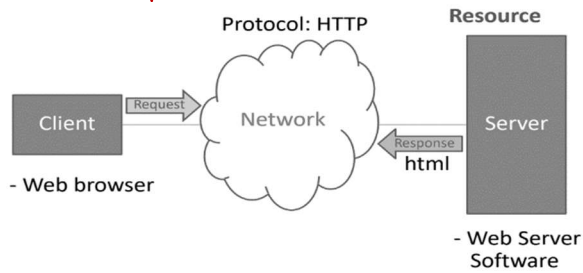


Web Concept



Hypertext Transfer Protocol

Request – Reply protocol (RR)

HTTP Resources are identify by URI (or URL)

2 types of message

- Request message

Request Line	General header	Response header	Blank line	Message body
--------------	----------------	-----------------	------------	--------------

- Response message

Status line	General header	Response header	Blank line	Message body
-------------	----------------	-----------------	------------	--------------

Example of HTTP Exchange

Request message:

GET /index.html HTTP/1.1 *Request line*
 Host: www.example.com *header line*
 [Blank Line]

Response message:

HTTP/1.1 200 OK *Status line*
 Date: Fri, 31 Dec 1999 23:59:59 GMT *Header lines*
 Content-Type: text/html *Header lines*
 Content-Length: 1354 *Header lines*
 [Blank Line]
 <html>
 ... *Body*

HTTP Protocol

2 common methods

- Get
- Post

HTTP GET Method

URL: <http://www.kmitl.ac.th/page.html>

- Requesting resource

Method Resource Protocol Version

GET/page.htmlHTTP/1.1
 Host: www.kmitl.ac.th
 User-Agent: Mozilla/5.0
 ...
 ...
 ...

Header

URL: <http://www.kmitl.ac.th/q.php?id=123&name=John>

Query string

Method Resource Protocol Version
GET/q.php?id=123&name=JohnHTTP/1.1
 Host: www.kmitl.ac.th
 User-Agent: Mozilla/5.0
 ...
 ...
 ...

Header

HTTP POST Method

```
<form action="kmitl/inputtest.php" method="post">
Username: <input type="text" name="uname" value=""
size="20" maxlength="20"> <br>
Password: <input type="password" name="pwd"
value="" size="20" maxlength="20">
<input type="submit" value="Submit">
</form>
```

Method Resource Protocol Version
POST/inputtest.phpHTTP/1.1
 Host: www.kmitl.ac.th
 User-Agent: Mozilla/5.0
 ...
 ...
 ...

Header

Empty line

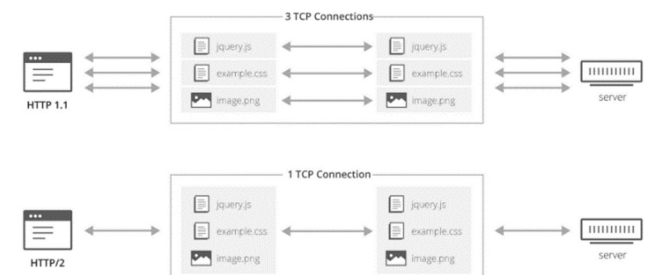
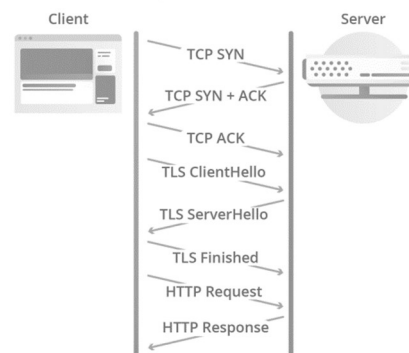
uname=john&pwd=1234 Body

Get VS Post

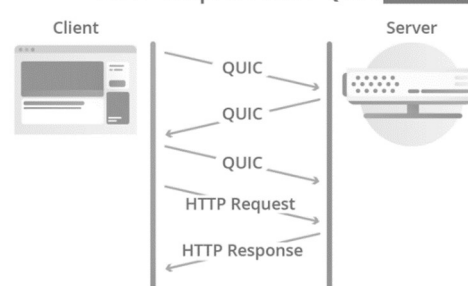
	Get	Post
Data length to be sent	~4000	unlimited
Can send a chunk of data	No	Yes
Location of data in message	URL	Body
Can access CGI without using Form	Yes	No
Can retrieve file or other resource	Yes	No

How does it work?

HTTP Request Over TCP + TLS



HTTP Request Over QUIC



WWW คืออะไร

WWW ย่อมาจาก World Wide Web คือ เครือข่ายที่เชื่อมต่อกันทั่วโลก เรามักเรียกย่อๆกันว่า เว็บ คือรูปแบบหนึ่งของระบบการเชื่อมโยงเครือข่ายข่าวสาร ใช้ในการค้นหาข้อมูลข่าวสารบน Internet จากแหล่งข้อมูลหนึ่ง ไปยังแหล่ง ข้อมูลที่อยู่ห่างไกล ให้มีความง่ายต่อการใช้งานมากที่สุด WWW จะแสดงผลอยู่ในรูปแบบของเอกสารที่เรียกว่า ไฮเปอร์เท็กซ์ (Hyper Text) ซึ่งเป็นฐานข้อมูลชนิดหนึ่งที่ทำหน้าที่รวบรวมข่าวสารข้อมูลที่อยู่กระจัดกระจายในที่ต่าง ๆ ทั่วโลกให้สามารถนำมาใช้งานได้เสมือนอยู่ในที่เดียวกัน โดยใช้เว็บเบราว์เซอร์

การรวมกลุ่มคอมพิวเตอร์อย่างกว้างขวางของ WWW ทำให้เว็บกลายเป็นแหล่งข้อมูลขนาดใหญ่ เป็นบริการข้อมูลแบบมีลติมีเดีย บนอินเทอร์เน็ตที่ได้รับความนิยมสูงสุดใน ปัจจุบัน จุดเด่น ได้แก่ความง่ายต่อการใช้งาน ที่เชื่อมโยงจากข้อมูลชุดหนึ่งไปยังข้อมูลอีกชุดหนึ่งได้ ซึ่งอาจอยู่ในศูนย์บริการข้อมูลเดียวกันหรือต่างศูนย์กัน จึงเป็นเสมือนเครือข่ายที่โยงใยข้อมูลทั่วโลกเข้าหากัน เมื่อใช้งานศูนย์บริการแห่งหนึ่งแล้วผู้ใช้สามารถเชื่อมต่อ เพื่อค้นข้อมูลที่ศูนย์อื่นๆได้ข้อมูลใน WWW มีทั้งข้อความปกติ หรือแบบมีลติมีเดีย ที่ประกอบด้วยเสียง ภาพนิ่ง และภาพเคลื่อนไหว WWW ยังได้ผนวกบริการอินเทอร์เน็ตอื่นไว้ภายใน

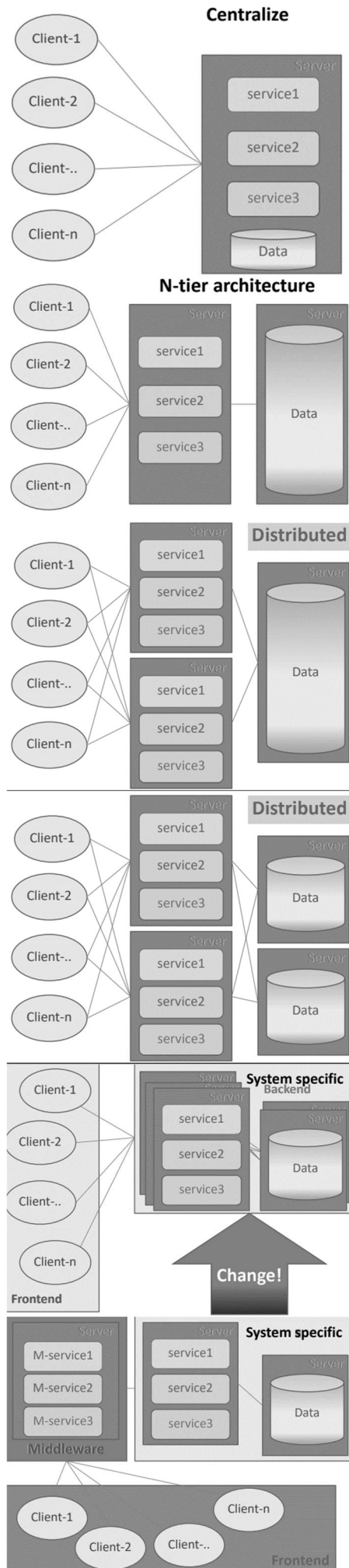
Webpage

เอกสารใน WWW มีชื่อเรียกว่า เว็บเพจ ถูกสร้างขึ้นจากภาษาคอมพิวเตอร์ ที่เรียกว่า HTML ภาษา HTML จะกำหนดรูปแบบและหน้าตาของเว็บเพจที่ปรากฏบนหน้าจอและส่วนที่เชื่อมต่อกับเว็บเพจอื่น

Link

เว็บเพจแต่ละหน้าใน WWW มีการเชื่อมต่อกันทำให้สามารถเรียกดูเว็บเพจหนึ่งจากเว็บเพจอื่นได้โดยในเว็บเพจจะมีจุดเชื่อมโยงที่เรียกว่า ลิงค์ ลิงค์อาจอยู่ในรูปแบบของข้อความ รูปภาพ หรือปุ่ม เมื่อเลื่อนเมาส์ไปเหนือ ลิงค์ มันจะเปลี่ยนเป็นรูปมือสีขาว ลิงค์เป็นคุณสมบัติที่ทำให้เว็บเพจมีความแตกต่างจากเอกสารทั่วไป เพราะผู้อ่านสามารถโต้ตอบกับข้อมูลได้โดยการคลิกเมาส์ เพื่อเปิดดูข้อมูลในส่วนที่ต้องการได้

โดย World Wide Web เป็นระบบเปิด ไม่มีผู้ใดเป็นเจ้าของ ทำให้ผู้ที่สร้างเว็บไซต์สามารถทำงานบนเวปด์ไวล์เว็บได้อย่างมีประสิทธิภาพ โดยการเข้าอินเทอร์เน็ตแต่ละครั้งต้องพิมพ์ URL ผ่าน Browser หลังจากนั้นเบราว์เซอร์จะขอใช้บริการของ WWW เพื่อเข้าถึงข้อมูลที่ต้องการผู้ใช้งานอินเทอร์เน็ตก็สามารถที่จะเห็นข้อมูลต่างๆผ่านหน้าจอคอมพิวเตอร์เป็นตัวอักษรและที่มนุษย์เข้าใจได้ โดยผู้ใช้งานสามารถโต้ตอบสื่อสารกับทาง WWW ได้ด้วยเช่นกัน การโต้ตอบเราจะทำผ่านเว็บเบราว์เซอร์เช่นเดียวกัน



HTML (HyperText Markup Language)

เป็นภาษาที่ใช้สำหรับสร้างเว็บเพจ มีโครงสร้างภาษาโดยใช้ตัวกำกับ (Markup Tag) เพื่อควบคุมการแสดงผลข้อมูล รูปภาพ และวัตถุอื่น ๆ ผ่านทาง Web Browser เช่น GoogleChrome , Firefox , Safari , Microsoft Edge เป็นต้น

ในแต่ละ Tag จะมีส่วนที่เรียกว่า Attribute เพื่อควบคุมการทำงานของ Tag แต่ละตัว

การสร้างไฟล์ HTML

จะต้องอาศัย Text Editor เพื่อใช้สำหรับเขียนคำสั่งต่าง ๆ ที่ต้องการแสดงผลทางจอภาพ / เว็บเบราว์เซอร์ และเก็บเป็นไฟล์โดยมีนามสกุล .html

HTML 5.0

มาตรฐานของภาษา HTML มีการจัดโครงสร้างและการแสดงผลของเนื้อหา

สำหรับ www มาตรฐานใหม่จะมีคุณลักษณะเด่นที่สำคัญ เช่น

- เล่นวิดีโอ
- แสดงตำแหน่งทางภูมิศาสตร์
- เก็บไฟล์ในลักษณะออฟไลน์
- แสดงกราฟิก
- การป้อนข้อมูลแบบใหม่ เช่น search, number, range, color, tel, url, email, date, month, week, time, datetime, datetime-local

DOCTYPE

การประกาศว่าเว็บเพจที่ได้สร้างขึ้นมาอ้างอิงตามมาตรฐานใด

HTML 5

```
<!DOCTYPE html>
```

HTML 4.01:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD
```

```
HTML 4.01 Transitional//EN" ">
```

XHTML 1.1:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
```

```
XHTML 1.1//EN" "">
```

กำหนดรูปแบบ Character encoding ในหน้าเว็บ

การกำหนดรูปแบบการเข้ารหัสอักขระ(Character encoding) โดยใช้แท็ก <meta> กำหนด Attribute charset ลงไป

- <meta charset="utf-8">
- <meta http-equiv="Content-Type" content="text/html; charset=utf-8">

โครงสร้างภาษา HTML

โครงสร้าง HTML จะแบ่งออกเป็น 2 ส่วน ได้แก่ ส่วน head และ ส่วน body โดยเรียงจากแท็ก <head> และ <body> ตามลำดับ โดยทั้ง 2 แท็กจะอยู่ภายใน <html> ... </html>

ส่วน head

เป็นส่วนที่อยู่ภายใน <head> ... </head> ใช้สำหรับอธิบายข้อมูลเกี่ยวกับเว็บ เช่น ชื่อเรื่องของเว็บเพจ (Title) ชื่อผู้จัดทำเว็บ (Author) คีย์เวิร์ด (Keywords) เพื่อใช้สำหรับให้ผู้ใช้ค้นหาข้อมูลเกี่ยวกับเว็บได้

ส่วน body

เป็นส่วนที่อยู่ระหว่าง <body> ... </body> ใช้อธิบายเนื้อหาของเว็บ เช่น ใส่ข้อความต่างๆ รูปภาพ แบบฟอร์ม วิดีโอและยังสามารถกำหนดคุณสมบัติ พื้นฐานของเว็บได้ เช่น รูปแบบของพื้นหลัง สีของตัวอักษร

HTML Element

- ทุกคำสั่งที่อยู่ระหว่างแท็กเปิดและแท็กปิด HTML element บางอย่างไม่มีเนื้อหา (content) ซึ่งจะจบคำสั่งในแท็กเปิดเลย
- โดยส่วนใหญ่ HTML element มักจะมี attribute ประกอบอยู่ในแท็กด้วย

Tag เปิด	Element Content	Tag ปิด
<h1>	หัวข้อเรื่อง	</h1>
	เข้าสู่เว็บไซต์	

Comment

ส่วนที่ใช้ในการการอธิบายโค้ด ซึ่งจะช่วยให้สามารถเข้าใจและสามารถแก้ไขโค้ดได้ในภายหลังได้

รูปแบบ <!-- ข้อความอธิบายโค้ด -->

การกำหนดหัวข้อเรื่อง (Heading)

จะใช้ Tag <h1> จนถึง <h6> โดย <h1> จะเป็นการกำหนดหัวข้อเรื่องที่มีขนาดใหญ่ที่สุดส่วน <h6> เป็นกำหนดหัวข้อเรื่องที่มีขนาดเล็กสุด

แสดงข้อมูลเป็น Paragraphs <p>

จุดเริ่มต้นของ Paragraphs จะเริ่มที่บรรทัดใหม่ และประโยคที่ไม่ได้อยู่ใน Paragraphs เดียวกัน แต่อยู่ในตำแหน่งที่ต่อจาก Paragraphs ก็จะถูกจัดให้ขึ้นบรรทัดใหม่ทันที

แท็กสำหรับขึ้นบรรทัดใหม่

 เพื่อให้เนื้อหาเป็นระเบียบและอ่านได้ง่ายขึ้น

แท็กสำหรับสร้างเส้นคั่นในแนวนอน

<hr> สร้างเส้นคั่นให้กับเนื้อหา

แท็กรูปภาพ (HTML Images)

ใส่ link ให้กับรูปภาพ

การแสดงรายการ (Lists)

ใช้แสดงข้อมูลในรูปแบบของรายการมี 2 รูปแบบ

- รายการแบบใช้ตัวเลข (Order List : Ol)
- รายการแบบใช้สัญลักษณ์ (Unorder List : Ul)

<ol type="รูปแบบการแสดงผล">

หัวข้อย่อยรายการที่ 1

หัวข้อย่อยรายการที่ 2

A. "A" - ตัวอักษรพิมพ์ใหญ่ เช่น A, B, C

a) "a" - อักษรพิมพ์เล็ก เช่น a, b, c

I. "I" - เลขแบบโรมัน เช่น I, II, III

<ul type="รูปแบบการแสดงผล">

หัวข้อย่อยรายการที่ 1

หัวข้อย่อยรายการที่ 2

- disc – จุดสีดำ
 - circle – จุดวงกลมโปร่ง
- square – สี่เหลี่ยมทึบดำ (ตัวเล็กทั้งหมด)

การสร้างตาราง (Table)

<table></table> ใช้กำหนดลำรับสร้างตาราง

<thead></thead> ใช้กำหนดกลุ่มเนื้อหาส่วนหัวตาราง

<tbody></tbody> ใช้กำหนดกลุ่มเนื้อหาตาราง

<tfoot></tfoot> ใช้กำหนดกลุ่มส่วนใต้ตาราง

<tr></tr> ใช้กำหนดแถวในตาราง

<td></td> กำหนดคอลัมน์

<th></th> กำหนดคอลัมน์ที่แสดงผลในส่วนหัวของตาราง

Attribute ของตาราง

- border="ความหนา" กำหนดเส้นขอบและความหนาของเส้นขอบตาราง ค่าเริ่มต้น = 0
- width="%" กำหนดความกว้างหน่วยเป็น %
- bgcolor="สี" กำหนดสีพื้นหลังในตาราง
- <table bgcolor="สี"> กำหนดสีทั้งแถวและคอลัมน์
- <tr bgcolor="สี"> สีของแถว
- <td bgcolor="สี"> สีของคอลัมน์

Attribute ของตาราง

- colspan="x" รวมคอลัมน์ ค่า x คือจำนวนคอลัมน์ที่ต้องการรวมเข้าด้วยกัน (ช่อง <td>)
- rowspan="x" รวมแถว ค่า x คือจำนวนแถวที่ต้องการรวมเข้าด้วยกัน
- align="left, center, right" จัดตำแหน่งของภาพ หรืออักษรภายในช่องตาราง <td> ค่าปกติคือ left
- Cellpadding แสดงข้อมูลภายในตาราง หากมีค่ามากก็จะมีพื้นที่การแสดงผลเป็นที่ว่างมากขึ้น โดยมีค่าเริ่มต้นเป็น 0 (หน่วย Pixel)
- Cellspacing กำหนดขนาดเส้นตาราง หากมีค่ามากขึ้นเส้นตารางก็จะมีขนาดมากตามไปด้วย โดยมีค่าเริ่มต้นเป็น 0 (หน่วย Pixel)

การจัดกลุ่มด้วย div , span

 ใช้จัดกลุ่มข้อความหรือแท็กต่าง ๆ เข้าเป็นกลุ่มเดียวกันเพื่อกำหนด สี รูปแบบตัวอักษร หรือ style ให้กับข้อความและแท็กภายใต้ ให้เป็นรูปแบบเดียวกัน

<div></div> ใช้จัดกลุ่มข้อความหรือแท็กต่าง ๆ เข้าเป็นกลุ่มเดียวกันลักษณะคล้ายๆกับ แต่แตกต่างกันตรงที่แท็ก div จะมีการขึ้นบรรทัดใหม่ก่อนเริ่มแสดงข้อความภายใต้แท็ก div

HTML FORM

การพัฒนาเว็บแอปพลิเคชันจำเป็นต้องมีการสร้างแบบฟอร์มที่ผู้ใช้งานสามารถป้อนข้อมูลต่างๆได้ เพื่อนำข้อมูลไปทำการประมวลผลอีกที โดยการรับค่าข้อมูลจะดำเนินการผ่าน <form>....</form>

Tag	คำอธิบาย
<input>	สร้างช่องรับข้อความต่างๆ
<select>	แสดงตัวเลือกในรูปแบบ Drop-down
<option>	สร้างตัวเลือก
<button>	สร้างปุ่ม
<label>	กำหนดป้ายชื่อให้ช่องรับข้อมูล
<textarea>	สร้างช่องรับข้อความแบบหลายบรรทัด

Block vs Inline

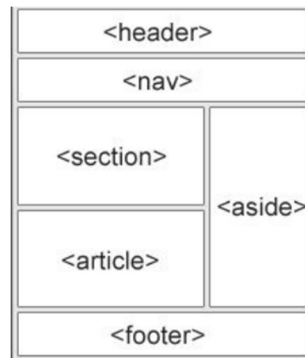
- Block คือ ความยาวเต็มบรรทัด
- Inline คือ ความกว้างเท่ากับข้อความที่แสดง

Class & ID

- Class การประกาศค่า Attribute "class" ในแท็กที่ต้องการ
- ID เป็นการกำหนดรหัสเฉพาะของแท็กด้วยการประกาศค่า Attribute "id" ในแท็กที่ต้องการ เพื่อนำไปแสดงผลเหมือนกับ Class แต่ค่า id จะไม่สามารถซ้ำกันได้

Semantic Tag

การใช้ Semantic tag ถูกนำมาใช้แทน div หลายๆ ชิ้นในหน้าเว็บจะส่งผลทำให้โครงสร้าง html มีความหมายตรงตัวชัดเจนมากยิ่งขึ้น



<header> คือ ส่วนหัวของเว็บ

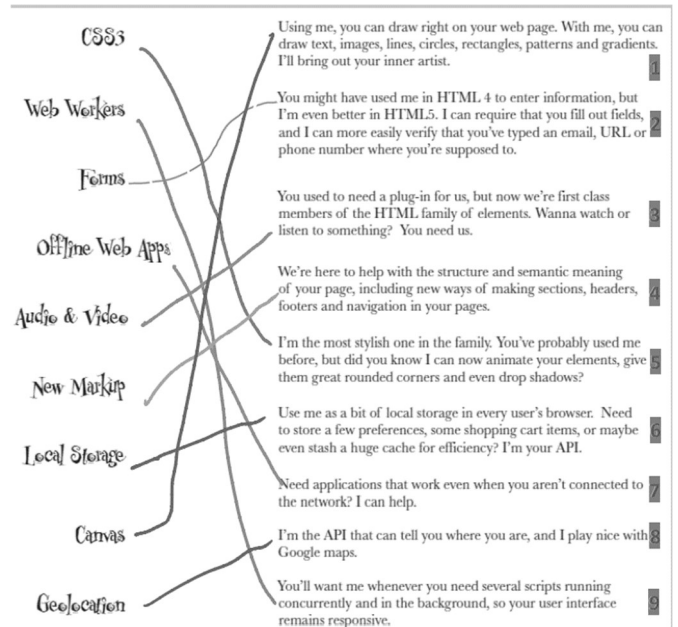
<nav> คือ เมนูของเว็บ หรือ ลิงค์ไปเว็บอื่นๆ

<article> คือ ส่วนที่แสดงเนื้อหาของเว็บ

<section> คือ กลุ่มหัวข้อย่อย

<aside> คือ เนื้อหาอื่นๆที่แยกจากเนื้อหาหลัก

<footer> คือ ส่วนท้ายของหน้าเว็บ



CSS (Cascading Style Sheet)

CSS (Cascading Style Sheets) เป็นเทคโนโลยีที่นำมาใช้จัดรูปแบบและควบคุมการแสดงผลของเว็บ เช่น การกำหนดขนาด สี แบบอักษร เส้นขอบ พื้นหลัง และตำแหน่งของข้อความ รูปภาพ เป็นต้น

โครงสร้างของ CSS

มีส่วนประกอบอยู่ 2 ส่วน คือ Selector และ Declaration

- Selector คือแท็ก HTML, ID และ Class ที่ต้องการกำหนด
- Declaration ใช้สำหรับกำหนดค่าให้กับ Selector

Selector{ Declaration }	แท็กอ้างอิง{ คุณสมบัติ }
-------------------------------	--------------------------------

Selector คือแท็ก HTML, ID และ Class ที่ต้องการกำหนด

คุณสมบัติ แบ่งออกเป็น 3 รูปแบบ

- Element Selector คือ การกำหนด style ให้แท็ก HTML โดยตรง
- Class Selector คือ การกำหนด style โดยใช้ชื่อ Class เป็น Selector โดยใช้เครื่องหมายจุด (.) สามารถมีค่าซ้ำกันได้
- ID Selector คือ การกำหนด style โดยระบุชื่อ ID เป็นรหัสเฉพาะของแท็กหลังเครื่องหมาย # โดย ID ไม่สามารถมีค่าซ้ำกันได้

Declaration ใช้กำหนดค่าให้กับ selector ซึ่งมีส่วนประกอบ

อยู่ด้วยกัน 2 ส่วน คือ

- Property หรือ Attribute คือ คุณสมบัติที่ต้องการกำหนดให้กับ Selector
- Value คือ ค่าที่กำหนดให้กับ Property หรือ Attribute

การกำหนดส่วนของ property กับ value จะแยกด้วยเครื่องหมาย colon (:) ถ้ามีมากกว่า 1 property จะต้องแยกด้วยเครื่องหมาย semi-colon (;) โดยมีรูปแบบดังนี้

selector { property : value; property : value ... }

รูปแบบการประกาศใช้ CSS

แบบ Internal

- Inline เป็นการแทรก attribute style ลงในแท็ก HTML
- Embedded การกำหนดรูปแบบ style sheet ภายในไฟล์เดียวกันกับ

ไฟล์เว็บโดยการเขียนภายใต้ <style> ... </style>

แบบ External

จะกำหนดไฟล์ style sheet แยกเป็นอีกไฟล์โดยมีนามสกุล .css แล้วค่อยเชื่อมเข้ามาใช้ภายในเว็บเพจ เหมาะสำหรับการกำหนดให้หลายเว็บเพจใช้รูปแบบ style เดียวกันผ่านการเขียน style แค่ครั้งเดียว

<link rel="stylesheet" href="style.css">

แบบ Inline

ใช้กำหนด style ให้กับอิลิเมนต์ของ HTML เพียงอันเดียวเป็นการเฉพาะ <p style="color:green">TEXT.</p>

การเขียน Comment

/* ข้อความอธิบายโค้ด */

หน่วย (Unit) ใน CSS (Absolute) แบบตายตัว

px	pixel (พิกเซล) เป็นหน่วยที่นิยมใช้มากที่สุด โดยที่ 1px = 0.75pt สัมพันธ์กับรายละเอียดหน้าจอ
pt	point โดยที่ 1 pt = 1/72 inches ใช้ในงานสิ่งพิมพ์
cm	เซนติเมตร
mm	มิลลิเมตร
in	inches (1 in = 96px = 2.54cm)
pc	picas (1pc = 12pt)

หน่วย (Unit) ใน CSS (Relative) แบบอัตราส่วน

%	เป็นการกำหนดขนาดเป็นเปอร์เซ็นต์ มักใช้กับความกว้างหรือสูง
em	อ้างอิงขนาดกับ element parent ที่ใกล้ที่สุด ใช้ในการกำหนดขนาดจำนวนเท่า ของขนาดปัจจุบันเช่น หากขนาดที่ใช้อยู่คือ 10px <ul style="list-style-type: none"> - ถ้ากำหนดขนาดเป็น 2em จะหมายถึง 2 เท่าของขนาดปัจจุบัน คือ 20px - ถ้ากำหนดขนาดเป็น 1.4em จะหมายถึง 1.4 เท่าของขนาดปัจจุบัน คือ 14px - ถ้ากำหนดเป็นขนาดเดิมที่กำหนดเป็น 1em = 10px
rem	กำหนดขนาดโดยอ้างอิงกับ root element ปกติ font-size จะอยู่ที่ 16px
vw	1% หรือ 1/100 ของ viewport width <ul style="list-style-type: none"> - width ของ browser viewport เท่ากับ 750px ค่า 1vw = 7.5px
vh	1% หรือ 1/100 ของ viewport height <ul style="list-style-type: none"> - height ของ browser viewport เท่ากับ 900px ค่า 1vh = 9px
vmin, vmax	กำหนดค่าต่ำสุด และค่าสูงสุดของ viewport

Universal Selector

กำหนด style sheet มีผลต่อทุกแท็กของเว็บเพจ โดยใช้เครื่องหมาย *

การกำหนดชนิดฟอนต์ (Font)

font-family กำหนดชนิดของฟอนต์ในหน้าเว็บกำหนดได้ทั้งแบบฟอนต์เดียวหรือหลายฟอนต์ โดยกำหนดได้ดังนี้ คือ

ฟอนต์แบบที่ 1;

ฟอนต์แบบที่ 1, ฟอนต์แบบที่ 2, ชนิดฟอนต์;

สามารถกำหนดฟอนต์ได้มากกว่า 1 ชนิดในกรณี Browser ไม่รองรับฟอนต์ชนิดแรก ก็จะนำฟอนต์ลำดับถัดไปมาใช้แทน

การกำหนดขนาดข้อความ

font-size กำหนดขนาดข้อความหรือตัวอักษรในหน้าเว็บโดย

กำหนด 2 รูปแบบ คือ

- แบบค่าคงที่ : xx-small, x-small, medium, large, x-large, xx-large
- แบบตัวเลข : 10px 20px

การกำหนดความหนาข้อความ

font-weight กำหนดได้ 2 รูปแบบ

- แบบค่าคงที่ : lighter, bold (ตัวหนา) bolder (ตัวหนากว่า)
- แบบตัวเลข : 100, 200, ..., 800, 900

font-style กำหนดตัวเอียง โดยค่าที่กำหนดได้คือ italic, oblique (45 องศา)

การกำหนดสี (Colors)

color กำหนดได้ 4 รูปแบบ

ชื่อสี : green, red, yellow, pink

RGB : rgb(red, green, blue) , rgb(30%, 20%, 45%)

Hexadecimal : #000 #FFF

HSL : (hue, saturation, lightness)

การกำหนดลักษณะข้อความ

Text-decoration กำหนดได้ทั้ง 4 แบบ

none : (ค่าว่าง)

underline : (ขีดเส้นใต้)

overline : (ขีดเส้นเหนือข้อความ)

line-through : (ขีดเส้นทับข้อความ)

การจัดแนวข้อความ

Text-align กำหนดได้ 4 รูปแบบ

- left, right, center (ชิดด้านซ้าย ขวา และกึ่งกลางตามลำดับ)
- justify (การกระจายข้อความให้ทุกบรรทัดมีความกว้างเท่ากัน)

กำหนดความกว้างและความสูง

width ใช้กำหนดความกว้างของวัตถุที่ต้องการ

- auto คือ browser จะกำหนดให้เอง
- length คือ ความกว้างแบบหน่วยวัด เช่น px cm เป็นต้น
- % คือ ความกว้างแบบ % (แบบยืดหยุ่น)

height ใช้กำหนดความสูงของวัตถุที่ต้องการ

- auto คือ browser จะกำหนดให้เอง
- length คือ ความสูงแบบหน่วยวัด เช่น px cm เป็นต้น
- % คือ ความสูงแบบ % (แบบยืดหยุ่น)

ความกว้างและความสูงด้วย max , min

- width-min ใช้กำหนดความกว้างต่ำสุดของวัตถุ
- width-max ใช้กำหนดความกว้างสูงสุดของวัตถุ
- height-min ใช้กำหนดความสูงต่ำสุดของวัตถุ
- height-max ใช้กำหนดความสูงสูงสุดของวัตถุ

หน่วยที่รองรับ

- none ไม่กำหนด (default)
- length คือ กำหนดแบบหน่วยวัด เช่น px cm เป็นต้น
- % คือ กำหนดแบบ % (แบบยืดหยุ่น)

การกำหนดเส้นขอบ

Border คือการกำหนดเส้นขอบ โดยมีรูปแบบดังนี้

border : <รูปแบบเส้น> <ขนาด> <สี>

รูปแบบเส้น ประกอบด้วย solid (เส้นทึบ) dotted (แบบจุด) dashed (เส้นปะ)

ขนาด คือ ความกว้างของเส้นกำหนดเป็นตัวเลขตามหน่วย

สี กำหนดรูปแบบเหมือน font เลย เช่น ชื่อสี , rgb , hex

การกำหนดรูปแบบเส้นขอบ (style)

Border คือการกำหนดเส้นขอบ โดยมีรูปแบบดังนี้

border-style : <รูปแบบเส้น>

รูปแบบเส้น ประกอบด้วย solid (เส้นทึบ) dotted (แบบจุด) dashed (เส้นปะ)

border-xxx-style : <รูปแบบเส้น>

xxx คือ ทิศทางของรูปแบบเส้นขอบ

การกำหนดสีเส้นขอบ (color)

Border คือการกำหนดเส้นขอบ โดยมีรูปแบบดังนี้

border-color : <สี>

สี ประกอบด้วย RGB , ชื่อสี , HEX

border-xxx-color: <สี>

xxx คือ ทิศทางของรูปแบบเส้นขอบ

การกำหนดขนาดเส้นขอบ (width)

Border คือการกำหนดเส้นขอบ โดยมีรูปแบบดังนี้

border-width : <ขนาด>

border-xxx-width: <ขนาด>

xxx คือ ทิศทางของรูปแบบเส้นขอบ

การกำหนดขนาดเส้นขอบ (width)

Border คือการกำหนดเส้นขอบ โดยมีรูปแบบดังนี้

border-width : <ขนาด>

แบบระบุค่าคงที่ :

- medium – ขอบปานกลาง
- thin – ขอบบาง
- thick – ขอบหนา
- initial – ค่าเริ่มต้น
- inherit – อ้างอิงตาม parent element

การกำหนดความโค้งเส้นขอบ (radius)

Border คือการกำหนดเส้นขอบ โดยมีรูปแบบดังนี้

border-radius : <ค่าความโค้ง>

border-xxx-radius: <ค่าความโค้ง>

xxx คือ ทิศทางของรูปแบบเส้นขอบ

การกำหนดความโค้งเส้นขอบ

เขียน 4 รูปแบบ

border-radius: 10px 5px 15px 20px;

- top-left = 10px

- top-right = 5px

- bottom-right = 15px

- bottom-left = 20px

เขียน 3 รูปแบบ

border-radius: 10px 5px 15px;

- top-left = 10px

- top-right , bottom-left = 5px

- bottom-right = 15px

เขียน 2 รูปแบบ

border-radius : 10px 5px;

- top-left , bottom-right = 10px

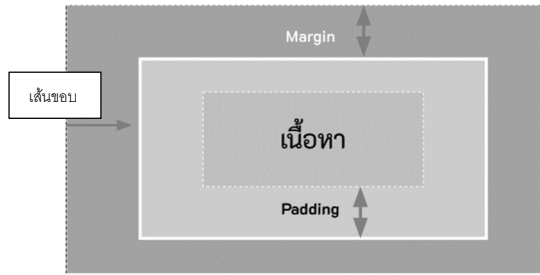
- top-right , bottom-left = 5px

เขียน 1 รูปแบบ

border-radius : 10px;

ทุกทิศทางมีค่า = 10px

Box Model



การกำหนดพื้นที่รอบ Element

Margin กำหนดระยะห่างจากเส้นขอบหรือพื้นที่ภายนอกของ Element เทียบกับวัตถุอื่นๆ (ช่องว่างข้างนอก)

Padding กำหนดพื้นที่ระยะห่างภายในของ Element (ช่องว่างข้างใน)

Border กำหนดเส้นขอบ

กำหนดได้ทั้งรูปแบบด้านบน (top) ด้านล่าง (bottom) ด้านขวา (right) และด้านซ้าย (left) ทั้ง margin และ padding เช่น margin-left, margin-right, padding-left, padding-bottom

หน่วยที่กำหนดได้ เช่น px, pt, cm อื่นๆ หรือแบบเปอร์เซ็นต์ (%) โดยมีค่า default เป็น 0

การกำหนดพื้นที่รอบ Element

Margin กำหนดระยะห่างจากเส้นขอบหรือพื้นที่ภายนอกของ Element

- margin: ค่าที่กำหนด ;
- margin-left: ค่าที่กำหนด;
- margin-right: ค่าที่กำหนด;
- margin-top: ค่าที่กำหนด;
- margin-bottom: ค่าที่กำหนด;
- margin:auto; (ให้ browser จัดให้เอง ปกติจะกำหนดกึ่งกลาง)

เขียน 4 รูปแบบ

margin: 10px 5px 15px 20px;

- top margin = 10px
- right margin = 5px
- bottom margin = 15px
- left margin = 20px

เขียน 3 รูปแบบ

margin: 10px 5px 15px;

- top margin = 10px
- right left = 5px
- bottom margin = 15px

เขียน 2 รูปแบบ

margin: 10px 5px;

- top bottom margin = 10px
- right left = 5px

เขียน 1 รูปแบบ

margin: 10px;

ทุกทิศทางมีค่า = 10px

** ลำดับ top, right, bottom, left

Padding กำหนดพื้นที่ระยะห่างภายในของ Element

- padding: ค่าที่กำหนด ;
- padding-left: ค่าที่กำหนด;
- padding-right: ค่าที่กำหนด;
- padding-top: ค่าที่กำหนด;
- padding-bottom: ค่าที่กำหนด;
- padding:auto; (ให้ browser จัดให้เอง ปกติจะกำหนดกึ่งกลาง)

การกำหนดสีพื้นหลัง (background)

background-color ใช้เปลี่ยนสีพื้นหลังหรือกำหนดความโปร่งใสให้กับพื้นหลังและ Element ต่าง ๆ

background-color : <สี> | transparent

สี ประกอบด้วย RGB , ชื่อสี , HEX

การกำหนดภาพพื้นหลัง (background)

background-image ใช้เปลี่ยนภาพพื้นหลัง

background-image : url(รูปภาพ)

url("image.png")

url("http://www.unsplashpng")

รูปแบบภาพพื้นหลัง (background-repeat)

background-image:url(xxx);

background-repeat : xxx;

repeat (ค่า default ทำให้ภาพเต็มพื้นที่ของอีลิเมนต์)

repeat-x (จัดเรียงเฉพาะในแนวนอน X)

repeat-y (จัดเรียงเฉพาะในแนวแกน Y)

no-repeat (ไม่จัดเรียงแนวแกน x,y แสดงภาพพื้นหลังภาพเดียว)

กำหนดคุณสมบัติพื้นหลัง

โดยทั่วไปภาพพื้นหลังจะถูกเลื่อนตามการ scroll ของเว็บเบราว์เซอร์ ถ้าต้องการตรึงภาพพื้นหลังให้อยู่กับที่สามารถทำได้โดยใช้ **background-attachment**

- scroll (คือการเลื่อนพื้นหลังตาม scrollbar)
- fixed (คือการตรึงพื้นหลังไว้กับที่)

กำหนดตำแหน่งพื้นหลัง

สามารถกำหนดได้ว่า จะเริ่มต้นวางพื้นหลังที่ตำแหน่งใด โดยใช้ **background-position** เช่น

background-position เช่น

background-position : center right ; (กลาง ขวา)

- left top หรือ left center หรือ left bottom
- right top หรือ right center หรือ right bottom
- center top หรือ center center หรือ center bottom

กำหนดขนาดภาพ (Background-size)

background-size : กำหนดค่า

- auto คอปกติภาพเท่าเดิม
- px หรือ %
- cover ขยายภาพทั้งความกว้างความสูงเต็มจอในอัตราส่วนเท่ากัน
- contain ขยายภาพทั้งความกว้างความสูงชนขอบ Browser

กำหนดพื้นหลังแบบหลายคุณสมบัติพร้อมกัน

สามารถกำหนดโดยใช้ property ชื่อว่า **background**

`background : white url(images/logo.jpg) no-repeat fixed center center;`

กำหนดตำแหน่งด้วย float

float กำหนดให้อิลิเมนต์สามารถลอยอยู่ด้านใดด้านหนึ่ง โดยค่าที่กำหนดได้ คือ

- left, right (ให้ลอยอยู่ทางด้านซ้ายและขวาตามลำดับ)
- inherit ลอยตาม parent element
- none ไม่ให้มีการลอย (default)
- clear ไม่อนุญาตให้มีการลอยของอิเลิเมนต์ โดยค่าที่กำหนดได้คือ
- left, right (ไม่อนุญาตให้มีการลอยทางด้านซ้ายและขวาตามลำดับ)
- both (ไม่อนุญาตให้มีการลอยทั้งสองด้าน)

กำหนด style ให้กับ Link

- a : link กำหนด style ให้กับ link ที่ยังไม่ถูกคลิก
- a : hover กำหนด style ให้กับ link เมื่อนำเมาส์ไปวาง
- a : visited กำหนด style ให้กับ link ที่ถูกคลิกแล้ว
- a : active กำหนด style ให้กับ link ขณะที่ถูกคลิก

Display Inline , Block , Inline-Block

display คือการจัดรูปแบบการแสดงผลข้อมูล layout

- none ไม่มีการแสดงผล
- block แสดงผลแบบ block โดยการขึ้นบรรทัดใหม่ก่อน (เรียงในแนวตั้ง)
- inline แสดงผลแบบ inline โดยไม่มีการขึ้นบรรทัดใหม่ (เรียงในแนวนอน)
- inline-block แสดงผลแบบแนวนอนและขยายพื้นที่ด้านใน

Visibility

visibility คือ การแสดงหรือซ่อนวัตถุโดยไม่กระทบ layout

- hidden ซ่อนวัตถุ
- visible แสดงวัตถุ

การจัดตำแหน่งด้วย Position

position ใช้ในการจัดตำแหน่งของวัตถุ

- static จัดวางแบบปกติเป็นค่า default
- relative ใช้ในการจัดวางและกำหนดตำแหน่งวัตถุโดยการนับจากจุดที่วัตถุนั้นๆอยู่
- absolute ใช้ในการจัดวางวัตถุให้ไปอยู่ตำแหน่งใดก็ได้ แต่ต้องกำหนดตำแหน่งจากขอบของ element ที่บรรจุวัตถุนั้นๆอีกที (ถ้าไม่มีอะไรครอบก็ถือว่า body เป็น element ที่ครอบวัตถุที่ระบุ absolute)
- fixed ใช้ในการจัดวางวัตถุให้อยู่ตำแหน่งเดิม
- sticky ให้วัตถุติดขอบเมื่อเลื่อนไปถึง

ลำดับความสำคัญของ Style

- เรียงลำดับความสำคัญจากล่างขึ้นบน
- !important กำหนดให้มีความสำคัญที่สุด

จำกัดการแสดงผลข้อมูลด้วย Overflow

overflow

- visible แสดงข้อมูลทั้งหมด
- hidden ซ่อนข้อมูลที่เกิน
- scroll ให้มี scroll bar แสดงผลเมื่อมีข้อมูลเกิน
- overflow:scroll
- overflow-x : scroll
- overflow-y : scroll
- auto ให้มี scroll bar แสดงผลออกมาอัตโนมัติ

กำหนดเงาให้วัตถุด้วย Box-shadow

box-shadow : x y blur spread color

- x คือ เงาแกน x (+ ขวา, - ซ้าย)
- y คือ เงาแกน y (+ ล่าง, - บน)
- blur คือ ขนาดความมัวของเงา
- spread คือ ขนาดของเงา
- color คือ สีของเงา (color name,rgb,...)

กำหนดค่าความทึบ (Opacity)

opacity : value

- ค่าอยู่ระหว่าง 0.0 - 1.0
- ยิ่งค่าน้อย ยิ่งจาง
- ยิ่งค่ามาก ยิ่งทึบ

Responsive Web Design

การออกแบบเว็บไซต์ที่ตอบสนองและแสดงผลได้ดีบนอุปกรณ์ต่างๆหรือขนาดหน้าจอที่หลากหลาย เช่น จอคอมพิวเตอร์ หรือ สมาร์ทโฟน เป็นต้น

Responsive Web Design (Media Query)

Media Query คือ รูปแบบการเขียน Style ให้แสดงผลตามขนาดหน้าจอที่แตกต่างกัน

@media ขนาดอุปกรณ์ {

.....style.....

}

@media screen , printer {

.....style.....

}

ความกว้างของขนาดอุปกรณ์

- 320px—480px: Mobile devices
- 481px—768px: iPads, Tablets
- 769px—1024px: Laptops
- 1025px—1200px: Desktops
- 1201px เป็นต้นไป — TV , Widescreen

Viewport Units

ในอดีตเว็บถูกพัฒนาให้ทำงานบนจอคอมพิวเตอร์อย่างเดียว ถ้านำมาทำงานบน Smart Phone , Tablet ในปัจจุบันอาจจะไม่สามารถแสดงผลได้ตามที่ต้องการเช่น ถ้าต้องการดูเนื้อหาในหน้าเว็บต้องทำการ zoom และ scroll หน้าจอเท่านั้น

ในปัจจุบันจึงได้มีการพัฒนาและเพิ่ม meta tag ชื่อว่า viewport เข้ามาในหน้าเว็บเพื่อบอกว่าให้เว็บแสดงผลบน Smart Phone , Tablet ได้ผ่านการอ้างอิงพื้นที่หรือ

สัดส่วนของ Web Browser ที่อยู่ในอุปกรณ์นั้นๆ ในการแสดงผล หน้าเว็บให้เหมาะสมกับอุปกรณ์ดังกล่าว

องค์ประกอบของ Viewport

- Vw แนวนอน(ความกว้าง)
- Vh แนวตั้ง(ความสูง)

Viewport Units

- vw บอกสัดส่วนความกว้างของ viewport (%) เช่น เรามีจอกว้าง 100px ต้องการแสดงผล 10% ของความกว้างทั้งหมด จะมีค่า 10vw หรือพื้นที่ 10px หรือ 10/100 เป็นต้น ถ้าอยากกำหนดความกว้างของ viewport มีค่าเท่ากับความกว้างของ browser จะมีค่าเท่ากับ 100vw
- vh บอกสัดส่วนความสูงของ viewport (%)
- vmin ค่า % ต่ำสุดของ viewport
- vmax ค่า % สูงสุดของ viewport

รู้จักกับ Flexbox

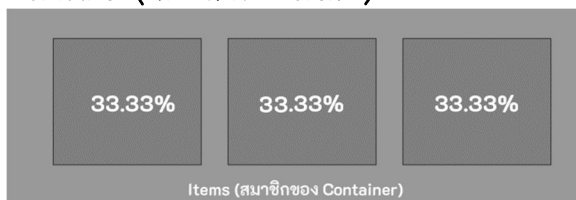
Flexbox คือเครื่องมือใน CSS ที่ช่วยในการจัดการ element ต่างๆ ในหน้าเว็บ มีความง่ายและยืดหยุ่นมากยิ่งขึ้น โดยทั่วไปการจัดการตำแหน่ง element ต่างๆ ต้องใช้ layout mode คือ block , inline , position และอื่นๆ

การพัฒนาเว็บในปัจจุบันมีความซับซ้อนมากยิ่งขึ้นทำให้แบบเดิมไม่ตอบโจทย์เท่าที่ควร จึงได้มีการพัฒนา flexbox ขึ้นมาเพื่อจัดใช้ในการจัดการ element ให้มีความยืดหยุ่นสูง โดยมีความสมบัติดังนี้

- จัดเรียงตำแหน่งของ element ได้ง่ายขึ้น เรียงจากบนลงล่าง ซ้ายไปขวา อื่นๆ
- กำหนดขนาดให้พอดีกับพื้นที่ว่างแบบอัตโนมัติ (Sizing)

องค์ประกอบของ Flexbox

Container (กล่องที่ครอบ Items)



Flexbox เบื้องต้น

```
.container{
display:flex;
box-sizing: border-box;
}
```

- flex: ให้จัดวางในรูปแบบ flex
- border-box – กำหนดขนาดให้พอดีกับพื้นที่ว่างโดยคำนวณจากค่าจริงที่ผู้ใช้กำหนด (border + padding) เพื่อไม่ให้ item แสดงผลเพี้ยน (ระบุหรือไม่ระบุก็ได้)

กำหนดทิศทางด้วย flex-direction

- row (ค่าเริ่มต้น) จัดวาง items ในแนวนอนทิศทางเดียวกับแกนหลัก
- column จัดวาง items ในแนวตั้งทิศทางเดียวกับแกนหลัก
- row-reverse จัดวาง items ในแนวนอนทิศทางตรงข้ามกับแกนหลัก

- column-reverse จัดวาง items ในแนวตั้งทิศทางตรงข้ามกับแกนหลัก

กำหนดขนาดด้วย flex-wrap

กรณีที่ขนาด items ใหญ่กว่าพื้นที่ container

- nowrap จัดวาง items ที่เกินพื้นที่ container ไปด้านขวา
- wrap จัดวาง items ที่เกินพื้นที่ container เรียงจากบนลงล่าง
- wrap-reverse จัดวาง items ที่เกินพื้นที่ container เรียงจากล่างขึ้นบน

Flex - properties (Items)

- flex-shrink : ให้ item หดตัวจำนวนเท่าใดเมื่อเทียบกับ item อื่นๆ ค่าเริ่มต้นเป็น 1
- flex-grow : ให้ item ขยายจำนวนเท่าใดเมื่อเทียบกับ item อื่นๆ ค่าเริ่มต้นเป็น 0
- flex-basis : กำหนดค่าความยืดหยุ่นเริ่มต้น
- flex:1 ทำให้ item ที่อยู่แถวเดียวกันมีขนาดเท่ากัน

Flex Justify (จัดวางตำแหน่ง item)

**เทียบกับแกนหลัก เช่น กำหนดแกนหลักเป็นแนวนอน

Justify-content

- flex-start ซิดซ้าย container ทิศทางตามแนวนอน
- center กึ่งกลาง container ทิศทางตามแนวนอน
- flex-end ซิดขวา container ทิศทางตามแนวนอน
- space-around ระยะห่างซ้ายขวาและขนาด item เท่ากัน
- space-between ระยะห่างซ้ายขวาและขนาด item เท่ากัน (ติดมุม)

Flex Alignment (จัดวางตำแหน่ง item)

**เทียบกับแกนตรงข้าม เช่น กำหนดแกนหลักเป็นแนวนอน

align-items (item ทุกตัว) และ align-self (Item ที่ต้องการ)

- stretch - กำหนดขนาด item เท่ากับขนาด container
- flex-start ด้านบน container ทิศทางตามแนวนอน
- center กึ่งกลาง container ทิศทางตามแนวนอน
- flex-end ด้านล่าง container ทิศทางตามแนวนอน

CSS Grid Layout

Flexbox ถูกออกแบบให้จัดการ layout แบบทิศทางเดียว คือ 1 มิติเช่น เรียงลำดับ

จากบนลงล่าง ซ้ายไปขวา เป็นต้น

--	--	--

แต่ Grid Layout ถูกออกแบบมาเพื่อจัดการ layout แบบ 2 มิติ คือ มีทั้งแบบแนวนอน และแนวตั้งในเวลาเดียวกับ หรือมองง่ายๆ ก็เป็นแบบตาราง

CSS Grid Layout

การใช้งาน

display : grid;

กำหนดขนาดแถว (ความสูง) :

grid-template-rows : ความสูงของแถวที่ 1 , 2 , 3;

กำหนดขนาดคอลัมน์ (ความกว้าง) :

grid-template-columns : ความกว้างของคอลัมน์ที่ 1, 2, 3;

Gird Properties อื่นๆ

- กำหนดสัดส่วนพื้นที่ด้วย span
- กำหนดสัดส่วนพื้นที่ด้วยหน่วย fr
- กำหนดระยะห่างพื้นที่ด้วย gap
- กำหนดชื่อพื้นที่ด้วย grid-template-area

กำหนดเงาให้ข้อความด้วย text-shadow

text-shadow : x y blur color

- x คือ เงาแกน x (+ ขวา, - ซ้าย)
- y คือ เงาแกน y (+ ล่าง, - บน)
- blur คือ ขนาดความมัวของเงา
- color คือ สีของเงา (color name,rgb,...)

CSS Variable

การสร้างตัวแปรใน css เพื่อช่วยอำนวยความสะดวกในการกำหนด style ให้แต่ละ element โดยลดขั้นตอนการทำงานที่ซ้ำซ้อนใน css ให้โค้ดมีความเป็นระเบียบและง่ายต่อการจัดการมากยิ่งขึ้น

การกำหนดตัวแปรสามารถกำหนดชื่อได้เอง (custom variable) โดยขึ้นต้นด้วยเครื่องหมาย - ตามด้วยชื่อตัวแปร ซึ่งตัวแปรส่วนใหญ่ใน css จะนำมาเก็บค่าที่ใช้เรียกทำงานซ้ำๆ เช่น สี เงา แอนิเมชัน เป็นต้น

จัดการ Element ด้วย Transform

- translate (x,y) กำหนดตำแหน่ง element
- scale(x,y) ขยาย element
- rotate (มุม deg) กำหนดการหมุนของ element
- skewX(มุม deg) กำหนดการเอียงของ element แกน x
- skewY (มุม deg) กำหนดการเอียงของ element แกน y

เปลี่ยนแปลง Element ด้วย Transition

transition คือการเปลี่ยนค่าใน element จากค่าหนึ่งไปสู่อีกค่าหนึ่งในช่วงเวลาที่กำหนดประกอบด้วย

- transition-properties คือ รูปแบบคุณสมบัติที่การเปลี่ยนแปลงค่า
- transition-duration คือ ระยะเวลาในการเปลี่ยนแปลงค่า
- transition-timing-fuction คือ รูปแบบฟังก์ชันของการเปลี่ยนแปลงค่า (ease-in)
- transition-delay คือ ระยะเวลาที่จะเริ่มต้นเปลี่ยนแปลงค่า

transition-timing-function



CSS Animation

- animation-name คือ ชื่อ animation
- animation-duration คือ ระยะเวลาของ animation จากเริ่มต้นไปสิ้นสุด
- animation-timing-function คือ รูปแบบการเล่น animation

- animation-iteration-count คือ จำนวนการเล่น animation (infinite คือ ไม่สิ้นสุด)
- animation-direction คือ ทิศทางการเล่น (เล่นจาก frame 1 ไป frame 10 หรือ แบบย้อนกลับก็ได้)
- animation-delay คือ ระยะเวลาที่จะเริ่มต้น

JavaScript เบื้องต้น ใช้ร่วมกับ HTML5 CSS3

JavaScript คืออะไร

เป็นภาษาคอมพิวเตอร์ที่ใช้ในการพัฒนาเว็บร่วมกับ HTML เพื่อให้เว็บมีลักษณะแบบ ไดนามิก คือ เว็บสามารถตอบสนองกับผู้ใช้งานหรือแสดงเนื้อหาที่แตกต่างกันไปโดยจะอ้างอิงตามเว็บเบราว์เซอร์ที่ผู้เข้าชมเว็บใช้งานอยู่

เป็นภาษาที่ทำงานฝั่งผู้ใช้ (Client Side Script) โดยเว็บเบราว์เซอร์จะทำหน้าที่ ประมวลผลคำสั่งที่ถูกเขียนขึ้นมาและตอบสนองต่อผู้ใช้ได้ทันที เช่น การแสดงข้อความแจ้งเตือน (Alert) การตรวจสอบข้อมูลของผู้ใช้ป้อน (Validation) เป็นต้น

ความสามารถของ JavaScript

- สามารถเปลี่ยนแปลงรูปแบบการแสดงผลของ HTML,CSS ได้
- ตรวจสอบความถูกต้องของข้อมูลได้
- ตรวจสอบ Browser ของผู้ใช้ได้
- เก็บข้อมูลผู้ใช้ได้ เช่น การใช้ Cookie , Local Storage เป็นต้น

รูปแบบการเขียน JavaScript

1.แบบ Internal คือ กำหนด JavaScript ไว้ในส่วน of <head></head> หรือ <body></body>

```
<script type="text/javascript">
document.write("Hello World"); </script>
```

2. แบบ External คือ กำหนด JavaScript ไว้เป็นไฟล์ด้านนอกที่มีนามสกุล .js จากนั้นก็นำเข้ามาทำงานในหน้าเว็บ หรือ HTML ไฟล์

```
<script src="ชื่อไฟล์.js"></script>
```

การแสดงผลข้อมูล

- document.write("ข้อความที่ต้องการแสดง") แสดงเป็นข้อความ ตัวเลข ตัวแปร หรือแท็ก HTML ก็ได้ในหน้าเว็บ
- alert("ข้อความแจ้งเตือน") สำหรับแจ้งเตือนผู้ใช้ในหน้าเว็บ
- Console.log("ข้อความ หรือ ตัวแปร") สำหรับ debug ค่าต่างๆ แต่จะไม่แสดงผลในหน้าเว็บ

การเขียนคำอธิบาย (Comment)

วิธีที่ 1 โดยใช้เครื่องหมาย Slash (/) ใช้ในการอธิบายคำสั่งสั้นๆในรูปแบบบรรทัดเดียว

วิธีที่ 2 เขียนคำอธิบายไว้ในเครื่องหมาย /* ... */ ใช้ในการอธิบายคำสั่งยาวๆหรือแบบหลายบรรทัด

ตัวแปรและชนิดข้อมูล

ตัวแปร คือ ชื่อที่ถูกนิยามขึ้นมาเพื่อใช้เก็บค่าข้อมูลสำหรับนำไปใช้งานในโปรแกรม โดยข้อมูลอาจจะประกอบด้วยข้อความ ตัวเลข ตัวอักษรหรือผลลัพธ์จากการประมวลผลข้อมูล

รูปแบบการตั้งชื่อ

1. ขึ้นต้นด้วยตัวอักษรในภาษาอังกฤษตามด้วยตัวอักษรหรือตัวเลข
2. ห้ามขึ้นต้นด้วยตัวเลขหรือสัญลักษณ์พิเศษ
3. ขึ้นต้นด้วย \$ (dollar sign) และ _ (underscore) ได้
4. มีลักษณะเป็น case sensitive คือ ตัวพิมพ์เล็กพิมพ์ใหญ่จะมีความหมายที่แตกต่างกัน
5. ไม่ซ้ำกับคำสงวน (Keyword)

ตัวแปรใน JavaScript เป็นรูปแบบ Dynamic Typing

- ตัวแปรแบบ Dynamic Typing คือชนิดตัวแปรจะเป็นอะไรก็ได้ตามค่าที่ตัวมันเก็บโดยไม่ต้องประกาศชนิดข้อมูล
- ตัวแปรแบบ Static Typing ต้องประกาศชนิดข้อมูลในตอนเริ่มต้น เช่น int, double, char เพื่อบอกว่าตัวแปรนี้จะเก็บข้อมูลชนิดไหน

การนิยามตัวแปร

```
var money;
var money=100;
money=200;
var a, b, c, d;
var x = 10, y = 20, z = 30;
***ตัวแปรที่ประกาศไว้แต่ยังไม่ได้กำหนดค่า จะมีค่าเป็น undefined โดยอัตโนมัติ
```

หัวข้อที่เกี่ยวข้องกับตัวแปร

typeof คือ ใช้ชนิดข้อมูล

null คือ ไม่มีการกำหนดค่าถูกกำหนดค่าโดยผู้เขียน

undefined ไม่มีการกำหนดค่า (เป็นค่าเริ่มต้นของโปรแกรม)

การแปลงชนิดข้อมูล (Type Conversion)

แปลงจาก String เป็น Number

- `x = parseInt('1.2');`
- `x = parseFloat('1.2');`
- ใช้เครื่องหมาย (+...) เพิ่มไปข้างหน้า

แปลงจาก Number เป็น String

- ใช้เครื่องหมาย " " + ตัวแปร หรือ ค่าที่เป็นตัวเลข
- ใช้ `toString()` เช่น `x.toString()`

โครงสร้างควบคุม (Control Structure)

```
if(เงื่อนไขที่ 1){
    คำสั่งเมื่อเงื่อนไขที่ 1 เป็นจริง ;
}else if(เงื่อนไขที่ 2){
    คำสั่งเมื่อเงื่อนไขที่ 2 เป็นจริง ;
}else{
    คำสั่งเมื่อทุกเงื่อนไขเป็นเท็จ ;
}
while(เงื่อนไข){
    คำสั่งที่จะทำซ้ำเมื่อเงื่อนไขเป็นจริง ;
}
for(let i = 1;i<=10;i++) {
    คำสั่งเมื่อเงื่อนไขเป็นจริง;
}
```

รูปแบบของฟังก์ชัน

1.ฟังก์ชันที่ไม่มีการรับและส่งค่า

```
function ชื่อฟังก์ชัน(){
    // คำสั่งต่างๆ
}
```

การเรียกใช้งานฟังก์ชัน

ชื่อฟังก์ชัน ();

2.ฟังก์ชันที่มีการรับค่าเข้ามาทำงาน

```
function ชื่อฟังก์ชัน(parameter1,parameter2,...){
    // กลุ่มคำสั่งต่างๆ
}
```

อาร์กิวเมนต์ คือ ตัวแปรหรือค่าที่ต้องการส่งมาให้กับฟังก์ชัน (ตัวแปรส่ง)

พารามิเตอร์ คือ ตัวแปรที่ฟังก์ชันสร้างไว้สำหรับรับค่าที่จะส่งเข้ามาให้กับฟังก์ชัน (ตัวแปรรับ)

การเรียกใช้งานฟังก์ชัน

ชื่อฟังก์ชัน (argument1,argument2,...);

3.ฟังก์ชันที่มีส่งค่าออกมา

```
function ชื่อฟังก์ชัน(){
    return ค่าที่จะส่งออกไป
}
```

4.ฟังก์ชันที่มีการรับค่าเข้ามาและส่งค่าออกไป

```
function ชื่อฟังก์ชัน(parameter1,parameter2,...){
    return ค่าที่จะส่งออกไป
}
```

ฟังก์ชันแบบกำหนดค่าเริ่มต้น

```
function ชื่อฟังก์ชัน(name="Mon",parameter2,...){
    // คำสั่งต่างๆ
}
```

ขอบเขตตัวแปร

- **local variable** ตัวแปรที่ทำงานอยู่ในฟังก์ชันมีขอบเขตการทำงานตั้งแต่จุดเริ่มต้นไปจนถึงจุดสิ้นสุดของฟังก์ชัน
- **global variable** ตัวแปรที่ทำงานอยู่นอกฟังก์ชันมีขอบเขตการทำงานตั้งแต่จุดเริ่มต้นไปจนถึงจุดสิ้นสุดของไฟล์ที่ประกาศใช้

Array Properties & Function

หาจำนวนสมาชิกและเรียงลำดับ

```
let color = ["แดง", "น้ำเงิน", "เหลือง"];
let x = color.length;
let y = color.sort();
```

สมาชิกตัวแรกและตัวสุดท้าย

```
let first = color[0];
let last = color[color.length-1];
```

การเพิ่มสมาชิก

```
color.push("สีเทา");
```


แปลง Array เป็น String

- .toString() //แปลงเป็น String
- .join(" * "); // นำค่าแต่ละค่าในตัวแปร array มารวมกันเป็นข้อความและส่งค่ากลับเป็นข้อความที่มีตัวคั่นค่าตัวแปรแต่ละค่าตามที่กำหนด
- color.pop(); // เอาตัวสุดท้ายออก
- let x = color.pop(); //เอาตัวสุดท้ายออกแล้วเก็บในตัวแปร x

เรียงลำดับใน Array

```
let fruits = ["ส้ม", "องุ่น"];
fruits.sort();
fruits.reverse();
```

JavaScript Object

```
let ชื่อวัตถุ = {propertyName:value}
ยกตัวอย่าง เช่น
let user = {
  name:"mon", age:20 email:"mon@gmail.com"
};
let product = {name:"มะม่วง",price:150,category:"ผลไม้"}
```

การเข้าถึงข้อมูล

```
objectName.propertyName
objectName["propertyName"]
ยกตัวอย่าง เช่น
user.name
user["name"]
```

JavaScript Object (Method)

```
let user = {
  name:"mon",
  age:20,
  email:"mon@gmail.com",
  getUser:function(){
    return this.name + " " + this.email;
  }
};
```

การเรียกใช้งาน

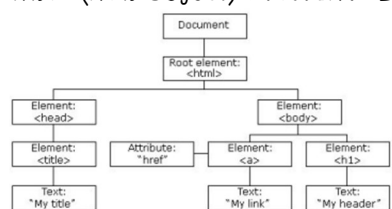
```
objectName.methodName();
let data = user.getUser();
```

ความแตกต่างของ Array และ Object

- Array มี Index เป็นตัวเลข , Object กำหนดเป็นชื่อ
- Array ใช้ [] , ส่วน Object ใช้ {}

HTML DOM (Document Object Model)

เมื่อหน้าเว็บโหลดเสร็จเรียบร้อย Web Browser มันจะสร้าง DOM ของหน้านั้นขึ้นมา โดยมอง HTML เป็นโครงสร้างต้นไม้ (ก่อน Object) หรือเรียกว่า DOM



Tag ต่าง ๆ ใน HTML จะเรียกว่า Element

คุณสมบัติของ HTML DOM

- เข้าถึงและเปลี่ยนคุณสมบัติทั้งหมดของ Element ในหน้าเว็บได้
- ควบคุมและเปลี่ยนรูปแบบ CSS ได้
- สามารถตอบสนองกับทุกเหตุการณ์ที่เกิดขึ้นหน้าเว็บได้

เข้าถึง Element ผ่าน Id , Tag , Class

- document.getElementById ("ชื่อไอดี");
- document.getElementsByTagName ("ชื่อแท็ก");
- document.getElementsByClassName("ชื่อคลาส");

DOM Document

- เปลี่ยนเนื้อหา HTML : element.innerHTML
- เปลี่ยนเนื้อหา Text : element.innerText
- เปลี่ยน style Element : element.style.properties = value

ดำเนินการผ่าน Method

- element.setAttribute(attribute, value)

DOM Nodes

- document.createElement(element) // สร้าง element ใหม่
- document.removeChild(element) // ลบ node ลูก
- document.appendChild(element) // นำ element ไปต่อใน node แม่
- document.replaceChild(new, old) แทนที่ element

DOM CSS Add & Remove Class

- element.classList.add("class"); // เพิ่ม class style
- element.classList.remove("class"); // ลบ class style
- element.classList.toggle("class"); // สลับ class style
- element.classList.contains("class");// เปรียบเทียบ class style

DOM Event

คือ เหตุการณ์หรือการกระทำบางอย่างที่เกิดขึ้นกับอิลิเมนต์ เช่น การคลิกเมาส์ การเคลื่อนย้ายเมาส์ การกดปุ่มคีย์บอร์ด เป็นต้น

โดยผู้พัฒนาสามารถใช้ไิเวนต์ที่เกิดขึ้นเป็นตัวกำหนดให้ตอบสนอง หรือกระทำบางอย่างได้ เช่น การคลิกแล้วแจ้งเตือน เป็นต้น

ชื่อ Event	ความหมาย	ทำงานร่วมกับแท็ก
onfocus=" "	เมื่ออิลิเมนต์นั้นได้รับการโฟกัส	select, text, textarea
onblur=" "	เมื่ออิลิเมนต์นั้นสูญเสียการโฟกัส หรือถูกย้ายโฟกัสไปยังอิลิเมนต์อื่น	select, text, textarea
onchange=" "	เมื่อผู้ใช้เปลี่ยนแปลงค่าในฟอร์มรับข้อมูล	select, text, textarea
onselect=" "	เมื่อผู้ใช้เลือกข้อความ (ใช้เมาส์ลาก) ในช่องข้อความ	text, textarea
onsubmit=" "	เมื่อผู้ใช้คลิกปุ่ม submit	form

ชื่อ Event	ความหมาย	ทำงานร่วมกับ
onmouseover=" "	เกิดเมื่ออนเจกต์นั้นถูกเลื่อน mouse pointer ไปทับ	a,div
onmouseout=" "	เกิดเมื่ออนเจกต์นั้นถูกเลื่อน mouse pointer ที่ทับอยู่ออกไป	a,div
onclick=" "	เกิดเมื่ออนเจกต์นั้นถูกคลิก	a, button, checkbox, radio, reset, submit
onload=" "	เกิดเมื่อโหลดเอกสารเสร็จ	body
onunload=" "	เกิดเมื่อยกเลิกการโหลด เช่น คลิกปุ่ม Stop	body

EventListener

คือ เหตุการณ์หรือการกระทำบางอย่างที่เกิดขึ้นกับอิลิเมนต์แต่รูปแบบการเขียนจะเขียนในฝั่ง javascript ทั้งหมด
โครงสร้าง : `element.addEventListener(event,callback)`

JavaScript ES6 (For React)

Block Scope (let/constant)

การประกาศใช้งาน `let` แทน `var` เนื่องจาก `var` เป็นตัวแปรที่ถูกมอง global variable สามารถทำงานทะลุขอบเขต (block scope) ได้

- การใช้งาน `const` ในการประกาศตัวแปรที่เก็บค่าคงที่

Arrow Function

เป็นรูปแบบการเขียน Function ให้มีความกระชับมากยิ่งขึ้น
แบบเดิม

```
function fullname(fname,lname){
    return fname+lname
}
```

Arrow Function เป็นรูปแบบการเขียน Function ให้มีความกระชับมากยิ่งขึ้น

แบบใหม่

```
fullname=(fname,lname)=>fname+lname
```

Object

```
const user {
    name:name,
    email:email,
    address:address
}
```

String

- MultiLine String สามารถทำงานกับข้อความยาวๆได้ โดยการขึ้นบรรทัดใหม่แล้วไม่มีข้อผิดพลาดเกิดขึ้นโดยใช้ `
- Interpolation สามารถแทรกตัวแปรลงในพื้นที่ String ได้โดยใช้`\${ชื่อตัวแปร}` ร่วมกับ `

Spread Operator

ใช้ในการกระจายสมาชิกใน Array ออกมาใช้งานโดยเดิมเครื่องหมาย ... ด้านหน้าตัวแปร Array

Rest Parameter

ใช้ในการส่งค่า Parameter เข้าไปทำงานใน Function โดยไม่จำกัดจำนวนโดยใช้เครื่องหมาย ...

Destructuring (การสลายโครงสร้าง)

คือ การกำหนดค่าที่อยู่ภายใน Array หรือ Object ให้กับตัวแปรโดยใช้วิธีการจับคู่ตัวแปรกับค่าใน array หรือ Object เช่น

```
const colors = ["ขาว","แดง"]
```

```
const [a,b]=colors
```

Default Parameter

การกำหนดค่าเริ่มต้นให้กับ Parameter ภายในฟังก์ชัน

Array เบื้องต้น

- `Join ()` แปลง Array ให้กลายเป็น string และคั่นด้วยเครื่องหมาย , (comma) หรือ เครื่องหมายอื่นๆที่กำหนดขึ้นมา
- `concat ()` รวม Array และ return Array ก้อนใหม่กลับมา โดยที่ไม่เปลี่ยนแปลงโครงสร้าง Array เดิม (ส่ง Array หรือ Element เข้าไปได้)
- `slice()` คัดลอก Array แล้ว return Array ออกไปใช้งาน

React

React คือ JavaScript Library ที่ใช้สำหรับสร้าง User Interface (UI) หรือหน้าจอของแอปพลิเคชัน ถูกพัฒนาขึ้นโดย Facebook

Framework vs Library

Framework ข้อดี

- มีรูปแบบที่ชัดเจนเป็นแบบแผนเหมาะกับงานที่ทำเป็นทีมต้องเขียนตามรูปแบบที่กำหนดไว้
- มีเครื่องมือทุกอย่างพร้อมให้เราใช้งานได้เลย

Framework ข้อเสีย

- ไม่มีความยืดหยุ่น
- ปรับปรุงแก้ไขการทำงานได้ยาก

Library

Library ข้อดี

- เลือกเครื่องมือหรือสิ่งที่ต้องการมาใช้งานในระบบของเราได้เลย
- มีความยืดหยุ่นสูง

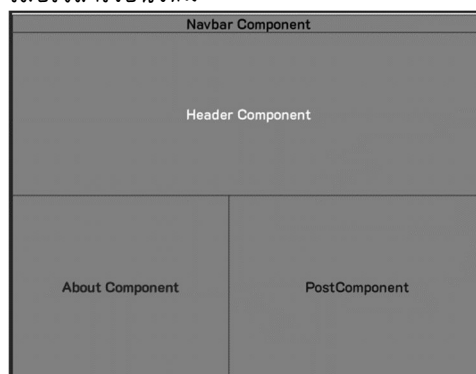
Library ข้อเสีย

- ต้องทำทุกอย่างด้วยตนเอง

แนวคิดของ React

React จะใช้เข้ามาสร้างหน้าเว็บ โดยที่สามารถแบ่งการแสดงผลหน้าเว็บออกเป็นส่วนย่อยหลายๆส่วนได้ โดยที่ไม่ต้องเขียนหน้าเว็บเก็บไว้ในไฟล์เดียว เพื่อให้ง่ายต่อการจัดการ แล้วค่อยนำส่วนย่อยดังกล่าวมาประกอบกันในภายหลัง เราจะเรียกองค์ประกอบที่แบ่งออกเป็นส่วนย่อยๆ นี้ว่า **"Component"**

ข้อดีของการสร้าง Component คือ สามารถที่จะออกแบบแล้วนำกลับมาใช้งานในภายหลังได้ โดยที่ไม่ต้องเสียเวลาเขียนใหม่



สำหรับการสร้าง Component นั้นก็จะมีรูปแบบการเขียนคล้ายกับ HTML เพียงแต่ว่า React จะใช้ส่วนที่เรียกว่า **JSX** ในการเขียนซึ่งมีความคล้ายคลึงกับ HTML มาก โดยการเขียน JSX นั้น คือการเขียนในไฟล์ JavaScript ไม่ใช่ในไฟล์ HTML ซึ่งจะลงรายละเอียดในหัวข้อของ Component

Imperative Programming

เป็นรูปแบบการเขียนโปรแกรมแบบเดิมที่ใช้ใน JavaScript ในการบอกว่าเว็บของเราต้องการอยากจะทำอะไรแล้วให้กระทำการอะไร เช่น

```
const btnEl = document.createElement('button')
btnEl.innerHTML = 'Send Data'
document.body.appendChild(btn)
```

Declarative Programming

เป็นรูปแบบการเขียนโปรแกรมที่ได้รับความนิยมในปัจจุบันเนื่องจากทำให้โค้ดมีความกระชับอ่านแล้วเข้าใจง่าย มีความหมายที่ตรงตัว

```
<Button>Send Data</Button>
```

แนวคิดของ React

- **Component** คือ ส่วนประกอบต่างๆที่ถูกนำมาประกอบรวมกันเป็นหน้าเว็บ (คล้ายๆ สร้าง tag ขึ้นมาใช้เอง เช่น <mon/> เป็นต้น)
- **State** คือ ข้อมูลที่อยู่ภายใน Component แต่ละตัว
- **Props** คือ ข้อมูลที่ถูกส่งจาก Component (แนวคิดมาจากการกำหนด Attribute หรือ Properties ใน HTML)

โครงสร้างโปรเจกต์

- **node_modules** เก็บโมดูลหรือไลบรารีที่ทำงานภายในโปรเจกต์ของเรา
- **public** เก็บไฟล์ public ต่างๆ เช่น รูปภาพ และ index.html เป็นต้น
- **src** สำหรับเก็บ Component หรือโครงสร้างของแอปพลิเคชัน
- **package.json** เก็บข้อมูลต่างๆรวมถึง package ที่จะใช้ทำงานภายในโปรเจกต์

/Public/Index.html

ไฟล์ที่ใช้สำหรับการแสดงผลข้อมูลบนเบราว์เซอร์ จะรันผ่านไฟล์ index.html สิ่งที่เราสนใจใน index.html ก็คือ คำสั่งส่วนที่อยู่ใน <body> ตรงส่วนของ <div id="root"></div>

/Src/App.js

เป็นไฟล์หลักในการรันแอปพลิเคชันขึ้นมา โดยนำส่วนประกอบต่างๆมาประกอบกันแล้วนำไปแสดงผลในเบราว์เซอร์

/Src/index.js

เป็นไฟล์สำหรับเชื่อมการทำงานระหว่าง App.js และ Index.html

ReactDOM Render HTML

ReactDOM เป็นไลบรารีเหมือนกับ React ทำหน้าที่เฉพาะในการจัดการกับ DOM และใช้เฉพาะกับ React เท่านั้น

คำสั่ง **ReactDOM.render()** จะทำการสร้าง DOM (Virtual DOM) ที่มีลักษณะของ โครงสร้างต้นไม้ แล้วนำโครงสร้างดังกล่าวไปส่งไปยัง DOM จริงๆ (Real DOM) ซึ่งเป็น

วิธีการในการ Render JSX ออกมาแสดงผลทางหน้าจอ ตัวอย่าง เช่น

```
ReactDOM.render(<p>Hello World</p>, document.getElementById('root'));
```

Virtual DOM

การทำงานของ Virtual DOM

Virtual DOM มีลักษณะคล้ายกับ DOM ใน HTML โดย Virtual DOM จะทำการคัดลอก DOM จริง (Real DOM) มาเก็บลง Memory ถ้ามีการเปลี่ยนแปลงเกิดขึ้นที่ Component ก็จะอัปเดตเฉพาะ Component ที่เปลี่ยนแปลงเท่านั้น โดยไม่จำเป็นต้องอัปเดต DOM ใหม่หมดทั้งหน้า ทำให้เกิดการทำงานได้อย่างรวดเร็ว

***ถ้าใช้ (DOM แบบปกติจะ Refresh ทั้งหน้าเพื่ออัปเดตหน้าเว็บที่เปลี่ยนแปลงไป)

รูปแบบการสร้าง Component

สามารถสร้างได้ 2 รูปแบบ

- **Class Component**
- **Functional Component**

โดยทั้งคู่จะ Return HTML ออกมาและเขียน JSX ด้านในส่วนของการ Return

Functional Component

สร้างแบบ Component ที่ง่ายที่สุด คือ เป็นรูปแบบฟังก์ชันโดยสร้างฟังก์ชันธรรมดาๆ และ Return HTML ออกมา โดย กำหนดให้ตัวอักษรตัวแรกของชื่อฟังก์ชันเป็นตัวพิมพ์ใหญ่เสมอ เช่น

```
function HelloComponent() {
  return <h1>สวัสดี React </h1>;
}
ReactDOM.render(<HelloComponent />, document.getElementById('root'));
```

Class Component

สร้าง Class ที่ extends มาจาก React.Component และ Return HTML ออกมา แต่ถ้าสร้างเป็นแบบ Class จะมีความสามารถในการใช้งานมากกว่าแบบ Functional

```
class HelloComponent extends React.Component {
  render() {
    return <h1>สวัสดี React</h1>;
  }
}
```

```
ReactDOM.render(<HelloComponent />, document.getElementById('root'));
```

External Component

การสร้าง Component ไว้เป็นไฟล์ด้านนอกที่มีนามสกุล .js จากนั้นก็นำเข้ามาทำงาน

ในหน้าเว็บ

```
function HelloComponent(){
  return <h1>สวัสดี Component แบบ External</h1>
}
export default HelloComponent
```


React JSX

รูปแบบการเขียน JSX

- สามารถเขียนใน `<div>` / `section` / `article` / `Fragment` / `<>` ก็ได้ และต้องมีการกำหนด Tag เปิด - ปิด ทุกครั้ง
- การใช้ Class Style ที่เป็น Attribute ใน JSX จะมีการกำหนด `className` แทน `class` เนื่องจากคำว่า `class` เป็น keyword

รูปแบบการเขียนแบบ div

```
function HelloComponent(){
  return (
    <div>
      <div><h3>สวัสดี</h3></div>
    </div>
  );
}
```

รูปแบบการเขียนแบบ section / article

```
function HelloComponent(){
  return (
    <section>
      <article><h3>สวัสดี </h3></article>
    </section>
  );
}
```

รูปแบบการเขียนแบบ React.Fragment

```
function HelloComponent(){
  return (
    <React.Fragment>
      <div><h3>สวัสดี React </h3></div>
    </React.Fragment>
  );
}
```

รูปแบบการเขียนแบบ Fragment แบบ <>

```
function HelloComponent(){
  return (
    <>
      <div><h3>สวัสดี React </h3></div>
    </>
  );
}
```

React Props

Props (Properties) คือ ตัวแปรที่สามารถส่งเข้าไปใน Components ได้ ผ่านการกำหนด Attribute ส่งผลให้ Component แต่ละตัวสามารถรับค่าจากด้านนอกเข้าไปทำงานได้ <ชื่อคอมโพเนนต์ ชื่อพร็อพเพอร์ตี้ = ค่าที่กำหนดให้พร็อพเพอร์ตี้/>

Keys

Keys เป็นพร็อพเพอร์ตี้ที่อยู่ใน JSX โดย Keys จะมีค่าไม่ซ้ำกัน กำหนดขึ้นเพื่อให้ React นำไปเช็คว่ามี Component ไດบ้างที่เปลี่ยนแปลงค่าไปในการ Render หน้าเว็บ

React PropTypes (Validation)

เป็นการประกาศรูปแบบหรือชนิดของ Props ที่ส่งเข้าไปทำงานใน Component เช่น กำหนดชนิดข้อมูลของ Props หรือ บังคับให้ต้องกำหนดค่า Props ทุกครั้งที่มีการเรียกใช้งาน Component เป็นต้น

การใช้งาน

```
ชื่อคอมโพเนนต์.propTypes = {
  ชื่อพร็อพเพอร์ตี้ : รูปแบบของพร็อพเพอร์ตี้
}
```

State

State คือ ตัวแปรที่เก็บข้อมูลภายใน Component คล้ายๆกับ Props แต่การใช้งาน Props นั้น ข้อมูลจะไม่สามารถเปลี่ยนแปลงค่าได้ แต่ State สามารถทำได้

ฉะนั้น ถ้าต้องการให้ข้อมูลภายในแอพสามารถเปลี่ยนแปลงค่าได้ในระหว่างรันแอพก็จะใช้ State ซึ่งรูปแบบเดิมจะเขียนภายใน

Stateless vs Stateful

- Stateless คือ State ที่ไม่มีการเปลี่ยนแปลงค่า
- Stateful คือ State ที่มีการเปลี่ยนแปลงค่า

REACT HOOK

Hook เป็นฟีเจอร์ที่มีอยู่ใน React เวอร์ชัน 16.8 เป็นต้นไปใช้สำหรับจัดการเกี่ยวกับ State หรือฟีเจอร์ต่างๆที่อยู่ภายใน React โดยที่ไม่ต้องเขียนใน Class Component แต่จะอยู่ใน Functional Component แทน

การใช้งาน React Hook

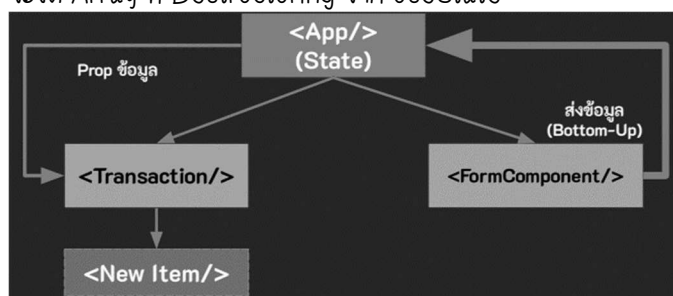
1. เขียนใน Functional Component
2. เขียนในส่วน of Top-Level ของ Function (อย่าเขียนใน Condition , Loop , Nested Function นอกจากจะเขียน Custom Hook)

ตัวอย่าง React Hook

```
import {useState} from 'react'
[ชื่อ State , ฟังก์ชันที่ใช้เปลี่ยนแปลงข้อมูลใน State] =
useState(ค่าเริ่มต้นของ State)
```

ตัวอย่าง

```
const [name, setName] = useState('kongruksiam')
const [age, setAge] = useState(30)
จะได้ Array ที่ Destructuring จาก useState
```



useEffect

**** Effect** คือ ผลกระทบ

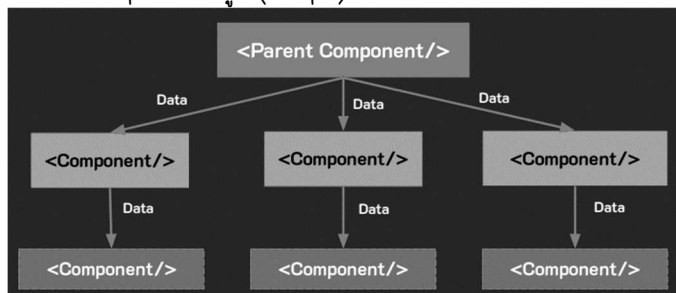
**** useEffect** คือ ผลกระทบที่เกิดขึ้นภายใน Component ใช้เพื่อต้องการทราบว่าเกิดการอัปเดตหรือเปลี่ยนแปลงอะไรขึ้นบ้างภายใน Component จนส่งผลให้ Component เกิดการ Render ใหม่

โดยสาเหตุหลักๆที่ Component Render ใหม่จะมาจาก การเปลี่ยนแปลงค่าที่อยู่ภายใน Props และ State นั้นเอง

การใช้งาน useEffect จึงนำมาใช้งานเพื่อตรวจสอบการเปลี่ยนแปลงที่เกิดขึ้นภายใน Application ของเรามาว่ามีข้อมูลส่วนใดบ้างที่เปลี่ยนแปลงไปจากค่าหนึ่งไปสู่อีกค่าหนึ่ง จนส่งผลให้ Render Component ใหม่อีกครั้ง

Context API (Global State)

ในกรณีที่ต้องการอยากให้แอปของเรามีข้อมูลกลางและอยากให้มีการแชร์ข้อมูลเกิดขึ้นใน Component จะทำอย่างไร เช่น ต้องการให้ทุกๆ Component สามารถเข้าถึงข้อมูลชุดเดียวกันได้ โดยที่ไม่ต้องให้รูปแบบส่งข้อมูลจาก Component แม่ไปยัง Component ลูก (Props)



Context API มีองค์ประกอบ 2 ส่วน คือ

- Provider (Parent) ดูแลและจัดการข้อมูลแล้วนำไปส่งให้ Consumer
- Consumer (Children) นำข้อมูลที่ได้จาก Provider ไปสร้างหรือแสดงผลใน Component

useReducer

******** เป็นการจัดการ State ในรูปแบบของ Redux

โดยทั่วไป State สามารถอ่านค่าได้อย่างเดียว ถ้าต้องการอยากเปลี่ยนแปลงค่า State จะใช้ useReducer โดยการกำหนดรูปแบบการกระทำที่เกิดขึ้นกับ State ของเราผ่านส่วนที่เรียกว่า Action

React Router

การพัฒนาแอปด้วย React ประกอบไปด้วยการแสดงผลมากกว่า 1 หน้าจอ การที่จะทำให้ผู้ใช้งานไปยังส่วนการแสดงผลต่างๆที่เกิดขึ้น จากการใช้งาน Component ในแอป เรา จะใช้ส่วนที่เรียกว่า Route (การสร้างเส้นทาง) !!!

- 1.สร้าง Router/ Route เพื่อกำหนดเส้นทางในการเข้าถึง Component ของการกำหนด Path
- 2.สร้าง Switch ในการเปลี่ยนเส้นทางในการเข้าถึง Component
- 3.กำหนด Link เพื่อเชื่อมโยง Path กับ Component

AJAX

AJAX ย่อมาจากคำว่า Asynchronous JavaScript and XML ซึ่งหมายถึงการพัฒนาเว็บแอปพลิเคชันที่ประมวลผลในเบื้องหลังเป็นเทคนิคในการพัฒนาเว็บแอปพลิเคชันเพื่อให้สามารถโต้ตอบกับผู้ใช้ได้ดีขึ้น ทำให้ความรู้สึกรการใช้งานโปรแกรมเหมือนกับเดสก์ท็อปแอปพลิเคชัน

ตัวอย่างโค้ดการทำงานของ AJAX ของโดยใช้ jQuery

```

$.ajax({
  type: 'GET',
  url: 'send-ajax-data.php',
  dataType: 'JSON', // data type expected from server
  success: function (data) {
    console.log(data);
  },
  error: function(error) {
    console.log('Error: ' + error);
  }
});

```

วิธีที่ 1 ใช้ XMLHttpRequest

```

var request = new XMLHttpRequest();
request.open('GET','https://tutorialzine.com/misc/files/my_url.html',
true);
request.onload = function (e) {
  if (request.readyState === 4) {

    // Check if the get was successful
    if (request.status === 200) {
      console.log(request.responseText);
    } else {
      console.error(request.statusText);
    }
  }
};

```

```

// Catch errors:
request.onerror = function (e) {
  console.error(request.statusText);
};
request.send(null);

```

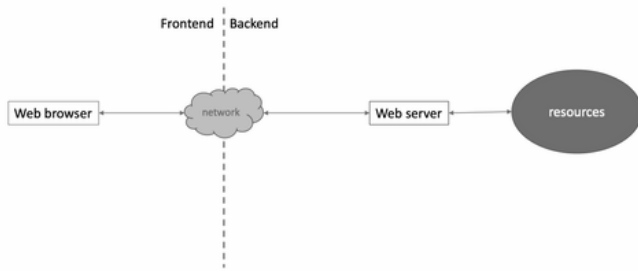
วิธีที่ 2 ใช้ Fetch API

```

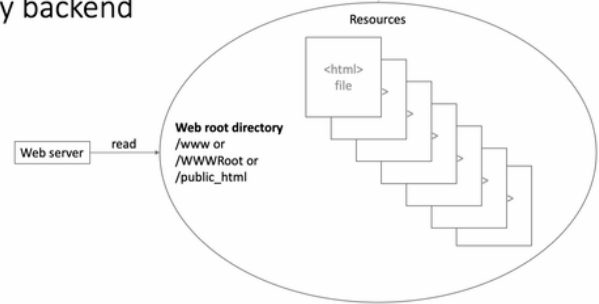
fetch('./api/some.json')
.then(
  function(response) {
    if (response.status !== 200) {
      console.log('Looks like there was a problem. Status Code: ' +
response.status);
      return;
    }
    // Examine the text in the response
    response.json().then(function(data) {
      console.log(data);
    });
  }
)
.catch(function(err) {
  console.log('Fetch Error :-S', err);
});

```

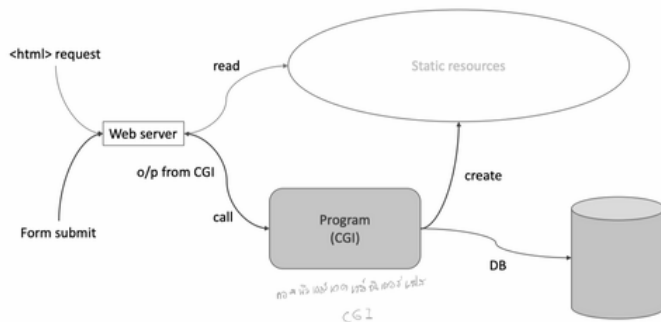
Web in general



Early backend

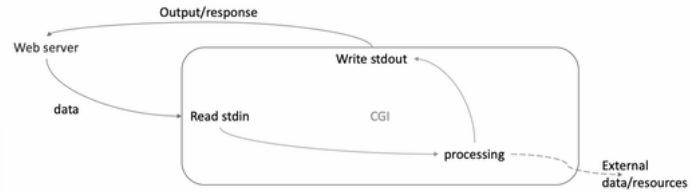


Early backend with data handling

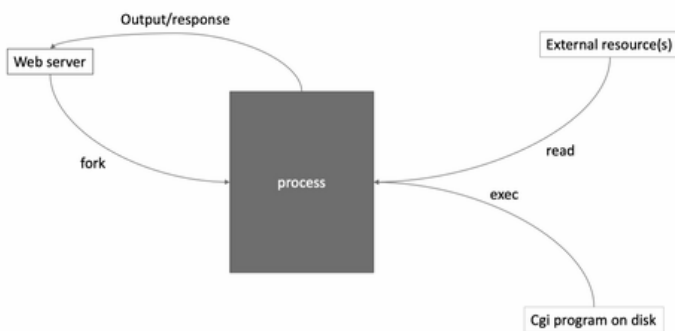


Early CGI

- Develop with language such as C/C++, perl, etc.
- Data transfer from web server to CGI via standard input
- Data transfer from CGI to web server via standard output



Execution



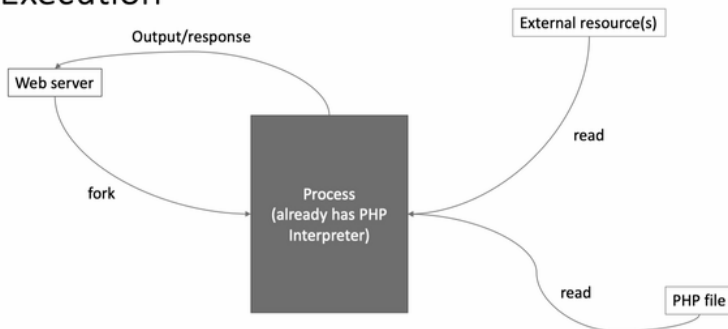
PHP (ASP)

- Easier data input handling
 - Automatic generate \$_GET or \$_POST
- Easier coding
 - Coding in html file
 - <?php ?>
- Integrates PHP interpreter into web server (option)
 - Speed up execution

ASP

- Classic ASP
- Active Server Page
- Developed by Microsoft
- Same concepts as PHP
- ASP command written within <% ... %>
- Based on VBScript syntax

Execution



Example: ASP code

```
<!DOCTYPE html>
<html>
<body>
```

```
<p>ASP example: output HTML tag</p>
```

```
<%
response.write("<h1>Hello World!</h1>")
%>
```

```
</body>
</html>
```

Example: what browser receive

```
<!DOCTYPE html>
<html>
<body>
```

```
<p>ASP example: output HTML tag</p>
```

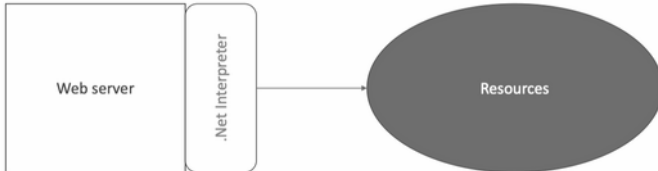
```
<h1>Hello World!</h1>
```

```
</body>
</html>
```


ASP.Net

- Release in 2002 as a successor to ASP X
- Normally used c# syntax
- Uses .aspx as an extension
- Tutorial: https://www.w3schools.com/asp/webpages_intro.asp

ASP/ASP.Net

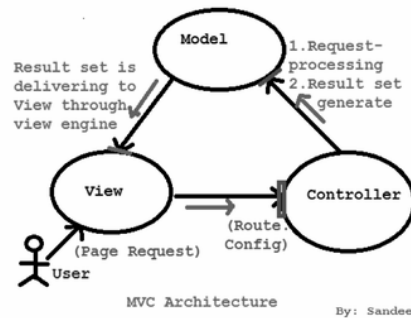


Example:

Lesson08_Backend : 2

```
<!DOCTYPE html>
<html>
<body>
@{
    if (IsPost)
    {
        string companyname = Request["CompanyName"];
        string contactname = Request["ContactName"];
        <p>You entered: <br>
        Company Name: @companyname <br>
        Contact Name: @contactname </p>
    }
    else
    {
        <form method="post" action="">
        Company Name:<br>
        <input type="text" name="CompanyName" value=""><br>
        Contact Name:<br>
        <input type="text" name="ContactName" value=""><br><br>
        <input type="submit" value="Submit" class="submit">
        </form>
    }
}
</body>
</html>
```

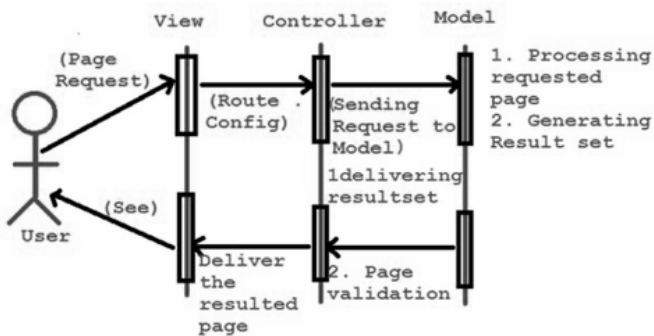
MVC Architecture



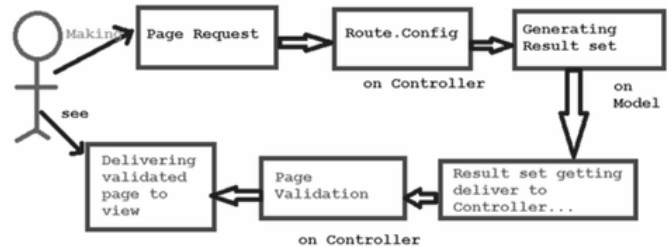
ASP.Net MVC → ASP.Net Core MVC (2016)

- MVC = Model View Controller
 - Model represent data
 - View is UI
 - Controller is the request handler

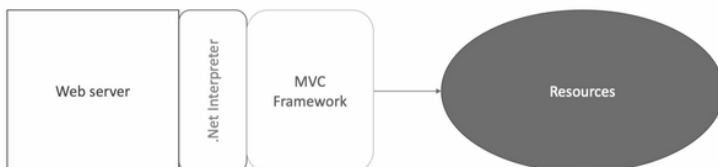
Life cycle of MVC



Live cycle of MVC



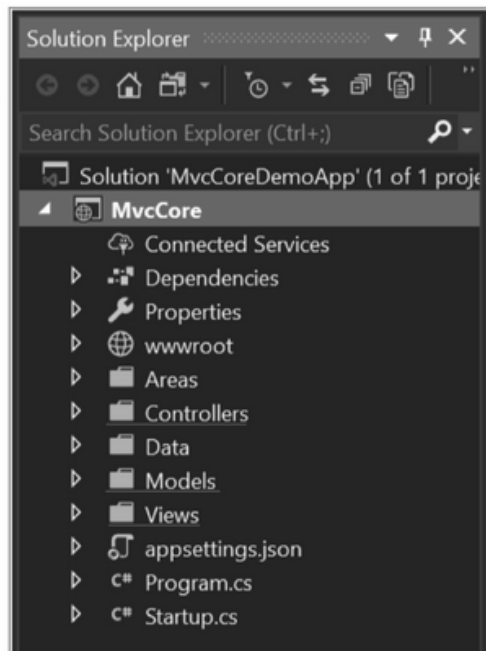
ASP.Net Core MVC



ASP.Net Core MVC Tutorial

- VDO Clip: ASP.Net Core Introduction: <https://www.youtube.com/watch?v=1ck9LIBxQ14>
- ASP.Net Core Web Tutorial:
 - <https://asp.mvc-tutorial.com/>
 - <https://www.tektutorialshub.com/asp-net-core-tutorial/>

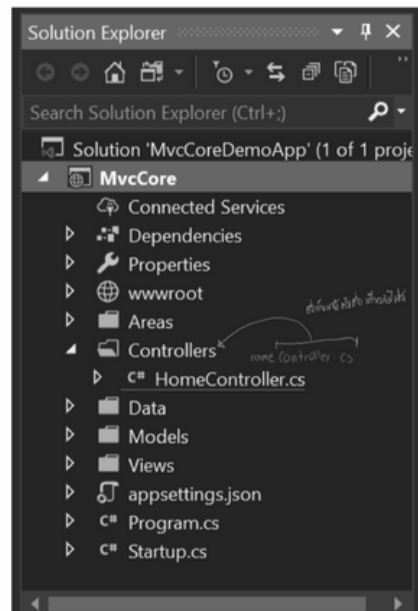
Project Structure



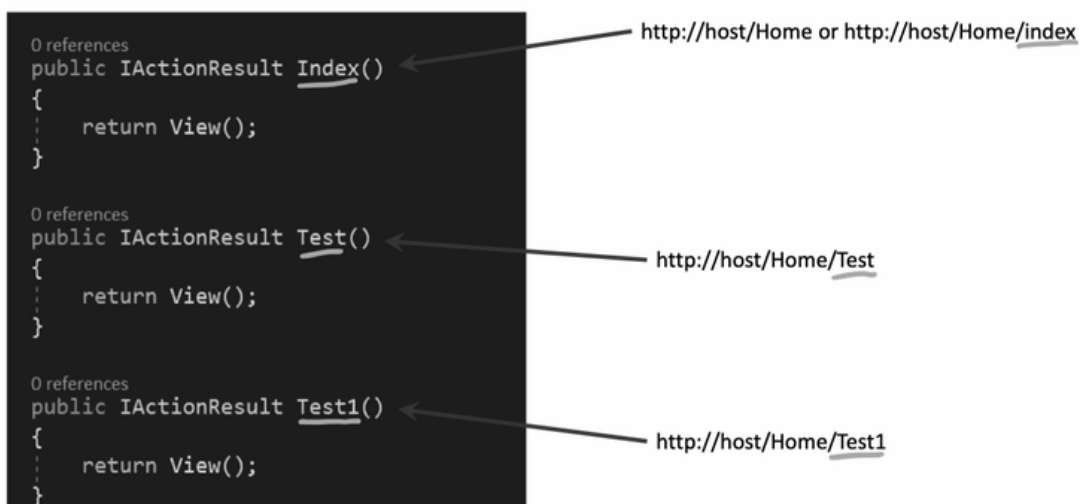
Controller

การกำหนดเส้นทางไปยังทรัพยากร

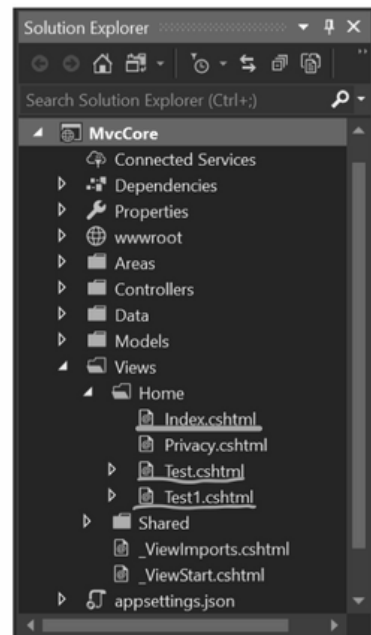
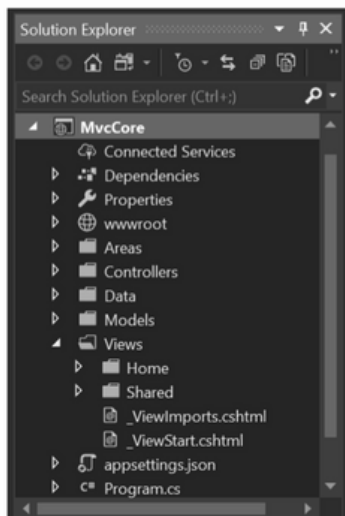
- **Routing to resource**
 - From picture there is 1 (virtual) directory in this web site
 - Named: Home
 - <http://host/home>
 - ใน HomeController.cs มีวิธีการแสดงผลหน้า
- In HomeController.cs contain methods for rendering pages
 - Each method correspond to 1 page



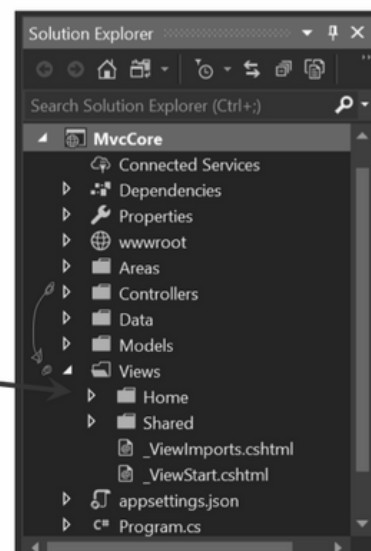
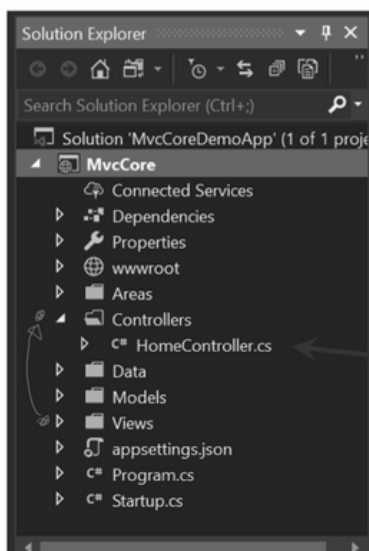
HomeController.cs



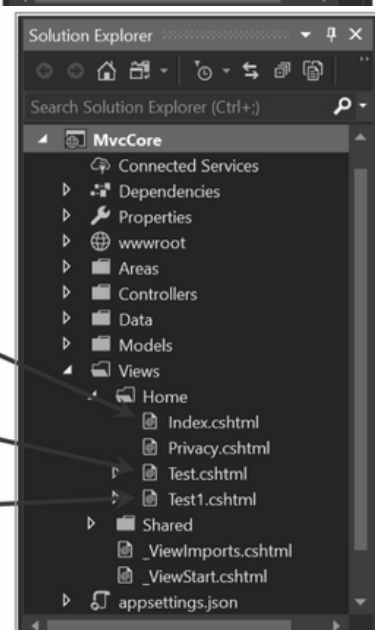
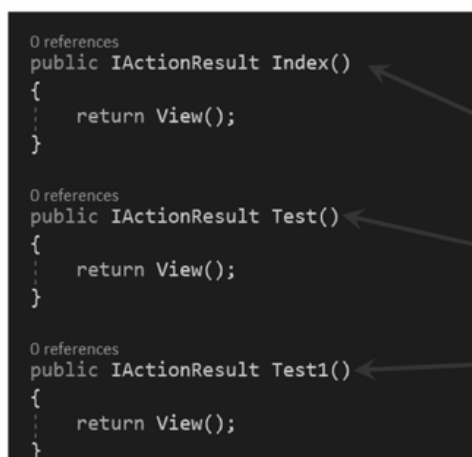
View



Controller - View



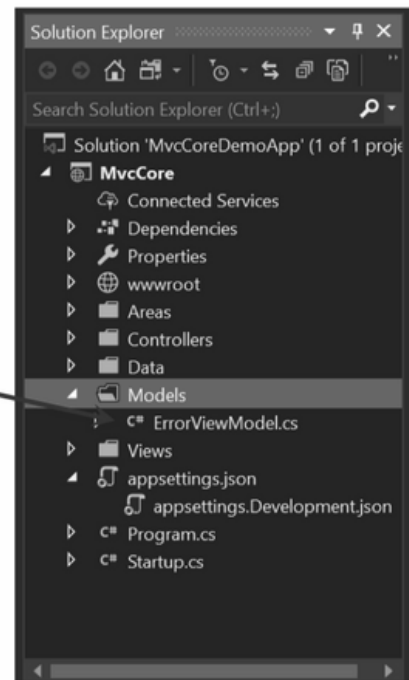
Controller - View



Model

```
namespace MvcCore.Models
{
    4 references
    public class ErrorViewModel
    {
        3 references
        public string RequestId { get; set; }

        1 reference
        public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);
    }
}
```



Controller-Model-View

```
public IActionResult Error()
{
    return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
}
```

HomeController.cs (Controller)

```
<h1 class="text-danger">Error.</h1>
<h2 class="text-danger">An error occurred while processing your request.</h2>

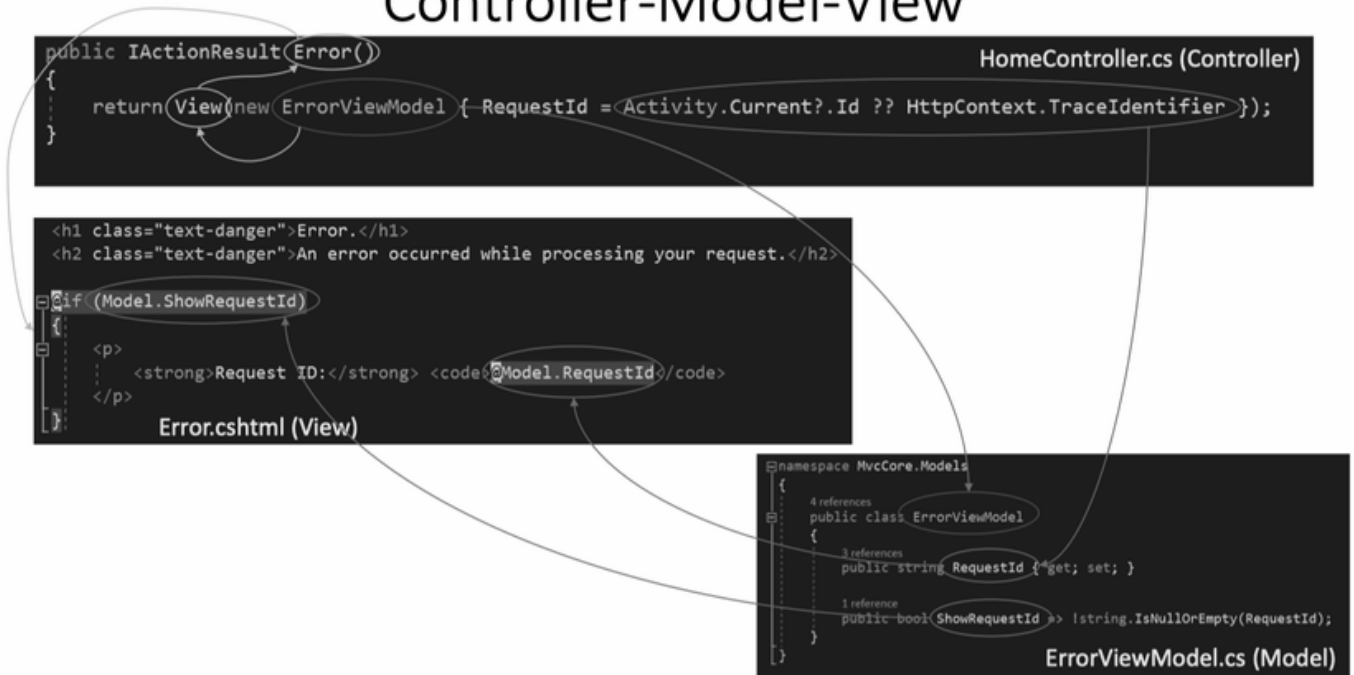
@if (Model.ShowRequestId)
{
    <p>
        <strong>Request ID:</strong> <code>Model.RequestId</code>
    </p>
}
```

Error.cshtml (View)

```
namespace MvcCore.Models
{
    4 references
    public class ErrorViewModel
    {
        3 references
        public string RequestId { get; set; }

        1 reference
        public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);
    }
}
```

ErrorViewModel.cs (Model)



แบบ #1 - Controller

ASP.Net Core MVC Handling Data

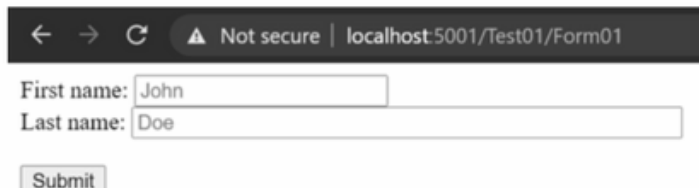
Lesson09_DataHandling : 1

```
0 references
public IActionResult Form01()
{
    return View();
}

2
0 references
public IActionResult Form01Handle(string fname, string lname)
{
    ViewBag.fname = fname;
    ViewBag.lname = lname;
    return View();
}
```

แบบ #1 - View – Display Form

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <form method="post" asp-controller="Test01" asp-action="Form01Handle">
        First name: <input type="text" id="fname" name="fname" placeholder="John"><br>
        <label for="lname">Last name:</label>
        <input type="text" id="lname" name="lname" placeholder="Doe" size="50"><br><br>
        <input type="submit">
    </form>
</body>
</html>
```



← → ↻ ⚠ Not secure | localhost:5001/Test01/Form01

First name:

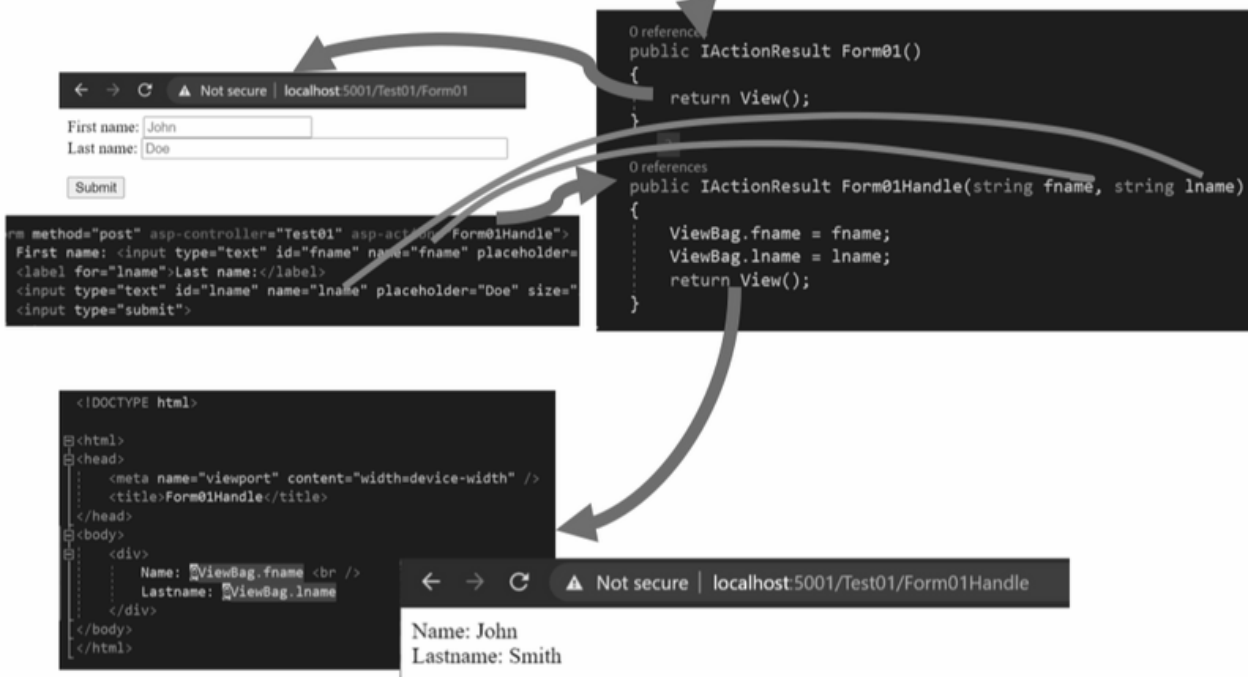
Last name:

แบบ #1 – View – Submit

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Form01Handle</title>
</head>
<body>
    <div>
        Name: @ViewBag.fname <br />
        Lastname: @ViewBag.lname
    </div>
</body>
</html>
```

แบบ #1 Conclusion

<http://localhost:5001/Test01/Form01>



แบบ #2 – Controller

```
0 references
public IActionResult Index()
{
    return View();
}

[HttpPost]
0 references
public IActionResult Index(string fname, string lname)
{
    //ViewBag.Message = fname + " " + lname;
    FLClass person = new FLClass();
    person.FName = fname;
    person.LName = lname;

    return View(person);
}
```

2 references

แบบ #2 – Model

```

public class FLClass
{
    1 reference
    public string FName { get; set; }
    1 reference
    public string LName { get; set; }
}

```

แบบ #2 – View

```

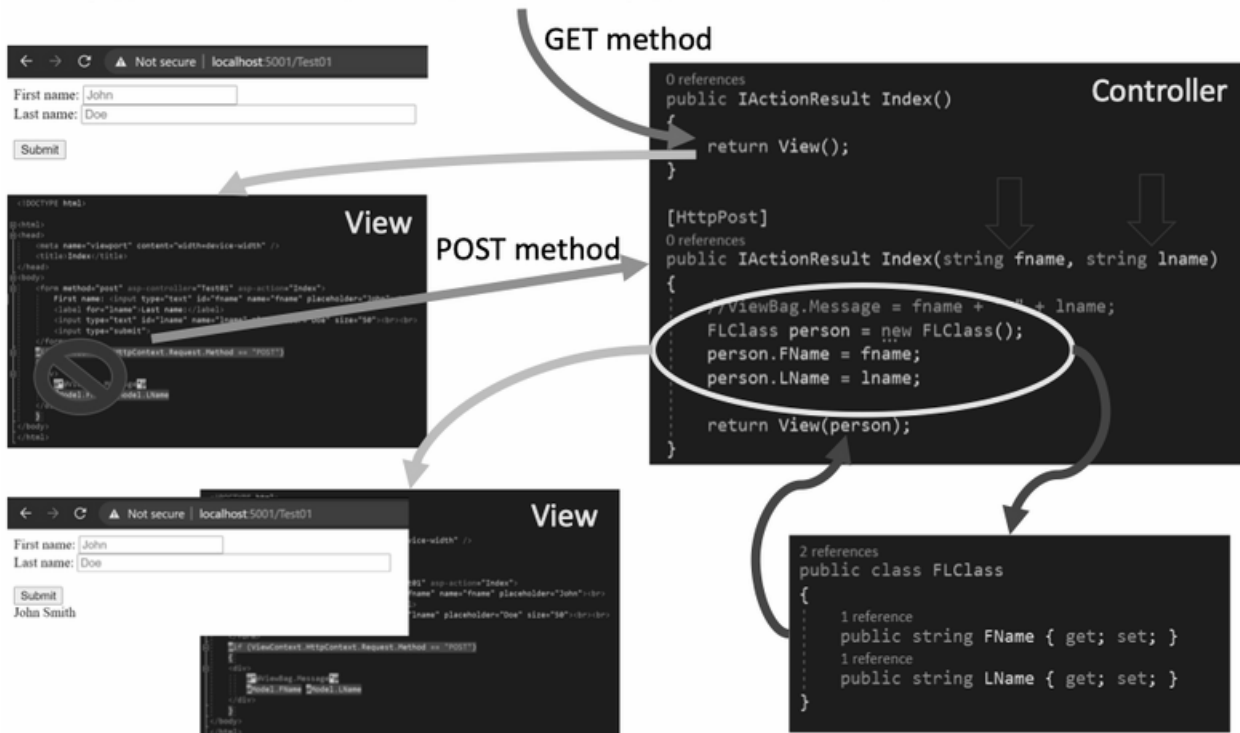
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width" />
<title>Index</title>
</head>
<body>
<form method="post" asp-controller="Test01" asp-action="Index">
First name: <input type="text" id="fname" name="fname" placeholder="John"><br>
<label for="lname">Last name:</label>
<input type="text" id="lname" name="lname" placeholder="Doe" size="50"><br><br>
<input type="submit">
</form>
<div>
<div>
@ViewBag.Message
@Model.FName @Model.LName
</div>
</div>
</body>
</html>

```

John Smith

แบบ #2 – Conclusion

<http://localhost:5001/Test01/Index> or <http://localhost:5001/Test01>



GET Method data handling

Controller

http://localhost:5001/Test01/Index/?fname=Jack
or http://localhost:5001/Test01/?fname=Jack

```
0 references
public IActionResult Index(string fname)
{
    ViewBag.Message = fname;
    return View();
}
```

View GET Method data handling

```
<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>Index</title>
</head>
<body>
  <form method="post" asp-controller="Test01" asp-action="Index">
    First name: <input type="text" id="fname" name="fname" placeholder="John"><br>
    <label for="lname">Last name:</label>
    <input type="text" id="lname" name="lname" placeholder="Doe" size="50"><br><br>
    <input type="submit">
    <div>@ViewBag.Message</div>
  </form>
</body>
</html>
```

← → ↻ ⚠ Not secure | localhost:5001/Test01/?fname=Jack

First name:

Last name:

Jack

Controller return

- View()
 - Web page -> html
 - View file in Views folder will be used to render response
- Content()
 - Can be anything that MIME supports

Example:

```
[Produces("text/plain")]
0 references
public IActionResult Index(string fname)
{
    //ViewBag.Message = fname;
    string x = "The received name is " + fname;
    return Content(x, "text/plain", Encoding.UTF8);
}
```

← → ↻ 🔒 localhost:5001/Test01/?fname=Jack

The received name is Jack

← → ↻ ⓘ view-source:https://localhost:5001/Test01/?fname=Jack

Line wrap ☐

1 The received name is Jack

Consume http

```
0 references
public IActionResult TestConsume()
{
    string output = "";
    HttpClient client = new HttpClient();
    client.BaseAddress = new Uri("https://localhost:5001/");
    var rtask = client.GetAsync("Test01/?fname=George");
    rtask.Wait();

    var result = rtask.Result;
    if (result.IsSuccessStatusCode)
    {
        var rcontent = result.Content.ReadAsStringAsync();
        rcontent.Wait();

        output = rcontent.Result;
    }
    return Content(output, "text/plain", Encoding.UTF8);
}
```

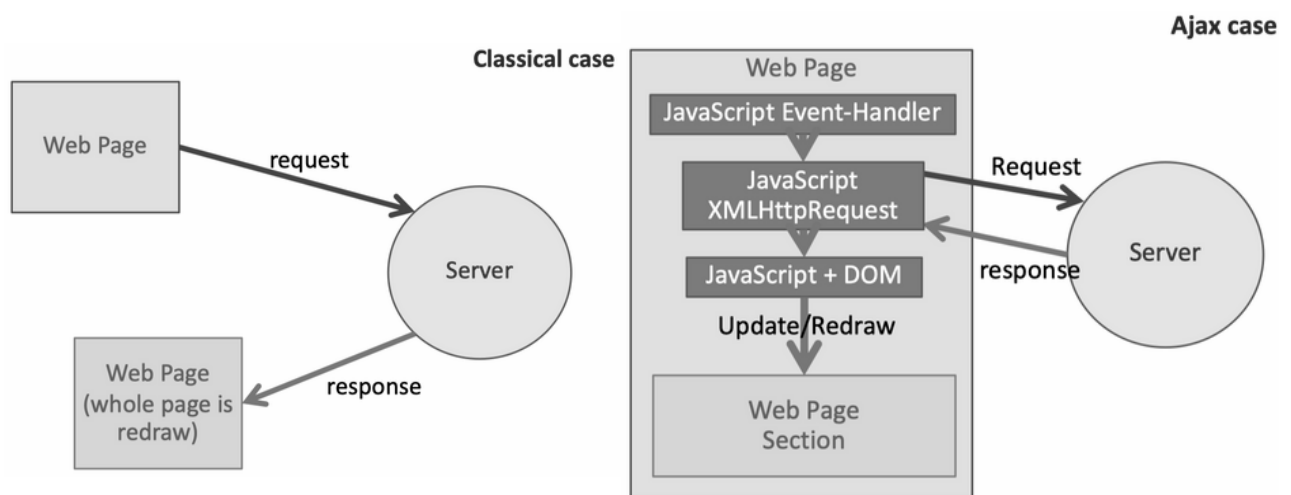
on Jack in George

Asynchronous JavaScript and XML

เทคโนโลยีขึ้นอยู่กับ:

- A technology based on:
 - JavaScript
 - DOM
 - Server side script
- Concept: **Client-Driven Communication, Server-Side Processing**

• แนวคิด: การสื่อสารที่ขับเคลื่อนด้วยไคลเอนต์ การประมวลผลฝั่งเซิร์ฟเวอร์



Client-Side Communication

การสื่อสารฝั่งไคลเอนต์

- Achieved by
 - XMLHttpRequest Object
 - XMLHttpRequest Object Methods

Method	Description
abort()	Cancels the current request ยกเลิก คำขอ ปัจจุบัน
getAllResponseHeaders()	Returns all HTTP headers as a String type variable ส่งคืน ส่วนหัว HTTP ทั้งหมด เป็น ตัวแปร ประเภท สตริง
getResponseHeader()	Returns the value of the HTTP header specified in the method คืนค่า ของ ส่วนหัว HTTP ที่ระบุ ใน เมธอด
open()	Specifies the different attributes necessary to make a connection to the server; allows you to make selections such as GET or POST (more on that later), whether to connect asynchronously, and which URL to connect to ระบุแอตทริบิวต์ต่างๆ ที่จำเป็นในการเชื่อมต่อแอตเตอร์เซิร์ฟเวอร์; ให้คุณทำการเลือก เช่น GET หรือ POST (เพิ่มเติมในภายหลัง)ว่าจะเชื่อมต่อแบบอะซิงโครนัสหรือไม่ และ URL ใดเพื่อเชื่อมต่อ
setRequestHeader()	Adds a label/value pair to the header when sent กับ เพิ่ม คู่ป้ายกำกับ/ค่า ให้กับ ส่วนหัว เมื่อส่ง
send()	Sends the current request ส่งคำขอปัจจุบัน

• XMLHttpRequest Object Properties

Property	Description
onreadystatechange	ใช้เป็นตัวจัดการเหตุการณ์สำหรับเหตุการณ์ที่ trigger เมื่อมีการเปลี่ยนแปลง Used as an event handler for events that trigger upon state changes
readyState	สถานะมีสถานะปัจจุบันของวัตถุ (0:ไม่ได้ กำหนดค่าเริ่มต้น, 1:กำลังโหลด 2:กำลังโหลด, 3:โต้ตอบ, 4: สมบูรณ์) Contains the current state of the object (0: uninitialized, 1: loading, 2: loaded, 3: interactive, 4: complete)
responseText	Returns the response in string format ส่งกลับการตอบสนอง ในรูปแบบสตริง
responseXML	Returns the response in proper XML format ส่งกลับการตอบสนองในรูปแบบ XML ที่เหมาะสม
status	Returns the status of the request in numerical format (regular page errors are returned, such as the number 404, which refers to a not found error) ส่งกลับสถานะของคำขอในรูปแบบตัวเลข (ส่งคืนข้อผิดพลาดของหน้าปกติ เช่น ตัวเลข 404 ซึ่งอ้างถึงข้อผิดพลาดที่ไม่พบ)
statusText	Returns the status of the request, but in string format (e.g., a 404 error would return the string Not Found)

ส่งกลับสถานะของคำขอแต่อยู่ในรูปแบบ สตริง (เช่น ข้อผิดพลาด 404 ข้อความจะส่งคืนสตริงไม่พบ)

• Cross Browser Usage

– IE and non-IE

ใช้เพื่อว่าเป็น Internet Explorer หรือไม่ใช่

```
var xmlhttp = false;

try {
    xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
    alert ("You are using Microsoft Internet Explorer.");
} catch (e) {
    xmlhttp = false;
}

if (!xmlhttp && typeof XMLHttpRequest != 'undefined') {
    xmlhttp = new XMLHttpRequest();
    alert ("You are not using Microsoft Internet Explorer");
}
```

```
var xmlhttp;
```

```
if (window.ActiveXObject) {
    xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
} else {
    xmlhttp = new XMLHttpRequest();
}
```

• Sending a request to Server

```
function makerequest(serverPage, objID) {
    var obj = document.getElementById(objID);
    xmlhttp.open("GET", serverPage);
    xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState == 4 && xmlhttp.status == 200)
        {
            obj.innerHTML = xmlhttp.responseText;
        }
    }
    xmlhttp.send(null);
}
```

.innerHTML มีประโยชน์สำหรับการส่งคืนหรือแทนที่เนื้อหาขององค์ประกอบ HTML
The **.innerHTML** is useful for returning or replacing the content of HTML elements

****innerHTML ไม่ใช่นับหนึ่งของ W3C DOM แต่รองรับเบราว์เซอร์หลักทั้งหมด****
****innerHTML is not part of W3C DOM but it is supported by all major browser****


```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

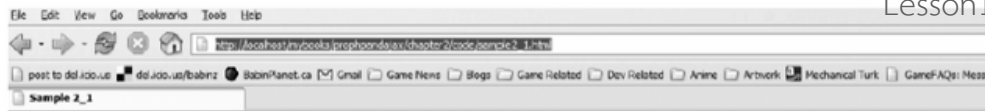
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Sample 2_1</title>
    <meta http-equiv="Content-Type"
      content="text/html; charset=iso-8859-1" />
    <script type="text/javascript">
      <!--
        var xmlhttp = false;
        try {
          xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
          alert ("You are using MS IE.");
        } catch (e) {
          xmlhttp = false;
        }

        if (!xmlhttp && typeof XMLHttpRequest != 'undefined') {
          xmlhttp = new XMLHttpRequest();
          alert ("You are not using MS IE");
        }

        function makerequest(serverPage, objID) {
          var obj = document.getElementById(objID);
          xmlhttp.open("GET", serverPage);
          xmlhttp.onreadystatechange = function() {
            if (xmlhttp.readyState == 4 && xmlhttp.status == 200)
            {
              obj.innerHTML = xmlhttp.responseText;
            }
          }
          xmlhttp.send(null);
        }
      //-->
    </script>
  </head>

  <body onload="makerequest ('content1.html','hw')">
    <div align="center">
      <h1>My Webpage</h1>
      <a href="content1.html" onclick=
        "makerequest('content1.html','hw');
        return false;"> Page 1</a> |
      <a href="content2.html" onclick=
        "makerequest('content2.html','hw');
        return false;">Page 2</a> |
      <a href="content3.html" onclick=
        "makerequest('content3.html','hw');
        return false;">Page 3</a> |
      <a href="content4.html" onclick=
        "makerequest('content4.html','hw');
        return false;">Page 4</a>
      <div id="hw"></div>
    </div>
  </body>
</html>

```



My Webpage

Page 1 | Page 2 | Page 3 | Page 4

Page 3

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

ASP.Net Core MVC & Ajax Example Controller

```
0 references
public IActionResult Index()
{
    return View();
}

0 references
public IActionResult Message01()
{
    string msg = "Hello, This is a test Ajax message";
    return Content(msg, "text/html", Encoding.UTF8);
}

2
0 references
public IActionResult Message02()
{
    string msg = "<ul><li>topic01</li><li>topic02</li><li>topic03</li><li>topic04</li><li>topic05</li></ul>";
    return Content(msg, "text/plain", Encoding.UTF8);
}
```

ASP.Net Core MVC & Ajax Example

```
<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>Ajax testing</title>
  <script>
    var xmlhttp = new XMLHttpRequest();
    ...
    function MakeReq(serverpage, objID) {
      let obj = document.getElementById(objID);
      xmlhttp.open("GET", serverpage);
      xmlhttp.onreadystatechange = function () {
        if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
          obj.innerHTML = xmlhttp.responseText;
        }
      }
      xmlhttp.send(null);
    }
  </script>
</head>
<body>
  <div>
    <h2>Part I</h2>
    click here ==>
    <a href="" onclick="MakeReq('https://localhost:5001/Ajax/Message01', 'p01'); return false;">Hello</a>
    <div id="p01"></div>
  </div>
  <div>
    <h2>Part II</h2>
    click here ==>
    <a href="" onclick="MakeReq('https://localhost:5001/Ajax/Message02', 'p02'); return false;">Test</a>
    <div id="p02"></div>
  </div>
</body>
</html>
```

View