Lección 5 - Uso de plantillas y carga de contenido estático

Tabla de contenido

Introducción

En esta lección, nos adentraremos en la integración de plantillas HTML y la carga de contenido estático, como archivos CSS y JavaScript, en nuestras aplicaciones web utilizando el framework Gorilla de Go. Estos son elementos esenciales para construir interfaces de usuario atractivas y dinámicas.

Objetivos

1. **Integrar y Configurar Plantillas HTML en Gorilla:**
   - Aprenderemos cómo configurar Gorilla 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 e 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 Gorilla

Configurar el Proyecto

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

```
# Crear directorio
$ mkdir templates

# Crear archivo HTML index.html
$ touch templates/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 `templates` 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 `templates`, 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

```
package main

import (
        "html/template"
        "net/http"

        "github.com/gorilla/mux"
)

func main() {
        // Crear un enrutador Gorilla
        r := mux.NewRouter()

        // Configurar el manejador para la carga de contenido estático
        r.PathPrefix("/static/").Handler(http.StripPrefix("/static/",
http.FileServer(http.Dir("./static/"))))

        // Configurar la ruta para cargar la página principal
        r.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
                // Renderizar la plantilla index.html con los datos
proporcionados
                renderTemplate(w, "index.html", map[string]interface{}{
                        "Title":   "Mi Aplicación",
                        "Heading": "¡Hola, mundo!",
                        "Message": "Bienvenido a mi aplicación web con
Gorilla y plantillas HTML.",
                })
        })

        // Iniciar el servidor en el puerto 8080
        http.ListenAndServe(":8080", r)
}

// renderTemplate renderiza una plantilla HTML con datos proporcionados
```

```go
func renderTemplate(w http.ResponseWriter, tmpl string, data interface{})
{
        tmpl = "templates/" + tmpl
        t, err := template.ParseFiles(tmpl)
        if err != nil {
                http.Error(w, err.Error(), http.StatusInternalServerError)
                return
        }
        err = t.Execute(w, data)
        if err != nil {
                http.Error(w, err.Error(), http.StatusInternalServerError)
        }
}
```

Cargar y Servir Contenido Estático

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

```css
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
```

```
Gorilla"
$ git push
```

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

Conclusión

Hemos aprendido cómo integrar plantillas HTML y cargar contenido estático en nuestras aplicaciones web utilizando Gorilla. Estos son elementos fundamentales para crear interfaces de usuario dinámicas y atractivas. Con este conocimiento, estamos listos para desarrollar aplicaciones web más complejas y escalables con Gorilla.

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