Data: จำวิดีโอprepare (worksheetออกบ้าง แต่น่าจะไม่ต้องท่องไป)

CPU: น้าจะออกแต่ Brookshear

Linux: ท่องคำสั่งใน worksheet

Embedded: ไม่รู้

Frontend: อ่านไอหน้าที่จารอติวงศ์ทำให้เพื่อไปตอบ prepare

เก็ง

- DOM คืออะไร
- หน้าที่คนทำ frontend คือ
- ถามว่า html css js อันไหนไว้ทำอะไร
- วิธีเชื่อม css กับ html อีกอัน

Backend: คิดว่าออกในworksheet + พวกIAAS

ชัญญา สิทธินันทวิทย์

รับบริจาคค่าขนม

ให้กำลังใจสรุป(หนังสือ)ของพี่ 🥺

Binary representations of data

คอมพิวเตอร์เก็บข้อมูลเป็น bit patterns (มีค่าเป็นได้แค่ 2 ค่า เช่น 0-1, เปิด-ปิด) ผ่านการ encoding (data -> bits), decoding (bits -> data)

เก็บข้อมูลได้จากหลายที่เช่น mechanical switches, vacuum tubes, electronics devices, lights, magnetic

ถ้าข้อมูล bit ยาว N ตัว จะเก็บข้อมูลได้ 2^N ตัว

Hex digit

แต่การเก็บข้อมูลเป็น bit (ฐาน 2) คนจะอ่านยาก เลยมีการแปลงจาก bits เป็น hexadecimal (ฐาน 16) เพื่อให้อ่าน & จัดการง่ายขึ้น -> 4 bits = 1 hex digit

โดยมี 16 ค่า ตั้งแต่ 0-9 & A-F (แทน 10-15)

แปลง Bit -> String

แต่ละอักขระจะถูกแปลงเป็นตัวเลข

วิธีการเข้ารหัสที่นิยม เช่น

- ASCII : 1 char = 8 bits, สามารถเก็บได้แค่ 256 ตัว
- UTF-8 : 1 char = 8 / 16 / 24 / 32 bits แล้วแต่ภาษา, ภาษาไทยใช้ 24 bits (3 bytes) ในการเก็บ

เช่น ก = E0 B8 81, space = 20 (1 byte)

Two's complement

การเข้ารหัสเลขจำนวนเต็ม ที่มีทั้งจำนวนบวก – ลบ

ถ้าข้อมูล bit ยาว N ตัว จะเก็บข้อมูล $-2^{(N-1)}$ ถึง $2^{(N-1)} - 1$

- การแปลงจาก dec -> two's complement

จำนวนบวก & ศูนย์ -> แปลงเหมือนฐาน 2 ปกติ

จำนวนลบ

1. ให้แปลงค่า absolute ของจำนวนเป็นฐาน 2
2. สลับค่าของทุก bit (1 เป็น 0, 0 เป็น 1)
3. บวก 1

- การแปลงจาก two's complement -> dec

หลักแรกเรียกว่า Sign bit -> 0 เป็นบวก-ศูนย์, 1 เป็นลบ

จำนวนบวก & ศูนย์ -> แปลงเหมือนฐาน 2 ปกติ

จำนวนลบ

1. สลับค่าของทุก bit (1 เป็น 0, 0 เป็น 1)
2. บวก 1
3. แปลงเป็นฐาน 2 ปกติ
4. คูณ -1 (เติมติดลบข้างหน้า)

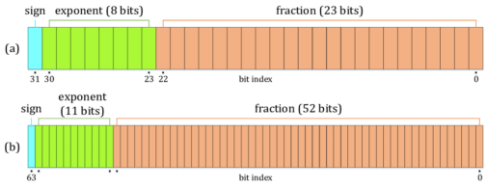
Excess-K / Bias-K

เป็นวิธีเก็บจำนวนเต็มคล้ายๆ ฐาน 2 แต่ขยับช่วงไป K ตัว ถ้าข้อมูล bit ยาว N ตัว จะเก็บข้อมูล -K ถึง $2^N - 1 - K$

ถ้าแปลงค่า x ฐาน 10 เป็น Excess-K จะได้ x + K แล้วแปลงเป็นฐาน 2 เช่น -3 in Excess-4 is -3+4 = 1 -> 001

การแปลงกลับเป็นฐาน 10 ให้แปลงเหมือนเลขฐาน 2 ปกติแล้วลบด้วย K เช่น 111 in Excess-4 = 7-4 = 3

IEEE 754



การเข้ารหัสจำนวนทศนิยม -> เลขจะถูกแบ่งเป็น 3 ช่วง คือ Sign bit, exponent, fraction

ถ้าเป็น single precision ใช้ 32 bits -> แบ่งเป็น 1, 8, 23

ถ้าเป็น double precision ใช้ 64 bits -> แบ่งเป็น 1, 11, 52

- Sign bit : 0 เป็นบวก & ศูนย์, 1 เป็นลบ
- fraction & exponent แปลงโดย

1. แปลงเลขทศนิยมเป็นฐาน 2

   วิธี 1 - เอาเลขฐาน 10 มาดูว่าลบด้วย 2** เท่าไหร่ได้มากที่สุด ไล่ไปเรื่อยๆ จนกว่าจะลบไม่ได้แล้ว

   2**-1 = 0.5 2**-2 = 0.25

   2**-3 = 0.125 2**-4 = 0.0625

   2**-5 = 0.03125 2**-6 = 0.015625

   2**-7 = 0.0078125

   วิธี 2 – แปลงตัวหน้าทศนิยมแบบฐานสองปกติ แล้วจะแปลงส่วนทศนิยมโดย เอาคูณ 2 ไปเรื่อยๆ พอเกิน 1 แล้ว ก่อนจะคูณตัวต่อไปให้ตัด 1 ทิ้ง แล้วคูณวนไปเรื่อยๆ จนกว่าจะพอใจ แล้วเอาเลขด้านหน้าทศนิยมแต่ละตัวมาเรียงกัน

   เช่น แปลง 3.14 จะได้ $2^1 + 2^0 + 0.14$

   0.14 จะได้ 0.28 -> 0.56 -> 1.12 -> 0.24 -> 0.48 -> 0.96 -> 1.92 -> 1.84 จะได้ 00100011

   เอามารวมกันจะได้ 3.14 = 11.00100011

2. เขียนเป็นสัญยากรณ์วิทยาศาสตร์ (1.xxxxxx * 2^N) xxxxxx = fraction (ไม่เอา 1. แรก), N = exponent

   เลื่อนขวาเลขยกกำลังติดลบ

3. เปลี่ยน N (exponent) เป็น Excess-127

เช่น 0.171875 = 0.125 + 0.03125 + 0.015625

= $2^{-3} + 2^{-5} + 2^{-6}$ -> 0.001011 ->

1.011 * 2^{-3} -> N = -3 -> -3+127 = 124 = 64+32+16+8+4 -> 0111 1100

Hex editor

บอกตำแหน่งด้วย 0x?? เป็นคอลัมน์+แถว (0 บอกว่าเป็นค่าคงที่, x บอกว่าข้างหลังเป็นฐาน 16, ?? เป็นเลขฐาน 16)

ดูขนาดไฟล์ได้โดยนับจำนวน byte ทั้งหมดในไฟล์หรือดูใน template > size

Bitcount -> ใช้บอกว่า 1 ตัว / pixel ใช้กี่ byte

ค่าที่เก็บใน Hex editor จะเก็บแบบ little endian -> เก็บแบบย้อนกลับ เช่นในไฟล์เขียน 00 10 59 30 จะต้องกลับค่าเป็น 30 59 10 00 ก่อนแปลงกลับเป็นฐาน 10

Image

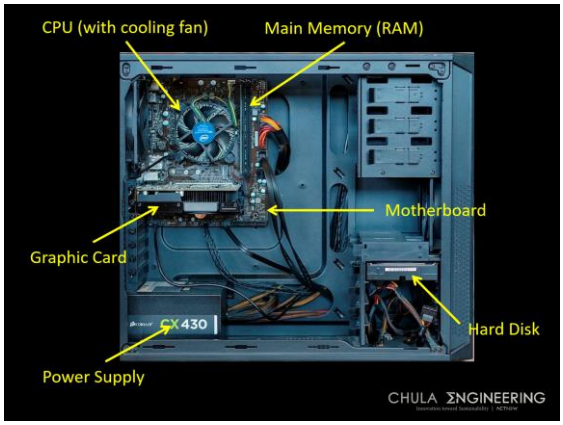
จะเก็บค่าเป็น 2 ส่วน คือ header, content ใน header ใช้บอกข้อมูลต่างๆ เช่นขนาดไฟล์ ขนาดภาพ ส่วนใน content จะเป็นสีของแต่ละ pixel (จำนวน byte ตาม bitcount) -> เก็บแบบ RGB (แปลงแบบ little endian ด้วย)

Game

เครื่องเกมสมัยก่อนจะแบ่งเป็น 2 ส่วน คือเครื่องเกม (RAM 8 KB : address ตั้งแต่ 0x0000 – 0x7FFF) กับแผ่น (ROM : address ตั้งแต่ 0x8000 เป็นต้นไป)

ในเกมก็จะเก็บค่าเป็นข้อมูลแต่ละตัว เช่น สีของผม, สีของเสื้อ, state ของตัวละคน, ตัวหนังสือแต่ละตัวใน UI

CPU (Central Processing Unit)



คอมพิวเตอร์ 1 เครื่องประกอบไปด้วย

1. Motherboard
   - CPU : หน่วยประมวลผลกลาง มักมาพร้อมกับ Cooling fan (เพราะทำงานเร็วจนร้อน เลยต้องทำค.เย็น)
   - Main memory (RAM) : เก็บโปรแกรม & ข้อมูลเพื่อใช้ประมวลผล (ถ้าปิดเครื่องไป ข้อมูลใน RAM จะหายไป)
   - อุปกรณ์อื่นๆ ที่ช่วยในการทำงานของ CPU, RAM

2. GPU (Graphic Card) : เป็นอุปกรณ์เสริม ช่วยประมวลผลทางกราฟฟิก & A.I. (ไม่มีก็ได้)

3. Hard disk : หน่วยความจำสำรอง ใช้เก็บข้อมูลเวลาปิดเครื่อง

4. Power supply : จ่ายกระแสไฟ เพื่อเลี้ยงอุปกรณ์ทั้งหมด

CPU มีหน้าที่ประมวลผลข้อมูลทั้งหมด ตามคำสั่ง Machine language ที่ถูกประกอบกันจนเป็นโปรแกรม

บาง CPU มีหลาย Cores (Processing units) ทำงานพร้อมกัน

บาง CPU จะมีหน่วยประมวลผลพิเศษ สำหรับประมวลเฉพาะทาง เช่น GPU (ด้านกราฟฟิก), Neural engine (ด้านเอไอ)

CPU จะถูกบรรจุอยู่ใน Package โดยประกอบด้วย

1. die : บรรจุ transistors & วงจร

ภายใน CPU จะประกอบด้วย Transistors หลายพันล้านตัว ทำงานร่วมกันตามจังหวะ "Clock cycle"

ยิ่ง CPU มี Clock cycle มาก (ถี่มาก) ก็ยิ่งเร็ว

2. pins : ขาไว้ต่อ die กับอุปกรณ์ภายนอก

ยิ่งมี pin มาก ยิ่งส่งสัญญาณได้เยอะ = เร็ว

รุ่น CPU เด่นๆ เช่น

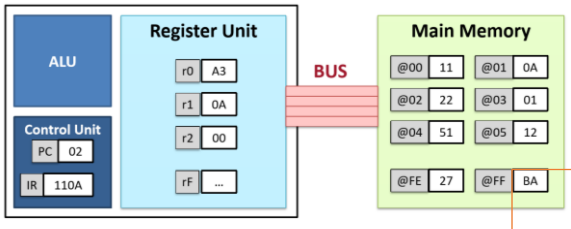
- 1978 - Intel 8086 : เริ่มต้นมาตรฐานการใช้ instruction set
- July 2019 - AMD Zen 2 / AMD RDNA 2 (Sony PS5) : เริ่มมี GPU ในตัว
- Sep 2020 - Nvidia GA102 (Ampere) : ผลิตมาไว้ใช้ทางด้าน Graphic & AI โดยเฉพาะ
- Sep 2020 - Apple S8 (Apple waych iltra) : มีขนาดเล็กใช้กับนาฬิกา

ขนาด 1 transistor ยิ่งเล็กยิ่งดี

- Apple M2 Ultra : มี 1.34 แสนล้าน stransistor (5nm)

24 CPU + 32 Neural + 76 GPU cores

การทำงาน



ตาม "Von Neumann architecture" ใน CPU จะมีองค์ประกอบในการทำงานหลักอยู่ 3 ส่วน คือ

1. CPU : มี 3 ส่วนย่อย คือ
   - ALU : ประมวลผลข้อมูล บวกลบคูณหาร ต่างๆ (สามารถดึงข้อมูลได้จาก Register unit เท่านั้น)

มี "Load-Store Architecture" เพื่อเคลื่อนย้ายข้อมูลจาก Main memory ↔ Register unit ↔ ALU

- Control unit : เป็นตัวควบคุมการทำงาน ของส่วนอื่นๆ ใน CPU

- Register unit : Memory ภายใน CPU ที่มีความเร็วเท่ากับส่วนอื่นๆ ใน CPU ทำหน้าที่ดึงข้อมูลจาก Main memory มาเก็บไว้ก่อนทำงาน (เพราะ Main memory ทำงานช้ามาก)

2. Main memory : เก็บ computer code, program, data

3. BUS : เชื่อมต่อ CPU และ Main memory เพื่อให้ CPU ดึงข้อมูลไปประมวลผล และส่งกลับมาเก็บที่ memory

CPU จะทำงานได้เมื่อมี Instruction set / Machine language (ต่างกันตามการออกแบบของแต่ละรุ่น-ยี่ห้อ) ซึ่งเป็นภาษาแบบ low-level (เครื่องทำงานตามได้โดยตรง ไม่ต้องตีความ เลยเร็ว แต่คนอ่าน - เขียนยาก)

> High-programming language เช่น python, java, c++ จะเขียนง่าย แต่เครื่องต้องแปลงเป็น machine language ก่อนถึงจะทำงานได้ โดยแปลงได้ 2 แบบ คือ compiled (แปลงก่อนแล้วเก็บเป็นไฟล์ใหม่ ค่อยใช้), interpreted (แปลงตอนจะใช้งาน / ทำงาน)

Brookshere's Simple Machine เป็นแบบจำลอง CPU ขนาดเล็กมาก ที่มี Main memory (256 bytes -> Address ละ 1 byte ตั้งแต่ @00-@FF), General-purpose register unit (16 ตัว -> ตั้งแต่ r0-rF) , $PC (Program Counter -> 1 byte -> เก็บตำแหน่งของ memory ที่จะเอามาประมวลผลตัวต่อไป), $IR (Instruction Register -> 2 bytes -> เก็บคำสั่งที่ประมวลผลอยู่, ดึงค่ามาจาก $PC)

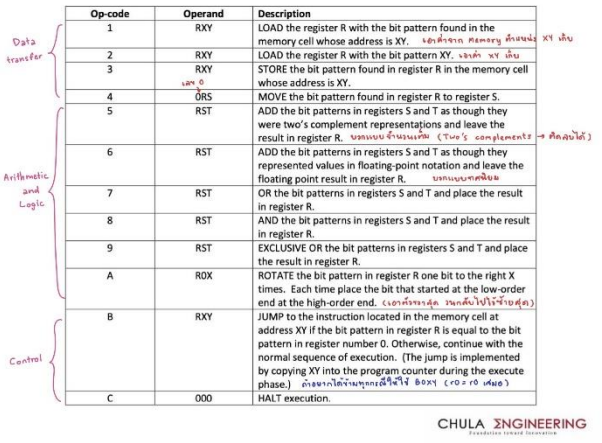
การทำงานจะเรียกว่า "Machine Cycle" (แบ่งเป็น 3 step)

1. Fetch : ดึงคำสั่งจาก Main memory ที่ถูกชี้ตำแหน่งใน $PC ทั้งหมด 2 byte (ตัวมันกับตัวถัดไป) เอาไปเก็บใน $IR แล้วเพิ่มค่าใน $PC ไป 2 (ชี้คำสั่งต่อไป)
2. Decode : ตีความว่าค่า bit pattern ใน $IR คือคำสั่งอะไร
3. Execute : Control unit สั่งให้ส่วนต่างๆ ทำงานตามคำสั่งที่ Decode มา

เมื่อจบรอบ จะวนกลับไปทำใหม่เรื่อยๆ จนกว่าจะเจอคำสั่ง HALT จึงจะหยุดการทำงาน

> ถ้าคำสั่งผิด จะแจ้งว่า illegal instruction หลังจาก decode

คำสั่ง (Instuctions) จะถูกเก็บเป็น 2 bytes แบ่งเป็น Op-code (Operation code) : 4 bits -> บอกว่าเป็นคำสั่งอะไร และ Operand : 12 bits -> บอกรายละเอียดของคำสั่งนั้นๆ



	Op-code	Operand	Description
Data transfer	1	RXY	LOAD the register R with the bit pattern found in the memory cell whose address is XY.
	2	RXY	LOAD the register R with the bit pattern XY.
	3	RXY	STORE the bit pattern found in register R in the memory cell whose address is XY.
	4	0RS	MOVE the bit pattern found in register R to register S.
Arithmetic and Logic	5	RST	ADD the bit patterns in registers S and T as though they were two's complement representations and leave the result in register R.
	6	RST	ADD the bit patterns in registers S and T as though they represented values in floating-point notation and leave the floating point result in register R.
	7	RST	OR the bit patterns in registers S and T and place the result in register R.
	8	RST	AND the bit patterns in registers S and T and place the result in register R.
	9	RST	EXCLUSIVE OR the bit patterns in registers S and T and place the result in register R.
	A	R0X	ROTATE the bit pattern in register R one bit to the right X times. Each time place the bit that started at the low-order end at the high-order end.
Control	B	RXY	JUMP to the instruction located in the memory cell at address XY if the bit pattern in register R is equal to the bit pattern in register number 0. Otherwise, continue with the normal sequence of execution. (The jump is implemented by copying XY into the program counter during the execute phase.)
	C	000	HALT execution.

CHULA ENGINEERING

Operating System เป็นโปรแกรมที่ใช้จัดการ hardware & software อื่นๆ ในเครื่องคอม, เป็นสื่อกลางในการติดต่อระบบ-รันโปรแกรม เพื่อให้ผู้ใช้ใช้งานง่ายขึ้น เช่น unix (1970 ต้นแบบของ linux ฯลฯ), linux, window, macOS, BSD

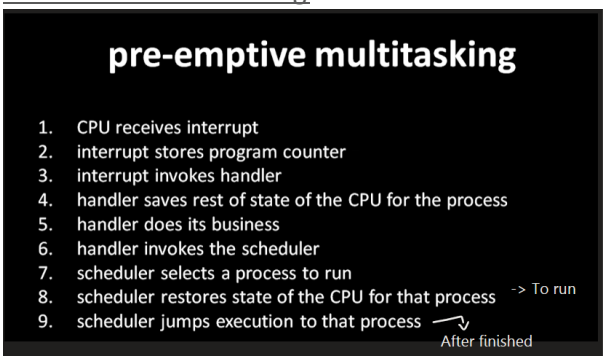
หน้าที่

1. จัดการการรันโปรแกรมต่างๆ (process = การรันโปรแกรมอื่นๆ)
2. ให้เราติดต่อ hardware ผ่าน "system call"

> "Device drivers" เป็น plug-in module ที่ใช้จัดการ i/o device ต่างๆ บางอุปกรณ์ใช้ driver ทั่วไปได้ (เช่น USB mouse) แต่ส่วนใหญ่ต้องลง driver พิเศษของรุ่น / อุปกรณ์ก่อนใช้

3. จัดการ filesystem -> process เข้าถึงไฟล์ได้ง่าย
4. สร้าง graphic user interface ให้ user ใช้งานง่าย

Process's time sharing



Brookshere's simple machine เป็น CPU อย่างง่ายที่ทำได้ทีละอย่าง แต่ CPU ปกติจะมีหลาย core ที่รัน process ต่างๆ สลับกัน -> จัดการโดย OS -> ผู้ใช้รู้สึกเหมือนรันทุกอย่างพร้อมกัน

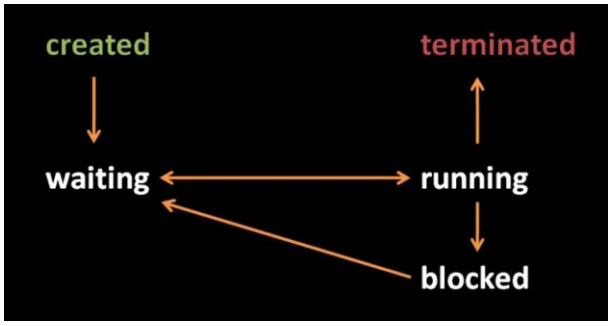
1 Core ทำงานได้ 1 อย่าง (1 process หรือ OS อย่างใดอย่างหนึ่ง) และ 1 process รันได้บน 1 core เท่านั้น (ในเวลาเดียวกัน) เลยมีตัวจัดการชื่อ "Scheduler" (OS) คอยตัดสินใจว่าจะรันอะไรต่อ ซึ่งจะรันระหว่าง process แต่ละอัน

ในการรันแต่ละ process ไม่ควรรันนาน (เกิน 10-20 ms) เลย ต้องมีการ interrupt - ถ้าถูก interrupt "interrupt

handler” จะจัดการให้การควบคุมกลับไปที่ scheduler เพื่อให้ scheduler ทำงานต่อ แต่ถ้าไม่มีการ interrupt นานเกินไป clock device ใน mainboard จะส่ง regular interrupt มา

การเลือก process ของ scheduler ทำได้หลายวิธี เช่น Round-Robin Algorithm (อย่างง่าย) จะรัน process ตามลำดับไปเรื่อยๆ, แต่ OS ใหม่ๆ จะใช้วิธีที่ซับซ้อนขึ้น สนใจ process ที่ใช้เวลาเยอะมากกว่าอันอื่น

Life cycle หลักๆ ของ process



1. Created : OS สร้าง process
2. Waiting : รอ scheduler เรียกใช้
3. Running : หลังจากถูก scheduler เรียก process จะทำงาน
4. Terminated : หลังจากที่ process ทำงานเสร็จเรียบร้อย
5. Blocked : รอ external event ใน system ทำงานเสร็จ เมื่อเสร็จแล้วจะถูก unblock แล้วกลับไป waiting อีกรอบ เช่นรอระบบอ่านไฟล์ (เพราะอ่านไฟล์ช้ากว่าการทำงาน + ไม่สามารถทำอย่างอื่นไปก่อนได้)

ถ้า scheduler เปลี่ยน process ที่จะทำงาน อันเก่าจะไป waiting เพื่อให้อันใหม่ running วนไปเรื่อยๆ จนกว่า process นั้นจะเสร็จและเปลี่ยนสถานะเป็น terminated

Memory sharing

OS จำกัด memory ที่แต่ละ process ใช้ เพื่อไม่ให้รบกวนกัน โดย OS จะเข้าถึงข้อมูล + i/o devices ได้ทุกส่วน ในขณะที่แต่ละ Process จะเข้าถึงได้แค่ memory ในส่วนตัวเองเท่านั้น

ใน 1 process memory จะประกอบด้วย

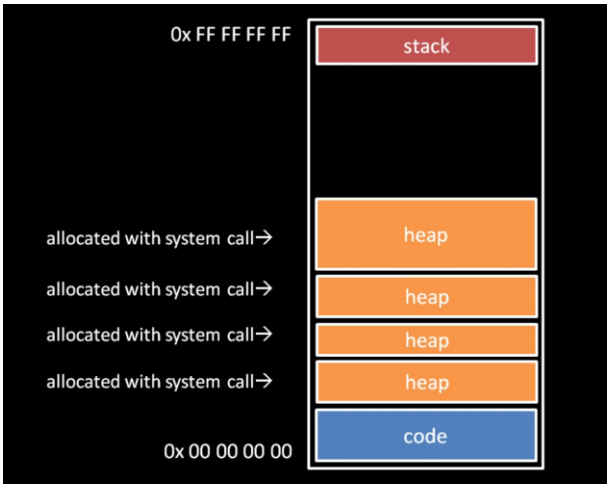
1. text : binary code, ไม่มีการเปลี่ยนแปลงค่า (ยกเว้น shared libraries ที่ถูกเชื่อมแบบ dynamic)
2. call stack : local variables, เริ่มต้นจากพื้นที่ว่าง แล้วจะสร้าง frame เก็บข้อมูลของแต่ละ function เมื่อถูกเรียกใช้ ใน frame จะประกอบด้วย local variable, return address (กลับไปฟังก์ชันก่อนหน้า หลังจากทำงานฟังก์ชันนั้นเสร็จแล้ว), size of frame (ใช้เลื่อน stack pointer (top of stack) หลังทำงานเสร็จ) ยกเว้น frame of main ที่จะไม่มี return address & size of frame

* ถ้าพื้นที่เต็มแล้ว คอมพิวเตอร์จะเก็บ frame ใหม่ทับของเดิมที่ไม่ได้ใช้ *

> CPU ส่วนใหญ่จะเก็บ stack แบบบนลงล่าง (ถ้าเกิด stack pointer มากกว่า stack boundary CPU จะตัดสินใจว่าจะขยายมั้ย ถ้าเกิด stack ใหญ่ไป (มากกว่า 1-2 mb) ก็จะไม่ขยายแล้วหยุดทำงาน process นั้น (เวลามี underlying error เช่นการเรียกใช้ recursive function ยาวเกินไป) โดยจะเรียกการหยุดว่า “Stack overflow” แต่บาง computer ที่ไม่ซับซ้อนเช่น embed system จะไม่มี stack overflow แล้วปล่อยให้ข้อมูลทับซ้อนกันเลย ซึ่งอาจทำให้ bug ได้)

3. heap : ข้อมูลอื่นๆ, ตอนแรกเป็นพื้นที่ว่าง แล้วจะถูกสร้างเมื่อ process request พื้นที่จาก OS แล้วให้ OS ตัดสินใจว่าจะสร้าง chunk (heap) ใหม่ไว้ตรงไหน ด้วย “System call” (เจาะจงขนาดของ chunk ที่ติดกัน) หลังจากที่ใช้งาน chunk เสร็จแล้ว จะทำการ “deallocate” ด้วย system call เพื่อคืนพื้นที่ให้ OS (OS ต้องจัดการดีๆ เพราะยิ่งทำงานไป chunks ต่างๆ ก็ยิ่งกระจายกัน ทำให้สร้าง chunk ใหม่ไม่ได้ ถ้าพื้นที่ที่เหลือน้อยกว่าที่ต้องการ -> อาจทำให้พื้นที่เต็มจนทำให้ process fail ได้ -> ถ้าจัดการไม่ดี จน process fail จะเรียกว่าเกิด “Memory leak”)

ปกติแล้ว CPU จะเก็บ stack ไว้บนสุด, text ไว้ล่างสุด และพื้นที่ที่เหลือจะเอาไว้เก็บ heap



ถ้า Process ต้องการเข้าถึง memory ภายนอก process ต้อง request ไปที่ OS เพื่อเข้าถึง “System call routines” เช่นการอ่าน-เขียนไฟล์, รับ-ส่งข้อมูลผ่าน network (จะเรียกใช้ system call ได้ ต้องใช้ “Cisco” ที่เป็นคำสั่งเฉพาะของ CPU แล้ว CPU จะหาที่อยู่ของ routine นั้นๆ แล้ว jump ไปทำงานคำสั่งนั้น)

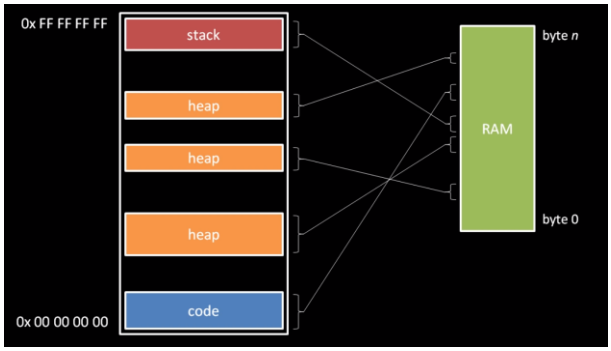
Address ของ memory ใน process ไม่ได้เป็นที่อยู่จริงใน system memory แต่ chunks ของ process address จะถูก OS เอาไป mapped กับ chunks ของ system memory (มักจะไม่ได้ mapped ด้วยกันทั้ง chunk แต่แต่ละ page ของ stack จะถูก mapped กับ page ของ RAM คนละอันและอาจจะไม่ได้เรียงกัน)

OS จะเก็บข้อมูลของ chunks ต่างๆ ที่ถูก mapped กันไว้เป็นตาราง และเมื่อ CPU จะแปลงที่อยู่ก็จะมาเรียกใช้ตารางนี้

* Map chunks ของ memory จะเรียกว่า “Page” ซึ่งมีขนาดต่างกันตาม CPU (32-bit x86processeors -> ใช้ 4kb pages) แต่ปกติ page เล็กๆ จะใช้ไม่เกิน 1-2 GB เพื่อจัดเก็บ*

ถ้า process พยายามเรียกใช้ memory ที่ไม่ได้รับอนุญาต CPU จะเรียกใช้ hardware exception แล้วหยุดทำงานพร้อมกับส่ง error message ไปให้

การเรียกคืน RAM -> OS จะเลือกบาง pages แล้ว copy ไปเก็บใน storage (hard drive) แล้วเปลี่ยนค่าในตารางเป็น swapped (ถ้า process พยายามจะเรียกข้อมูลที่ถูก swapped -> OS จะ swap page นั้นกลับมา แล้วแก้ค่าในตาราง -> อนุญาตให้ process ทำงานต่อ) ถึงจะทำงานช้า แต่ต้อง swap ไม่งั้น RAM จะเต็มแล้วทำให้ process หยุดทำงาน (แต่ก็จะ swap เท่าที่จำเป็น)



File system (Storage devices)

เป็น layer พิเศษของ OS ที่ใช้ในการจัดการ storage device โดยเก็บไฟล์ในพื้นที่ว่างเป็นลำดับขั้น (hierarchy directories) เมื่อโปรแกรมเรียกใช้งานพื้นที่ใน hard drive จะไปใช้ฟังก์ชันที่ OS มีไว้ให้ ทำให้สามารถอ่าน-เขียน-เรียกใช้ไฟล์จาก storage แต่ละแบบได้ด้วยวิธีเดียวกัน

Storage แต่ละอันจะถูกแบ่งออกเป็น partition สำหรับการใช้งานที่เฉพาะเจาะจง (เช่นมีหลาย OS ใน 1 drive) แต่ส่วนใหญ่จะเป็น 1 partition ต่อ storage รวมถึงจะมีบางส่วนที่ว่างหรือ unformatted

Format ในปัจจุบันส่วนใหญ่จะกำหนด identified number ให้กับแต่ละ file / directories ที่ไม่ซ้ำกันใน partition (ต่อให้ข้อมูลจะต่อเนื่องกัน ก็อาจจะจัดเก็บแยกกันบน disk แล้ว File system จะประกอบข้อมูลขึ้นมาใหม่ก่อนส่งไปให้โปรแกรมใช้)

Directory คือลิสต์ของไฟล์และ directories อื่นๆ จะเชื่อม id ของไฟล์และ directory ด้วยชื่อ (ที่ไม่ซ้ำกัน ใน directory แต่ชื่อ file เหมือนกับชื่อ directory ได้)

เมื่อ partition ถูกสร้างใหม่ จะมีแค่ root directories

File path เป็นข้อความที่แสดงที่อยู่ของไฟล์หรือ directory บน system

- ใน Window : แต่ละ partition จะถูกกำหนดเป็นตัวอักษรตามด้วย : (เช่น D:, H:) ซึ่ง root path จะเป็นชื่อ partition ตามด้วย / หรือ \ (ใช้ได้เหมือนกัน) เช่น C:/
- ใน Unix : ใช้ได้แค่ /, มี 1 partition ที่ถูกสร้างเป็น root (path /) แล้ว partition อื่นๆ จะเข้าถึงได้ผ่าน directory อื่นๆ ที่อยู่บน partition ที่ถูกติดตั้งไว้แล้ว (เช่น partition 2 (root dir = /banana) -> root, partition 1 (root dir = /) -> directory banana ที่อยู่บน partition 2, partition 3 (root dir= /banana/apple) -> directory apple ที่อยู่บน banana อีกที)

ในการเข้าถึงไฟล์ / directory ต่างๆ เรียกว่าการ mount ผ่าน mount point โดยที่ directory ของ mount point ต้องมีอยู่ก่อนจะใช้ได้ และเมื่อ mount ไฟล์หรือ directory แล้ว mount point จะถูกซ่อน (เช่นถ้าเข้า /banana ก็จะเข้าถึงเข้าข้อมูลใน partition 2 ไม่ได้แล้ว เพราะย้ายไปเข้าถึงข้อมูลใน partition 1 แล้ว mount point ก็เปลี่ยนไปเป็น /banana ด้วย)

IPC เป็น mechanism ของ CPU ที่ช่วยให้ process ต่างๆ สามารถติดต่อกันได้ เช่น files (สามารถอ่าน-เขียนไฟล์ได้จากหลายๆ process & สร้าง channel สำหรับติดต่อสื่อสารให้กับ processes), pipes, sockets, signals, shared memory

Linux ถูกสร้างขึ้นในเดือนสิงหาคม 1991 โดย Linus Torvalds ที่ได้โพสต์ข้อความว่า เขาจะทำ free OS (แค่ทำเป็นงานอดิเรก เป็นโปรเจ็กต์เล็กๆ ไม่เหมือน GNU) ที่ไม่น่าจะซัพพอร์ตอะไรนอกจาก AT-hard disk โดยมีเพนกวินเป็น mascot -> ข้อความ & idea เกี่ยวกับ Open-source project นี้ก็ดังไปทั่วโลก ทำให้ developers ทั่วโลกมาช่วยกันเขียนโค้ด -> พัฒนามาเป็นบริษัทที่ประสบความสำเร็จอย่างมากและมี business อื่นๆ มาสร้างโดยใช้ linux เช่น Red hat stock (1999) : linux company แรกที่เปิดตัว และ IBM (1999) : ลงทุนพันล้านดอลลาร์เพื่อพัฒนาและโฆษณา linux

โดยมีคอนเสปหลักคือการมีอิสระ ได้แก่

1.อิสระที่จะใช้งาน software สำหรับวัตถุประสงค์ใดก็ได้

2. อิสระที่จะใช้งาน software ที่เหมาะกับความต้องการ

3. อิสระที่จะเผยแพร่ software ให้กับคนอื่นๆ

4. อิสระที่จะเผยแพร่ความเปลี่ยนแปลงที่คุณทำ

มีจุดเด่นคือเป็น free & open-source software (เราจะใช้ทำอะไรก็ได้ ไม่ถูกจำกัดเหมือน window) & ปลอดภัยและเสถียรมากกว่า window (นิยมใช้ทำ server + application ที่สำคัญ) & เบาและใช้ทรัพยากรน้อยกว่า window

แต่มีจุดด้อยคือไม่เหมาะกับการใช้งานทั่วไป เพราะ set up + ใช้งานยากกว่า ต้องใช้ความรู้เยอะกว่า (สามารถตั้งค่าได้เยอะกว่า) เช่น ไม่สามารถโหลด software ผ่าน browser ตรงๆ ได้ ต้องใช้ package manager / โหลดเกมเพื่อเล่นยากกว่า window

linux ใช้ GPL license (พัฒนาโดย Richard Stallman, 1989) ร่วมกับตัว linux และเทคโนโลยีอื่นๆ

linux มีการปล่อย version (distributions / distro) ใหม่ทุกๆ 3 เดือน -> มี stacks of software, tools, desktop environment รวมกันเป็น OS ที่เป็นได้หลายอย่างตั้งแต่ ubuntu, mint (popular desktop distros), android

นอกจาก linux จะปฏิวัติวงการ internet แล้ว ยังสามารถประสบความสำเร็จท่ามกลางการแข่งขันในตลาด (สู้กับบริษัทใหญ่อย่าง microsoft ได้) รวมถึง development community ในปัจจุบันก็ยังมีจำนวนเพิ่มขึ้น และมีบริษัทมากมายเข้ามา collab ด้วย, ถูกใช้งานใน 75% ของ การเทรดหุ้นทั่วโลก, ใช้งานใน 95% ของ supercomputer, ถูกใช้ทำ server ของ amazon facebook twitter eBay และ google, ถูกใช้ทำระบบปฏิบัติการ android และในอุปกรณ์อื่นๆ อย่างทีวี ATM

สามารถศึกษาเพิ่มเติมได้ เช่น linux newbie guide

Activity *ยังไม่เสร็จ*

ใช้ secure shell (ssh) -> puTTy เพื่อ remote เข้าไปควบคุม linux system ที่เราเปิด instance ไว้ (เซิฟอยู่ที่ north virginia)-> EC2 ใน AWS (เลือกเป็น ubuntu แล้วโหลด key pair (RSA) เป็น .ppk (window) หรือ .pem (mac / linux) เมื่อเปิด instance แล้ว ให้เอาไปเปิดใน puTTy โดยใช้ host name เป็น “ubuntu@PUBLIC_IP_ADDRESS” แล้วไปที่ SSH -> Auth -> Credentials -> เปิดไฟล์ key pair

Command	Description
ls -l -a -al	List ไฟล์ใน directory นั้น - longer format - all files (รวมที่ซ่อนด้วย) - สองคำสั่ง -a + -l รวมกัน
pwd	แสดง path ของ directory ปัจจุบัน
Cd [PATH]	เปลี่ยน directory เป็น [PATH] เช่น cd .. (), cd / (เปลี่ยนเป็น root directory), cd /root (), cd ~ (เปลี่ยนเป็น home directory)
clear	เคลียร์ display
Man [COMMAND]	วิธีการใช้งานของ [COMMAND]
Echo [VALUE]	แสดงผลค่า [VALUE]
Cp [SOURCE] [TARGET]	Copy ไฟล์ [SOURCE] ไปยังไฟล์ [TARGET]
Cp -r [SRCDIR] [TARGETDIR]	
Mv [SOURCE] [DESTINATION]	ย้ายไฟล์ [SOURCE] ไปไว้ที่ [DESTINATION] (ใช้เปลี่ยนชื่อไฟล์ได้)
Rm [FILE]	ลบไฟล์ [FILE]

Arduino คือระบบฝังตัว (Embed system) ประเภทหนึ่ง ที่ hardware & software ถูกควบคุมโดย microcontroller (microprocessor), ใช้ไฟฟ้า, อ่านค่าจากเซนเซอร์, ควบคุมอุปกรณ์ต่างๆ, ประมวลผลต่างๆ และเชื่อมต่อ wifi ได้ -> ถูกออกแบบมาเพื่อให้ทำงานเพียงงานเดียว, มักจะเป็นส่วนหนึ่งในเครื่องจักร / ระบบไฟฟ้าอื่นๆ, ถูกนับเป็นหนึ่งในอุปกรณ์ IoT

(Internet of things) -> ทำให้เครื่องใช้ต่างๆ ทำงานได้มากขึ้น เช่น Smart TV, smart watch, smart bulb

ถูกผลิตครั้งแรกในปี 2005 ที่อิตาลี และขายไปกว่า 10 ล้านบอร์ดทั่วโลก โดยรุ่นที่นิยม + หาซื้อง่ายสุดคือ Uno

Arduino มีหลายรุ่นเช่น Uno, Mega, Nano, Lilypad, Yun

มีข้อดีคือ ถูก, ต่อกับคอมพิวเตอร์ง่าย, set up และใช้งานง่าย, ใช้พลังงานน้อย

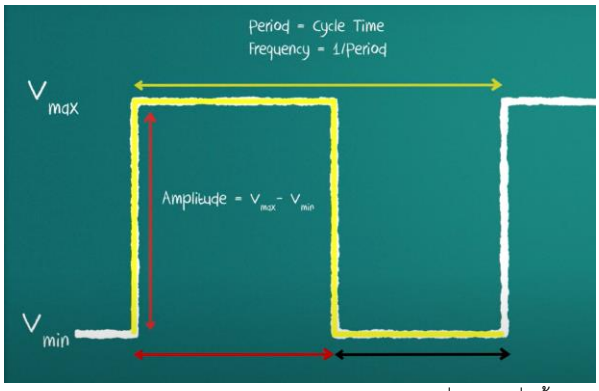
ใน Arduino มี digital pin 13 pin (ใช้เชื่อม i/o devices) และมี USB พอร์ตสำหรับต่อคอมพิวเตอร์ เพื่ออัพโหลดโค้ด

Arduino เป็น open-source platform ที่ถูกทำมาให้ใช้งานง่ายกับทั้ง hardware & software -> สามารถเขียนโปรแกรมได้ด้วยภาษา Arduino (มีรากฐานมาจากภาษา C, C++ แต่เพิ่มฟังก์ชันเกี่ยวกับการควบคุม hardware เพิ่ม) หรือภาษาอื่นๆ (มีตัวแปลงต่างๆ รองรับ แต่ไม่นิยมเท่า Arduino)

การเขียนโปรแกรมลง Arduino มีขั้นตอนคือ

1. เตรียมบอร์ด Arduino
2. โหลด Arduino IDE / Arduino vs code plug in (ใช้เขียนโปรแกรม)
3. ต่อสาย Arduino กับ USB port
4. ตั้งค่า board type (เลือกเวอร์ชันของ board ที่ใช้), serial port (port ของคอมที่ใช้ต่อ arduino) ใน IDE
5. เขียนโปรแกรม (เรียกว่า “Sketches”) แล้ว run
   - มีโครงสร้างหลัก คือ setup() -> ทำครั้งเดียวตอนเริ่มทำงาน (power up / reset board) -> ใช้ set ค่าต่างๆ เช่น pinMode Serial.begin, loop() -> ทำงานไปเรื่อยๆ ตลอดเวลา (คล้ายๆ while true)
   - มีคำสั่ง เช่น pinMode(pin, type) -> กำหนดประเภทของ pin, delay(x) -> หน่วยเวลาเป็น ms, digitalWrite(pin, state) -> เขียนค่าให้ pin เป็นไม่ high ก็ low, analogWrite(pin, value) -> เขียนค่าให้ pin เป็น value, digitalRead(pin) -> อ่านค่าจาก pin เป็น digital ถ้าไม่ได้ต่อกับอะไร จะสุ่มค่ามาให้, Serial.begin(x) -> กำหนดว่าจะแสดง output ที่ไหน, Serial.println(x) -> แสดงผลค่า x, Serial.read() -> รับค่าจาก keyboard ใน serial
   - คล้ายภาษา C หลายอย่าง เช่นต้องประกาศตัวแปรก่อนใช้, สร้างฟังก์ชันได้, if, for
   - สามารถแสดงผลต่างๆ ได้ใน Serial monitor
   - Input
     - Digital : มี 2 ค่าคือเปิด-ปิด (HIGH / LOW)
     - Analog : มีค่าตั้งแต่ 0-1023 (10 bit)
   - Output
     - Digital : มี 2 ค่าคือเปิด-ปิด (HIGH / LOW)
     - Analog : ใช้ Pulse width modulation (PWM) เพื่อเปลี่ยนค่าจาก digital (5V & 0V) เป็น analog signal

PWM สร้างคลื่นที่เป็นสี่เหลี่ยมที่ประกอบด้วย High-time & Low-time



มี Maximum & minimum voltage เป็นลิมิตของคลื่น โดยเมื่อขึ้น-ลงครบ 1 รอบจะเรียกว่า 1 cycle ซึ่งจะสามารถเอามาคำนวณ Output voltage ได้จาก Max voltage * Duty cycle (Duty cycle คือเปอร์เซ็นของเวลาที่สัญญาณเป็น High) เช่น high 6 ms, low 4 ms จะได้ว่า duty cycle = (6 / (4+6)) * 100 -> High อยู่ 60%

การเขียนโปรแกรมทำ PWM : ปรับค่าเป็น High -> delay (10 * duty cycle) -> ปรับค่าเป็น Low -> delay (1000 - (10 * duty cycle))

6. ต่อวงจรต่างๆ แล้วอัพโหลด sketch เพื่อใช้งาน

หรือสามารถใช้ simulation ได้บนเว็ป เพื่อจำลองวงจรไฟฟ้า + การเขียนโปรแกรม (https://www.tinkercad.com)

อุปกรณ์ทางไฟฟ้าอื่นๆ ที่น่ารู้

- oscilloscope : วัดสัญญาณไฟฟ้า คลื่นไฟฟ้า วัดค่าแรงดันของไฟฟ้า การใช้วัดความถี่ วัดเฟสของสัญญาณ และใช้สำหรับการวัดคาบเวลา
- breadboard : ช่วยให้ออกแบบ + ต่อวงจรอิเล็กทรอนิกส์ได้ง่ายขึ้น

Frontend

Internet คือ network ที่ซับซ้อนที่เชื่อมต่อทั่วโลกไว้ด้วยกัน โดยมีส่วนประกอบสำคัญ คือ server (คอมพิวเตอร์ที่มีประสิทธิภาพที่คอยเก็บ / ประมวลผลข้อมูล), communication network (คอยส่ง (route) ข้อมูลผ่านเน็ตเวิร์คไปยัง server ที่ถูกต้อง เช่นผ่าน fiber optic cable, wireless connection, satellite link), IP address (เป็นสิ่งที่ใช้ระบุตัวตนของอุปกรณ์แต่ละอย่างที่เชื่อมเน็ต ใช้ในการรับ-

ส่งข้อมูลให้ถูกที่อยู่), DNS (domain name system : เป็น ip address ที่อ่านได้ง่ายเช่น www.google.com โดยที่ไม่ต้องจำตัวเลข IP ตรงๆ), web browser (UI ของ internet ที่ช่วยแสดงผลเว็ปต่างๆ ให้ผู้ใช้ดูได้ง่าย+สวย+โต้ตอบได้), URLs (Uniform resource locators : address ที่สามารถอ่านได้ที่ใช้เข้าถึง resource บางอย่าง เป็นเหมือนไกด์ใน internet), HTTP/HTTPS Protocol (HyperText Transfer Protocol : กฎ / วิธีการส่งขอ-ข้อมูลระหว่างหน้าเว็ปกับ server ต่างๆ โดยถ้าเติม S จะเป็นการส่งข้อมูลแบบถูกเข้ารหัสก่อน (ด้วยวิธี SSL, TLS) -> ปลอดภัยกว่า)

ส่วนประกอบของ URL

โครงสร้างจะช่วยให้สามารถระบุข้อมูลและตำแหน่งของมันได้, ถูกออกแบบมาให้ใช้งานง่ายทั้งกับมนุษย์ & ง่ายกับเครื่องจักรในการทำงาน

1. Protocol : บอก method ที่จะส่งข้อมูลระหว่าง user & server -> ตัวที่นิยมสุดคือ HTTP, HTTPS ที่จะส่งข้อมูลเป็น plain text (และเข้ารหัสสำหรับแบบมี S)
2. Domain name : เป็น IP Address ที่สามารถอ่านออกได้ (DNS) ของ server แต่ละเซิฟ โดยจะถูกเก็บเป็ยลำดับขั้นที่ประกอบด้วย TLD (Top level domain : อยู่ท้ายสุด เพื่อบอกประเภทเว็ป เช่น .org .com .net), subdomain (อยู่ข้างหน้าสุด -> www.) และ domain name (อยู่ตรงกลาง)
3. Path : ส่วนประกอบที่อยู่ข้างหลัง Domain name เพื่อระบุที่อยู่ที่เฉพาะเจาะจง เพื่อเรียกใช้ resource ใน directory ของ server นั้นๆ (ปกติมักจะถูกตั้งชื่อให้อ่านรู้เรื่อง เพื่อให้ผู้ใช้เข้าใจ + ใช้งานได้ง่าย)
4. Query parameters : ใช้ในการส่งข้อมูลเพิ่มเติมไปให้ server ผ่าน URL มักใช้ในเว็ป dynamic เพื่อจัดการข้อมูลที่เฉพาะเจาะจง, จะอยู่ต่อหลังส่วนอื่นๆ ที่คั่นด้วย ? แล้วตามด้วย query parameter เป็น key, value เช่น http://www.example.com/search?query=computer&category=laptops
5. Fragment identifier : อยู่ต่อหลังจาก # เพื่อระบุ section ในเว็ป มักใช้เพื่อย้ายผู้ใช้ไปส่วนที่ต้องการ (ในเพจที่ยาวๆ) หรือเพื่อ highlight element ต่างๆ ของเพจ

การเขียน Web page

HTML เปรียบเสมือนโครงร่าง CSS เปรียบเสมือนของตกแต่ง JavaScript เปรียบเสมือนฟังก์ชันการทำงาน

HTML (HyperText Markup Language) เป็นโครงสร้างพื้นฐานของเว็ป โดยใช้ tag ต่างๆ เพื่อแสดงว่า แต่ละส่วนเป็น element อะไรบ้าง, มีหน้าที่หลักในการจัดการโครงสร้างและข้อมูลให้เป็นแบบ semantically และมีความหมาย เพื่อให้เข้าถึงได้ถูกต้องและทำงานต่อใน CSS, JS ได้ง่าย

headings, paragraphs, images, links, forms, etc

CSS (Cascading Style Sheets) เป็นส่วนที่จัดการเกี่ยวกับหน้าตาและความสวยงามของเว็ป โดยมีกฎ & properties ต่างๆ ให้เลือกใช้กับ HTML element, ช่วยให้สามารถทำ responsive design ได้และช่วยดึงความสนใจจากผู้ใช้ได้

colors, fonts, spacing, layout, etc

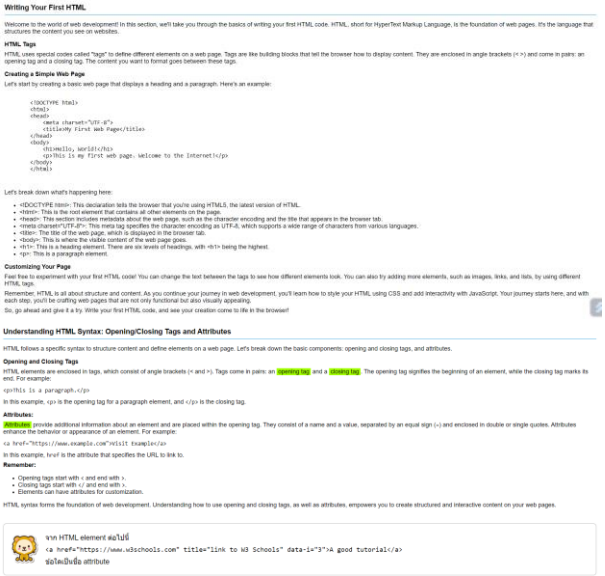
JS (JavaScript) มีหน้าที่ทำให้เว็ปสามารถโต้ตอบกับผู้ใช้ได้และเป็น dynamic ด้วยการสร้างฟังก์ชันต่างๆ, ตอบโต้กับ user actions & events, ทำให้สามารถติดต่อกับ server ได้และจัดการกับ DOM (Document Object Model) เพื่อแก้ไขข้อมูลแบบ real-time ได้โดยไม่ต้องรีเฟรชหน้าเพจใหม่

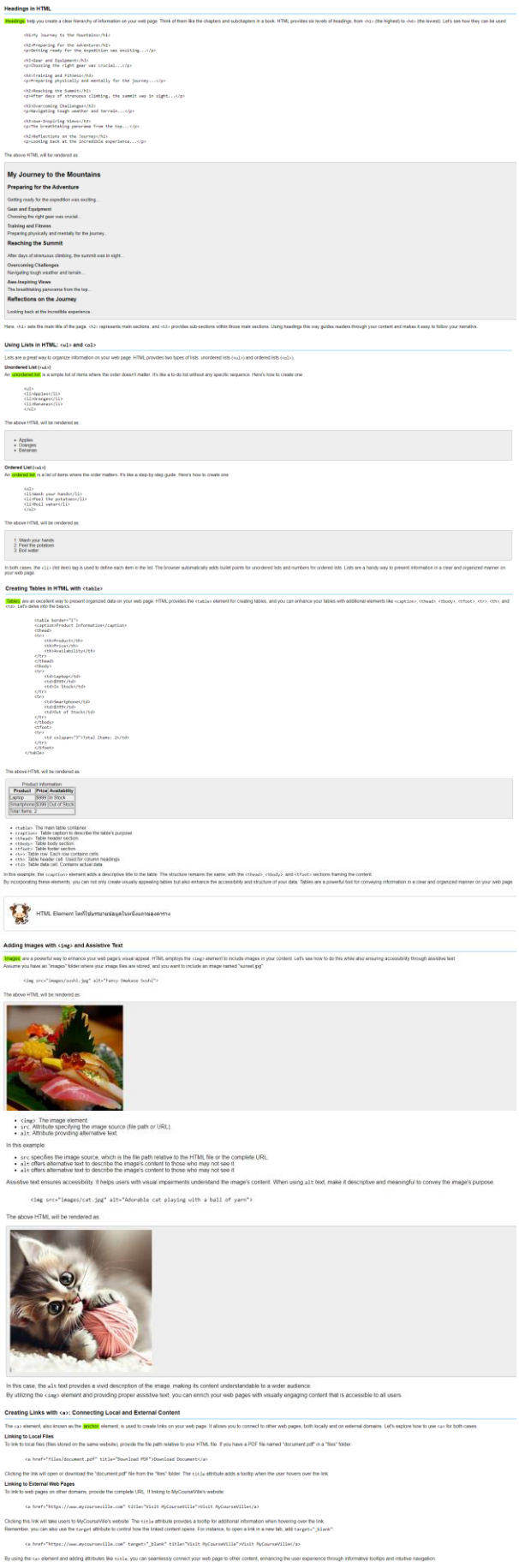
เช่นตรวจสอบความถูกต้องของข้อมูล, ทำ animation, ประมวลผล form, dynamic content loading, real-time update

หน้าที่ของ Frontend - Backend

- Frontend : พัฒนาเกี่ยวกับหน้าเว็ป, โครงสร้าง HTML, UI ให้สวย, เข้าถึงแต่ละเพจได้ง่ายและการตอบโต้กับผู้ใช้ เช่นกดปุ่ม
- Backend : เป็นระบบหลังบ้าน เกี่ยวกับการประมวลผลทางเทคนิคต่างๆ, การเก็บข้อมูล, การจัดการ content เพื่อให้ frontend ทำงานได้ เช่นการ จ่ายเงิน, เก็บข้อมูลในฟอร์ม

การเขียน HTML : โครงสร้าง, tags, attributes (heading, list -> ol, ul, table, img, a) * ไปดูเอาเอง *





Semantic vs Non-semantic (ยังไม่เสร็จ)

DOM tree

เป็น concept พื้นฐานที่บอกว่าโครงสร้างของ HTML จะจัดเรียงเป็นแบบลำดับขั้น คล้ายต้นไม้ โดยที่เริ่มต้นจาก root node = document ที่จะมี node อื่นๆ ข้างใน

```
<!DOCTYPE html>
<html>
<head>
<title>DOM Tree Example</title>
</head>
<body>
<header>
    <h1>Welcome to My Website</h1>
</header>
<section>
    <p>This is a sample section.</p>
</section>
</body>
</html>
```

In this case, the DOM tree structure is as follows:

```
- Document (root node)
- html
    - head
    - title
        - Text: "DOM Tree Example"
    - body
    - header
        - h1
        - Text: "Welcome to My Website"
    - section
        - p
        - Text: "This is a sample section."
```

ซึ่งเราควรจะทำตาม concept นี้ เพราะจะง่ายกว่าในการจัดการ + เชื่อมต่อกับ CSS / JS

CSS

ก่อนจะเขียนโค้ดต้อง import เข้าไปในไฟล์ HTML ด้วย <link rel=”stylesheet” href=”FILE_PATH”> ในส่วน head ในการเขียน CSS กำหนด style ให้ tags HTML จะใช้ CSS selector (#id -> มีได้แค่ตัวเดียว, .class -> มีได้หลายตัว, tag -> ทุกตัวที่ใช้ tag นั้น) จากนั้นใน { } จะสามารถปรับค่าต่างๆ ได้เช่น color, size, font, background เช่น .highlight { background-color : yellow; }, h1 { size : 24px; color : #007BFF; font-family : Arial, sans-serif; }

JS

ก่อนจะเขียนโค้ดได้ต้อง import เข้าไปในไฟล์ HTML ก่อนด้วย <script src=”FILE_PATH”></script> ในส่วนล่างของ body

การเขียนโค้ดให้ทำฟังก์ชันเมื่อกดปุ่ม ใช้คำสั่ง document.addEventListener(“DOMContentLoaded”, function() {

  const NAME = document.getElementById(“id”)

  NAME.addEventListener(“click”, FUNCTION_NAME)

})

การเลือกว่าสนใจอะไรจะใช้ GetElementById, GetElementByClass แล้วเอามาเก็บไว้ในตัวแปร ทำให้สามารถแก้ไข element นั้นๆ หรือ AddEventListener ได้ เมื่อกดปุ่มที่เราสนใจ จะไปเรียกใช้ FUNCTION_NAME

Activity * ยังไม่เสร็จ *

Backend

JSON (JavaScript Object Notation)

เป็นรูปแบบการเก็บข้อมูลอย่างนึง (คล้ายกับ XLM, yamo) ถูกใช้อย่างกว้างขวาง (เช่นการส่ง API, config files, games, text editor -> VS code) เนื่องจากมีข้อดีหลายอย่าง ได้แก่ ใช้ทรัพยากรน้อย, อ่านง่าย, เหมาะกับการใช้กับ JS, สามารถใช้กับภาษาอื่นได้ เพราะมี library / extension รองรับเยอะ

ประเภทข้อมูล

Object : กลุ่มของข้อมูลที่เป็นคู่ (key, value) โดยที่ key เป็น string ที่ถูกครอบด้วย “ ” แล้วตามด้วย : ตามด้วย value (เป็นข้อมูลประเภทอะไรก็ได้) ถ้ามีข้อมูลตัวอื่นอีกจะคั่นด้วย , แล้วใช้ { } ครอบข้อมูลทั้งหมดเพื่อรวมเป็น 1 object

เช่น {

```
"Key1" : "Value1",
"Key2" : 10,
"Key3" : false,
"Key4" : null
```

}

ปล. สามารถใส่ข้อมูลหลายประเภทลงใน array / object ตัวเดียวกันได้, สามารถเข้าถึงค่าใน array ได้ด้วย [INDEX] และเข้าถึงค่าใน object ได้ด้วย .KEY

สามารถใช้ `` เพื่อกำหนดให้ตัวข้างในเป็น string ได้ข

ใช้ json.parse(STRING) เพื่อเปลี่ยน string กลับเป็น json

วิธีการใช้

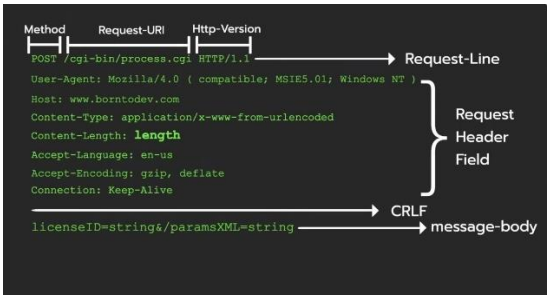
1. สร้างไฟล์ .json
2. ใส่ค่าที่ต้องการเก็บ (มักใช้เก็บ array / object แล้วมีค่าอื่นข้างใน)

HTTP Request

คือ protocol ที่ทำให้เครื่องเราคุย รับ-ส่งข้อมูลกับเซิฟเวอร์ได้ เครื่องเราจะส่ง HTTP Request ไปหาเซิฟเวอร์ แล้วเซิฟเวอร์จะส่ง HTTP Response กลับมาหลังจากประมวลผลแล้ว

HTTP จะประกอบด้วย

1. Request-line
   - HTTP method / verb มี 4 แบบที่ใช้บ่อยคือ POST (create), Get (read), POST (update), DELETE -> ถูกเรียกรวมๆ ว่า CRUD Operation
   - Path / Request-URI
   - HTTP Protocol / Version
2. Header : กำหนด properties ต่างๆ ของสิ่งที่เราร้องขอข้อมูล กฎกติกา รูปแบบการเชื่อมต่อ ขนาดต่างๆ ที่ใช้คุย
3. CRLF : ขึ้นบรรทัดใหม่ คั่นระหว่าง header & body
3. Body : กำหนดเนื้อหาด้านใน content ต่างๆ ที่ร้องขอ (ข้อมูลที่จะส่งให้ปลายทาง)



REST API

Api = application programming interface

เนื่องจากในปัจจุบันแอพลิเคชันจะใช้โครงสร้างแบบ client (frontend) - server (backend) ซึ่งทั้งสองฝั่งจะต้องติดต่อกันเพื่อรับ-ส่งข้อมูล สำหรับการประมวลผลหรือเก็บข้อมูลนั้นเลยจะใช้การติดต่อกันด้วย HTTP Protocol โดยที่ server จะมี service ต่างๆ ให้ client มาเรียกใช้งาน ผ่าน “REST API”

* Rest API = Representational State Transfer API *

โดยที่ server จะมี endpoint (คล้ายๆ URL เช่น http://vidly.com/api/customers) ที่เรียกว่า URI (Uniform resource identifier) เพื่อให้ client ส่ง HTTP request หา

ซึ่งจะประกอบด้วย 3 ส่วนคือ http / https, domain name, /api (จริงๆ ไม่จำเป็นต้องมี แต่นิยมใช้ และอาจจะอยู่ตำแหน่งไหนก็ได้ -> subdomain / path), ชื่อของ api ที่จะเรียกใช้

Response message จาก server จะประกอบด้วย Status code (บอกว่าเกิดอะไรขึ้นบ้าง เช่น 2xx : สำเร็จ, 4xx : client request error, 5xx : server มีปัญหา), header (ข้อมูลเกี่ยวกับ server), body (ข้อมูลที่ส่งกลับ เป็น Json data payload)

การใช้ REST API ทำให้ทั้งสองฝั่งสามารถติดต่อกันได้อิสระ ไม่ต้องเก็บข้อมูลกันและกันไว้ ไม่ผูกมัดกับการติดต่อแบบอื่น

การสร้าง API

มักจะใช้ framework ที่ชื่อ Express.js

1. npm init & install express
2. สร้างไฟล์ index.js

แล้วเขียนโค้ด เปิดเซิฟเวอร์ (appp.listen(PORT, callback function)), import express package และสร้าง express object (const app = express();)

3. รัน / เปิดเซิฟเวอร์ด้วยคำสั่ง node . ใน terminal
4. สร้าง endpoint ด้วย app.METHOD(ENDPOINT_NAME, CALLBACK_FUNCTION) โดยที่ CALLBACK_FUNCTION คือ (req, res) => { SOME_CODE }

ถ้าจะรับค่าผ่าน URI -> ให้ใส่ใน ENDPOINT_NAME เช่น ‘/tshirt/:id’ แล้วไปสร้างตัวแปรที่ดึงค่ามาใช้ด้วย const { id } = req.params

ถ้าจะรับค่าผ่าน body ต้องใช้ middleware แปลง body เป็น json ก่อนใช้ด้วย const app = express(); app.use(express.json()); -> const { data } = req.body

การ Debug API

ใน browser จะ debug ยาก เลยมักจะใช้ Postman, Insomnia, REST Client (VS code extension), curl [URL]

Open API Specification เช่น swagger hub

เป็นมาตรฐานการอธิบาย API ที่ให้ทั้งมนุษย์ & เครื่องจักรเข้าใจทำให้ผู้ใช้งานใช้ได้ง่ายขึ้น, สามารถเขียนโค้ดทั้งฝั่ง server - client ได้อัตโนมัติ, สามารถใช้กับ API gateway (เช่น AWS, google cloud) เพื่อทำให้ปลอดภัยมากขึ้นในการตรวจสอบและเชื่อมต่อกับ back-end

NPM เป็น package manager หลักของ node.js ใช้สำหรับเอาโค้ดที่คนอื่นเขียนไว้แล้วมาใช้ (จะได้ไม่ต้องเขียนเองทั้งหมด) ซึ่งใน NPM มี package เยอะมาก และส่วนใหญ่เป็น open-software

โดยจะมีคนเขียนโค้ดแล้วเอามา publish บน NPM registry แล้วผู้ใช้คนอื่นสามารถไป install มาไว้บนเครื่องตัวเองได้ผ่าน NPM CLI (command line interface)

ก่อนใช้งานต้องติดตั้ง npm & node.js ก่อน แล้วถึงจะติดตั้ง package ที่ต้องการได้ โดยสามารถติดตั้งได้ 2 แบบคือ

1. local -> อยู่ในแค่โฟลเดอร์ของโปรเจ็กต์นั้นและจะถูก deploy ไปด้วย, ใช้คำสั่ง npm install [PACKAGE]
2. global -> สามารถเข้าถึงได้จากทุกที่, มักใช้กับสิ่งที่จะไปรันบน Command Line, ใช้คำสั่ง npm install -g [PACKAGE]

หลังจากติดตั้งแล้ว ใน project จะต้องสร้างไฟล์ package.json เพื่อเก็บข้อมูลของโปรเจ็กต์ (เช่นชื่อ, เวอร์ชัน) และ package ที่ใช้ ซึ่งจะช่วยเวลาทำงานกับคนอื่น / มีคนเอางานเราไปใช้ต่อ โดยสร้างเองหรือใช้คำสั่ง npm init ก็ได้ (ถ้ายังไม่อยากใส่ข้อมูลโปรเจ็กต์สามารถใช้ -y ต่อท้ายเพื่อข้ามได้)

ถ้าหลังจากนั้นเราติดตั้ง package เพิ่ม ไฟล์ package.json จะเพิ่มข้อมูล package นั้น ลงในไฟล์ (ส่วน dependencies) ให้อัตโนมัติ และจะดาวน์โหลด package นั้นมาใส่ในโฟลเดอร์ node_modules (ถ้ายังไม่มีโฟลเดอร์ ก็จะสร้างให้เลย)

* แต่ละ package สามารถติดตั้ง package อื่นอัตโนมัติได้ *
* สามารถใช้คำสั่ง npm list เพื่อดู package ทั้งหมดที่เราติดตั้งได้ (ชื่อ + เวอร์ชันที่ติดตั้ง) *

เมื่อเราเอางานคนอื่นมาทำต่อ ให้ใช้ npm install เพื่อติดตั้ง package ทั้งหมดที่ต้องใช้ (เขียนไว้ใน package.json) ได้เลย

การอัพเดต

ต้องระมัดระวังเพราะอาจทำให้โค้ดที่เขียนไว้พังได้ (เพราะแต่ละ package มีคนเขียนคนละคน พออัพเดตอันนึง อีกอันอาจจะไม่ได้อัพเดตตาม ทำให้เกิดปัญหาได้)

ปกติเวอร์ชันของ package จะมี 3 ส่วนคือ major.minor.patch (ถ้าเวลาอัพเดตจะเพิ่มเลขที่ช่องนั้น 1 แล้วเปลี่ยนเลขตัวข้างหลังมันทั้งหมดเป็น 0) โดยที่

- Major change : อัพเดตใหญ่, มีผลกระทบกับโค้ดเก่าๆ, มักทำให้โค้ดเก่าที่เขียนไว้พัง
- Minor change : อัพเดตเล็ก, ไม่มีผลกระทบกับโค้ดเก่า (ปกติแล้วจะไม่ทำให้โค้ดพัง)

- Patch : แก้ bug เล็กๆ น้อยๆ

* สามารถเช็ค version ทั้งหมดของ package ได้โดยใช้ npm view [PACKAGE] versions หรือเข้าไปดูใน github repo *

เวลาเราติดตั้งครั้งแรก จะติดตั้งเวอร์ชันที่ใหม่ที่สุดที่มี และเราจะสามารถอัพเดตได้โดยใช้ npm update

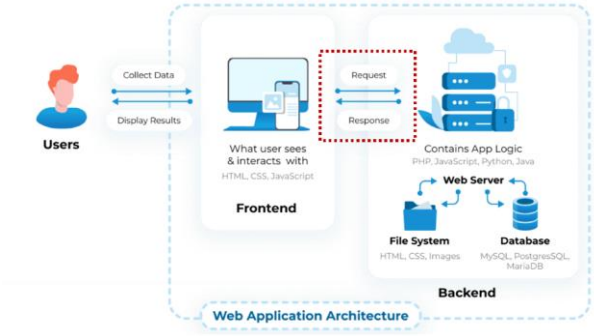
แต่เราไม่ควร update major version จึงควรจะมี ^ อยู่หน้าข้อมูล version ของ package ใน package.json เพื่อบอกว่าจะไม่อัพเดต major (อัพแค่ minor เท่านั้น)

ถ้าอยากจะอัพเดต major version สามารถใช้ npm update [PACKAGE]@lastest ได้

หรือถ้าอยากระบุเวอร์ชันที่ต้องการไปเลยก็สามารถใช้ npm update [PACKAGE]@[MAJOR_VERSION] หรือ npm update [PACKAGE]@~[MAJOR_VER].[MINOR_VER] เพื่ออัพเดตเป็นเวอร์ชันล่าสุดของ major (+minor) ที่ต้องการได้

Package.lock.json เป็นไฟล์ที่ใช้เก็บเวอร์ชันเป๊ะๆ ของ package เพื่อป้องกัน package.json มีปัญหาเวลาเอาไปลงใน environment อื่นๆ หรือ package ถูกอัพเดตตอนเรากำลังติดตั้งพอดี

การเชื่อมต่อกับ frontend



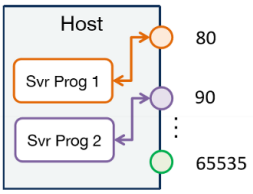
ผู้ใช้จะติดต่อกับส่วน frontend (เก็บข้อมูลจากผู้ใช้และแสดงผลกลับไปให้ผู้ใช้) โดยที่ frontend จะติดต่อกับ backend (โดยใช้ request, response) ส่วนที่เป็น web server แล้ว web server จะติดต่อกับ file system, database อีกที

แต่ละ server ที่เชื่อมต่อกับอินเตอร์เน็ตจะถูกเรียกว่า “Host”

แต่ละ host จะต้องมีอย่างน้อย 1 IP address (บางอันอาจจะมี hostname ด้วย) -> บน 1 host จะมีได้มากสุด 65536 “Port”

แต่ละ port จะรันโปรแกรมแยกกันและจะติดต่อกันผ่าน port เราต้องกำหนดว่าโปรแกรมไหนจะรันบน port อะไร ถ้าไม่กำหนดก็จะใช้ default port

เราจะแยก frontend กับ backend เป็น 2 server (port) ก็ได้ หรือจะรวมกันเป็น 1 อันก็ได้ เพราะว่าการที่ผู้ใช้ request HTML ก็เป็นการ request แบบนึงเหมือนกัน



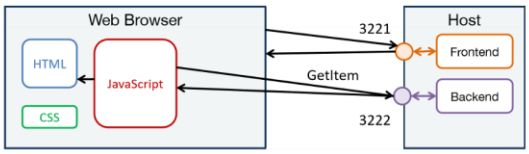
IP = 18.136.165.66
Name = lab.aimet.tech

บนหน้าเว็ปมีได้ทั้ง static object (from files) หรือจะสร้างจาก on-the-fly ก็ได้

และสามารถเป็นได้ทั้งแบบ ready-for-rendering (HTML, CSS) หรือจะเป็นข้อมูลที่ rendered by client (JSON, XML)

Activity 7

Server in our Activity



- **Frontend Server** – a server program (port 3221) that serves web static objects for frontend e.g. HTML, CSS, javascript
- **Backend server** – a server program (port 3222) that serves data to frontend via REST API
- First web browser get static objects from frontend server and run JavaScript
- Once web browser runs JavaScript, it will get data from backend server and render with HTML

- ใช้ curl -v [IP_ADDRESS] เพื่อให้ curl แสดงข้อมูลรายละเอียดของการทำงานของการเชื่อมต่อไปที่ IP_ADDRESS โดยจะแสดงออกมาเป็นหลายบรรทัด แบ่งเป็น 2 ส่วนหลัก คือ request (เริ่มบรรทัดด้วย >) คั่นด้วยบรรทัดเปล่าและ response (เริ่มบรรทัดด้วย <, จะมี header-body ที่ใน body จะมีโค้ดต่างๆ อยู่)

- ใช้ Postman ในการลองเรียกใช้ API ที่มี (เหมือนเราส่ง req ที่กำหนดเองไป แล้วเซิฟเวอร์ จะส่ง response กลับมา)

File	Description
backend/src/app.js	initialize express object and register routes
backend/src/routes/ItemRoute.js	register methods (GET, POST, DETELE) for the endpoint
backend/src/controllers/itemController.js	contain functions that handle methods

- creates an express object ด้วยคำสั่ง const app = express();

- คำสั่ง app.get("/items/:id", (req, res) => { ... }) ใช้หาข้อมูลที่มี id เหมือนใน URI

- คำสั่ง const { id } = req.params

สร้างตัวแปรชื่อ id โดยมีค่าเหมือนตัวแปร id ใน URI

Cloud computing

คล้ายกับการเก็บข้อมูล + จัดการ service ต่างๆ บน local storage แต่ว่าอยู่บนอินเตอร์เน็ต ซึ่งเป็นแบบตามความต้องการ ใช้เท่าไหร่จ่ายเท่านั้น (pay-as-you-go)

เช่น AWS (Amazon web services), Microsoft Azure Google cloud platform

Models

แบ่งเป็น deploy model 3 ประเภท คือ

- public : โครงสร้างสามารถเข้าถึงได้ผ่านอินเตอร์เน็ต (เป็นของ cloud providers)
- private : โครงสร้างถูกจัดการโดย 1 บริษัท / third-party
- hybrid : แบบผสมระหว่าง public + private

และ service model 3 ประเภท คือ

- IaaS (Infrastructure as a service) : สามารถเข้าถึง infrastructure พื้นฐานของคอมพิวเตอร์ได้ เหมาะกับการทำงานที่ต้องแก้ไข storage & virtual machine เช่น Amazon Web Services, Microsoft Azure, and Google Compute Engine, EC2
- PaaS (Platform as a service) : เหมาะกับการสร้าง application แต่ไม่ต้องการปรับแก้โครงสร้างเอง (ผู้ให้บริการ cloud จะจัดการ platform, runtime environment ให้เหมาะกับการ developing, testing, manage application) เช่น Vercel, Netlify, GitHub, GitLab, Docker, Red Hat OpenShift Online, Google App Engine
- SaaS (Software as a service) : ผู้ใช้แค่มาใช้งาน service ที่ผู้ให้บริการทำให้เฉยๆ ผู้บริการจัดการทั้ง hardware, software เช่น Gmail, Slack, and Microsoft Office 365

ข้อดีข้อเสีย

Pros of cloud computing

Topics	On-premises	Cloud
Scalability	1. Pay more 2. Lesser options 3. Difficult to scale down 4. Heavy losses for infrastructure & maintenance	1. Pay for how much you use 2. Easier for scaling up-down
Storage	1. Needs a lot of space 2. More power & maintenance hassles	1. Don't need space (provider manage them)
Security	1. Less	1. Better 2. Don't need much monitoring and managing security protocol
Data loss	1. Low chance to recover loss data	1. Have a good recovery system
Maintenance	1. Require hardware & software team 2. More cost	1. Don't need to maintain (provider manage them)

Database

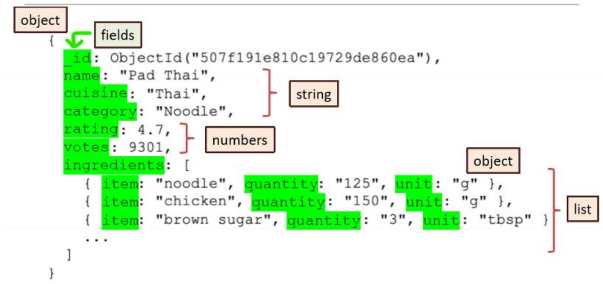
no-sql -> ข้อมูลไม่จำเป็นต้องเกี่ยวข้องกัน (ต่างจาก SQL ที่เป็น relational database)

เป็น database แบบ document-oriented & no-sql* ที่เป็นที่นิยม ซึ่งใช้ฟรีและสามารถ host ด้วยตัวเองได้ หรือจะ host ด้วย atlas ได้ (มีแบบที่ฟรีและเสียตัง)

Atlas (serverless / cloud platform ของ mongoDB) สามารถ scale อัตโนมัติได้ -> มีข้อดีคือจ่ายแค่เท่าที่ใช้ และมี UI และ feature สำหรับจัดการข้อมูล (เช่น text search, triggers -> ดูการเปลี่ยนแปลงของข้อมูล) เหมาะกับการใช้กับ Apache Lucene

สร้างในปี 2007 โดย DoubleClick company หลังจากเจอปัญหาเรื่องความสเถียรและความยืดหยุ่นของ database อื่น

MongoDB สามารถจัดการข้อมูลจำนวนมากได้



ข้อมูลไม่ได้ถูกเก็บในตาราง แต่จะเก็บใน document (ทำหน้าที่เหมือนตารางแถวเดียว) ซึ่งจะเก็บข้อมูล object เป็นคู่ field & value (เก็บเป็น BSON ที่คล้ายๆ กับ JSON)

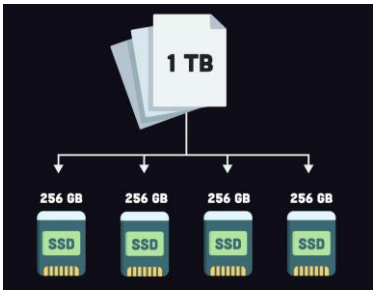
แต่ละ object จะมี unique id ของมันเอง (สร้างให้อัตโนมัติ) ที่ไม่ซ้ำกับตัวอื่นใน collection

โดยที่จะมีหรือไม่มี schema (โครงสร้างข้อมูลหรือนิยามข้อมูล รวมถึงความสัมพันธ์ของข้อมูลในแต่ละตัวว่ามี ความสัมพันธ์กันอย่างไร) ก็ได้ ทำให้สามารถเปลี่ยนโครงสร้างข้อมูลได้ง่าย

โดยมีรูปแบบในการเก็บคือ Database > Collections > Documents

ถ้าข้อมูลไหนถูกเรียกใช้บ่อยๆ ก็จะเอามาเก็บไว้ด้วยกัน -> อ่านข้อมูลเร็ว (เพราะไม่ต้อง join เหมือนใน SQL database)

สามารถปรับขนาดได้ง่าย ด้วยการแบ่งเป็นส่วนๆ (แนวนอน) เพราะแต่ละ collection จะเก็บข้อมูลแค่ในตัวมันเอง



เวลาเราเรียกใช้ API แล้วมาต่อกับ database ก็จะเป็นการแก้ข้อมูลใน document นั้นๆ

ซึ่งใน document ก็จะสามารถ query ได้

Query API -> เป็นเครื่องมือพื้นฐานที่ใช้ในการอ่าน-เขียนและจัดการข้อมูลใน database เช่นการ sort, filter

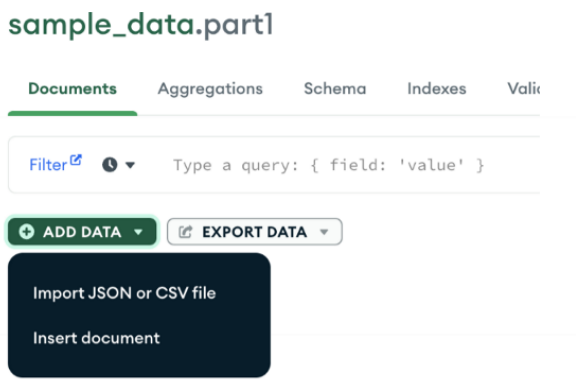
มี index ที่เป็น lookup table พิเศษที่ใช้ในการหาข้อมูลที่ผู้ใช้หาบ่อยๆ

ซึ่ง Query API มีฟังก์ชัน geospatial queries (หาข้อมูลที่อยู่ใกล้กัน -> ที่อยู่ทาง geographic)

และมีฟังก์ชัน data aggregation pipeline ที่ใช้รวม document ไว้ด้วยกัน เพื่อลดผลลัพธ์ลงมาเป็นอันเดียว

การใช้ MongoDB

1. เชื่อมต่อ MongoDB บน cloud / database ที่เรา host เอง (default port คือ 27017)
2. สร้าง Database & collection
3. Add and import data



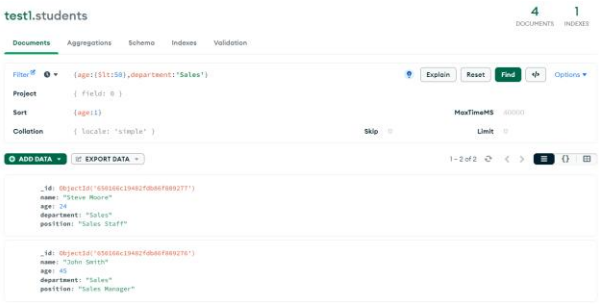
(อันบนเพิ่มข้อมูลหลายตัวพร้อมกัน, อันล่างเพิ่มทีละตัว)

4. การใช้ Query

- Simple query : พิมพ์ในช่อง filter ด้วย { field : value } เพื่อเลือกเฉพาะ object ที่มี field นั้น เป็นค่า value เช่น { department : “sales” }
- Numerical query : ถ้าต้องการเปรียบเทียบค่า value ในการ filter จะเปลี่ยนจากค่า value เฉยๆ เป็น { $gt: value } (ค่าที่มากกว่า value), { $lt: value } (ค่าที่น้อยกว่า value)
- AND query : ให้ใช้ , ต่อด้วย filter ที่ต้องการในปีกกาเดียวกัน เช่น { department : “sales”, age : 30 }
- OR query : พิมพ์ในช่อง filter ด้วย { $or: [FILTERS] } เช่น { $or: [{department: "Sales"}, {department: "HR"}]}

* สามารถใช้รวมกันได้ เช่น { Universe: “Marvel”, Appearance : {$gt:500}, $or: [{Alignment: "Good"}, {Alignment: "Neutral"}] } -> คือเอาตัวละคร Marvel ที่ดูอายุเกิน 500 และอยู่ฝั่ง good / neutral *

- Sort : พิมพ์ในช่อง sort ด้วย { field : value } เพื่อเรียงลำดับข้อมูลใหม่ตามค่า field ถ้า value เป็น -1 จะเรียงจากมากไปน้อย แต่ถ้าเป็น 1 จะเรียงจากน้อยไปมาก
- Limit : เพื่อกำหนดว่าจะแสดงข้อมูลทั้งหมดกี่ตัว



การใช้งานกับ Frontend