Echo เป็น web framework สำหรับภาษา Go พัฒนาขึ้นโดย LabStack ที่เครมว่ามีประสิทธิภาพสูง (High performance), ขยายเพิ่มเติมได้ (Extensible), มีความเรียบง่าย (Minimalist)

- มี built-in middleware หลากหลายให้ใช้ หรือสร้างใหม่ได้ และ middleware สามารถตั้งค่าในระดับ root, group หรือ route ได้
- มี data binding สำหรับ HTTP request ที่รองรับ JSON, XML หรือ form-data
- API สามารถส่ง HTTP response ได้หลากหลาย ทั้ง JSON, XML, HTML, file, attachment, inline, stream หรือ blob.

อ้างอิง : https://phayao.medium.com/มาสร้าง-restful-web-service-ด้วย-echo-go-web-framework-140dcf74d30e

Start echo server

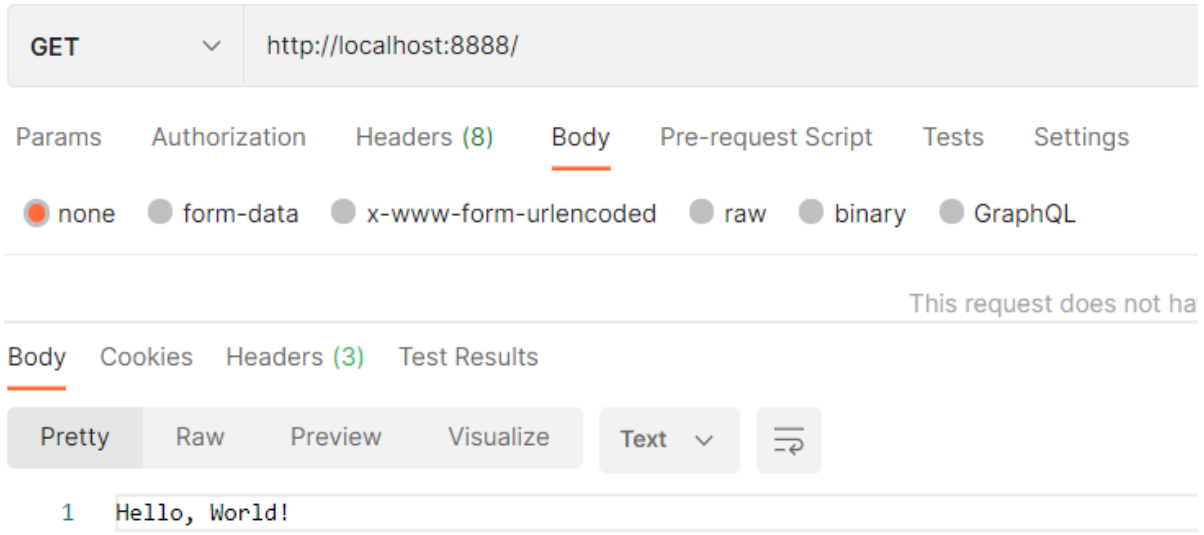
```
C:\Users\acer\Desktop\basic_echo\basic_hello>go get github.com/labstack/echo/v4
```

```go
package main

import (
    "github.com/labstack/echo/v4"
)

func main() {
    e := echo.New()
    e.Logger.Fatal(e.Start(":8888"))
}
```

```
C:\Users\acer\Desktop\basic_echo>go run .


   ____    __
  / __/___/ /  ___
 / _// __/ _ \/ _ \
/___/\__/_//_/\___/ v4.4.0
High performance, minimalist Go web framework
https://echo.labstack.com
____________________________________O/_______
                                    O\
⇨ http server started on [::]:8888
```

Http Method (GET, POST, PUT, DELETE)

```go
e := echo.New()
e.GET("/", func(c echo.Context) error {
    return c.String(http.StatusOK, "Hello, World!")
})
e.Logger.Fatal(e.Start(":8888"))
```



Binding request data

```go
type Employee struct {
    ID        string `json:"id"`
    Firstname string `json:"firstname"`
    Lastname  string `json:"lastname"`
    Salary    int    `json:"salary"`
}
```

query - source is request query parameters.
param - source is route path parameter.
form - source is form. Values are taken from query and request body. Uses Go standard library form parsing.
json - source is request body. Uses Go json package for unmarshalling.
xml - source is request body. Uses Go xml package for unmarshalling.

อ้างอิง: https://echo.labstack.com/guide/binding/

Context

echo.Context represents the context of the current HTTP request. It holds request and response reference, path, path parameters, data, registered handler and APIs to read request and write response.

```go
e.GET("/:name", func(c echo.Context) error {
    resp := fmt.Sprintf("Hello %s", c.Param("name"))
    return c.String(http.StatusOK, resp)
})
e.POST("/", func(c echo.Context) error {
    var request struct {
        Name string `json:"name"`
    }
    if err := c.Bind(&request); err != nil {
        return err
    }
    resp := fmt.Sprintf("Hello %s", request.Name)
    return c.String(http.StatusOK, resp)
})
```

อ้างอิง: https://echo.labstack.com/guide/context/

สร้างกลุ่มของ endpoint

```
routes := []route{
    {
        Group:          "employee",
        Path:           "/byId",
        HttpMethod:     http.MethodPost,
        HandlerFunc:    emp.GetEmployeeById,
        MiddlewareFunc: nil,
    },
    {
        Group:          "employee",
        Path:           "/add",
        HttpMethod:     http.MethodPost,
        HandlerFunc:    emp.AddEmployee,
        MiddlewareFunc: nil,
    },
```

```
e.Group(rt.Group).Add(rt.HttpMethod, rt.Path, rt.HandlerFunc, mw...)
```

middleware

Middleware is a function chained in the HTTP request-response cycle with access to Echo#Context which it uses to perform a specific action, for example, logging every request or limiting the number of requests.

Handler is processed in the end after all middleware are finished executing.

Usage

```
e := echo.New()
e.GET("/", <Handler>, <Middleware...>)
```

อ้างอิง: https://echo.labstack.com/middleware/

middleware function

```
e.Use(handler.Recover)
```

```go
func Recover(next echo.HandlerFunc) echo.HandlerFunc {
    return func(c echo.Context) error {
        defer func() {
            if rec := recover(); rec != nil {
                err, ok := rec.(error)
                if !ok {
                    err = fmt.Errorf("%v", rec)
                }
                stack := make([]byte, 4<<10) // 4KB
                length := runtime.Stack(stack, false)

                log.Errorf("[PANIC RECOVER] %v: %s", err, stack[:length])
            }
        }()
        //ส่งต่อ context ให้ hander fuction ต่อไป
        return next(c)
    }
}
```

Files

Endpoint -> รับ request มาจาก user ส่งให้ Service ทำการประมวลผลจากนั้นจึงส่ง response กลับไปให้ user

Service -> ทำการประมวลผลตามข้อมูลที่ request ส่งมา แล้วส่งผลลัพธ์ของการประมวลผลคืนให้ endpoint

Repository -> เป็น data access ทำหน้าที่ เพิ่ม, ลบ, แก้ไข และเรียกดูข้อมูลจาก database

GORM

```
go get -u gorm.io/gorm
go get -u gorm.io/driver/[driver name]
```

Connect database

```go
func (pg *Postgres) connectPostgres() error {

    var err error

    DB, err = gorm.Open(postgres.Open(pg.Uri), &gorm.Config{})

    if err != nil {
        fmt.Println(err)
    }
    return nil
}
```

```go
import (
  "gorm.io/driver/postgres"
  "gorm.io/gorm"
)

dsn := "host=localhost user=gorm password=gorm dbname=gorm port=9920 sslmode=disable TimeZone=Asia/Shanghai"
db, err := gorm.Open(postgres.Open(dsn), &gorm.Config{})
```

อ้างอิง: https://gorm.io/docs/connecting_to_the_database.html

Create data

```go
func (repo *Repo) AddEmployee(ctx context.Context, emp model.Employee) error {
    // if err := repo.db.Exec("INSERT INTO Employee VALUES(?, ?, ?, ?)", emp.ID, e
    //  Error; err != nil {
    //  fmt.Println(err)
    //  return err
    // }
    if err := repo.db.Table("employee").Create(&emp).Error; err != nil {
        fmt.Println(err)
        return err
    }
    return nil
}
```

Read data

```go
func (repo *Repo) GetEmployeeById(ctx context.Context, empId string) ([]model.Employee, error) {
    var em []model.Employee
    // if err := repo.db.Table("employee").Where("id = ?", empId).Find(&em).Error; err != nil {
    //  fmt.Println(err)
    //  return nil, err
    // }
    if err := repo.db.Raw("SELECT * FROM Employee WHERE ID = ?", empId).Scan(&em).Error; err != nil {
        fmt.Println(err)
        return nil, err
    }
    return em, nil
}
```

Update data

```go
func (repo *Repo) UpdateSalary(ctx context.Context, emp model.Employee) error {
    // if err := repo.db.Exec("UPDATE Employee SET Salary = ? WHERE ID = ?", emp.Salary, emp.ID).Error; err != nil {
    //  fmt.Println(err)
    //  return err
    // }
    if err := repo.db.Table("employee").Model(&model.Employee{}).
        Where("ID = ?", emp.ID).
        Update("salary", emp.Salary).Error; err != nil {
        fmt.Println(err)
        return err
    }
    return nil
}
```

Delete data

```go
func (repo *Repo) DeleteEmployeeById(ctx context.Context, empId string) error {
    // if err := repo.db.Table("employee").Exec("DELETE FROM Employee WHERE ID = ?", empId).Error; err != nil {
    //  fmt.Println(err)
    //  return err
    // }
    if err := repo.db.Table("employee").Where("ID = ?", empId).Delete(model.Employee{}).Error; err != nil {
        fmt.Println(err)
        return err
    }
    return nil
}
```

END