Lección 5 - Manejo de solicitudes HTTP

Tabla de contenido

Introducción

En esta clase, nos sumergiremos en el mundo del manejo de solicitudes HTTP utilizando el framework Gin de Go. Las solicitudes HTTP son la columna vertebral de cualquier aplicación web, ya que permiten la comunicación entre el cliente y el servidor.

Objetivos

1. **Comprender el manejo de solicitudes HTTP en Gin:** Aprenderemos los conceptos básicos del manejo de solicitudes HTTP en Gin, incluyendo cómo definir rutas y cómo asociar controladores a estas rutas para manejar las solicitudes entrantes.
2. **Explorar los diferentes tipos de solicitudes HTTP y su manejo en Gin:** Analizaremos cómo Gin maneja diferentes métodos HTTP, como GET, POST, PUT y DELETE, y cómo podemos implementar lógica específica para cada tipo de solicitud.
3. **Instalar y utilizar la extensión Thunder Client en Visual Studio Code:** Aprenderemos cómo instalar y configurar la extensión Thunder Client en Visual Studio Code, y cómo utilizarla para realizar solicitudes HTTP y probar nuestras aplicaciones web de manera eficiente.

Desarrollo

Partimos de que ya tenemos creadas dos rutas, donde ambas son peticiones GET, una llama a la raíz (`/`) y la otra a `/saludo/:nombre` .

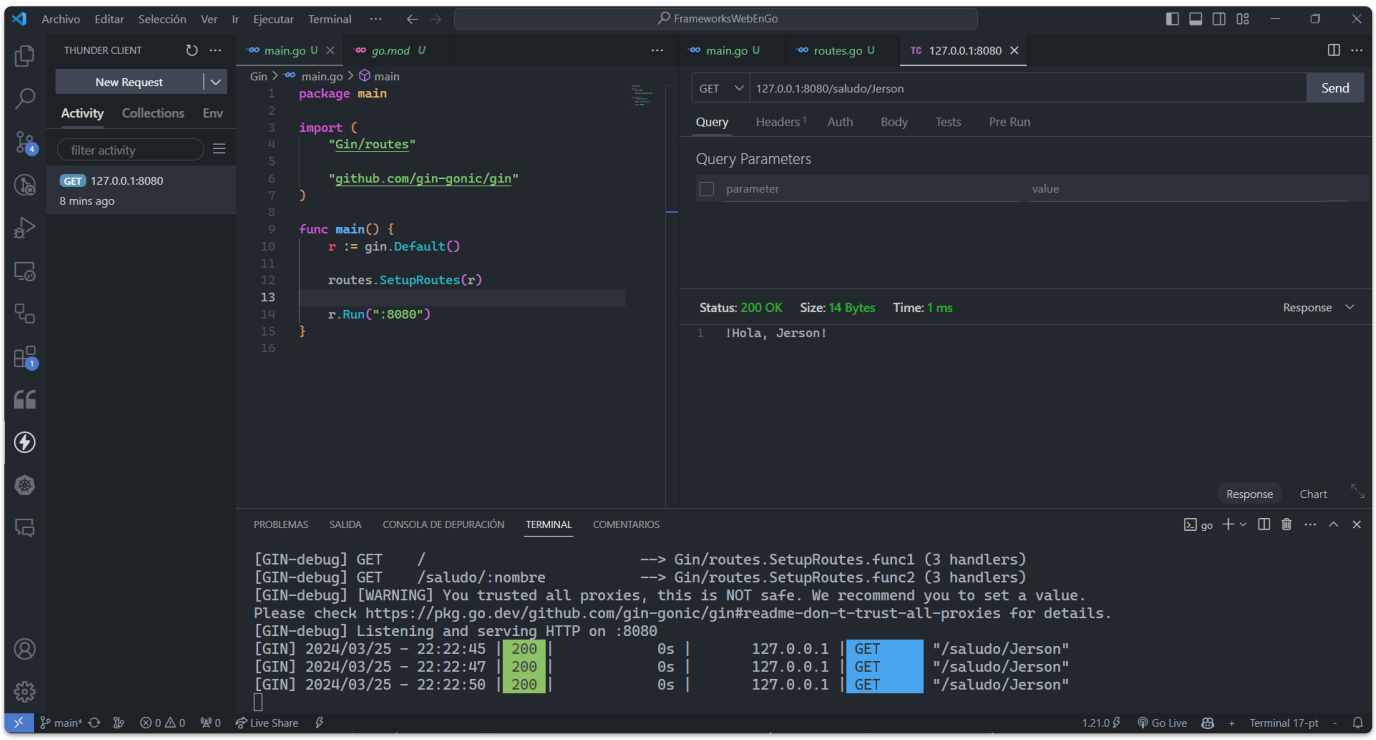
- La primera tiene que devolver un String "¡Hola, mundo!".
- La segunda tiene que devolver un "!Hola, ", acompañado del nombre de persona que se haya ingresado desde la barra de direcciones del navegador.

En este sentido, vamos a dejar de ir al navegador tan recurrentemente, por lo que si ya conoces aplicaciones como `Postman` , `Insomnia` , `Hoppscotch` , que sirven de cliente HTTP o bien, como herramientas para hacer pruebas de API.

En este caso, vamos a utilizar una alternativa muy interesante y que ha ganado popularidad en los últimos tiempos. Además, es una extensión de Visual Studio Code, llamada `Thunder Client` .

Instalación y uso de "Thunder Client" en Visual Studio Code

1. Abre la pestaña de extensiones en VS Code.
2. Busca "Thunder Client" y haz clic en "Instalar".
3. Una vez instalado, abre un nuevo archivo y haz clic en la pestaña de "Thunder Client" en la barra lateral izquierda.
4. Aquí podrás crear nuevas solicitudes HTTP, enviarlas y ver las respuestas recibidas.



Realizamos algunas pruebas, corremos el servidor actual con `go run main.go` .

- 127.0.0.1:8080

- 127.0.0.1:8080/saludo/
- 127.0.0.1:8080/saludo/Jerson

Todo lo que se ha probado, ha sido por el tipo de solicitud GET.

Ahora vamos a probar haciendo una solicitud por método POST.

Ejemplo de registro de un nuevo usuario

Actualizamos el fichero `routes.go` para hacer un registro.

```
// Usuario struct para representar la información del usuario
type Usuario struct {
    Nombre string `json:"nombre"`
    Email  string `json:"email"`
}

var usuarios []Usuario
```

Dentro de la función `SetupRoutes`, agregamos la nueva ruta por medio de POST:

```
// Ruta para crear un nuevo usuario
    r.POST("/usuarios", func(c *gin.Context) {
        // Obtener los datos del cuerpo de la solicitud
        var nuevoUsuario Usuario
        if err := c.BindJSON(&nuevoUsuario); err != nil {
            c.JSON(http.StatusBadRequest, gin.H{"error": "Error al
decodificar el JSON"})
            return
        }

        // Validar los datos del usuario
        if nuevoUsuario.Nombre == "" || nuevoUsuario.Email == "" {
            c.JSON(http.StatusBadRequest, gin.H{"error": "Nombre y correo
electrónico son campos requeridos"})
            return
        }

        // Guardar el nuevo usuario en la estructura de datos (en este
caso, en un slice)
        usuarios = append(usuarios, nuevoUsuario)

        // Responder con el nuevo usuario creado
```

```
        c.JSON(http.StatusOK, gin.H{"mensaje": "Usuario registrado",
"datos": usuarios})
    })
```

El script completo queda de la siguiente manera: `routes.go`.

```
package routes

import (
    "net/http"
    "github.com/gin-gonic/gin"
)

// Usuario struct para representar la información del usuario
type Usuario struct {
    Nombre string `json:"nombre"`
    Email  string `json:"email"`
}

var usuarios []Usuario

// SetupRoutes configura las rutas de la aplicación
func SetupRoutes(r *gin.Engine) {
    // Ruta básica
    r.GET("/", func(c *gin.Context) {
        c.String(http.StatusOK, "¡Hola, mundo!")
    })

    // Ruta con parámetros
    r.GET("/saludo/:nombre", func(c *gin.Context) {
        nombre := c.Param("nombre")
        c.String(http.StatusOK, "!Hola, "+nombre+"!")
    })

    // Ruta para crear un nuevo usuario
    r.POST("/usuarios", func(c *gin.Context) {
        // Obtener los datos del cuerpo de la solicitud
        var nuevoUsuario Usuario
        if err := c.BindJSON(&nuevoUsuario); err != nil {
            c.JSON(http.StatusBadRequest, gin.H{"error": "Error al
decodificar el JSON"})
            return
        }
```

```
        // Validar los datos del usuario
        if nuevoUsuario.Nombre == "" || nuevoUsuario.Email == "" {
            c.JSON(http.StatusBadRequest, gin.H{"error": "Nombre y correo
electrónico son campos requeridos"})
            return
        }

        // Guardar el nuevo usuario en la estructura de datos (en este
caso, en un slice)
        usuarios = append(usuarios, nuevoUsuario)

        // Responder con el nuevo usuario creado
        c.JSON(http.StatusOK, gin.H{"mensaje": "Usuario registrado",
"datos": usuarios})
    })
}
```

Pruebas por medio de Thunder Client

1. Seleccionamos el método POST.
2. Escribimos la URL: `127.0.0.1:8080/usuarios`
3. En el apartado de JSON, pasamos lo siguiente:

```
{
  "nombre": "Jerson Martínez",
  "email": "jerson@crashell.com"
}
```

Vamos a hacer una cuenta a Leonel Messi y a Cristiano Ronaldo ¿Por qué no?

```
{
  "nombre": "Leo Messi",
  "email": "leo@jersonmartinez.com"
}
// Primero uno y después el otro
{
  "nombre": "Cristiano Ronaldo",
  "email": "cr7@jersonmartinez.com"
}
```

El método POST ofrece una forma segura y versátil de enviar datos sensibles y grandes cantidades de información a un servidor web, lo que lo hace preferible en muchas situaciones sobre el método GET. Sin embargo, es importante elegir el método adecuado según los requisitos específicos de cada aplicación.

Subir los cambios a GitHub:

```
$ git add .
$ git commit -m "Manejo de solicitudes HTTP"
$ git push
```

Revisar en la web de GitHub que las actualizaciones estén disponibles. Detallaré mayor contenido sobre cada lección.

Conclusión

Hemos explorado cómo instalar y utilizar la extensión `Thunder Client` en Visual Studio Code para probar nuestras solicitudes HTTP de manera interactiva, lo que nos proporciona una herramienta valiosa para el desarrollo y depuración de aplicaciones web.

Hemos experimentado en el manejo de solicitudes HTTP utilizando el framework Gin en Go, tanto para solictudes GET y POST. Además de destacar la importancia de validar datos y gestión de errores para garantizar un manejo adecuado de las solicitudes entrantes.

Conoce más del autor

¡Encuéntrame en las siguientes redes sociales para estar al tanto de mis proyectos y actividades!

📱 **Red Social**	🌐 **Enlace**
🌐 Página web	jersonmartinez.com
👔 LinkedIn	Jerson Martínez - DevOps Engineer
📺 Canales de YouTube	DevOpsea | Side Master
💻 GitHub	Perfil en GitHub
🐓 Twitter (X)	@antoniomorenosm