The Go Programming Language

API

Anuchit Prasertsang
Developer

Pre-required

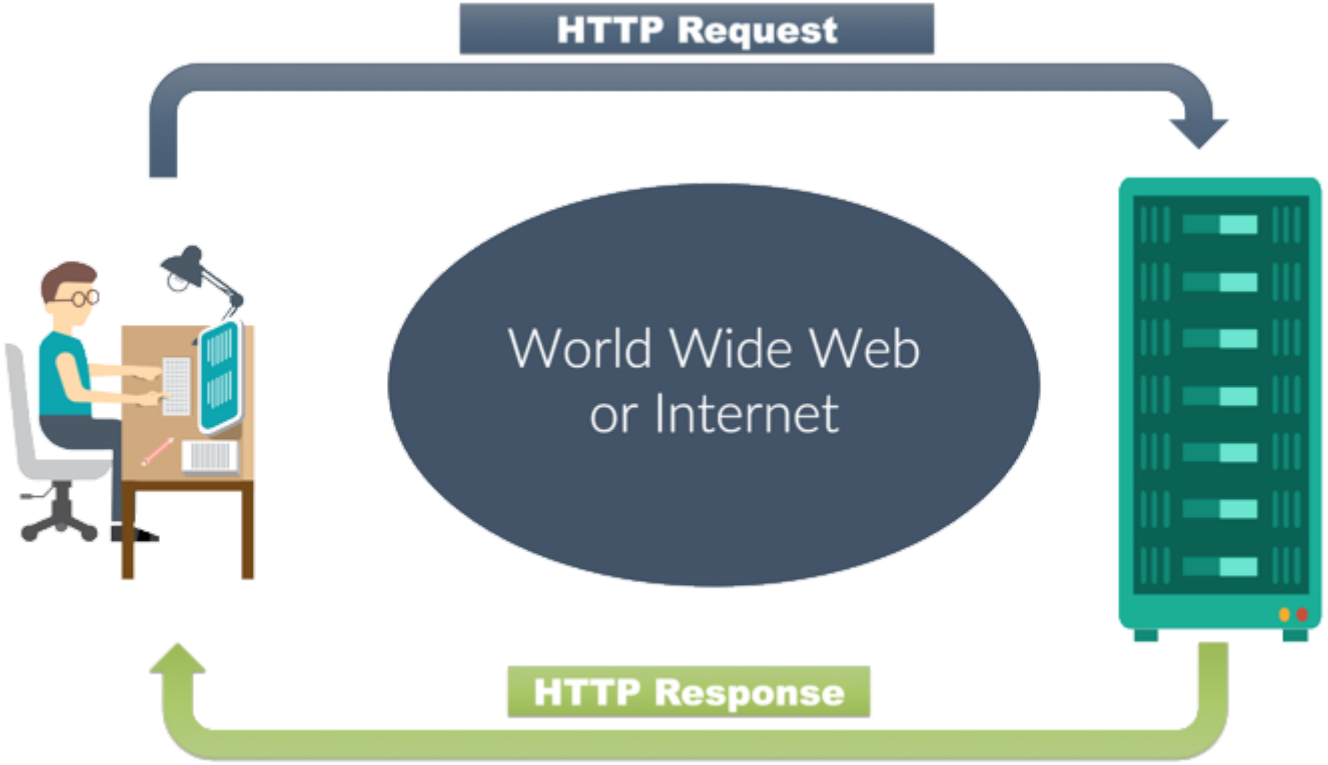
- postman
- docker

Book


Mat Ryer
Go Programming
Blueprints
Second Edition
Build real-world, production-ready solutions in Go using cutting-edge technology and techniques
Packt

protocol http

HTTP stands for "Hyper Text Transfer Protocol"



REST API

REST stands for "Representational State Transfer"
API stands for “Application Programming Interface”

Design concepts:

- HTTP methods describe the kind of action to take; for example, *GET* methods will only ever read data, while *POST* requests will create something.
- Data is expressed as a collection of resources.
- Actions are expressed as changes to data.
- URLs are used to refer to specific data.
- HTTP header are used to describe the kind of representation comming into and going out of the server.

HTTP methods and URLs

- GET /todos

```
Read all todos => Show a list of todos to the uses
```

- GET /todos/{id}

```
Read the todo => Show details or results of a specific todo
```

- POST /todos

```
Create a todo => Create a new todo
```

- DELETE /todos/{id}

```
Delete a todo => Delete a specific todo
```

{id} placeholder represent where in the path the unique ID for a poll. e.g **/poll/1**

purely HTTP in Go.

- package net/http

```
package main

import (
    "net/http"
)

func helloHandler(w http.ResponseWriter, req *http.Request) {
    resp := []byte(`{"name": "anuchit"}`)
    w.Write(resp)
}

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

    http.ListenAndServe(":1234", nil)
}
```

How about this?

```
http.ListenAndServe(":1234567890", nil)
```

Handle Error when start API

```
package main

import (
    "io"
    "log"
    "net/http"
)

func helloHandler(w http.ResponseWriter, req *http.Request) {
    resp := []byte(`{"name": "anuchit"}`)
    w.Write(resp)
    io.WriteString(w, "Hello")
}

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

    err := http.ListenAndServe(":1234", nil)
    log.Fatal(err)

    // normally make it one line
    // log.Fatal(http.ListenAndServe(":8080", nil))
}
```

GET /todos

- create func **todosHandler**
- create api path **/todos**
- response "hello GET todos"
- go to browser `` `http://localhost:1234/todos` ``

example - GET /todos

```
package main

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

func todoHandler(w http.ResponseWriter, req *http.Request) {
    method := "GET"
    fmt.Fprintf(w, "hello %s todos", method)
}

func main() {
    http.HandleFunc("/todos", todoHandler)

    log.Fatal(http.ListenAndServe(":1234", nil))
}
```

POST /todos

- What happen if we request with POST methods?
- Open **postman**
- POST /todos

Handle Method POST
- response "hello POST created todos"

example - POST /todos

```
package main

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

func todosHandler(w http.ResponseWriter, req *http.Request) {
    if req.Method == "POST" {
        fmt.Fprintf(w, "hello %s created todos", "POST")
        return
    }

    fmt.Fprintf(w, "hello %s todos", "GET")
}

func main() {
    http.HandleFunc("/todos", todosHandler)

    log.Fatal(http.ListenAndServe(":1234", nil))
}
```

POST /todos with body string

```
func todosHandler(w http.ResponseWriter, req *http.Request) {
    if req.Method == "POST" {
        body, err := ioutil.ReadAll(req.Body)
        if err != nil {
            fmt.Fprintf(w, "error : %v", err)
            return
        }

        fmt.Printf("body : %s\n", body)

        fmt.Fprintf(w, "hello %s created todos", "POST")
        return
    }

    fmt.Fprintf(w, "hello %s todos", "GET")
}
```

JSON (JavaScript Object Notation)

- JSON Object

```
{
  "name": "John",
  "age": 30,
  "weight": null
}
```

- JSON Array

```
[
  {
    "name": "John",
    "age": 30,
    "weight": null
  },
  {
    "name": "Nong",
    "age": 15,
    "weight": 59
  }
]
```

JSON in Go

- Todo json

```
{
  "id": "1",
  "title": "pay phone bills",
  "status": "active"
}
```

- use struct with tags

```
type Todo struct {
  ID     string `json:"id"`
  Title  string `json:"title"`
  Status string `json:"status"`
}
```

JSON to struct

```
import (
    "encoding/json"
    "fmt"
)

type Todo struct {
    ID     string `json:"id"`
    Title  string `json:"title"`
    Status string `json:"status"`
}

func main() {
    data := []byte(`{
            "id": "1",
            "title": "pay credit card",
            "status": "active"
        }`)

    var t Todo
    err := json.Unmarshal(data, &t)

    fmt.Printf("% #v\n", t)
    fmt.Println(err)
}
```

struct to JSON

```
type Todo struct {
    ID     string `json:"id"`
    Title  string `json:"title"`
    Status string `json:"status"`
}

func main() {
    t := Todo{
        ID:     "1",
        Title:  "pay credit card",
        Status: "completed",
    }

    b, err := json.Marshal(t)
    fmt.Printf("%T => %v \n %s \n", b, b, b)
    fmt.Println(err)
}
```

POST /todos with body json

```
    if req.Method == "POST" {
        body, err := ioutil.ReadAll(req.Body)
        if err != nil {
            fmt.Fprintf(w, "error : %v", err)
            return
        }

        t := Todo{}
        err = json.Unmarshal(body, &t)
        if err != nil {
            fmt.Fprintf(w, "error: ", err)
        }
        fmt.Printf("body : % #v\n", t)

        fmt.Fprintf(w, "hello %s created todos", "POST")
        return
    }
```

```
{
  "id": "1",
  "title": "pay phone bills",
  "status": "active"
}
```

exercise - store todo

- declare new variable `var todos []Todo`
- when receive POST /todos use json.Unmarshal body and append new todo into **todos**
- print it out

POST /todos store new todo

```
    if req.Method == "POST" {
        body, err := ioutil.ReadAll(req.Body)
        if err != nil {
            fmt.Fprintf(w, "error : %v", err)
            return
        }

        t := Todo{}
        err = json.Unmarshal(body, &t)
        if err != nil {
            fmt.Fprintf(w, "error: ", err)
        }

        todos = append(todos, t)
        fmt.Printf("% #v\n", todos)

        fmt.Fprintf(w, "hello %s created todos", "POST")
        return
    }
```

exercise - get all todos

- when receive GET

GET /todos - all todos list

```
    if req.Method == "GET" {
        b, err := json.Marshal(todos)
        if err != nil {
            w.WriteHeader(http.StatusInternalServerError)
            fmt.Fprintf(w, "error: ", err)
            return
        }

        w.Header().Set("Content-Type", "application/json")
        w.Write(b)
    }
```

exercise - PUT /todos/{1}

- update status of element 1

exercise - DELETE /todos/{1}

- delete element 1

Testing HTTP

```
func TestTodosHandler(t *testing.T) {
    t.Run("GET /todos", func(t *testing.T) {
        req, _ := http.NewRequest("GET", "/todos", nil)
        rr := httptest.NewRecorder()
        handler := http.HandlerFunc(todosHandler)

        handler.ServeHTTP(rr, req)

        status := rr.Code
        if http.StatusOK != status {
            t.Errorf("should response status code %v but got %v \n", http.StatusOK, status)
        }

        expected := `hello GET todos`
        resp := rr.Body.String()
        if expected != resp {
            t.Errorf("should response body %q but got %q \n", expected, resp)
        }
    })
}
```

Gin HTTP web framework

Gin example

```
package main

import (
    "net/http"

    "github.com/gin-gonic/gin"
)

func helloHandler(c *gin.Context) {
    c.JSON(http.StatusOK, gin.H{
        "message": "hello",
    })
}

func main() {
    r := gin.Default()
    r.GET("/hello", helloHandler)
    r.Run()                       // listen and serve on 127.0.0.0:8080
}
```

```
PORT=1234 go run serve.go
```

Using GET,POST,PUT,DELETE

exercise - Gin - GET /todos

- create func **getTodosHandler**
- create api path GET **/todos**
- response "hello Gin GET todos"
- go to browser `http://localhost:1234/todos`

next step
change response to be

```
[
  {
    "id": "1",
    "title": "pay phone bills",
    "status": "active"
  }
]
```

Gin GET /todos

```
type Todo struct {
    ID     string `json:"id"`
    Title  string `json:"title"`
    Status string `json:"status"`
}

var todos = map[string]*Todo{
    "1": &Todo{ID: "1", Title: "pay phone bills", Status: "active"},
}

func getTodosHandler(c *gin.Context) {
    items := []*Todo{}
    for _, item := range todos {
        items = append(items, item)
    }
    c.JSON(http.StatusOK, items)
}

func main() {
    r := gin.Default()
    r.GET("/todos", getTodosHandler)
    r.Run()
}
```

Gin POST /todos store new todo

- create func **createTodosHandler**
- create api path POST **/todos**
- store new todo into the todo list.
- postman post `http://localhost:1234/todos` with body

```
{
  "title": "pay phone bills",
  "status": "active"
}
```

Gin POST /todos

```
func createTodosHandler(c *gin.Context) {
    t := Todo{}
    if err := c.ShouldBindJSON(&t); err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
        return
    }

    i := len(todos)
    i++
    id := strconv.Itoa(i)
    t.ID = id
    todos[id] = &t

    c.JSON(http.StatusCreated, "created todo.")
}

func main() {
    r := gin.Default()
    r.GET("/todos", getTodosHandler)
    r.POST("/todos", createTodosHandler)

    r.Run(":1234")
}
```

Gin GET /todos/1 - get by id

```
func getTodoByIdHandler(c *gin.Context) {
    id := c.Param("id")

    t, ok := todos[id]
    if !ok {
        c.JSON(http.StatusOK, gin.H{})
        return
    }
    c.JSON(http.StatusOK, t)
}

func main() {
    r := gin.Default()
    r.GET("/todos", getTodosHandler)
    r.GET("/todos/:id", getTodoByIdHandler)
    r.POST("/todos", createTodosHandler)

    r.Run(":1234")
}
```

Gin GET /todos?status=value - filter by status

```
var todos = map[string]*Todo{
    "1": &Todo{ID: "1", Title: "pay phone bills", Status: "active"},
    "2": &Todo{ID: "2", Title: "pay credit", Status: "completed"},
    "3": &Todo{ID: "3", Title: "homework go language", Status: "active"},
    "4": &Todo{ID: "4", Title: "buy new shoes", Status: "completed"},
}
```

Gin GET /todos?status=active - filter by status

```
func getTodosHandler(c *gin.Context) {
    status := c.Query("status")
    items := []*Todo{}

    for _, item := range todos {
        if status != "" {
            if item.Status == status {
                items = append(items, item)
            }
        } else {
            items = append(items, item)
        }
    }

    c.JSON(http.StatusOK, items)
}

func main() {
    r := gin.Default()
    r.GET("/todos", getTodosHandler)
    r.Run(":1234")
}
```

Gin PUT /todos - update todo

```
func updateTodosHandler(c *gin.Context) {
    id := c.Param("id")
    t := todos[id]
    if err := c.ShouldBindJSON(t); err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
        return
    }

    c.JSON(http.StatusOK, t)
}

func main() {
    r := gin.Default()
    r.GET("/todos", getTodosHandler)
    r.GET("/todos/:id", getTodoByIdHandler)
    r.POST("/todos", createTodosHandler)
    r.PUT("todos/:id", updateTodosHandler)

    r.Run(":1234")
}
```

Gin DELETE /todos/:id - delete todo

```
func deleteTodosHandler(c *gin.Context) {
    id := c.Param("id")
    delete(todos, id)
    c.JSON(http.StatusOK, "deleted todo.")
}

func main() {
    r := gin.Default()
    r.GET("/todos", getTodosHandler)
    r.GET("/todos/:id", getTodoByIdHandler)
    r.POST("/todos", createTodosHandler)
    r.PUT("todos/:id", updateTodosHandler)
    r.DELETE("todos/:id", deleteTodosHandler)

    r.Run(":1234")
}
```

Testing Gin

```
package main

import "github.com/gin-gonic/gin"

func setupRouter() *gin.Engine {
    r := gin.Default()
    r.GET("/hello", func(c *gin.Context) {
        c.String(200, "hi.")
    })
    return r
}

func main() {
    r := setupRouter()
    r.Run(":1234")
}
```

Test filter

```
package main

import (
    "net/http"
    "net/http/httptest"
    "testing"

    "github.com/stretchr/testify/assert"
)

func TestPingRoute(t *testing.T) {
    router := setupRouter()
    w := httptest.NewRecorder()
    req, _ := http.NewRequest("GET", "/hello", nil)

    router.ServeHTTP(w, req)

    assert.Equal(t, 200, w.Code)
    assert.Equal(t, "hi.", w.Body.String())
}
```

Grouping routes

```
func main() {
    router := gin.Default()

    // Simple group: v1
    v1 := router.Group("/v1")
    v1.POST("/login", loginEndpoint)
    v1.POST("/submit", submitEndpoint)
    v1.POST("/read", readEndpoint)

    // Simple group: v2
    v2 := router.Group("/v2")
    v2.POST("/login", loginEndpoint)
    v2.POST("/submit", submitEndpoint)
    v2.POST("/read", readEndpoint)

    router.Run(":1234")
}
```

Gin Middleware

```
func helloHandler(c *gin.Context) {
    log.Println("in helloHandler")
    c.JSON(http.StatusOK, gin.H{
        "message": "hello",
    })
}

func main() {
    r := gin.Default()

    r.Use(func(c *gin.Context) {
        log.Println("start middleware")
        c.Next()
        log.Println("end middleware")
    })

    r.GET("/hello", helloHandler)
    r.Run(":1234")
}
```

Gin Middleware - Authorization

```
func helloHandler(c *gin.Context) {
    log.Println("in helloHandler")
    c.JSON(http.StatusOK, gin.H{"message": "hello"})
}

func authMiddleware(c *gin.Context) {
    log.Println("start middleware")
    authKey := c.GetHeader("Authorization")
    if authKey != "Bearer token123" {
        c.JSON(http.StatusUnauthorized, http.StatusText(http.StatusUnauthorized))
        c.Abort()
        return
    }

    c.Next()
    log.Println("end middleware")
}

func main() {
    r := gin.Default()
    r.Use(authMiddleware)
    r.GET("/hello", helloHandler)
    r.Run(":1234")
}
```

exercise

How to build

```
go build -o api
```

Homework - install docker

Thank you

Anuchit Prasertsang
Developer
anuchit.prasertsang@gmail.com (mailto:anuchit.prasertsang@gmail.com)
https://github.com/AnuchitO (https://github.com/AnuchitO)
@twitter_AnuchitO (http://twitter.com/twitter_AnuchitO)