GO WEB PROGRAMMING:
BEGINNER
(INTEGRATIVE-GENERATIVE AI EDITION)

Author: Student Price Book Center

คำนำ

โลกของการพัฒนาเว็บแอปพลิเคชันในปัจจุบันเติบโตอย่างรวดเร็วและมีความซับซ้อนมากขึ้นทุกวัน นักพัฒนาจำเป็นต้องมีความรู้รอบด้าน ตั้งแต่แนวคิดพื้นฐานของการทำเว็บ การสื่อสารระหว่าง Client และ Server ไปจนถึงการออกแบบระบบที่มีความเสถียรและขยายตัวได้ สำหรับผู้เริ่มต้น ภาษา Go (Golang) เป็นเครื่องมือที่ยอดเยี่ยม ด้วยความเรียบง่าย ประสิทธิภาพสูง และรองรับการประมวลผลพร้อมกัน (Concurrency) ทำให้สามารถสร้างเว็บแอปพลิเคชันได้รวดเร็วและมีประสิทธิภาพ

หนังสือเล่มนี้ถูกออกแบบมาเพื่อเป็นคู่มือ **สำหรับผู้เริ่มต้น (Beginner)** ที่ต้องการเรียนรู้การพัฒนาเว็บด้วย Go ตั้งแต่พื้นฐานไปจนถึงการสร้างโปรเจกต์จริง โดยแต่ละบทถูกจัดเรียงอย่างเป็นลำดับขั้นตอน เริ่มจากแนวคิดพื้นฐานของ Web Programming ในบทที่ 1 ผู้อ่านจะได้ทำความเข้าใจ **สถาปัตยกรรม Client-Server, HTTP Protocol** และประโยชน์ของการใช้ Go ในการพัฒนาเว็บ พร้อมแนวทางการติดตั้ง Go สำหรับเริ่มต้นพัฒนาเว็บแอปพลิเคชัน

บทที่ 2 จะนำเสนอวิธีการเริ่มต้นใช้งาน **package net/http** ซึ่งเป็นหัวใจของการสร้าง HTTP server ด้วย Go โดยเน้นการสร้าง HTTP server แบบง่าย การรับคำร้องขอและส่งข้อมูลตอบกลับ รวมถึงการจัดการ HTTP Methods พื้นฐานอย่าง GET, POST, PUT, DELETE นอกจากนี้ยังมีตัวอย่างโปรเจกต์บูรณาการเพื่อให้ผู้อ่านได้ฝึกปฏิบัติจริง

ในบทที่ 3 หนังสือเล่มนี้จะพาผู้อ่านเจาะลึกเรื่อง **Routing** ซึ่งเป็นการกำหนดเส้นทางของคำร้องขอไปยัง Handler ที่เหมาะสม ทั้งการใช้ http.HandleFunc การสร้าง Custom Router ของตัวเอง และการใช้ Third-party routers เช่น gorilla/mux และ chi เพื่อจัดการ URL pattern และ Middleware สำหรับระบบที่ซับซ้อน

บทที่ 4 มุ่งเน้นการ **Handling Requests and Responses** ซึ่งเป็นพื้นฐานสำคัญของเว็บเซิร์ฟเวอร์ ผู้อ่านจะได้เรียนรู้วิธีอ่านข้อมูลจากคำร้องขอหลายรูปแบบ เช่น query parameters, form data และ JSON payload พร้อมทั้งการตั้งค่า Response Headers, Status Code และการส่งข้อมูลกลับในรูปแบบ JSON, HTML และไฟล์ต่าง ๆ ทั้งนี้ยังมีตัวอย่างโปรเจกต์บูรณาการที่ช่วยให้เข้าใจแนวคิดได้ชัดเจน

บทที่ 5 นำเสนอ **Templates in Go** ผ่าน package html/template เพื่อสร้าง dynamic HTML ผู้อ่านจะได้เรียนรู้การ render HTML ตามข้อมูลจริง การใช้ Template Functions สำหรับประมวลผลข้อมูลภายใน template และ Template Inheritance สำหรับสร้างโครงสร้างหน้าเว็บที่ซ้ำกัน เช่น header และ footer การใช้ template อย่างถูกต้องช่วยให้เว็บแอปพลิเคชันทั้ง **สวยงาม ปลอดภัย และยืดหยุ่น**

หนังสือเล่มนี้เน้นทั้ง **ความเข้าใจเชิงลึกและการปฏิบัติจริง** โดยแต่ละบทมีตัวอย่างโค้ดและโปรเจกต์บูรณาการ เพื่อให้ผู้อ่านสามารถนำความรู้ไปสร้างเว็บแอปพลิเคชันจริงได้ทันที เหมาะสำหรับผู้ที่ต้องการก้าวจากการเรียนรู้ภาษา Go ไปสู่การพัฒนาเว็บแอปพลิเคชันที่ครบวงจร

ด้วยโครงสร้างและการจัดเรียงเนื้อหาอย่างเป็นระบบ หนังสือเล่มนี้จะเป็นทั้งคู่มือและแนวทางปฏิบัติสำหรับผู้เริ่มต้น ให้สามารถเรียนรู้ **Web Programming ด้วย Go** ตั้งแต่พื้นฐานจนถึงการสร้างเว็บแอปพลิเคชันที่มีคุณภาพ พร้อมต่อยอดไปสู่การพัฒนาเว็บแอปขนาดใหญ่ในอนาคต

ด้วยความปรารถนาดี

ศูนย์หนังสือราคานักเรียน

สารบัญ

หน้า

บทที่ 1

Introduction to Web Programming in Go

(Introduction to Web Programming in Go)

เนื้อหา

บทที่ 1: Introduction to Web Programming in Go

การพัฒนาเว็บแอปพลิเคชันในปัจจุบันเป็นหนึ่งในทักษะพื้นฐานที่นักพัฒนาควรมีความเชี่ยวชาญ การทำความเข้าใจหลักการและเทคโนโลยีที่อยู่เบื้องหลังเว็บเป็นสิ่งสำคัญไม่ว่าจะเป็นการออกแบบระบบการจัดการข้อมูล หรือการสื่อสารระหว่างผู้ใช้และเซิร์ฟเวอร์ ในบทนี้ ผู้อ่านจะได้เรียนรู้แนวคิดเบื้องต้นเกี่ยวกับ **Web Programming** และวิธีการสร้างเว็บแอปพลิเคชันที่มีประสิทธิภาพและยืดหยุ่น

พื้นฐานสำคัญของการพัฒนาเว็บคือ **สถาปัตยกรรม Client-Server** ซึ่งอธิบายถึงการทำงานของผู้ใช้ (Client) และเซิร์ฟเวอร์ (Server) ในการแลกเปลี่ยนข้อมูล ผู้ใช้ส่งคำร้องขอข้อมูลไปยังเซิร์ฟเวอร์ และเซิร์ฟเวอร์ตอบกลับด้วยข้อมูลหรือผลลัพธ์ตามคำร้องนั้น การเข้าใจสถาปัตยกรรมนี้ช่วยให้นักพัฒนาสามารถออกแบบระบบที่สามารถรองรับผู้ใช้จำนวนมากและจัดการทรัพยากรได้อย่างมีประสิทธิภาพ

อีกหนึ่งองค์ประกอบสำคัญของการพัฒนาเว็บคือ **HTTP Protocol** ซึ่งเป็นมาตรฐานสำหรับการสื่อสารระหว่าง Client และ Server HTTP กำหนดรูปแบบคำร้องขอและการตอบสนอง ทำให้การแลกเปลี่ยนข้อมูลเป็นไปอย่างมีระบบ การเข้าใจพื้นฐานของ HTTP เช่น Method, Status Code และ Header เป็นสิ่งสำคัญในการสร้างเว็บแอปพลิเคชันที่เสถียรและปลอดภัย

ภาษา Go หรือ Golang ได้รับความนิยมอย่างรวดเร็วในวงการพัฒนาเว็บเนื่องจากมีประสิทธิภาพสูงและเรียบง่าย Go ถูกออกแบบให้รองรับ **Concurrent Programming** ทำให้สามารถ

จัดการคำร้องขอจากผู้ใช้จำนวนมากพร้อมกันได้อย่างมีประสิทธิภาพ อีกทั้งมีเครื่องมือและไลบรารีที่ช่วยให้นักพัฒนาสามารถสร้างเว็บเซิร์ฟเวอร์และ API ได้ง่าย

ข้อดีอีกประการของ Go คือ **ความเรียบง่ายของภาษาและความเร็วในการคอมไพล์** เมื่อเทียบกับภาษาอื่น ๆ ทำให้เหมาะสำหรับการพัฒนาเว็บแอปพลิเคชันที่ต้องการทั้งประสิทธิภาพและความง่ายในการดูแลรักษา โครงสร้างภาษาที่ชัดเจนยังช่วยลดข้อผิดพลาดและทำให้โค้ดอ่านง่ายสำหรับทีมพัฒนา

ในบทนี้ ผู้อ่านจะได้ทดลองสร้างเว็บเซิร์ฟเวอร์ง่าย ๆ ด้วย Go เพื่อทำความเข้าใจการทำงานของ HTTP Request และ Response รวมถึงการจัดการ Client-Server Interaction การปฏิบัติจริงเหล่านี้จะช่วยให้ผู้อ่านเห็นภาพรวมของการพัฒนาเว็บและสร้างพื้นฐานที่มั่นคงสำหรับบทต่อ ๆ ไป

ท้ายที่สุด บทนี้มุ่งเน้นให้ผู้อ่านเข้าใจ **เหตุผลและคุณค่าของการใช้ Go ในการพัฒนาเว็บ** ผ่านการเรียนรู้แนวคิดพื้นฐานและการทดลองสร้างโปรเจกต์เล็ก ๆ ความรู้ที่ได้จากบทนี้จะเป็นก้าวแรกที่สำคัญในการพัฒนาเว็บแอปพลิเคชันที่ทันสมัยและมีประสิทธิภาพ

Introduction to Web Programming in Go

- การทำความเข้าใจ Web Programming
- Client-Server Architecture
- HTTP Protocol พื้นฐาน
- ข้อดีของ Go ในการทำ Web Development

บทที่ 1: Introduction to Web Programming in Go อย่างละเอียด

1. การทำความเข้าใจ Web Programming

Web Programming คือการพัฒนาโปรแกรมที่ทำงานผ่านเครือข่าย (โดยเฉพาะอินเทอร์เน็ต) และสื่อสารผ่าน **โปรโตคอล HTTP/HTTPS**

สิ่งสำคัญที่ควรรู้:

- **Web Application** ประกอบด้วย
    - **Frontend (Client Side)** ⟶ ส่วนติดต่อผู้ใช้ (UI) ที่รันบนเว็บเบราว์เซอร์ (HTML, CSS, JavaScript)
    - **Backend (Server Side)** ⟶ ส่วนที่ทำงานฝั่งเซิร์ฟเวอร์ เช่น จัดการ Request, Business Logic, Database

ตัวอย่างการทำงาน:

1. ผู้ใช้เปิดเว็บเบราว์เซอร์ ⟶ พิมพ์ URL

2. Browser ส่ง **HTTP Request** ไปที่ Server
3. Server ประมวลผล ⟶ ส่ง **HTTP Response** กลับ
4. Browser แสดงผลลัพธ์ (HTML, JSON, Image ฯลฯ)

 ใน Go เราจะโฟกัสที่ **Backend Web Programming** เช่น การสร้าง REST API, Web Service, Web Server ที่มีประสิทธิภาพสูง

---

2. Client-Server Architecture

สถาปัตยกรรม **Client-Server** เป็นรากฐานของ Web Programming

- **Client**
  - อุปกรณ์หรือซอฟต์แวร์ที่ร้องขอข้อมูล (เช่น Browser, Mobile App, REST Client)
  - ส่ง **Request** ไปยัง Server
- **Server**
  - เครื่องหรือแอปพลิเคชันที่ให้บริการข้อมูล/ฟังก์ชัน
  - รับ Request ⟶ ประมวลผล ⟶ ส่ง Response กลับ

ลักษณะการทำงาน:

Client ---- HTTP Request ----> Server

Client <--- HTTP Response --- Server

ใน Go:

- เราสร้าง **Server** โดยใช้แพ็กเกจ net/http ซึ่งเป็น Standard Library
- Client สามารถเป็น Browser หรือใช้ http.Client ของ Go เพื่อทดสอบ

---

3. HTTP Protocol พื้นฐาน

HTTP (HyperText Transfer Protocol) คือโปรโตคอลมาตรฐานที่ใช้สื่อสารระหว่าง Client ⟷ Server

องค์ประกอบหลักของ HTTP

- **HTTP Request**
  - Method ⟶ เช่น GET, POST, PUT, DELETE
  - URL ⟶ เส้นทางของ resource เช่น /users/1
  - Headers ⟶ Metadata เช่น Content-Type, Authorization
  - Body ⟶ ข้อมูลที่ส่งไป (มักใช้กับ POST, PUT)
- **HTTP Response**
  - Status Code ⟶ เช่น 200 OK, 404 Not Found, 500 Internal Server Error
  - Headers ⟶ Metadata เช่น Content-Type: application/json
  - Body ⟶ ข้อมูลที่ตอบกลับ เช่น JSON, HTML

ตัวอย่างการทำงานจริง

1. Browser ส่ง Request:
2. GET /users/1 HTTP/1.1
3. Host: api.example.com
4. Server ตอบ Response:
5. HTTP/1.1 200 OK
6. Content-Type: application/json
7.
8. {"id":1,"name":"Alice"}

ใน Go:

```
package main

import (
	"fmt"
	"net/http"
)

func handler(w http.ResponseWriter, r *http.Request) {
	fmt.Fprintf(w, "Hello, Go Web Programming!")
}

func main() {
	http.HandleFunc("/", handler)
	http.ListenAndServe(":8080", nil)
}
```

- เมื่อรันแล้วเปิด http://localhost:8080 ⟶ จะแสดง "Hello, Go Web Programming!"

---

4. ข้อดีของ Go ในการทำ Web Development

Go (Golang) ถูกออกแบบมาให้เหมาะกับ **ระบบที่ต้องการประสิทธิภาพสูง** และ **รองรับ concurrent connections จำนวนมาก** ทำให้โดดเด่นในการพัฒนา Web Server และ API

☐ **จุดเด่นสำคัญ**

1. **ประสิทธิภาพสูง (High Performance)**
   - Go เป็นภาษา **Compiled** ⟶ แปลงเป็น Binary ⟶ ทำงานเร็วใกล้เคียง C
   - การประมวลผล HTTP Requests ทำได้เร็วกว่า Python/Node.js

2. **Concurrency แบบเบา (Goroutines)**
    - รองรับ **การทำงานพร้อมกัน (concurrency)** ด้วย goroutine และ channel
    - เหมาะสำหรับระบบที่มีการเชื่อมต่อพร้อมกัน (เช่น API ที่มีผู้ใช้เยอะ)
3. **Standard Library ครบครัน**
    - มี net/http ในตัว ⟶ ไม่ต้องติดตั้ง framework เสริม
    - สามารถสร้าง HTTP Server ได้ง่ายมากในไม่กี่บรรทัด
4. **Cross-Platform & Deployment ง่าย**
    - Compile ได้ Binary เดียว ⟶ รันบน Windows, Linux, macOS
    - ง่ายต่อการ deploy ใน Docker/Kubernetes
5. **Ecosystem แข็งแรง**
    - มี Framework เช่น **Gin, Echo, Fiber**
    - ORM เช่น **GORM, sqlx**
    - รองรับการเชื่อมต่อ Database, Redis, Kafka, gRPC

☐ **สรุป**

ในบทนี้เราได้เรียนรู้:

- ความหมายของ Web Programming ⟶ การสร้างระบบ Client-Server ผ่าน HTTP
- สถาปัตยกรรม Client-Server ⟶ Client ส่ง Request ⟶ Server ตอบ Response
- HTTP Protocol พื้นฐาน ⟶ Method, Headers, Body, Status Code
- ทำไม Go เหมาะกับ Web Development ⟶ เร็ว, Concurrent, Standard Library ดี, Deployment ง่าย

Introduction to Web Programming in Go (Deep Dive)

---

1. การทำความเข้าใจ Web Programming (เชิงลึก)

Web Programming ไม่ใช่แค่ “ทำเว็บให้แสดงผล” แต่คือ **การสร้างระบบที่มี Client และ Server สื่อสารกันผ่าน Network Protocol (หลัก ๆ คือ HTTP/HTTPS)**

ประเภทของ Web Application

1. **Static Website** ⟶ เช่น Landing Page
    - HTML/CSS/JS คงที่ ไม่ต้องประมวลผลฝั่งเซิร์ฟเวอร์
    - ตัวอย่าง: Personal Blog
2. **Dynamic Web Application**
    - มี Backend คอยประมวลผล

- ดึงข้อมูลจาก Database ⟶ Render ⟶ ตอบกลับ
- ตัวอย่าง: Facebook, Twitter

3. **API-Driven Web Application**
   - Server ให้บริการ **API (Application Programming Interface)** เช่น REST, GraphQL, gRPC
   - Client (เช่น React, Vue, Mobile App) เรียก API เพื่อดึงข้อมูล

ใน Go เรามักพัฒนาแบบที่ 2–3 โดยเฉพาะ **REST API** และ **Microservices**

---

2. Client-Server Architecture (เชิงลึก)

Client-Server เป็น **สถาปัตยกรรม 2 ฝั่ง**

- **Client** ⟶ ส่ง request (เช่น เบราว์เซอร์, แอปมือถือ)
- **Server** ⟶ ประมวลผล request แล้วตอบ response

ตัวอย่าง Flow

1. User เข้า https://api.example.com/users/123
2. Browser (Client) ⟶ ส่ง **GET Request**
3. Server ⟶ Query Database ⟶ คืน JSON
4. Client ⟶ แสดงผลข้อมูล

ข้อดีของ Client-Server Model

- **Separation of Concern** ⟶ Client สนใจ UI, Server สนใจ Logic
- **Scalability** ⟶ สามารถแยก Client กับ Server บนเครื่องคนละเครื่องได้
- **Maintainability** ⟶ อัปเดตฝั่งหนึ่งไม่กระทบอีกฝั่ง

---

3. HTTP Protocol พื้นฐาน (เชิงลึก)

HTTP เป็น **Stateless Protocol** ⟶ หมายความว่าแต่ละ Request ไม่จำเป็นต้องจดจำข้อมูลของ Request ก่อนหน้า

ส่วนประกอบสำคัญ

HTTP Methods

- GET ⟶ ดึงข้อมูล
- POST ⟶ สร้างข้อมูลใหม่
- PUT ⟶ อัปเดตข้อมูลทั้งหมด
- PATCH ⟶ อัปเดตบางส่วน
- DELETE ⟶ ลบข้อมูล

Status Codes

- 200 OK ⟶ สำเร็จ
- 201 Created ⟶ สร้าง resource สำเร็จ
- 400 Bad Request ⟶ ข้อมูลไม่ถูกต้อง
- 401 Unauthorized ⟶ ไม่มีสิทธิ์
- 404 Not Found ⟶ ไม่เจอ resource
- 500 Internal Server Error ⟶ เซิร์ฟเวอร์พัง

HTTP Headers

- Content-Type: application/json ⟶ บอกว่า response เป็น JSON
- Authorization: Bearer <token> ⟶ ใช้ JWT/OAuth
- Cache-Control ⟶ ควบคุม caching

☐ **ตัวอย่าง Request/Response จริง**

Request (POST):

POST /users HTTP/1.1

Host: api.example.com

Content-Type: application/json

```
{
  "name": "Alice",
  "email": "alice@example.com"
}
```

Response:

HTTP/1.1 201 Created

Content-Type: application/json

```
{
  "id": 1,
  "name": "Alice",
  "email": "alice@example.com"
}
```

---

4. ข้อดีของ Go ในการทำ Web Development (เชิงลึก)

Go ถูกเลือกใช้ใน Production โดยบริษัทใหญ่ ๆ เช่น **Uber, Dropbox, Google, Cloudflare**

□ จุดแข็งของ Go

1. **ความเร็ว & ประสิทธิภาพ**
   - Go เป็น **compiled language** ⟶ แปลงเป็น machine code ⟶ เร็วกว่า interpreted languages (เช่น Python, PHP)
   - เหมาะสำหรับระบบที่ต้องรองรับผู้ใช้จำนวนมาก
2. **Concurrency (Goroutines + Channels)**
   - Go รองรับการทำงานพร้อมกันโดยไม่ต้องใช้ threads หนัก ๆ
   - ตัวอย่าง: API Server ที่รองรับ request พร้อมกัน 100,000 connections ได้สบาย
3. **Standard Library ที่ครบเครื่อง**
   - มี net/http ⟶ สร้าง Web Server ได้ทันที
   - ไม่ต้องพึ่งพา Framework ภายนอก
4. **Ecosystem และ Frameworks**
   - Framework ยอดนิยม: **Gin, Echo, Fiber**
   - ORM: **GORM, sqlx**
   - Tools: Swagger (API Doc), Wire (Dependency Injection), Cobra (CLI)
5. **Deployment ง่าย**
   - Compile ⟶ ได้ Binary เดียว
   - Deploy ขึ้น **Docker/Kubernetes** ได้ง่าย
   - เหมาะกับ Cloud-Native Development

□ ตัวอย่างเชิงลึก: Web Server ด้วย Go

```
package main

import (
	"encoding/json"
	"fmt"
	"net/http"
)

type User struct {
	ID   int    `json:"id"`
	Name string `json:"name"`
}
```

```
func helloHandler(w http.ResponseWriter, r *http.Request) {
        fmt.Fprintln(w, "Welcome to Go Web Programming!")
}

func userHandler(w http.ResponseWriter, r *http.Request) {
        user := User{ID: 1, Name: "Alice"}

        // ตั้งค่า Header ให้ response เป็น JSON
        w.Header().Set("Content-Type", "application/json")
        json.NewEncoder(w).Encode(user)
}

func main() {
        http.HandleFunc("/", helloHandler)
        http.HandleFunc("/user", userHandler)

        fmt.Println("Server started at http://localhost:8080")
        http.ListenAndServe(":8080", nil)
}
```

วิธีทดสอบ

1. รัน go run main.go
2. เข้า Browser ⟶ http://localhost:8080/ ⟶ จะเห็นข้อความ:
3. Welcome to Go Web Programming!
4. เข้า http://localhost:8080/user ⟶ จะได้ JSON:
5. {
6. "id": 1,
7. "name": "Alice"
8. }

☐ **สรุปเชิงลึก**

1. Web Programming = การสื่อสารระหว่าง Client ⟷ Server ผ่าน HTTP
2. Client-Server Model = Separation of UI/Logic ⟶ Scale ได้ง่าย
3. HTTP Protocol = Stateless, มี Methods + Headers + Status Codes

4. Go เด่นมากใน Web Dev เพราะ: **เร็ว, Concurrency ดี, Standard Library ครบ, Deploy ง่าย, Ecosystem แรง**
5. เราสามารถสร้าง HTTP Server ได้เพียงไม่กี่บรรทัดด้วย net/http

---

การทำความเข้าใจ Web Programming แบบละเอียดที่สุด

 การทำความเข้าใจ Web Programming (Understanding Web Programming)

1. ความหมายของ Web Programming

Web Programming คือกระบวนการสร้างโปรแกรมหรือระบบที่ทำงานผ่านเครือข่าย (โดยเฉพาะอินเทอร์เน็ต) โดยสื่อสารกันผ่าน **โปรโตคอลมาตรฐาน เช่น HTTP/HTTPS**
ในเชิงโครงสร้าง ระบบ Web จะมี 2 ส่วนใหญ่ ๆ:

1. **Client (ผู้ใช้งาน)** ⟶ เว็บเบราว์เซอร์, แอปมือถือ, REST Client (Postman, curl)
2. **Server (ผู้ให้บริการ)** ⟶ เครื่องหรือโปรแกรมที่รอรับคำร้อง (Request) แล้วส่งคำตอบ (Response)

 Web Programming คือการสร้าง **ตรรกะฝั่งเซิร์ฟเวอร์** ที่รองรับการสื่อสารเหล่านี้ เช่น REST API, Web Service, หรือแม้แต่การให้บริการ HTML ธรรมดา

---

2. การพัฒนาเว็บแบ่งได้เป็น 3 ประเภทหลัก

1. **Static Web (เว็บคงที่)**
    - เสิร์ฟไฟล์ HTML/CSS/JS ตรง ๆ
    - ไม่มีการประมวลผลฝั่ง Server
    - เช่น เว็บ Portfolio, Blog
2. **Dynamic Web (เว็บไดนามิก)**
    - มีการประมวลผลฝั่ง Server
    - ดึงข้อมูลจาก Database และตอบกลับแบบ Realtime
    - เช่น Facebook, E-commerce
3. **API-Centric Web Application**
    - Server ทำหน้าที่เป็น **API Provider** เช่น REST, GraphQL, gRPC
    - Client ไม่จำเป็นต้องเป็น Browser เสมอไป ⟶ อาจเป็น Mobile App, IoT Device
    - เช่น Twitter API, GitHub API

Go มักถูกใช้กับ **Dynamic Web** และ **API-Centric** เนื่องจาก Go มีประสิทธิภาพสูงและรองรับการทำงานพร้อมกันได้ดี

---

3. Workflow ของ Web Programming

ตัวอย่างการทำงานแบบ Request ⟶ Response

1. ผู้ใช้พิมพ์ http://example.com/users/1 ใน Browser
2. Browser ส่ง **HTTP Request** ไปยัง Server
3. Server รับ Request ⟶ ประมวลผล (เช่น ค้นหาข้อมูลใน DB)
4. Server ส่ง **HTTP Response** กลับมาในรูปแบบ HTML หรือ JSON
5. Browser แสดงผลให้ผู้ใช้เห็น

แผนภาพ:

[Client: Browser] -- Request --> [Server: Go Application] -- Database Query -->

[Client: Browser] <-- Response -- [Server: Go Application]

---

4. ตัวอย่างพื้นฐาน: Hello World Web Server ใน Go

```
package main

import (
	"fmt"
	"net/http"
)

// ฟังก์ชัน Handler สำหรับ URL "/"
func helloHandler(w http.ResponseWriter, r *http.Request) {
	fmt.Fprintln(w, "Hello, Go Web Programming!")
}

func main() {
	// กำหนดเส้นทาง (Route)
	http.HandleFunc("/", helloHandler)

	// สตาร์ท Web Server บนพอร์ต 8080
	fmt.Println("Server started at http://localhost:8080")
	http.ListenAndServe(":8080", nil)
}
```

การทำงาน

1. Run โปรแกรม ⟶ Go จะสร้าง Web Server

2. เปิด Browser ไปที่ http://localhost:8080/
3. คุณจะเห็นข้อความ
4. Hello, Go Web Programming!

นี่คือ **จุดเริ่มต้นของ Web Programming ใน Go** ⟶ จาก Handler ง่าย ๆ นี้ เราสามารถขยายไปสู่ REST API, การเชื่อมต่อฐานข้อมูล, Authentication, Middleware และอื่น ๆ

---

5. สรุปความเข้าใจ

- **Web Programming** = การเขียนโปรแกรมที่ทำงานบนเครือข่าย โดยใช้ **Client-Server Model**
- Client ⟶ ส่ง Request, Server ⟶ ตอบ Response
- มีทั้ง Static, Dynamic, และ API-Centric Application
- Go ทำให้เริ่มต้นได้ง่ายมาก แค่ไม่กี่บรรทัดก็สร้าง Web Server ได้แล้ว

Client-Server Architecture (สถาปัตยกรรมลูกค้า-เซิร์ฟเวอร์)

---

1. ความหมายพื้นฐาน

Client-Server Architecture คือสถาปัตยกรรมที่แยกการทำงานของระบบออกเป็น **สองฝั่งหลัก**:

1. **Client (ลูกค้า)**
    - ส่งคำร้องขอ (Request) ไปยัง Server
    - แสดงผลลัพธ์ให้ผู้ใช้งาน
    - ตัวอย่าง:
        - Web Browser (Chrome, Firefox)
        - Mobile App (iOS, Android)
        - REST Client (Postman, curl)
2. **Server (เซิร์ฟเวอร์)**
    - รับคำร้องขอจาก Client
    - ประมวลผลข้อมูล, ติดต่อ Database, ทำ Business Logic
    - ส่งคำตอบกลับ (Response)

---

2. Flow การทำงานของ Client ⟷ Server

ขั้นตอนพื้นฐาน

1. Client ส่ง **HTTP Request** เช่น GET /users/1
2. Server รับ Request ⟶ ประมวลผล (เช่น Query Database)
3. Server ส่ง **HTTP Response** กลับ Client

4. Client แสดงผลลัพธ์ (HTML/JSON/Images)

แผนภาพ

Client (Browser, Mobile, API)

|

| HTTP Request

v

Server (Go Web Application)

|

| DB Query / Business Logic

v

Client <--- HTTP Response ---

---

3. ข้อดีของสถาปัตยกรรม Client-Server

1. **Separation of Concerns**
   - Client ⟶ UI/UX
   - Server ⟶ Business Logic/Data Management

     ⟶ ทำให้พัฒนาและบำรุงรักษาง่ายขึ้น
2. **Scalability**
   - สามารถเพิ่ม Server หรือแยก Database Server เพื่อรองรับผู้ใช้จำนวนมาก
   - ทำให้ระบบรองรับ Traffic สูงได้
3. **Security**
   - เซิร์ฟเวอร์ควบคุมการเข้าถึงข้อมูล
   - สามารถทำ Authentication/Authorization
4. **Centralized Management**
   - Update Logic หรือ Data บน Server ครั้งเดียว ⟶ Client ทุกตัวใช้ได้ทันที

---

4. ข้อจำกัด / ความท้าทาย

1. **Single Point of Failure**
   - ถ้า Server ล่ม ⟶ Client ทุกตัวใช้งานไม่ได้
   - ต้องมี Load Balancer / Failover
2. **Network Dependency**
   - Client ต้องเชื่อมต่อ Server ⟶ Offline ไม่ได้
3. **Latency**

- ต้องรอ Request-Response ⟶ ระบบต้อง Optimize Response Time

---

5. ตัวอย่าง Client-Server ด้วย Go

ตัวอย่าง: สร้าง REST API ง่าย ๆ

```
package main

import (
	"encoding/json"
	"fmt"
	"net/http"
)

type User struct {
	ID   int    `json:"id"`
	Name string `json:"name"`
}

// Handler สำหรับดึงข้อมูลผู้ใช้
func userHandler(w http.ResponseWriter, r *http.Request) {
	user := User{ID: 1, Name: "Alice"}
	w.Header().Set("Content-Type", "application/json")
	json.NewEncoder(w).Encode(user)
}

func main() {
	// กำหนด Route
	http.HandleFunc("/user", userHandler)

	fmt.Println("Server running at http://localhost:8080")
	http.ListenAndServe(":8080", nil)
}
```

การทดสอบ

1. Run Server: go run main.go
2. Client ส่ง Request: GET http://localhost:8080/user

3. Response:

```
{
  "id": 1,
  "name": "Alice"
}
```

ตัวอย่างนี้แสดงให้เห็นชัดเจนว่า **Client (Browser/Postman)** ส่ง Request ⟶ **Server (Go Web App)** ประมวลผล ⟶ ส่ง Response กลับ

---

6. ขยายความ: Multi-Tier Architecture

ในระบบใหญ่ มักมี **หลายชั้น (Layer)**:

1. **Presentation Layer** ⟶ Client UI
2. **Application Layer** ⟶ Business Logic (Go Web Server)
3. **Data Layer** ⟶ Database / Cache / External Services

ข้อดี: **แยกความรับผิดชอบชัดเจน, Maintainable, Scale ง่าย**

---

7. สรุปเชิงลึก

- **Client-Server Architecture** = โครงสร้างพื้นฐานของ Web Programming
- Client ⟶ ส่ง Request, Server ⟶ ประมวลผล & ตอบ Response
- ข้อดี: Separation, Scalability, Security, Centralized Management
- ข้อจำกัด: Single Point of Failure, Network Dependency, Latency
- ใน Go ⟶ สามารถสร้าง Server รองรับ Client ได้ง่ายด้วย net/http หรือ Framework เช่น Gin/Echo

---

HTTP Protocol พื้นฐาน (Basic HTTP Protocol)

HTTP Protocol พื้นฐาน (Basic HTTP Protocol)

HTTP (HyperText Transfer Protocol) เป็น **โปรโตคอลมาตรฐานสำหรับการสื่อสารระหว่าง Client ↔ Server** บนเว็บ

---

1. ความสำคัญของ HTTP

- HTTP ทำให้ Client (Browser, Mobile App) และ Server (Go Web Application) สื่อสารกันได้
- เป็น **Stateless Protocol** ⟶ แต่ละ Request ถูกประมวลผลแยกจาก Request อื่น ๆ
- ใช้ได้ทั้ง **Text (HTML, JSON)** และ **Binary (Image, File, Video)**

2. ส่วนประกอบหลักของ HTTP

2.1 HTTP Request

Client ส่งคำร้องไปยัง Server

โครงสร้าง

METHOD URL HTTP_VERSION

Headers

Body (Optional)

องค์ประกอบสำคัญ

- **Method (HTTP Verb)**
  - GET ⟶ ดึงข้อมูล
  - POST ⟶ ส่งข้อมูลใหม่
  - PUT ⟶ แก้ไขข้อมูลทั้งหมด
  - PATCH ⟶ แก้ไขข้อมูลบางส่วน
  - DELETE ⟶ ลบข้อมูล
- **URL (Uniform Resource Locator)**
  - ระบุ resource ที่ต้องการ เช่น /users/1
- **Headers**
  - Metadata ของ Request เช่น
    - Content-Type: application/json
    - Authorization: Bearer <token>
- **Body** (Optional)
  - ข้อมูลที่ส่งไป เช่น JSON, Form Data

ตัวอย่าง HTTP Request

POST /users HTTP/1.1

Host: api.example.com

Content-Type: application/json

Authorization: Bearer abc123

```
{
  "name": "Alice",
  "email": "alice@example.com"
}
```

2.2 HTTP Response

Server ส่งข้อมูลตอบกลับ Client

โครงสร้าง

HTTP_VERSION STATUS_CODE STATUS_TEXT

Headers

Body (Optional)

องค์ประกอบสำคัญ

- **Status Code** ⟶ บอกผลลัพธ์การประมวลผล
  - 200 OK ⟶ สำเร็จ
  - 201 Created ⟶ สร้าง resource สำเร็จ
  - 400 Bad Request ⟶ ข้อมูลผิดพลาด
  - 401 Unauthorized ⟶ ต้อง login
  - 404 Not Found ⟶ ไม่เจอ resource
  - 500 Internal Server Error ⟶ Server Error
- **Headers** ⟶ Metadata ของ Response เช่น
  - Content-Type: application/json
  - Cache-Control: no-cache
- **Body** ⟶ ข้อมูลจริง เช่น JSON, HTML

ตัวอย่าง HTTP Response

HTTP/1.1 201 Created

Content-Type: application/json

```
{
  "id": 1,
  "name": "Alice",
  "email": "alice@example.com"
}
```

3. HTTP Methods (เชิงลึก)

Method	การใช้งาน	ตัวอย่าง
GET	ดึงข้อมูล	/users

Method	การใช้งาน	ตัวอย่าง
POST	สร้าง resource ใหม่	/users
PUT	แก้ไข resource ทั้งหมด	/users/1
PATCH	แก้ไข resource บางส่วน	/users/1
DELETE	ลบ resource	/users/1

Tip: GET ไม่ควรมีผลลัพธ์เปลี่ยนแปลง server ⟶ idempotent

POST, PUT, PATCH มีผลเปลี่ยน server ⟶ ต้องระวัง side effect

---

4. ตัวอย่าง HTTP Server ใน Go

```
package main

import (
	"encoding/json"
	"fmt"
	"net/http"
)

type User struct {
	ID   int    `json:"id"`
	Name string `json:"name"`
}

// GET /user
func getUser(w http.ResponseWriter, r *http.Request) {
	if r.Method != http.MethodGet {
		http.Error(w, "Method not allowed", http.StatusMethodNotAllowed)
		return
	}

	user := User{ID: 1, Name: "Alice"}
	w.Header().Set("Content-Type", "application/json")
	json.NewEncoder(w).Encode(user)
}
```

```
// POST /user
func createUser(w http.ResponseWriter, r *http.Request) {
	if r.Method != http.MethodPost {
		http.Error(w, "Method not allowed", http.StatusMethodNotAllowed)
		return
	}

	var user User
	err := json.NewDecoder(r.Body).Decode(&user)
	if err != nil {
		http.Error(w, "Bad request", http.StatusBadRequest)
		return
	}

	user.ID = 2 // จำลองการสร้าง ID
	w.Header().Set("Content-Type", "application/json")
	w.WriteHeader(http.StatusCreated)
	json.NewEncoder(w).Encode(user)
}

func main() {
	http.HandleFunc("/user", func(w http.ResponseWriter, r *http.Request) {
		switch r.Method {
		case http.MethodGet:
			getUser(w, r)
		case http.MethodPost:
			createUser(w, r)
		default:
			http.Error(w, "Method not allowed", http.StatusMethodNotAllowed)
		}
	})

	fmt.Println("Server running at http://localhost:8080")
```

```
    http.ListenAndServe(":8080", nil)
}
```

การทดสอบ

1. GET http://localhost:8080/user ⟶ รับ JSON ของผู้ใช้
2. POST http://localhost:8080/user ⟶ ส่ง JSON { "name": "Bob" } ⟶ Server คืน 201 Created พร้อมข้อมูลผู้ใช้ใหม่

---

5. Headers & Content-Type

Content-Type ⟶ บอก Server หรือ Client ว่า Body เป็นประเภทอะไร

- application/json ⟶ JSON
- text/html ⟶ HTML
- application/x-www-form-urlencoded ⟶ Form
- multipart/form-data ⟶ File Upload

Authorization Header ⟶ ใช้สำหรับ Authentication/Token

Authorization: Bearer <token>

---

6. สรุปเชิงลึก

- HTTP = โปรโตคอลหลักในการสื่อสาร Client ↔ Server
- Request = Method + URL + Headers + Body
- Response = Status Code + Headers + Body
- Method แต่ละตัวมีจุดประสงค์เฉพาะ (GET, POST, PUT, PATCH, DELETE)
- ใน Go เราสามารถสร้าง HTTP Server รองรับ Method ต่าง ๆ ได้ง่าย ๆ ด้วย net/http

---

ข้อดีของ Go ในการทำ Web Development

☐ **ข้อดีของ Go ในการทำ Web Development (Advantages of Go for Web Development)**

Go (Golang) เป็นภาษาที่ Google พัฒนาขึ้น โดยออกแบบมาสำหรับ **ระบบที่ต้องการประสิทธิภาพสูง, Concurrent, และ Deployment ง่าย** ซึ่งทำให้เหมาะมากสำหรับ Web Development

---

1. ประสิทธิภาพสูง (High Performance)

- Go เป็น **Compiled Language** ⟶ แปลงเป็น **Binary Machine Code**
- ทำงานเร็วกว่า **Interpreted Languages** เช่น Python, PHP
- ไม่ต้องใช้ Virtual Machine (เช่น JVM) ⟶ ลด Overhead

ตัวอย่าง:

- Server ที่รับ HTTP Request ด้วย Go สามารถประมวลผลได้ **หลายแสน Request ต่อวินาที** ขึ้นอยู่กับ Hardware

```
// ตัวอย่าง Web Server ง่าย ๆ
package main

import (
	"fmt"
	"net/http"
)

func handler(w http.ResponseWriter, r *http.Request) {
	fmt.Fprintln(w, "Fast response!")
}

func main() {
	http.HandleFunc("/", handler)
	http.ListenAndServe(":8080", nil)
}
```

- Server ตัวนี้สามารถรองรับ **การเชื่อมต่อพร้อมกันจำนวนมาก** โดยไม่ต้องเขียนโค้ดซับซ้อน

---

2. Concurrency & Goroutines

- Go รองรับ **Concurrent Programming** ในตัวด้วย **Goroutines**
- **Goroutine** คือ Thread เบา ๆ ที่ใช้หน่วยความจำน้อย ⟶ สามารถสร้างได้เป็นล้าน ๆ ตัว

ตัวอย่าง: Concurrent Requests

```
package main

import (
	"fmt"
	"net/http"
	"time"
)

func slowHandler(w http.ResponseWriter, r *http.Request) {
```

```
        go func() {
                time.Sleep(2 * time.Second) // Simulate heavy work
                fmt.Println("Task done")
        }()
        fmt.Fprintln(w, "Request received!")
}

func main() {
        http.HandleFunc("/", slowHandler)
        http.ListenAndServe(":8080", nil)
}
```

- Client จะได้รับ Response ทันทีโดยไม่ต้องรอ Goroutine ทำงานเสร็จ
- เหมาะสำหรับ **API Server** ที่ต้องรองรับผู้ใช้จำนวนมาก

---

3. Standard Library ครบถ้วน

- Go มี **Standard Library** สำหรับ Web Development ครบ
- net/http ⟶ สร้าง HTTP Server / Client
- html/template ⟶ Render HTML แบบปลอดภัย (ป้องกัน XSS)
- encoding/json ⟶ แปลง JSON ↔ Struct
- database/sql ⟶ เชื่อมต่อ SQL Database
- context ⟶ จัดการ Timeout, Cancellation

ข้อดี: ไม่ต้องพึ่ง External Library ขั้นพื้นฐาน

---

4. Deployment ง่าย & Cross-Platform

- Compile ⟶ ได้ **Binary เดียว** ไม่ต้องติดตั้ง Interpreter หรือ Runtime
- รองรับทุก OS: Linux, macOS, Windows
- ง่ายต่อ **Docker / Kubernetes Deployment**

ตัวอย่าง: Compile Web Server

```
go build -o myserver main.go
./myserver   # Run Binary
```

- Deploy Binary ขึ้น Cloud หรือ Server ใดก็ได้

---

5. Ecosystem และ Community

- มี **Framework**: Gin, Echo, Fiber ⟶ ช่วยจัดการ Routing, Middleware, JSON Response
- มี **ORM/Database**: GORM, sqlx ⟶ จัดการ Database ง่าย
- มี **Tools**: Swagger (API Docs), Cobra (CLI), Wire (Dependency Injection)

Go Web Dev ecosystem เติบโตเร็ว ⟶ หา Package / Solutions ง่าย

---

6. Safety & Maintainability

- **Static Typing** ⟶ ลด Bug ก่อน Runtime
- **Garbage Collection** ⟶ จัดการ Memory อัตโนมัติ
- Code ของ Go มี **Style ชัดเจน** ⟶ Maintainable, Readable

---

7. ตัวอย่าง Web Server ที่ครบ Feature พื้นฐาน

```
package main

import (
	"encoding/json"
	"fmt"
	"net/http"
	"time"
)

type User struct {
	ID   int    `json:"id"`
	Name string `json:"name"`
}

func main() {
	http.HandleFunc("/users", func(w http.ResponseWriter, r *http.Request) {
		switch r.Method {
		case http.MethodGet:
			users := []User{{ID: 1, Name: "Alice"}, {ID: 2, Name: "Bob"}}
			w.Header().Set("Content-Type", "application/json")
			json.NewEncoder(w).Encode(users)

		case http.MethodPost:
```

```
                var user User
                json.NewDecoder(r.Body).Decode(&user)
                user.ID = int(time.Now().Unix()) // Dummy ID
                w.Header().Set("Content-Type", "application/json")
                w.WriteHeader(http.StatusCreated)
                json.NewEncoder(w).Encode(user)

        default:
                http.Error(w, "Method not allowed", http.StatusMethodNotAllowed)
        }
    })

    fmt.Println("Server running at http://localhost:8080")
    http.ListenAndServe(":8080", nil)
}
```

คุณสมบัติ:

- รองรับ GET/POST
- ส่ง/รับ JSON
- ใช้ Standard Library เพียงอย่างเดียว
- พร้อมต่อยอดเป็น REST API ขนาดใหญ่

☐ **สรุปข้อดีของ Go สำหรับ Web Development**

ข้อดี	รายละเอียด
Performance	Compiled ⟶ ทำงานเร็ว, รองรับผู้ใช้จำนวนมาก
Concurrency	Goroutines + Channels ⟶ รองรับการเชื่อมต่อพร้อมกัน
Standard Library	ครบถ้วน ⟶ HTTP, JSON, Template, Database
Deployment	Binary เดียว ⟶ Cross-Platform, Docker-Friendly
Ecosystem	Framework, ORM, Tools พร้อมใช้งาน
Safety & Maintainability	Static Typing, Garbage Collection, Code Style

สรุป: Go เหมาะมากกับ **Web API / Microservices / High Performance Web Server**

การติดตั้ง Go สำหรับ Web Programming

การติดตั้ง Go for Web Programming

1. ตรวจสอบความต้องการระบบ (System Requirements)

- **RAM:** ≥ 2 GB
- **Disk:** ≥ 100 MB free space
- **OS:** Windows 10+, macOS 10.12+, Linux Kernel 4+
- **Network:** ต้องเชื่อมต่อ Internet สำหรับดาวน์โหลด

---

2. ดาวน์โหลด Go

1. เข้าเว็บไซต์ทางการ: https://go.dev/dl
2. เลือก **Version ล่าสุด** (เช่น Go 1.22.x)
3. เลือกไฟล์ตาม OS:
    - **Windows:** go1.22.x.windows-amd64.msi
    - **macOS:** go1.22.x.darwin-amd64.pkg
    - **Linux:** go1.22.x.linux-amd64.tar.gz

⚠ อย่าลืมเลือก Architecture ให้ตรงกับเครื่อง (64-bit vs 32-bit)

---

3. ติดตั้ง Go

3.1 Windows

1. รันไฟล์ .msi ⟶ กด **Next** ⟶ **Install**
2. เลือก **Destination Folder** (เช่น C:\Go)
3. ติ๊กเลือก **Add Go to PATH**
4. เสร็จแล้ว ⟶ กด Finish

3.2 macOS

1. รันไฟล์ .pkg ⟶ กด **Continue** ⟶ **Install**
2. ตรวจสอบว่า /usr/local/go/bin ถูกเพิ่มใน PATH
3. ปิดและเปิด Terminal ใหม่

3.3 Linux

1. เปิด Terminal
2. ดาวน์โหลดไฟล์ tar.gz เช่น:

```
wget https://go.dev/dl/go1.22.linux-amd64.tar.gz
```

3. แตกไฟล์ไปที่ /usr/local

```
sudo tar -C /usr/local -xzf go1.22.linux-amd64.tar.gz
```

4. เพิ่ม Go ลง PATH

```
echo "export PATH=$PATH:/usr/local/go/bin" >> ~/.bashrc
source ~/.bashrc
```

---

4. ตรวจสอบการติดตั้ง

เปิด Terminal / Command Prompt แล้วพิมพ์

```
go version
```

Expected Output

```
go version go1.22 linux/amd64
```

---

5. ตั้งค่า Workspace สำหรับ Web Programming

5.1 สร้าง Project Folder

```
mkdir ~/go-web-project
cd ~/go-web-project
```

5.2 สร้าง Module ของ Go

```
go mod init github.com/username/go-web-project
```

- สร้างไฟล์ go.mod ⟶ จัดการ Dependencies
- ตัวอย่าง go.mod:

```
module github.com/username/go-web-project

go 1.22
```

---

6. สร้างตัวอย่าง Web Server แบบง่าย

6.1 สร้างไฟล์ main.go

```
package main

import (
        "fmt"
        "net/http"
)

func helloHandler(w http.ResponseWriter, r *http.Request) {
        fmt.Fprintln(w, "Hello, Go Web Programming!")
```