST

RPC:

```
POST /SendUserMessage HTTP/1.1
Host: api.example.com
Content-Type: application/json

{"userId": 501, "message": "Hello!"}
```

VS

REST:

```
POST /users/501/messages HTTP/1.1
Host: api.example.com
Content-Type: application/json

{"message": "Hello!"}
```

Định dạng trả lời theo đặc tả JSend*

- JSend là một đặc tả cho các phản hồi JSON cho các API RESTful
- JSend phân tách các câu trả lời thành một số loại cơ bản và xác định các khóa bắt buộc và tùy chọn cho từng loại: Success, Fail, Error

*https://github.com/omniti-labs/jsend

Định dạng trả lời theo đặc tả JSend*

- Ví du:
  - Thành công (2xx): yêu cầu được xử lý thành công

```
{
   "status" : "success",
   "data" : {
      "post" : { "id" : 1,
           "title" : "A blog post",
           "body" : "Some useful content" }}
}
```

```
{
   "status" : "success",
   "data" : null
}
```


*https://github.com/omniti-labs/jsend

Định dạng trả lời theo đặc tả JSend*

- Thất bại (4xx): yêu cầu không hợp lệ (lỗi client)

```
{
   "status" : "fail",
   "data" : { "title" : "A title is required" }
}
```

- Lỗi (5xx): lỗi xảy trong quá trình xử lý yêu cầu (lỗi server)

```
{
   "status" : "error",
   "message" : "Unable to communicate with database"
}
```


*https://github.com/omniti-labs/jsend

Làm việc với JSON trong JavaScript

- JS object → JSON string: *JSON.stringify(obj)*

```
let j={"name":"binchen"};
JSON.stringify(j); // '{"name":"binchen"}'
```

- JSON string → JS object: *JSON.parse(json)*

```
let json = '{"result":true,"count":1}';
let obj = JSON.parse(json);
```

Nhận dữ liệu json với express

```
const express = require("express");
const app = express();

app.use(express.json());

app.post("/api/contacts", (req, res) {
  // req.body: đối tượng JS ứng với JSON gửi về
  ...
}
...
```

Gửi dữ liệu json với express

```
const express = require("express");
const app = express();

app.use(express.json());

app.post("/api/contacts", (req, res) {
  ...
  res.json({ status : "success", data : null, });
}
...
```

Chứng thực người dùng HTTP API

- ✓ Xác định những gì người dùng có thể và không thể truy cập
- ✓ Xác minh xem có được phép truy cập thông qua các chính sách và quy tắc hay không (policies and rules)

- ✓ Thường được thực hiện sau khi authentication thành công
- ✓ Thường truyền thông tin qua Mã thông báo truy cập (Access Token)

- ✓ Xác định xem người dùng có phải là người mà họ tuyên bố hay không
- ✓ Yêu cầu người dùng xác thực thông tin đăng nhập (ví dụ: thông qua mật khẩu, câu trả lời cho câu hỏi bảo mật hoặc nhận dạng khuôn mặt ...)
- ✓ Thường được thực hiện trước authorization

- ✓ Thường, truyền thông tin qua Mã thông báo ID (ID Token)

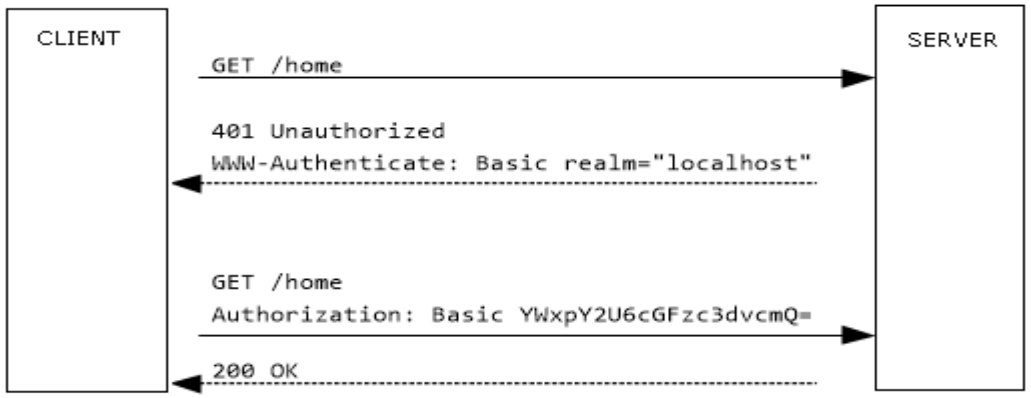
Chứng thực người dùng HTTP API

- 3 phương pháp xác thực phổ biến cho các API REST.
  - HTTP Basic Authentication
  - JWT (JSON Web Tokens)
  - OAuth 2.0 (Open Authorization)

Chứng thực người dùng HTTP API

- HTTP Basic Authentication
  - Cách xác thực người dung đơn giản nhất
  - Yêu cầu gửi thông tin đăng nhập như tên người dùng và mật khẩu dưới dạng username:password vào header
  - Được mã hóa bằng Base64 và được chuyển vào Authorization header như sau:

```
Authorization: Basic AKsdKfsdljOf1POs
```



Chứng thực người dùng HTTP API

- JWT (JSON Web Tokens)
  - Một tiêu chuẩn mở (RFC 7519), đại diện cho các xác nhận quyền sở hữu một cách an toàn giữa các bên dưới dạng đối tượng JSON
  - JWT cũng chuyển thông tin xác thực trong Authorization header
  - Thông tin xác thực là hình thức của mã thông báo (token) và nó có thể hết hạn

Chứng thực người dùng HTTP API

- JWT (JSON Web Tokens): https://jwt.io/introduction
- Khi nào nên sử dụng chứng thực JWT?
  - Authentication: JWT phù hợp với hầu hết các nhu cầu xác thực ứng dụng
    - Được sử dụng cho các ứng dụng di động, web hoặc phía máy chủ.
    - Hoạt động tốt với Express hoặc các ứng dụng có kiến trúc MVC
    - Đăng nhập một lần (Single Sign On) là một tính năng sử dụng rộng rãi JWT ngày nay vì chi phí thấp và dễ dàng sử dụng trên các domains khác nhau
    - Khi người dùng đã đăng nhập, mỗi request tiếp theo được gởi từ Client sẽ bao gồm JWT, cho phép người dùng truy cập các routes, dịch vụ và tài nguyên được phép với token đó

Chứng thực người dùng HTTP API

Chứng thực người dùng HTTP API

- JWT (JSON Web Tokens): https://jwt.io/introduction
  - Khi nào nên sử dụng chứng thực JWT?
    - Information Exchange (Trao đổi thông tin)
      - JWT một cách tốt để truyền thông tin giữa các bên một cách an toàn nhờ vào phần " signature " của nó
        - sử dụng cặp khóa công khai/riêng tư -> có thể chắc chắn rằng người gửi.
        - chữ ký (signature) được tạo ra bằng việc kết hợp cả phần header, payload lại nên thông qua đó ta có thể xác nhận được chữ ký có bị giả mạo hay không

Chứng thực người dùng HTTP API

- JWT (JSON Web Tokens): https://jwt.io/introduction
  - Cấu trúc JWT?
    - Bao gồm 3 phần, được ngăn cách nhau bởi dấu chấm (.)
      - Header
      - Payload
      - Signature
    - Một JWT có dạng tổng quát như sau: xxxxx.yyyyy.zzzzz

Chứng thực người dùng HTTP API

- JWT (JSON Web Tokens): https://jwt.io/introduction
  - Cấu trúc JWT?
    - Header: bao gồm hai phần
      - typ – Loại token (mặc định là JWT – cho biết đây là một Token JWT)
      - alg – Thuật toán đã dùng để mã hóa (HMAC SHA256 – HS256 hoặc RSA).
    - Payload: chứa các thông tin mình muốn đặt trong chuỗi Token như username , userId , author, …
      - Chứa tập hợp các claims
      - Claim là các xác nhận về một thực thể (thường là người dùng) và dữ liệu bổ sung.

```
{
  “alg”: “HS256”,
  “typ”: “JWT”
}
```

```
{
  “iss”: “techmaster”,
  “exp”: 1426420800,
  “https://www.techmaster.vn/jwt_claims/is_admin”: true,
  “user”: “paduvi”,
  “awesome”: true
}
```

Chứng thực người dùng HTTP API

- JWT (JSON Web Tokens): https://jwt.io/introduction
  - Cấu trúc JWT?
    - Signature
      - Được sử dụng để xác minh rằng người gửi
      - Để đảm bảo rằng thư không bị thay đổi trong quá trình truyền
      - Signature được tạo bằng cách kết hợp 2 phần Header + Payload, rồi mã hóa nó lại bằng 1 giải thuật được chỉ định trong header

```
$encodedContent = base64UrlEncode(header) + “.” + base64UrlEncode(payload);
$signature = hashHmacSHA256($encodedContent);
```

```
var header = {
  "alg": "HS256",
  "typ": "JWT"
};

var stringifiedHeader = CryptoJS.enc.Utf8.parse(JSON.stringify(header));
var encodedHeader = base64url(stringifiedHeader);

var data = {
  "iss": "techmaster",
  "exp": 1426420800,
  "https://www.techmaster.vn/jwt_claims/is_admin": true,
  "user": "paduvi",
  "awesome": true
};

var stringifiedData = CryptoJS.enc.Utf8.parse(JSON.stringify(data));
var encodedData = base64url(stringifiedData);

var token = encodedHeader + "." + encodedData;

var secret = "My very confidential secret!";

var signature = CryptoJS.HmacSHA256(token, secret);
signature = base64url(signature);

var signedToken = token + "." + signature;
```

Chứng thực người dùng HTTP API

- OAuth 2.0
  - Hoạt động bằng cách ủy quyền xác thực cho một máy chủ ủy quyền (tức là Facebook, Google, Github, v.v.) lưu trữ tài khoản người dùng
  - Sau đó, máy chủ tạo mã thông báo và gửi đến máy chủ tài nguyên (API) để cho phép người dùng truy cập các tuyến (route) được bảo vệ

Câu hỏi?