# MySql và Mongodb

Tài liệu:

<https://viblo.asia/p/so-sanh-mongodb-va-mysql-Qbq5QWWEZD8>

<https://www.w3schools.com/mongodb/?fbclid=IwAR2D-Rd6oy2c_pW9CIeu2Xd9JmdSpNW85CK4qNr8TMTgLIIw_2gC6ywN6uY>

Trong MySQL (hệ quản trị cơ sở dữ liệu quan hệ), **dữ liệu được tổ chức theo các bảng có cấu trúc**, với các hàng và cột cố định. **Mỗi dòng trong bảng có cùng số cột, và mỗi cột phải chứa kiểu dữ liệu nhất định**.

Ngược lại, MongoDB (hệ quản trị cơ sở dữ liệu không quan hệ) sử dụng mô hình dữ liệu không cấu trúc**, được biểu diễn dưới dạng tập tài liệu JSON/BSON**. MongoDB không yêu cầu một cấu trúc cố định cho mỗi bản ghi; thay vào đó, **mỗi tài liệu có thể chứa các trường khác nhau**.

MySQL (Cơ sở dữ liệu quan hệ):

| id | name | age | email

| 1 | John | 25 | john@email.com

| 2 | Alice | 30 | alice@email.com

MongoDB (Cơ sở dữ liệu không quan hệ):

[

{

"\_id": 1,

"name": "John",

"age": 25,

"email": [john@email.com](mailto:john@email.com),

“nationality”: [“Vietnamese”, “Chinese”],

“audio”: “abc”,

},

{

"\_id": 2,

"name": "Alice",

"age": 30,

"email": "alice@email.com"

}

]

* MongoDB thích hợp khi cần tính linh hoạt cao trong việc lưu trữ dữ liệu, đặc biệt là khi dữ liệu có cấu trúc không cố định hoặc thay đổi thường xuyên. MySQL thích hợp hơn cho các tình huống yêu cầu cấu trúc cố định và quan hệ giữa các bảng.

Note:

Từ "quan hệ" trong RDBMS nói về **cách dữ liệu được tổ chức và lưu trữ trong một bảng** cụ thể của cơ sở dữ liệu quan hệ. 1 Bảng sẽ chứa dữ liệu được tổ chức thành các hàng và cột. Các hàng trong bảng đại diện cho các bản ghi cụ thể và các cột đại diện cho các thuộc tính của bản ghi đó.

# BSON VÀ JSON

JSON là một định dạng dữ liệu phổ biến dùng để truyền tải và lưu trữ dữ liệu. Nó giống như cách chúng ta viết đối tượng trong JavaScrip

BSON tương tự như JSON, nhưng có thêm một số kiểu dữ liệu và đặc điểm tối ưu hóa để lưu trữ và truy xuất hiệu quả hơn.

* Chứa dữ liệu dưới dạng các byte
* BSON hỗ trợ một số kiểu dữ liệu mà JSON không hỗ trợ, bao gồm nhưng không giới hạn các kiểu dữ liệu như Date, Binary data, ObjectId
* BSON được thiết kế để mã hóa dữ liệu một cách hiệu quả, giúp tối ưu hóa kích thước dữ liệu và cải thiện hiệu suất truy xuất.

Ví dụ

{

"name": "John",

"age": 25,

"city": "New York",

"isStudent": false,

"hobbies": ["reading", "coding", "gaming"]

}

BSON cũng là một định dạng dữ liệu, nhưng nó được biểu diễn dưới dạng nhị phân (binary)

Ví dụ

\x16\x00\x00\x00 // Độ dài của mảng

\x02 // Kiểu dữ liệu: String

name\x00 // Tên trường: "name"

\x04\x00\x00\x00John\x00 // Giá trị: "John"

\x10\x00\x00\x00 // Kiểu dữ liệu: Int32

age\x00 // Tên trường: "age"

\x19\x00\x00\x00 // Giá trị: 25

\x00 // Kiểu dữ liệu: EOO (End of Object)

# PyMongo

Pip install pymongo

Tài liệu:

<https://pymongo.readthedocs.io/en/stable/atlas.html>

Pymongo là một thư viện Python dùng để tương tác với MongoDB, một hệ quản trị cơ sở dữ liệu (DBMS) phi quan hệ, không dùng ngôn ngữ truy vấn SQL. MongoDB lưu trữ dữ liệu dưới dạng BSON (Binary JSON)

Pymongo cho phép bạn thực hiện các thao tác CRUD (Create, Read, Update, Delete) và các thao tác khác trực tiếp trên cơ sở dữ liệu MongoDB.

**Tại sao dùng Pymongo**

Tương thích với MongoDB: Pymongo được phát triển và duy trì chính bởi đội ngũ MongoDB, đảm bảo tính tương thích và hỗ trợ tốt khi bạn làm việc với cơ sở dữ liệu MongoDB.

Đồng bộ với Django ORM: Bạn có thể sử dụng pymongo để thực hiện các truy vấn MongoDB, trong khi vẫn giữ được sự linh hoạt của Django ORM (Object-Relational Mapping) cho cơ sở dữ liệu quan hệ.

* Django ORM (Object-Relational Mapping) giúp ta tương tác với cơ sở dữ liệu một cách dễ dàng và thuận tiện, bằng cách sử dụng đối tượng và các hàm thay cho các câu lệnh SQL phức tạp và dài dòng

Ví Dụ:

Django ORM: Book.objects.filter(isbn=isbn).exists()

SQL: SELECT EXISTS (SELECT 1 FROM book WHERE isbn = 'isbn');

**Các phương thức phổ biến của PyMongo thường được sử dụng để tương tác với cơ sở dữ liệu MongoDB.**

**insert\_one():** Chèn một tài liệu vào một bộ sưu tập.

# Ví dụ: Chèn một sách mới vào bộ sưu tập "books"

new\_book = {

"title": "Python Programming",

"author": "John Smith",

"year": 2022

}

result = db.books.insert\_one(new\_book)

**insert\_many():** Chèn nhiều tài liệu vào một bộ sưu tập.

# Ví dụ: Chèn nhiều sách mới vào bộ sưu tập "books"

new\_books = [

{"title": "Machine Learning", "author": "Jane Doe", "year": 2021},

{"title": "Data Science", "author": "Alice Johnson", "year": 2020}

]

result = db.books.insert\_many(new\_books)

**find\_one():** Tìm và trả về một tài liệu đầu tiên trong bộ sưu tập.

# Ví dụ: Tìm sách đầu tiên có tên là "Python Programming"

book = db.books.find\_one({"title": "Python Programming"})

**find():** Tìm và trả về tất cả các tài liệu trong bộ sưu tập.

# Ví dụ: Tìm tất cả các sách có năm xuất bản là 2022

books = db.books.find({"year": 2022})

for book in books:

print(book)

**update\_one():** Cập nhật một tài liệu đầu tiên trong bộ sưu tập.

# Ví dụ: Cập nhật năm xuất bản của sách đầu tiên có tên là "Python Programming"

result = db.books.update\_one({"title": "Python Programming"}, {"$set": {"year": 2023}})

**update\_many():** Cập nhật nhiều tài liệu trong bộ sưu tập.

# Ví dụ: Cập nhật năm xuất bản của tất cả các sách có tác giả là "John Smith"

result = db.books.update\_many({"author": "John Smith"}, {"$set": {"year": 2023}})

**delete\_one():** Xóa một tài liệu đầu tiên khớp với điều kiện.

# Ví dụ: Xóa sách đầu tiên có năm xuất bản là 2022

result = db.books.delete\_one({"year": 2022})

**delete\_many():** Xóa nhiều tài liệu khớp với điều kiện.

# Ví dụ: Xóa tất cả các sách có năm xuất bản nhỏ hơn 2020

result = db.books.delete\_many({"year": {"$lt": 2020}})

# API

## API là gì

API là viết tắt của "Application Programming Interface" (Giao diện Lập trình Ứng dụng). API là một trung gian phần mềm cho phép hai ứng dụng giao tiếp với nhau thông qua các quy tắc và giao thức đã định sẵn.

Giao tiếp trong ngữ cảnh API được hiểu là việc truyền và nhận dữ liệu, chia sẻ chức năng, hoặc thực hiện các hành động cụ thể.

Một số quy tắc và giao thức quan trọng:

1. HTTP (Hypertext Transfer Protocol):

Mô Tả: HTTP là giao thức truyền tải thông tin giữa máy chủ và người dùng trên web. Nó được sử dụng rộng rãi trong việc xây dựng API RESTful.

Quy Tắc Liên Quan: RESTful APIs thường sử dụng phương thức HTTP (GET, POST, PUT, DELETE) để thực hiện các thao tác CRUD.

2. JSON (JavaScript Object Notation):

Mô Tả: JSON là định dạng dữ liệu nhẹ và dễ đọc, thường được sử dụng để truyền tải dữ liệu giữa ứng dụng và máy chủ.

Quy Tắc Liên Quan: Nhiều API sử dụng JSON để định dạng dữ liệu trong các yêu cầu và phản hồi.

3. TCP/IP (Transmission Control Protocol/Internet Protocol):

Mô Tả: TCP/IP là bộ giao thức mạng cơ bản của Internet, đảm bảo giao tiếp giữa các thiết bị khác nhau trên mạng.

Quy Tắc Liên Quan: API sử dụng TCP/IP để thiết lập kết nối và truyền tải dữ liệu giữa client và server.

## API làm việc như thế nào?

API được xây dựng trên chính 2 thành phần: Request và Reponse

**Về request(phía client)**

Một cái request đúng chuẩn cần có 4 thứ:

1. URL

URL là địa chỉ duy nhất cho 1 request, thường là đường dẫn tới một hàm xử lí logic.

1. Method

HTTP request có tất cả 9 loại method , 2 loại được sử dụng phổ biến nhất là GET và POST

GET (dữ liệu dạng công khai): Phương thức này được sử dụng để yêu cầu dữ liệu từ server. Dữ liệu được chuyển dưới dạng query parameters(tham số truy vấn) trong URL.

POST(dữ liệu dạng riêng tư): Gửi thông tin tới sever thông qua các parameters HTTP. Các parameters HTTP được chứa trong phần body của yêu cầu HTTP

1. Headers

Là nơi chứa các thông tin cần thiết của 1 request nhưng end-users, độ dài của request body, thời gian gửi request, loại thiết bị đang sử dụng, loại định dạng cái response mà client có đọc được

1. Body

Là nơi chứa thông tin mà client sẽ điền.

**Về response(phía server)**

Sau khi nhận được request từ phía client, server sẽ xử lý cái request đó và gửi ngược lại cho client 1 cái response. Cấu trúc của 1 response gồm 3 phần:

* Status code
* Headers
* Body

**Phần Header và body tương đối giống với request.**

**Status code của response**

1xx: Information (Thông tin): Khi nhận được những mã như vậy tức là request đã được server tiếp nhận và quá trình xử lý request đang được tiếp tục.

2xx: Success (Thành công): Khi nhận được những mã như vậy tức là request đã được server tiếp nhận, hiểu và xử lý thành công

3xx: Redirection (Chuyển hướng): Mã trạng thái này cho biết client cần có thêm action để hoàn thành request

4xx: Client Error (Lỗi Client): Nó nghĩa là request chứa cú pháp không chính xác hoặc không được thực hiện.

5xx: Server Error (Lỗi Server): Nó nghĩa là Server thất bại với việc thực hiện một request nhìn như có vẻ khả thi.

Link: <https://viblo.asia/p/tim-hieu-kien-thuc-co-ban-ve-api-maGK7A4Mlj2>

# API Django

## Requests trong Django(nhận các yêu cầu HTTP):

Trong Django, "requests" thường đề cập đến quá trình gửi yêu cầu HTTP(HyperText Transfer Protocol: giao thức truyền tải siêu văn bản) từ máy khách đến máy chủ Django và nhận phản hồi từ máy chủ.

**Cách nó hoạt động?**

1. Người Dùng Tương Tác với Ứng Dụng Django:

Người dùng tương tác với ứng dụng Django bằng cách gửi yêu cầu, ví dụ: khi họ truy cập trang web.

2.Yêu Cầu Được Gửi Đến Máy Chủ Django:

Yêu cầu HTTP, method GET hoặc POST, được gửi từ trình duyệt của người dùng đến máy chủ Django.

3. Xử Lý Yêu Cầu Trong Django:

Django xử lý yêu cầu, thực hiện các chức năng như truy vấn cơ sở dữ liệu, xử lý biểu mẫu và nhiều tác vụ khác.

* Nhận Request
* Xử lý URL và xác định view tương ứng
* Gọi view
* Xử lý logic trong view
* Tạo HttpResponse

4. Phản Hồi Được Gửi Về Trình Duyệt:

Máy chủ gửi phản hồi HTTP, chẳng hạn như trang HTML được tạo ra, trở lại trình duyệt của người dùng.

## Một số phương thức quan trọng của đối tượng HttpRequest trong Django:

1. request.GET: Trả về một đối tượng QueryDict chứa tất cả các tham số trên URL (dữ liệu từ phương thức GET).

Ví Dụ:

# URL: /my\_view/?name=John&age=25

name = request.GET.get('name') # Kết quả: 'John'

age = request.GET.get('age') # Kết quả: '25'

2. request.POST: Trả về một đối tượng QueryDict chứa dữ liệu từ phương thức POST (dữ liệu form).

Ví Dụ:

# Form data: name=John&age=25

name = request.POST.get('name') # Kết quả: 'John'

age = request.POST.get('age') # Kết quả: '25'

3. request.method: Xác định phương thức HTTP của yêu cầu (GET, POST, PUT, DELETE, vv.), trả về dữ liệu dạng str.

Ví Dụ:

method = request.method # Kết quả: 'GET' hoặc 'POST'

4. request.path: Trả về đường dẫn của URL được yêu cầu mà không bao gồm domain.

Ví Dụ:

# URL: "https://thaihabooks.com/tu-sach/"

path = request.path # Kết quả: '/tu-sach/'

Note: Domain bao gồm tên miền cấp cao nhất (top-level domain - TLD) và tên miền cụ thể của trang web đó.

Ví dụ, trong URL "https://thaihabooks.com/tu-sach/", phần " thaihabooks.com " là domain.

5. request.FILES: Trả về một đối tượng MultiValueDict chứa dữ liệu file được gửi lên (nếu có).

Ví Dụ:

uploaded\_file = request.FILES.get('file\_field\_name')

6. request.headers: Trả về tất cả các tiêu đề HTTP từ yêu cầu

Ví Dụ:

request.headers # Kết quả: {'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_12\_6'}

## Response trong Django(gửi các yêu cầu HTTP)

HttpResponse là đối tượng đại diện cho phản hồi mà máy chủ gửi về máy khách.

Một số phương thức quan trọng của đối tượng HttpResponse trong Django:

1. HttpResponse(content='', content\_type=None, status=200):

Khởi tạo một đối tượng HttpResponse.

Kiểu Dữ Liệu Trả Về: HttpResponse object.

2. HttpResponseRedirect(redirect\_to):

Tạo một phản hồi chuyển hướng (HTTP 302).

Kiểu Dữ Liệu Trả Về: HttpResponseRedirect object.

Ví Dụ:

from django.http import HttpResponseRedirect

def my\_view(request):

# Chuyển hướng đến trang '/my/redirect/target/'

return HttpResponseRedirect('/my/redirect/target/)

3. JsonResponse(data, encoder=DjangoJSONEncoder, safe=True, json\_dumps\_params=None, \*\*kwargs):

Tạo một phản hồi JSON với dữ liệu đã cho.

Kiểu Dữ Liệu Trả Về: JsonResponse object.

Ví Dụ:

from django.http import JsonResponse

def my\_json\_view(request):

data = {'key': 'value'}

return JsonResponse(data)

4. FileResponse(file, as\_attachment=False, filename=None):

Tạo một phản hồi cho file (HTTP 200).

Kiểu Dữ Liệu Trả Về: FileResponse object.

Ví Dụ:

from django.http import FileResponse

from django.core.files import File

def my\_file\_view(request):

file\_path = '/path/to/my/file.txt'

my\_file = File(open(file\_path, 'rb'))

return FileResponse(my\_file, as\_attachment=True, filename='download.txt')

#### Chi Tiết Quá Trình Xử Lý Request Trong Django:

Khi máy chủ Django nhận được một yêu cầu từ máy khách, quá trình xử lý diễn ra theo các bước sau:

1. \*\*Nhận Request:\*\*

- Máy chủ Django lắng nghe các yêu cầu được gửi đến qua mạng. Khi một yêu cầu đến, máy chủ xử lý nó để trích xuất thông tin như phương thức (GET, POST), URL, dữ liệu truyền kèm, và các thông tin khác từ yêu cầu HTTP.

2. \*\*Phân Đoạn URL (URL Dispatching):\*\*

- Django sử dụng hệ thống định tuyến (URL routing) để phân đoạn URL và xác định view (chức năng xử lý yêu cầu) tương ứng.

3. \*\*Gọi View (Xử Lý Yêu Cầu):\*\*

- View là một phần quan trọng trong Django, nó là nơi xử lý logic của ứng dụng. View nhận thông tin từ yêu cầu (như dữ liệu form, tham số trên URL), thực hiện các tác vụ như truy vấn cơ sở dữ liệu, xử lý logic kinh doanh, và trả về một đối tượng HttpResponse.

4. \*\*Xử Lý Logic Trong View:\*\*

- Trong view, các tác vụ được thực hiện dựa trên yêu cầu. Điều này có thể bao gồm:

- Truy vấn cơ sở dữ liệu để lấy hoặc cập nhật thông tin.

- Xử lý dữ liệu đầu vào từ người dùng (nếu có).

- Thực hiện logic kinh doanh như tính toán, xử lý hình ảnh, hoặc các tác vụ khác.

5. \*\*Tạo HttpResponse:\*\*

- Kết quả từ view là một đối tượng HttpResponse, chứa dữ liệu và thông tin cần trả về cho máy khách. HttpResponse có thể chứa HTML để hiển thị trên trình duyệt, dữ liệu JSON cho ứng dụng JavaScript, hoặc bất kỳ loại dữ liệu nào khác.

6. \*\*Trả Về Response:\*\*

- Máy chủ gửi đối tượng HttpResponse này như là phản hồi cho yêu cầu gốc. Dữ liệu trong response được chuyển từ máy chủ đến máy khách qua mạng.

#### Lưu Trữ Dữ Liệu Trong Response:

Dữ liệu trong đối tượng HttpResponse của Django không được lưu trữ trên máy chủ sau khi nó đã được gửi đi. Django hoạt động theo kiểu "request-response" và không duy trì trạng thái giữa các yêu cầu khác nhau từ cùng một người dùng.

Ví dụ, nếu bạn gửi một trang web cho người dùng, các thông tin về trang web đó (như HTML, CSS, JavaScript) sẽ được gửi đến trình duyệt của họ. Các yêu cầu sau đó không giữ lại dữ liệu này trên máy chủ.

Dữ liệu từ yêu cầu đến và phản hồi đi được truyền qua mạng và không được duy trì trên máy chủ trung tâm. Các ứng dụng thường sử dụng cơ sở dữ liệu hoặc lưu trữ tạm thời để duy trì thông tin giữa các yêu cầu khác nhau từ cùng một người dùng.

# CSRF

CSRF là gì?

CSRF là một kiểu tấn công mà kẻ tấn công gửi các yêu cầu từ một trang web mà người dùng đã đăng nhập. Điều này có thể dẫn đến việc thực hiện các hành động không mong muốn mà người dùng không biết.

Bảo vệ CSRF trong Django:

Django cung cấp một lớp bảo vệ chống CSRF mặc định bằng cách sử dụng token CSRF, và nó được kích hoạt cho tất cả các yêu cầu POST mặc định.

@csrf\_exempt là gì và tại sao sử dụng nó:

Khi bạn đánh dấu một view với @csrf\_exempt, bạn đang nói với Django rằng không áp dụng bảo vệ CSRF cho view đó. Điều này có thể hữu ích trong một số trường hợp đặc biệt, như khi bạn cần chấp nhận các yêu cầu từ các nguồn không tin cậy và không muốn quá trình xác thực CSRF.

Khi nào nên sử dụng @csrf\_exempt:

Bạn chắc chắn rằng việc bỏ qua bảo vệ CSRF không gây ra vấn đề bảo mật trong trường hợp cụ thể.

Bạn có yêu cầu nhất định từ một nguồn không tin cậy mà bạn muốn chấp nhận mà không cần xác thực CSRF.

Các bước cơ bản của quá trình xác thực CSRF:

Tạo Token CSRF:

Khi người dùng truy cập một trang web, máy chủ sẽ tạo ra một token CSRF duy nhất và gửi nó đến trình duyệt của người dùng.

Lưu Token CSRF:

Token CSRF này thường được lưu trữ trong một cookie trên trình duyệt của người dùng hoặc trong các trường dữ liệu ẩn trong các biểu mẫu web.

Gửi Token CSRF với Mỗi Yêu Cầu:

Khi người dùng thực hiện một hành động gửi yêu cầu (ví dụ: bấm nút "Gửi" trên một biểu mẫu), token CSRF sẽ được gửi kèm theo yêu cầu. Điều này có thể được thực hiện thông qua cookie hoặc các trường dữ liệu ẩn.

So Sánh Token CSRF:

Khi yêu cầu đến máy chủ, máy chủ sẽ kiểm tra token CSRF được gửi kèm với yêu cầu và so sánh nó với token được lưu trữ. Nếu chúng khớp, yêu cầu được coi là hợp lệ và tiếp tục xử lý.

Không Khớp Token CSRF:

Nếu token CSRF không khớp hoặc không tồn tại, máy chủ có thể từ chối yêu cầu, giả mạo không được xác thực, và bảo vệ người dùng khỏi tấn công CSRF.

# Django ORM

Django ORM cung cấp nhiều phương thức để thực hiện các thao tác tương tác với cơ sở dữ liệu.

**filter():** Lấy các bản ghi thoả mãn điều kiện được chỉ định.

# Ví dụ: Lấy tất cả sách có tác giả là 'John Doe'

books = Book.objects.filter(author='John Doe')

**get():** Lấy một bản ghi thoả mãn điều kiện. Nếu không có bản ghi hoặc có nhiều hơn một bản ghi, sẽ raise exception.

# Ví dụ: Lấy sách có ISBN là '123456789'

book = Book.objects.get(isbn='123456789')

**exclude():** Lấy các bản ghi không thoả mãn điều kiện.

# Ví dụ: Lấy tất cả sách không phải của tác giả 'Jane Doe'

books = Book.objects.exclude(author='Jane Doe')

order\_by(): Sắp xếp các bản ghi theo một hoặc nhiều trường.

# Ví dụ: Lấy tất cả sách và sắp xếp theo ngày xuất bản giảm dần

books = Book.objects.all().order\_by('-publication\_date')

**count():** Đếm số lượng bản ghi thoả mãn điều kiện.

# Ví dụ: Đếm số lượng sách có tác giả là 'John Doe'

count = Book.objects.filter(author='John Doe').count()

**create():** Tạo một bản ghi mới và lưu vào cơ sở dữ liệu.

# Ví dụ: Tạo một sách mới

new\_book = Book.objects.create(title='New Book', author='John Doe', publication\_date='2022-02-01', isbn='987654321')

**update():** Cập nhật các bản ghi thoả mãn điều kiện.

# Ví dụ: Cập nhật ngày xuất bản của tất cả sách của tác giả 'John Doe'

Book.objects.filter(author='John Doe').update(publication\_date='2022-03-01')

**delete():** Xóa các bản ghi thoả mãn điều kiện.

# Ví dụ: Xóa tất cả sách của tác giả 'Jane Doe'

Book.objects.filter(author='Jane Doe').delete()

# Django

<https://aws.amazon.com/vi/what-is/django/>

# JWT(Json web token): dữ liệu được mã hóa thành chuỗi và được gửi đi Dưới dạng json trên Web

## Các kịch bản áp dụng để triển khai JWT

Có một số tình huống mà JWT hữu ích:

● Trong quá trình ủy quyền, chúng ta có thể sử dụng JWT. Ngay khi người dùng đăng nhập vào máy chủ với sự trợ giúp của mã thông báo, nó có thể truy cập tất cả các tài nguyên thông qua mã thông báo như dịch vụ, tài nguyên và tuyến đường. Ngày nay, ở nhiều trang web, chỉ cần xác thực một lần, sau đó với sự trợ giúp của mã thông báo, họ không cần phải viết lại mật khẩu.

● Khi chúng tôi cần trao đổi thông tin giữa máy khách và máy chủ, điều đó giúp chúng tôi duy trì kênh liên lạc an toàn. Nó sử dụng mã thông báo đã ký để liên lạc, có thể là sự kết hợp giữa khóa chung và khóa riêng. Nó cũng có các phần bổ sung được sử dụng để tính toán tiêu đề chữ ký và tải trọng.

## Cấu trúc của mã thông báo JWT

JWT có ba phần và tất cả đều được phân tách bằng dấu chấm (.), như sau:

● Tiêu đề

● Tải trọng

● Chữ ký

Nó trông giống như được đề cập bên dưới, được phân biệt bằng dấu chấm (.):

aaaaaaa.bbbbbbb.ccccccc

Trên đây là ví dụ về JWT. Trong ví dụ aaaaaaa phần là tiêu đề, bbbbbbb là phần tải trọng và ccccccc là phần chữ ký của JWT.

### Tiêu đề

Gồm 2 phần là loại mã là JWT và thông tin về thuật toán mã hóa

### Tải trọng

Chứa thông tin về xác nhận quyền sở hữu của người dùng và dữ liệu bổ sung đã được mã hóa

### Chữ ký

Đây là phần thứ ba của JWT. Nó được sử dụng để xác thực mã thông báo. Nó kiểm tra xem mã thông báo có đáng tin cậy và không bị sửa đổi giữa các lần giao tiếp hay không.

Chữ ký là sự kết hợp giữa tiêu đề được mã hóa, tải trọng được mã hóa, bí mật và thuật toán được chỉ định của tiêu đề.

Ví dụ: Nếu chúng ta đang sử dụng thuật toán HMAC-SHA256 trong JWT header thì chữ ký sẽ như sau:

HMACSHA256(

base64UrlEncode(tiêu đề) + "." + base64UrlEncode(tải trọng), bí mật)

Nó được sử dụng để kiểm tra xem tin nhắn có bị thay đổi giữa các lần giao tiếp hay không.

Dưới đây là một ví dụ cụ thể:

Tài liệu hướng dẫn sử dụng(header):loại hàng, tên loại mã.

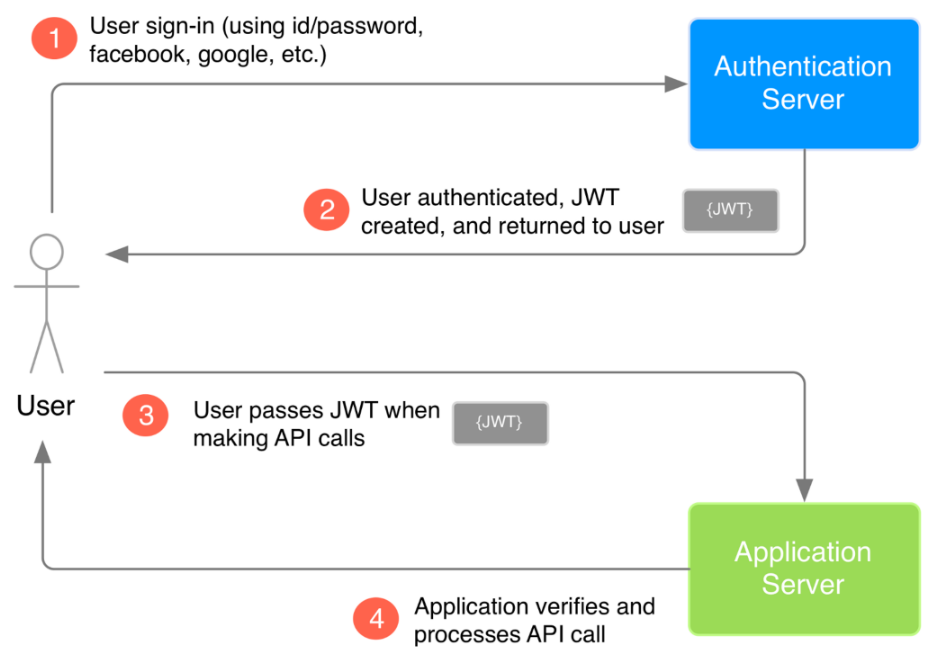
Gói hàng (Payload): Thông tin trong gói hàng bao gồm:

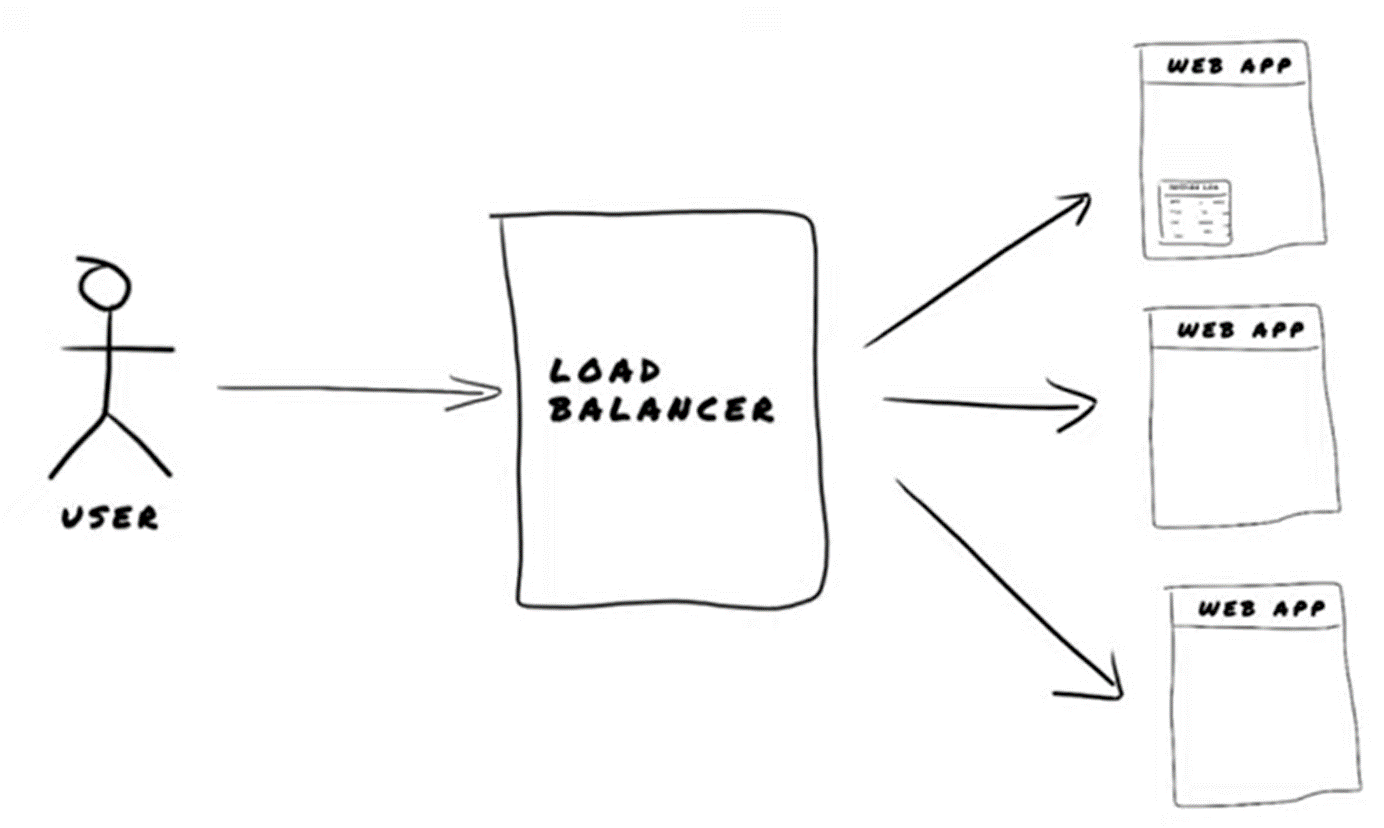
* Người gửi (Bạn)
* Người nhận (Bạn bè của bạn)
* Nội dung gói hàng (Sản phẩm được gửi)
* Thời gian gửi (Ngày và giờ)

Chữ ký (Signature)(mã tem -> đọc mã tem để biết gói hàng này có những cái gì): Được tạo ra bằng cách sử dụng một mã bí mật (secret key), chữ ký này sẽ được gửi cùng với gói hàng.

Khi bạn bè của bạn nhận được gói hàng, họ có thể kiểm tra chữ ký (signature) để đảm bảo rằng gói hàng không bị thay đổi và rằng nó thực sự được gửi từ bạn.

### Cách hoạt động







### Ưu điểm:

1. Dùng cho kiến trúc microservice
2. Giảm tải cho server khi thông tin được lưu trữ ở phía người dùng
3. Bảo mật tài nguyên hệ thống và dữ liệu gủi nhân khỏi những truy cập trái phép