

Go (Golang) for Web Services

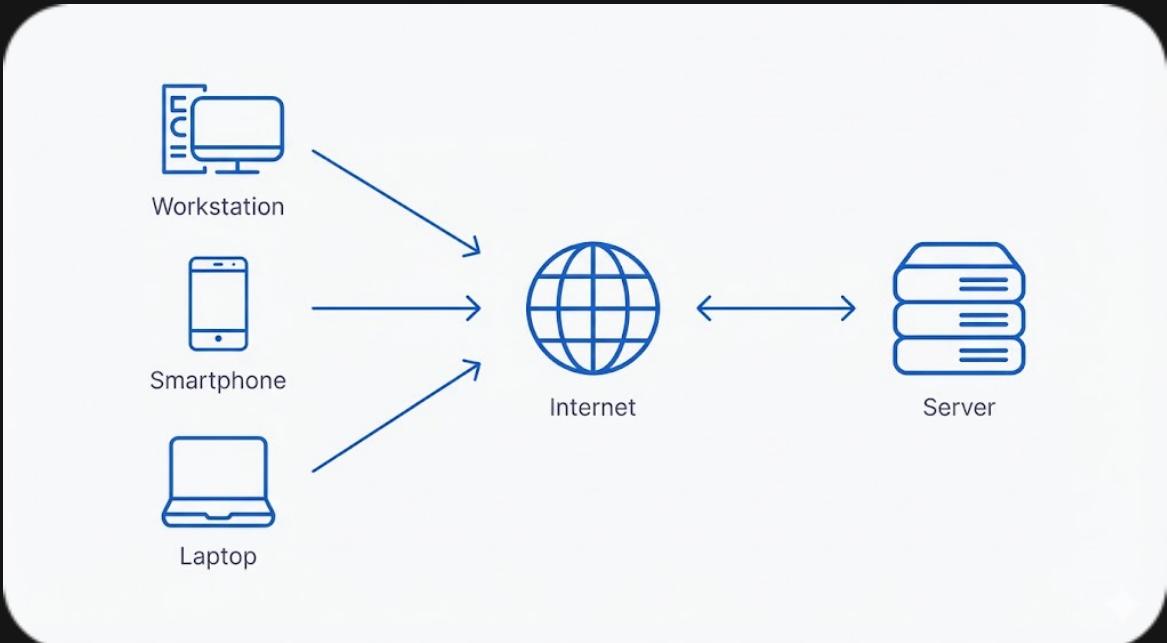
I-bitz Company Limited
Oakkharit Phichitart (Software Engineer)

Content

- 01 HTTP
- 02 REST API
- 03 OGC APIs
- 04 Go Basic
- 05 SQL & No SQL

HTTP

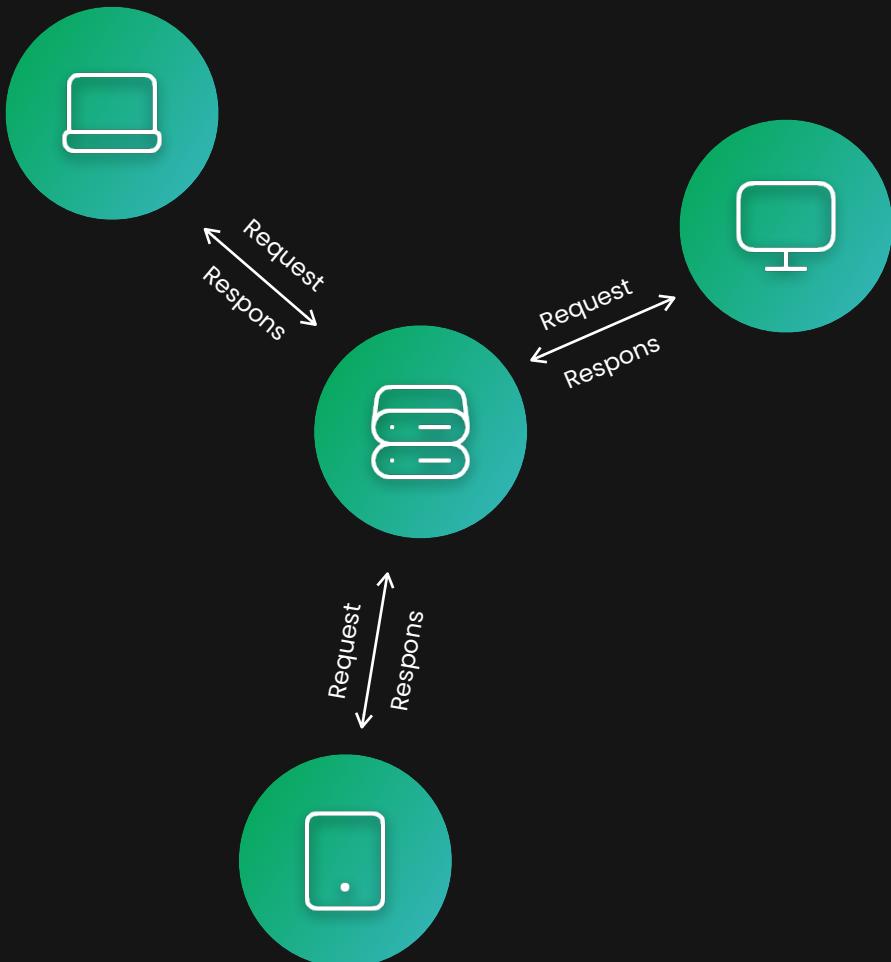
HTTP



API (Application Programming Interface) คือ ช่องทางที่ให้ระบบอื่น สามารถเรียกใช้เพื่อเข้าถึงข้อมูลหรือฟังก์ชันการทำงานได้ API เปรียบเสมือน "สะพาน" เชื่อมระหว่างโปรแกรมหรือระบบที่แตกต่างกันทำให้สามารถสื่อสารและส่งข้อมูลถึงกันได้

Protocol หมายถึง รูปแบบหรือข้อกำหนดภาษาที่ทั้งสองระบบตกลงใช้ร่วมกัน เพื่อให้สื่อสารและเข้าใจตรงกันได้

HTTP | Server & Client



ระบบ **Client-Server** ประกอบด้วย 2 บทบาทหลัก คือ **Client** และ **Server**

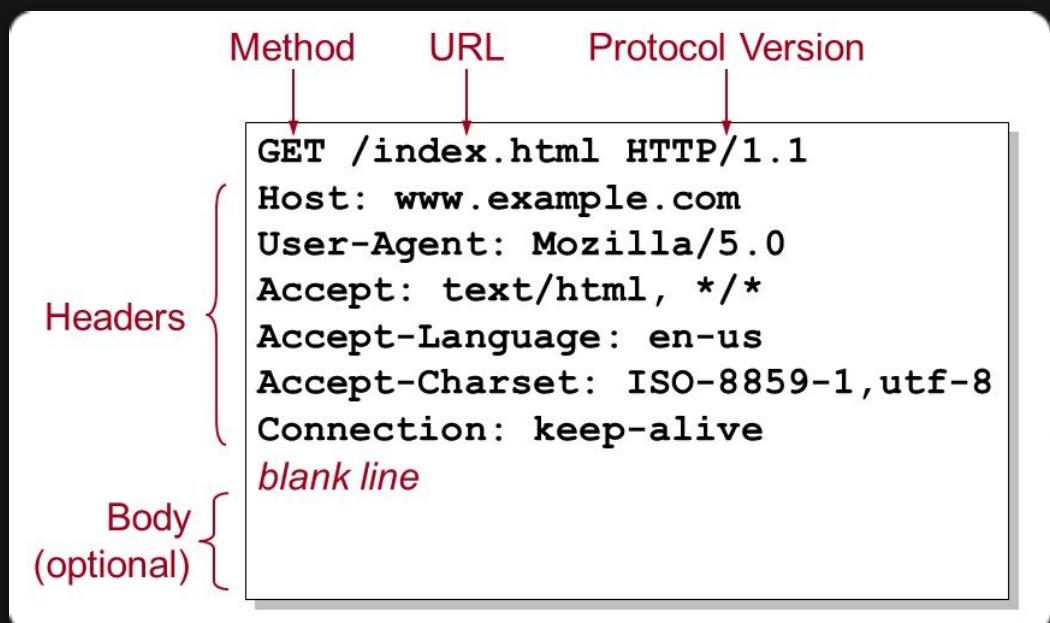
Server

ทำหน้าที่รอรับคำร้องขอ (**Request**) จาก Client และประมวลผลก่อนส่งผลลัพธ์กลับไป (**Response**) Server จะเปิดทำงานตลอดเวลา และ **ไม่สามารถเริ่มการสื่อสารเองได้** ต้องรอให้ Client ส่ง Request มา ก่อนเสมอ

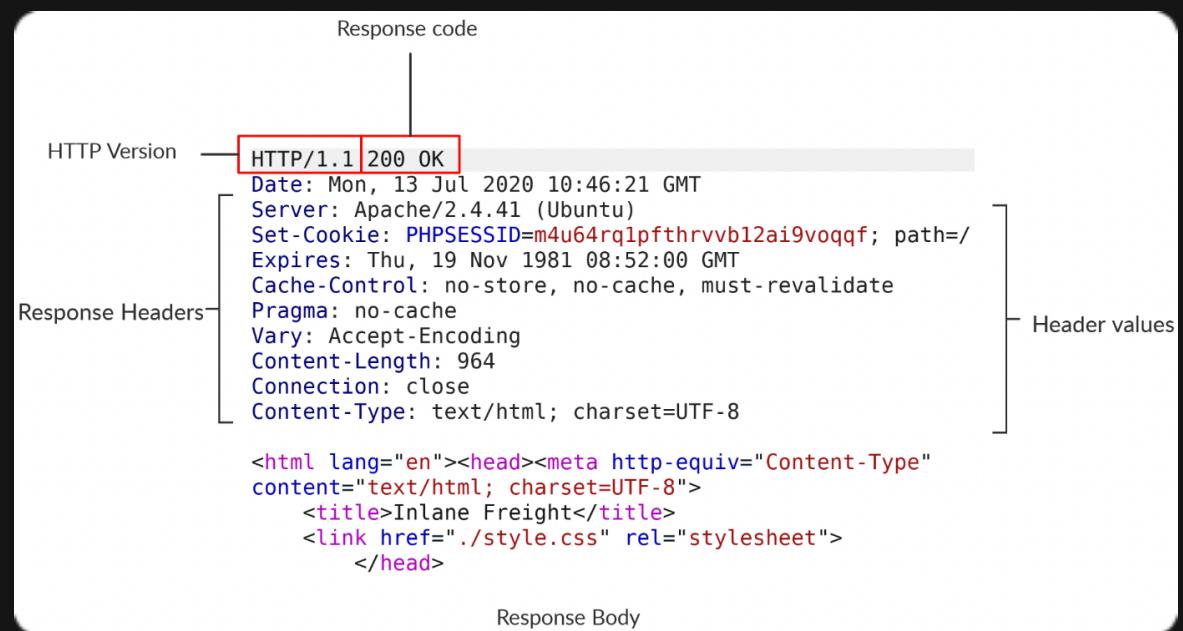
Client

ทำหน้าที่เป็นผู้เริ่มต้นการสื่อสาร โดยส่ง Request ไปยัง Server ผ่านเครือข่าย (เช่น Internet) จากนั้น Client จะรอรับ Response ที่ Server ส่งกลับมา เช่น ข้อมูล หน้าเว็บ หรือผลลัพธ์ต่าง ๆ

HTTP | Request & Response



HTTP Request



HTTP Response

HTTP | Method

Method	Description
POST	ใช้สำหรับสร้างข้อมูลใหม่
GET	ใช้สำหรับเรียกดูหรือดึงข้อมูล
PUT	ใช้สำหรับอัพเดตหรือแทนที่ข้อมูลเดิมที่มีอยู่แล้ว
PATCH	ใช้สำหรับอัพเดตหรือแก้ไขข้อมูลเฉพาะบางส่วน
DELETE	ใช้สำหรับลบข้อมูล

REST API

REST API



REST (Representational State Transfer) คือแนวคิดในการออกแบบ API สำหรับสื่อสารกันผ่าน HTTP ไม่ใช่ โปรโตคอลใหม่ แต่เป็น design principle ว่าควรออกแบบ API ยังไงให้เรียบง่าย ใช้งานง่าย และขยายระบบได้ดี

REST = แนวคิด (architectural style)

REST API = API ที่ออกแบบตามแนวคิด

REST บน HTTP REST ≠ framework ≠ protocol

REST API | API Design

ออกแบบ endpoint โดยไม่ยึดรูปแบบ REST

PUT: /findAllUser

DELETE: /findAllUser

POST: /findOneUser

GET: /createUser

ออกแบบ endpoint โดยยึดรูปแบบ REST

POST: /users

GET: /users

GET: /users/{id}

PUT: /users/{id}

DELETE: /users/{id}

OGC APIs

OGC APIs



Open
Geospatial
Consortium

เป็นมาตรฐานกลาง ที่องค์กรต่าง ๆ นำมาใช้ร่วมกันเพื่อการแลกเปลี่ยนข้อมูลภูมิศาสตร์ (Spatial Data) อย่าง
เป็นระบบและสอดคล้องกัน

มาตรฐานนี้ครอบคลุมตั้งแต่ การออกแบบ API, โครงสร้างข้อมูลที่ใช้ลีอสารภายในระบบ ไปจนถึงการกำหนด
ระบบพิกัด (Coordinate System) เพื่อให้สามารถใช้งานและแลกเปลี่ยนข้อมูลระหว่างระบบได้อย่างถูกต้องและ
มีประสิทธิภาพ

OGC APIs | Geo JSON

ตัวอย่างการออกแบบ API จัดการข้อมูลท่อ

สร้างข้อมูลท่อ

POST: /collections/pipes/items

เรียกดูข้อมูลท่อทั้งหมด

GET: /collections/pipes/items?bbox=100.4,13.6,100.7,13.9

เรียกดูข้อมูลท่อ 1 รายการ

GET: /collections/pipes/items/pipe-001

แก้ไขข้อมูล

PUT: /collections/pipes/items/pipe-001

ลบข้อมูลท่อ

DELETE: /collections/pipes/items/pipe-001

Geo JSON

```
{  
  "type": "Feature",  
  "geometry": {  
    "type": "Polygon",  
    "coordinates": [  
      [  
        [100.5, 13.7],  
        [100.6, 13.7],  
        [100.6, 13.8],  
        [100.5, 13.8],  
        [100.5, 13.7]  
      ]  
    ]  
  },  
  "properties": {  
    "titleDeedNo": "12345",  
    "owner": "dev",  
    "area": 1200  
  }  
}
```

Go Basic

Go (Golang)



Go หรือ Golang คือภาษาโปรแกรมที่ Google ออกแบบมาเพื่อแก็บปัญหาใหญ่ ๆ ของระบบยุคใหม่ เช่น server ที่รองรับผู้ใช้งานจำนวนมาก หรือ ระบบที่ต้องเร็ว เสถียร และ deploy ง่าย

ก่อน Go จะเกิดขึ้น นักพัฒนาที่ทำในส่วนของหลังบ้านจะมักเจอกาชาลักษณะดังนี้

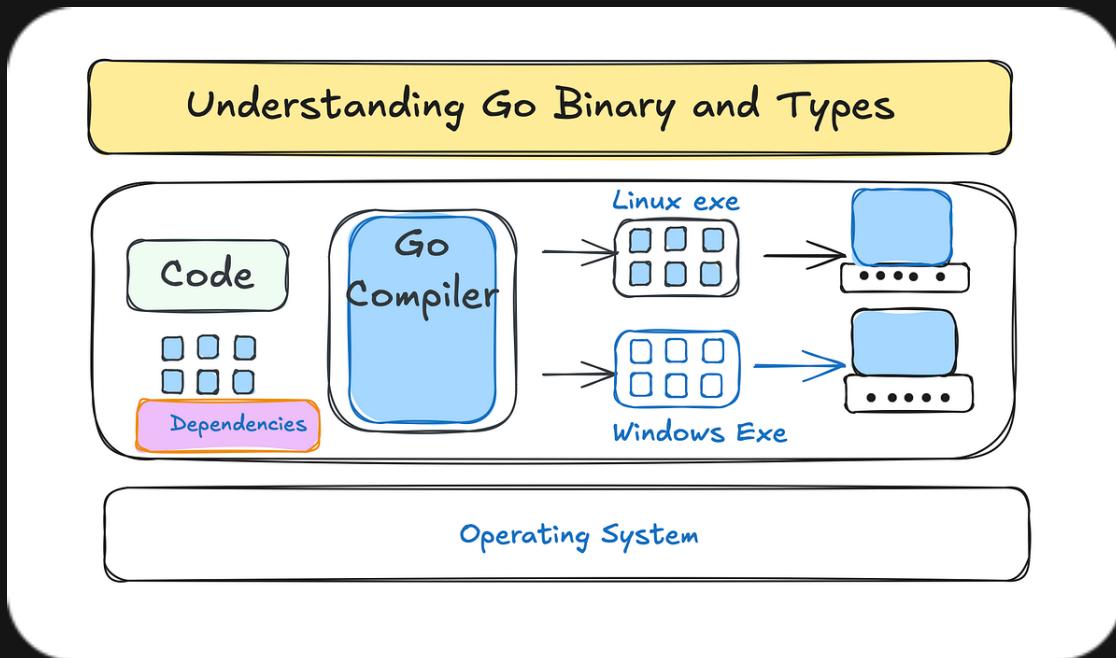
C/C++ เร็ว แต่ซับซ้อน ดูแลยาก

Java ปลอดภัยกว่า แต่ช้า Compile นาน

Python / JS เขียนง่าย แต่ช้าสำหรับงานหนัก

“ซึ่ง Go เป็นภาษาที่เร็วแบบภาษา C/C++ แต่เขียนง่ายแบบ Python”

Go : Compiler



OS	GOOS	ไฟล์ที่ได้	Run Commands
Windows	windows	app.exe	ดับเบิลคลิก / app.exe
macOS	darwin	app	./app
Linux	linux	app	./app

*นอกจาก Complier แล้วยังมี Interpreter ที่สามารถแปลง code เป็นคำสั่งให้ Computer ทำงานได้

compile เป็น binary ได้ไฟล์ binary

- go build -o app

compile เป็น binary แต่ไม่สร้างไฟล์

- go run . หรือ go run main.go

Go : Go Tour

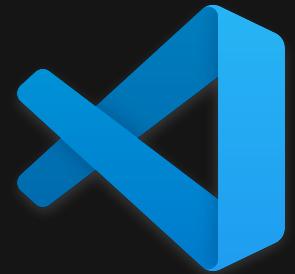
The screenshot shows a web browser window titled "A Tour of Go". The main content area displays the first page of the Go Tour, which includes a greeting message "Hello, สวัสดี" and some introductory text in Thai. To the right of the text is a code editor window titled "hello.go" containing the following Go code:

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Println("Hello, สวัสดี")
7 }
```

The code editor has tabs for "Imports off" and "Syntax off". Below the code editor is a small cartoon gopher character. At the bottom of the page, there are navigation links for "previous" and "next", and a page number indicator "< 1/5 >".

<https://go.dev/tour/welcome/1>

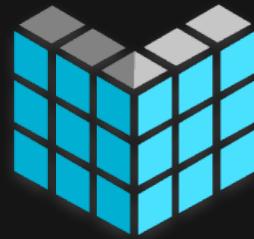
Go : Tools



Visual Studio Code



Go Compiler



Echo
(Go Framework)



Postman

Go Workshop Basic Syntax

Go Workshop : Training

พื้นที่ทดลองเขียนโค้ด
<https://go.dev/play>

เอกสาร Echo Framework
<https://echo.labstack.com>

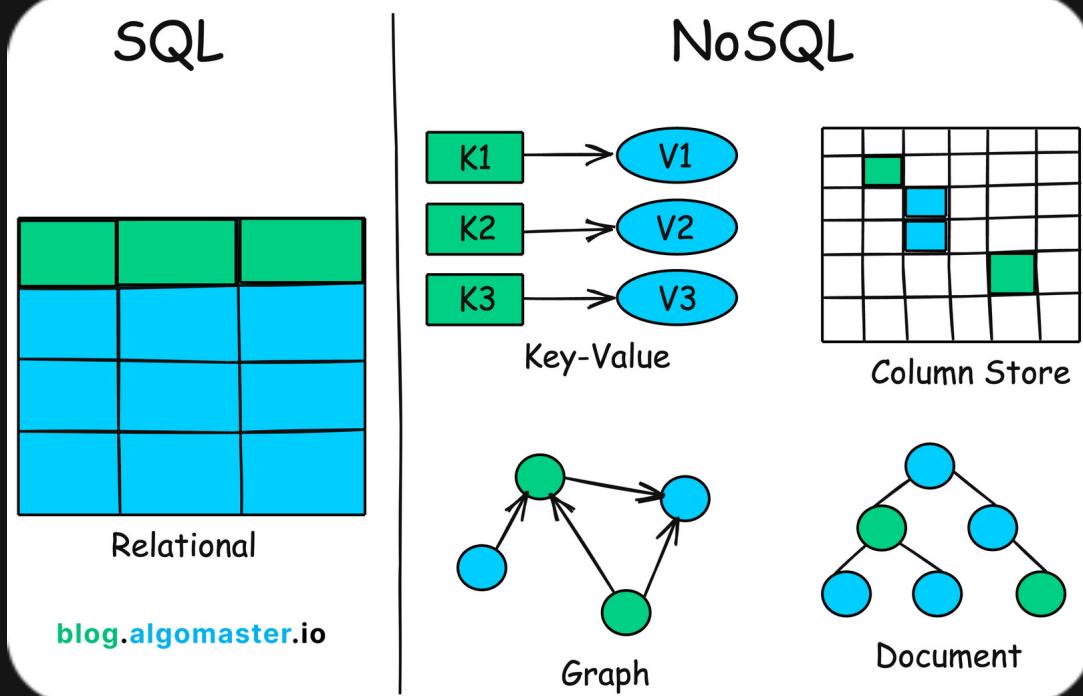
เอกสารสรุปตัวอย่าง โค้ดพื้นฐาน
<https://devhints.io/go>

Go Workshop : สร้าง REST API ด้วย Golang



SQL & No SQL

SQL & No SQL



SQL และ NoSQL เป็นระบบจัดการฐานข้อมูลที่มีแนวคิดและโครงสร้างแตกต่างกัน

SQL เป็นฐานข้อมูลเชิงสัมพันธ์ (**Relational Database**) ใช้โครงสร้างแบบตาราง (**Table**) ที่มีรูปแบบชัดเจนและกำหนดโครงสร้างล่วงหน้า เมามะสำหรับข้อมูลที่ต้องการความถูกต้องและความสอดคล้องสูง เช่น ระบบบัญชี ระบบการเงิน หรือข้อมูลธุรกิจในขณะที่

NoSQL เป็นฐานข้อมูลที่ไม่ใช้เชิงสัมพันธ์ (**Non-Relational Database**) มีความยืดหยุ่นสูง (**Dynamic Schema**) และสามารถจัดเก็บข้อมูลได้หลายรูปแบบ เช่น Document, Key-Value หรือ Graph เมามะสำหรับข้อมูลขนาดใหญ่หรือข้อมูลที่มีโครงสร้างไม่ตายตัว เช่น ข้อมูลจากโซเชียลมีเดีย หรือระบบที่ต้องรองรับการเปลี่ยนแปลงบ่อย

SQL & No SQL | Compare

หัวข้อ	SQL (PostgreSQL)	NoSQL (MongoDB)
Model	Table / Row	Document
Schema	Fixed	Flexible
Query	SQL	JSON-like
Transaction	Strong (ACID)	จำกัดกว่า
Join	ดีมาก	ไม่เน้น
Scale	Vertical	Horizontal

SQL & No SQL | Tools



PostgreSQL
pgAdmin



MongoDB
Mongo Compass

SQL & No SQL | Go Database Connecting

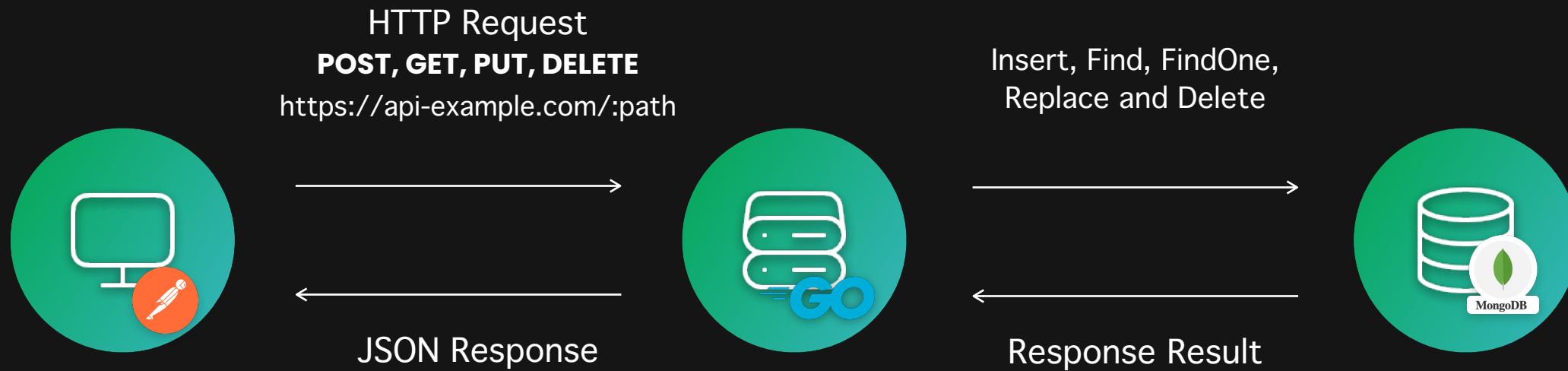
```
1 // คำสั่งติดตั้งไ/drเวอร์ PostgreSQL:  
2 // go get github.com/jackc/pgx/v5/stdlib  
3  
4 func connectPostgres(uri string) (*sql.DB, error) {  
5     db, err := sql.Open("pgx", uri)  
6     if err != nil  
7         return nil, err  
8 }  
9  
10    // timeout เป้องกัน connection ล้าง  
11    ctx, cancel := context.WithTimeout(context.Background(), 5*time  
 .Second)  
12    defer cancel()  
13  
14    if err := db.PingContext(ctx); err != nil  
15        return nil, err  
16    }  
17  
18    return db, nil  
19 }  
20 }
```

MongoDB

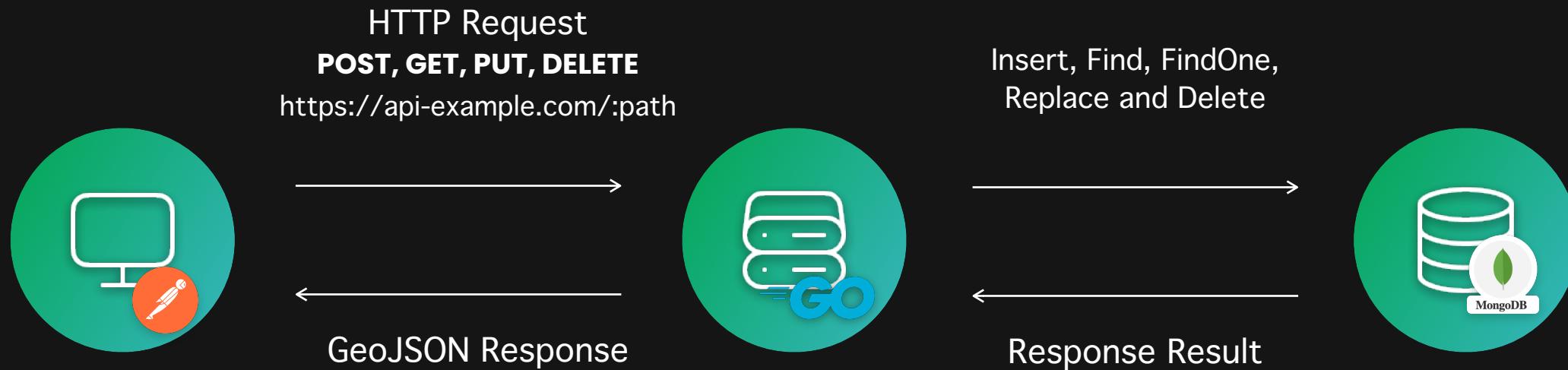
```
1 // คำสั่งติดตั้งไ/drเวอร์ MongoDB:  
2 // go get go.mongodb.org/mongo-driver/mongo  
3  
4 func connectMongo(uri string) (*mongo.Client, error) {  
5  
6     ctx, cancel := context.WithTimeout(context.Background(), 10*  
 time.Second)  
7     defer cancel()  
8  
9     client, err := mongo.Connect(ctx, options.Client().ApplyURI(uri  
 ))  
10    if err != nil  
11        return nil, err  
12    }  
13  
14    if err := client.Ping(ctx, nil); err != nil  
15        return nil, err  
16    }  
17  
18    return client, nil  
19 }
```

PostgreSQL

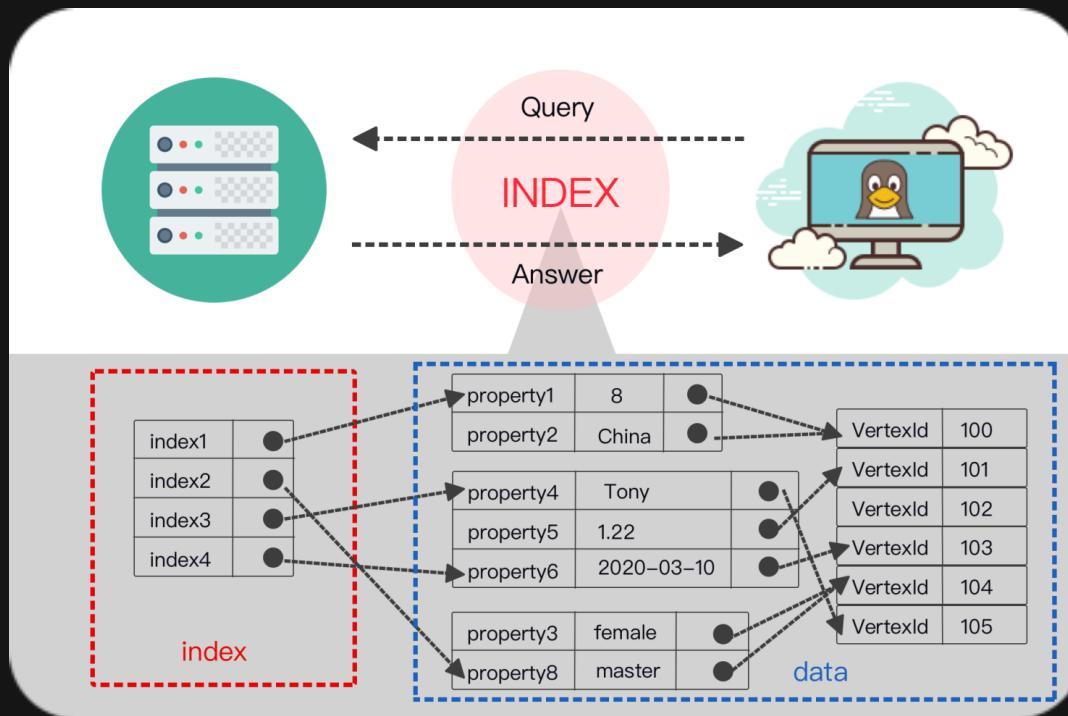
Go Workshop : จัดการข้อมูลด้วย REST API (CRUD)



Go Workshop : สร้าง REST API จัดการข้อมูล GeoJSON



SQL & No SQL | Performance and Indexing

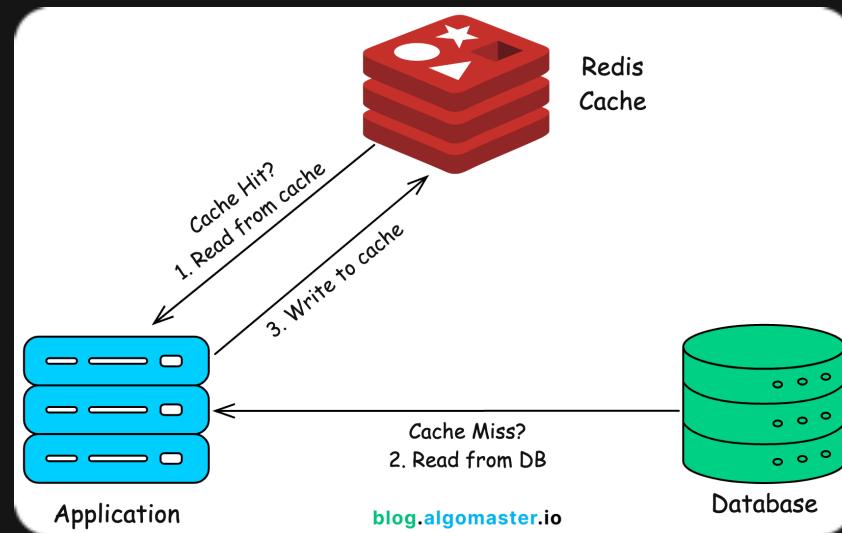


ประสิทธิภาพการทำงานของ **API (Application Programming Interface)** เป็นปัจจัยสำคัญที่ส่งผลโดยตรงต่อประสบการณ์ของผู้ใช้งาน ความรวดเร็วในการพัฒนา รวมถึงต้นทุนในการดำเนินงานของระบบ โดยประสิทธิภาพของ API สามารถพิจารณาได้จากหลายด้าน เช่น ความเร็วในการตอบสนอง (**Response Time**), ความสามารถในการรองรับคำขอจำนวนมาก (**Throughput**) และความเสถียรของระบบ (**Stability**)

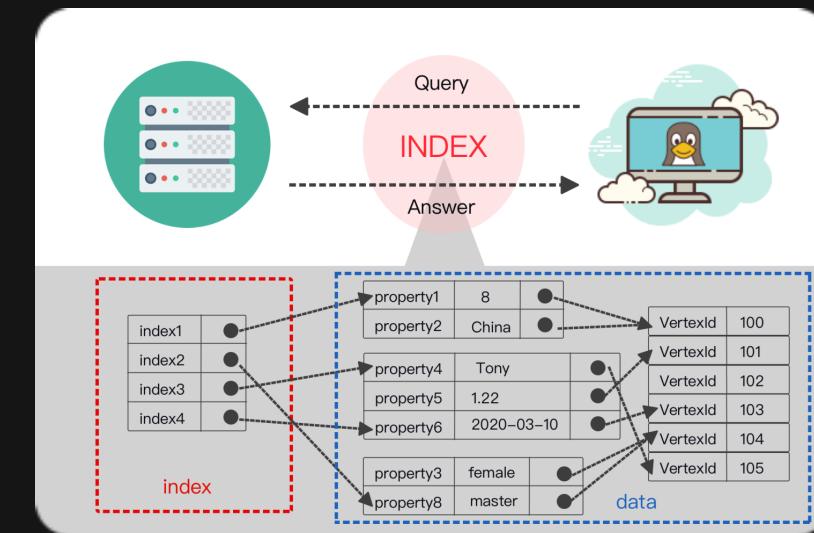
SQL & No SQL | Performance Metrics

ตัวชี้วัดประสิทธิภาพของ API (API Performance Metrics)

- **Response Time / Latency** ระยะเวลาที่ API ใช้ในการตอบสนอง หลังจากได้รับคำขอจาก Client
- **Throughput (Requests per Second)** จำนวนคำขอ (Requests) ที่ API สามารถประมวลผลได้ต่อหน่วยเวลาซึ่งสะท้อนความสามารถในการรองรับผู้ใช้งานพร้อมกันจำนวนมาก
- **Error Rate** อัตราการเกิดข้อผิดพลาดของ API เช่น การตอบกลับด้วย Error Code หรือกรณีที่ระบบไม่สามารถให้บริการได้
- **Resource Utilization** การใช้ทรัพยากรของเครื่องแม่ข่าย เช่น CPU, Memory และทรัพยากระบบอื่น ๆ ระหว่างการให้บริการ



Caching



Indexing

หัวข้อแนะนำศึกษาเพิ่มเติม

Authentication & Authorization

- JWT
- OAuth2

Go Advanced Programming

- Go Context
- Go Routine (Concurrent Programming)
- Go Pointer
- Error handling

Database Design

- Database normalisation (SQL)
- Schema design (Mongo)
- Transaction Lock / Deadlock

API Performance & Scalability

- Caching (Redis)
- Connection pool (Reuse Connection)
- Scaling API & Database

Advanced Programming

- Object Oriented Programming (OOP)
- Data structure & Algorithm
- Design Patterns > Dependency Injection

ขอบคุณครับ