Lección 5 - Uso de plantillas y carga de contenido estático

Tabla de contenido

Introducción

En esta lección, exploraremos cómo integrar plantillas HTML y cargar contenido estático, como archivos CSS y JavaScript, en nuestras aplicaciones web utilizando el framework `Fiber` en Go. Estos elementos son esenciales para construir interfaces de usuario atractivas y dinámicas.

Objetivos

1. **Integrar y Configurar Plantillas HTML en Fiber:**
   - Aprenderemos cómo configurar `Fiber` para usar un motor de plantillas y organizar los archivos de plantillas HTML en nuestra aplicación.
2. **Cargar y Servir Contenido Estático:**
   - Exploraremos cómo cargar y servir archivos estáticos, como estilos CSS y scripts JavaScript, para mejorar la apariencia y funcionalidad de nuestra aplicación.
3. **Pasar Datos a las Plantillas y Realizar Renderizado Dinámico:**
   - Comprenderemos cómo pasar datos desde nuestros controladores a las plantillas HTML y realizar un renderizado dinámico para mostrar información actualizada en nuestras páginas web.

Desarrollo

Integrar y Configurar Plantillas HTML en Fiber

Configurar el Proyecto

Crearemos un directorio `views` en la raíz del proyecto para almacenar nuestras plantillas HTML.

```
# Crear directorio
$ mkdir views

# Crear archivo HTML index.html
$ touch views/index.html
```

También, crearemos el directorio `static` y dentro de él los subdirectorios `css` y `js`, con sus respectivos archivos de estilo y script.

```
$ mkdir static
$ mkdir static/css static/js
$ touch static/css/styles.css
$ touch static/js/script.js
```

En este ejemplo, hemos creado un directorio llamado `views` para almacenar nuestras plantillas HTML y un directorio llamado `static` para almacenar nuestro contenido estático, como archivos CSS y JavaScript.

Crear una Plantilla HTML

Dentro del directorio `views`, crearemos el archivo `index.html`. Este será nuestro archivo de plantilla principal donde renderizaremos nuestro contenido dinámico.

```
<!DOCTYPE html>
<html lang="en">
        <head>
            <meta charset="UTF-8">
            <meta name="viewport" content="width=device-width, initial-
scale=1.0">
            <title>{{.Title}}</title>
            <link rel="stylesheet" href="/static/css/styles.css">
        </head>
        <body>
            <h1>{{.Heading}}</h1>
            <p>{{.Message}}</p>
            <script src="/static/js/script.js"></script>
```

```
        </body>
</html>
```

Cargar y Servir Contenido Estático

Crear Rutas y Manejar Solicitudes

Necesitamos descargar el paquete para renderizar `html`.

```
go get -u github.com/gofiber/template/html/v2
```

El fichero `main.go` queda de la siguiente manera:

```
package main

import (
    "log"
    "Fiber/routes"
    "github.com/gofiber/fiber/v2"
    "github.com/gofiber/template/html/v2"
)

func main() {
    // Crea una instancia de la aplicación Fiber
    app := fiber.New()
    engine := html.New("./views", ".html")

    app = fiber.New(fiber.Config{
        Views: engine,
    })

    // Configurar el manejador para la carga de contenido estático
    app.Static("/static", "./static")

    // Middleware para registrar cada solicitud
    app.Use(loggingMiddleware)
    routes.SetupRoutes(app)

    // Inicia el servidor en el puerto 3000
    log.Fatal(app.Listen(":3000"))
}

// Middleware para registrar cada solicitud
```

```
func loggingMiddleware(c *fiber.Ctx) error {
    log.Printf("Solicitud recibida: %s %s", c.Method(), c.Path())

    // Llama al siguiente middleware en la cadena
    return c.Next()
}
```

El fichero `routes.go` queda de la siguiente manera:

```
package routes

import (
    "github.com/gofiber/fiber/v2"
)

type User struct {
    ID       int    `json:"id"`
    Username string `json:"username"`
    Email    string `json:"email"`
}

// SetupRoutes configura las rutas de la aplicación
func SetupRoutes(app *fiber.App) {
    app.Get("/", handlerInicio)
    app.Get("/about", handlerAcercaDe)
    app.Get("/saludo/:nombre", handlerSaludo)

    // Define una ruta para manejar solicitudes POST
    app.Post("/api/usuarios", func(c *fiber.Ctx) error {
        // Parsea los datos del cuerpo de la solicitud JSON
        var usuario User
        if err := c.BodyParser(&usuario); err != nil {
            return err
        }

        // Retorna una respuesta JSON
        return c.JSON(usuario)
    })
}

// Controladores para las rutas definidas
func handlerInicio(c *fiber.Ctx) error {
    return c.Render("index", fiber.Map{
```

```
        "Title":   "Mi Aplicación",
        "Heading": "¡Hola, mundo!",
        "Message": "Bienvenido a mi aplicación web con Fiber y plantillas
HTML.",
    })
}

func handlerAcercaDe(c *fiber.Ctx) error {
    return c.SendString("Acerca de nosotros: Somos una aplicación
construida con Fiber.")
}

func handlerSaludo(c *fiber.Ctx) error {
    nombre := c.Params("nombre")
    return c.SendString("¡Hola, " + nombre + "!")
}
```

Cargar y Servir Contenido Estático

- Dentro del directorio raíz del proyecto, crearemos un directorio `static` para almacenar nuestros archivos estáticos, como CSS y JavaScript.

```
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    height: 100vh;
    background: linear-gradient(to bottom, #6bdd95, #38c32e);
}

h1 {
    color: #3a3a3a;
    margin-bottom: 20px;
}

p {
    color: #4e4e4e;
    font-size: 18px;
    line-height: 1.5;
```

```
    margin-bottom: 20px;
}
```

Actualizar el contenido del archivo `.js`:

```
function changeMessage() {
    const messageElement = document.getElementById("message");
    const currentMessage = messageElement.textContent;

    // Lista de mensajes alternativos
    const alternateMessages = [
        "¡Hola de nuevo!",
        "¡Bienvenido de vuelta!",
        "¡Qué tengas un buen día!"
    ];

    // Generar un mensaje aleatorio que no sea igual al mensaje actual
    let newMessage;
    do {
        newMessage = alternateMessages[Math.floor(Math.random() *
alternateMessages.length)];
    } while (newMessage === currentMessage);

    // Actualizar el contenido del elemento de mensaje
    messageElement.textContent = newMessage;
}
```

El body del HTML para que funcione el botón, tiene que quedar de la siguiente manera:

```
    <body>
        <h1>{{.Heading}}</h1>
        <p id="message">{{.Message}}</p>
        <button onclick="javascript: changeMessage();">Actualizar
mensaje</button>
        <script src="/static/js/script.js"></script>
    </body>
```

Hacer las pruebas en el navegador.

Subir los cambios a GitHub:

```
$ git add .
$ git commit -m "Uso de plantillas y carga de contenido estático con
Fiber"
$ git push
```

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

Conclusión

En esta lección, hemos aprendido cómo integrar plantillas HTML y cargar contenido estático en nuestras aplicaciones web utilizando el framework `Fiber` en Go. Estos elementos son fundamentales para crear interfaces de usuario dinámicas y atractivas. Con este conocimiento, estamos listos para desarrollar aplicaciones web más complejas y escalables con `Fiber`.

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