Hypertext Transfer Protocol

Request

เมื่อเปิด web browser ขึ้นมา พิมพ์ web ที่ต้องการ เช่น icode.run กด enter ตัว browser จะหา IP address ของ web

```
Name:    icode.run
Address: 128.199.119.79
```

จากนั้น browser จะเชื่อมต่อไปที่ port หรือ ช่อง ที่ 80 และส่งข้อความไปตามรูปแบบการสื่อสาร หรือ protocol

```
GET / HTTP/1.1
Host: icode.run
```

- GET คือ method ในการขอข้อมูล
- / คือ root เป็น path หรือ ชื่อ file ที่ต้องการ
- HTTP/1.1 คือ version ของ protocol
- Host: เพื่อบอกว่าต้องการ web ไหน เพราะ server สามารถเก็บได้หลาย web ถ้าไม่มีบรรทัดนี้อาจจะได้ web อื่นมาแทน

Response

จากนั้น web server จะส่งข้อความกลับมา เช่น

```
HTTP/1.1 200 OK
Content-Type: text/html

<!DOCTYPE html>
<html>
<head><title>Welcome</title></head>
<body><i>Welcome to iCode.run</i></body>
</html>
```

- 200 คือ status code ที่บอกว่า request สำเร็จ
- OK เป็น response message
- Content-Type: เพื่อบอกว่าข้อมูลที่ตามมาจะเป็นอะไร
- บรรทัดว่างเพื่อแยกระหว่าง header และ ข้อมูล
- ที่เหลือเป็นข้อมูล ในกรณีนี้คือ HTML นั่นเอง

ถ้าพิมพ์ airbnb.com/hello.jpg แล้วกด enter จะเกิด request

```
GET /hello.jpg HTTP/1.1
Host: airbnb.com
```

ถ้า server หา file ชื่อ hello.jpg ไม่พบ จะส่ง response ออกมา

```
HTTP/1.1 404 Not Found
...
```

Form

สมมุติว่า form นี้อยู่ที่ path ชื่อ /vat มีช่องใส่ข้อมูลและปุ่ม OK

```
<form>
    <input name="price">
    <input type="submit" value="OK">
</form>
```

เมื่อใส่ข้อมูล 499 เข้าไปแล้วกด enter ตัว browser จะเรียกไปที่ file เดิมหรือ path เดิม แต่เพิ่มข้อมูลต่อท้ายด้วยคือ /vat?price=499 ตัว browser สร้าง request คือ

```
GET /vat?price=499 HTTP/1.1
Host: icode.run
```

ถ้าต้องการให้เรียก path อื่น ต้องใส่ action เข้าไป

```
<form action="/result">
...
```

สำหรับข้อมูลที่สำคัญ เช่น รหัสผ่าน หรือ เลขบัตรเครดิต ต้องส่งผ่าน method post และเข้ารหัสผ่าน HTTPS

```
<form method="post">
    <input name="user">
    <input name="pass" type="password">
    <input type="submit" value="OK">
</form>
```

เมื่อใส่ข้อมูลแล้วกด enter ตัว browser จะสร้าง request คือ

```
POST /login HTTP/1.1
Host: icode.run
user=user@email.com&pass=secret
```

request แทบจะไม่แตกต่างกัน แต่ข้อดีของ POST คือ ไม่ถูก cache, ไม่เก็บไว้ใน history และ ไม่มีข้อจำกัดเรื่องขนาดข้อมูล

ถ้าต้องการส่ง file ไปที่ server ต้องใช้ POST และมี enctype="multipart/form-data"

```
<form method="post"
    enctype="multipart/form-data">
    <input type="file" name="photo">
    <input type="submit" value="OK">
</form>
```

Request Method

นอกจาก GET และ POST แล้ว method ยังมีอีกหลายอย่าง เช่น PUT, DELETE, PATCH, HEAD, TRACE, OPTIONS, CONNECT หรือจะสร้าง method ขึ้นมาเองก็ได้

ตัวอย่างการใช้กับ CRUD (Create, Read, Update, Delete)

- POST มักใช้สร้างข้อมูล หรือ Create
- GET มักใช้อ่านข้อมูล หรือ Read
- PUT มักใช้แก้ไขข้อมูล หรือ Update
- DELETE มักใช้ลบข้อมูล หรือ Delete

Header

Request header ถูกส่งจาก browser ไป server เช่น

- Host: บอกว่าต้องการ web ไหน เช่น

Host: icode.run

- User-Agent: ชื่อ browser และ ระบบ เช่น

User-Agent: Mozilla/5.0 (Intel Mac OS X)

Response header ถูกส่งจาก server ไป browser เช่น

- Content-Type: ชนิดของข้อมูล เช่น

Content-Type: text/html; charset=UTF-8

- Access-Control-Allow-Origin: ถ้าต้องการสร้าง web service ให้ทุก web เรียกข้อมูลได้ ต้องใส่บรรทัดนี้ไว้ใน header ด้วย เช่น

```
HTTP/1.1 200 OK
Content-Type: text/html
Access-Control-Allow-Origin: *
```

Status Code

status code ที่ควรรู้ เช่น

- 200 Request Success
- 301 Moved Permanently
- 403 Forbidden
- 404 Not Found
- 500 Internal Server Error

XML & JSON

ตัวอย่างข้อมูลแบบ XML หรือ eXensible Markup Language

```
<?xml version="1.0" encoding="UTF-8"?>
<data>
  <product>
    <name>Latte</name><price>80</price>
  </product>
  <product>
    <name>Mocha</name><price>90</price>
  </product>
</data>
```

ตัวอย่างข้อมูลแบบ JSON หรือ JavaScript Object Notation

```
[ {"name": "Latte", "price": 80},
  {"name": "Mocha", "price": 90} ]
```

Cookie

cookie คือข้อมูลที่ server สั่งให้ browser เก็บไว้ แล้วส่งกลับมาใน request ครั้งต่อไป ถ้า server ต้องการสั่งให้เก็บข้อมูลว่า I love you แล้วส่งกลับมา ทาง server ต้องส่ง header นี้ไปด้วย

```
HTTP/1.1 200 OK
Content-Type: text/html
Set-Cookie: I love you

<!DOCTYPE html>
...
```

ใน request ครั้งต่อไป browser จะส่งข้อความ I love you กลับมาด้วย เช่น

```
GET / HTTP/1.1
Host: icode.run
Cookie: I love you
```

ตัว server อาจจะส่ง Set-Cookie ออกมาหลายครั้ง เพื่อทำงานแตกต่างกัน ดังนั้นเมื่อ browser ส่งกลับมาจะถูกรวมกันมาเป็นบรรทัดเดียวกัน และแยกกันด้วย ; หรือ semicolon

```
HTTP/1.1 200 OK
Content-Type: text/html
Set-Cookie: I love you
Set-Cookie: token=1234567890
```

เมื่อ browser ส่งกลับจะรวมกันเป็นบรรทัดเดียว แยกกันด้วย ;

```
GET / HTTP/1.1
Host: icode.run
Cookie: I love you; token=1234567890
```

ถ้าต้องการให้ cookie มีความปลอดภัยมากขึ้น ใช้ Secure ต่อท้าย และ ส่งผ่าน HTTPS เช่น

```
HTTP/1.1 200 OK
Content-Type: text/html
Set-Cookie: token=1234567890; Secure
```

ถ้าต้องการปิดการทำงานของ AJAX ใช้ HttpOnly ต่อท้าย

```
HTTP/1.1 200 OK
Content-Type: text/html
Set-Cookie: token=1234567890; HttpOnly
```

ทั้ง Secure และ HttpOnly สามารถใช้ร่วมกันได้ อย่าลืมใส่ ; เพื่อแยกข้อมูลออกจากกัน เช่น

```
HTTP/1.1 200 OK
Content-Type: text/html
Set-Cookie: token=1234; Secure; HttpOnly
```