

Lección 6 - Adaptación de plantilla completa con diferentes páginas

Tabla de contenido

- [Introducción](#)
- [Objetivos](#)
- [Desarrollo](#)
- [Conclusión](#)
- [Conoce más del autor](#)

Introducción

En esta lección, exploraremos cómo integrar una plantilla HTML completa con diferentes páginas en una aplicación web utilizando el framework `Fiber` de Go. Esto nos permitirá crear una aplicación web completa con múltiples vistas y un diseño coherente.

Objetivos

1. **Configurar el Motor de Renderizado de Plantillas HTML:**
 - Profundizar en la configuración del motor de renderizado de plantillas HTML en Fiber. Aprenderemos cómo cargar y renderizar plantillas HTML para diferentes páginas de nuestra aplicación.
2. **Servir Contenido Estático:**
 - Aprender a servir contenido estático, como archivos CSS, JavaScript e imágenes, utilizando Fiber. Esto mejorará la apariencia y la interactividad de la aplicación.
3. **Manejar Rutas Dinámicas:**
 - Implementar rutas dinámicas en Fiber que acepten diferentes parámetros y carguen páginas específicas según la solicitud del usuario.

Desarrollo

Descargar una Plantilla Completa HTML:

- Buscaremos una plantilla HTML completa en Internet que se adapte a nuestras necesidades. En este ejemplo, utilizaremos una plantilla de Start Bootstrap llamada "SB Admin 2". **Buscar en Google:** StartBootstrap SB Admin 2

Descargar el repositorio:

- [GitHub - StartBootstrap/startbootstrap-sb-admin-2: A free, open source, Bootstrap admin theme created by Start Bootstrap](https://github.com/StartBootstrap/startbootstrap-sb-admin-2)

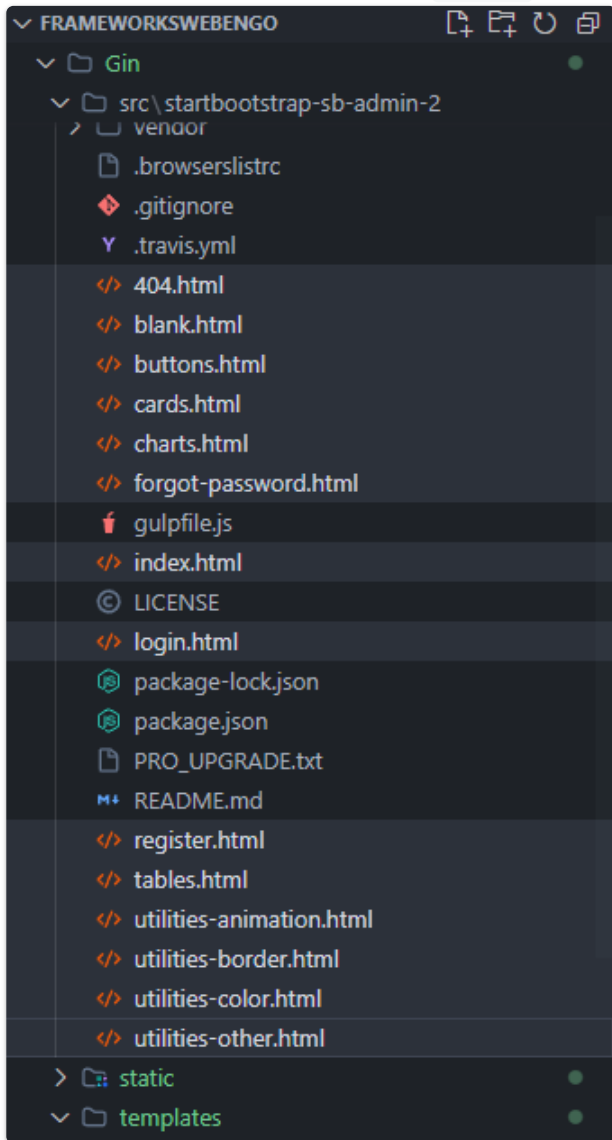
Creamos un directorio `/Fiber/src/` para hacer el `git clone` del repositorio.

```
$ mkdir src
$ cd src

$ git clone https://github.com/StartBootstrap/startbootstrap-sb-admin-2.git
Cloning into 'startbootstrap-sb-admin-2'...
remote: Enumerating objects: 8826, done.
remote: Counting objects: 100% (16/16), done.
remote: Compressing objects: 100% (12/12), done.
remote: Total 8826 (delta 8), reused 11 (delta 4), pack-reused 8810
Receiving objects: 100% (8826/8826), 28.10 MiB | 6.81 MiB/s, done.
Resolving deltas: 100% (3300/3300), done.
```

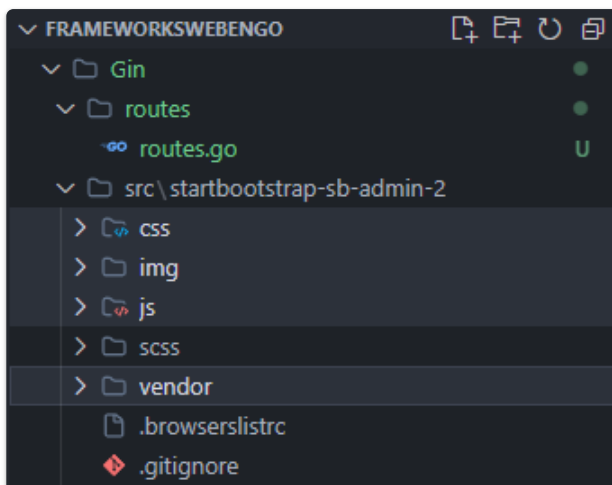
Al `index.html` que teníamos, le llamaremos `index-old.html`, para no colicionar con el `index.html` de la plantilla.

Seleccionar todos los ficheros `.html` y copiarlos al directorio `templates\`:



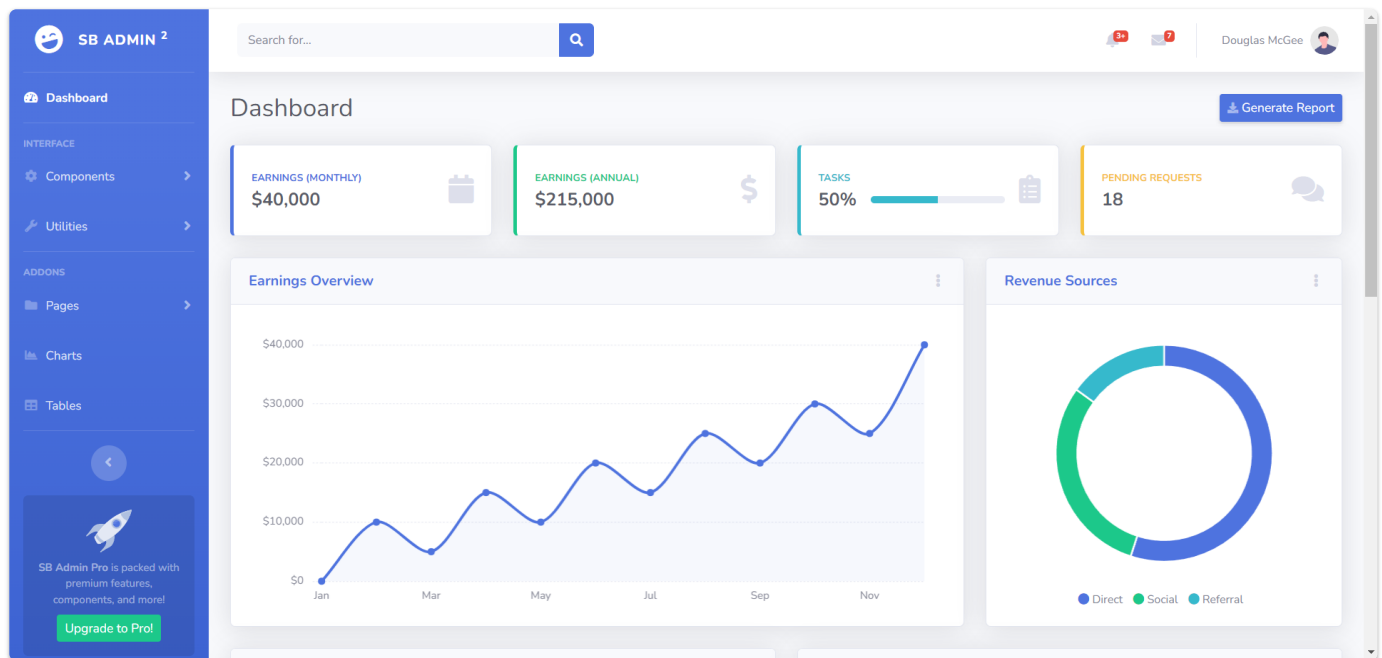
Antes también renombramos los directorios `css` y `js` a `css-old` y `js-old`, para proceder a copiar los demás recursos.

Copiar los directorios `css`, `img`, `js` y `vendor` al directorio `static`.



En todos los ficheros `.html` que se han colocado dentro del directorio templates, hay que modificar las rutas de llamadas de los recursos estáticos.

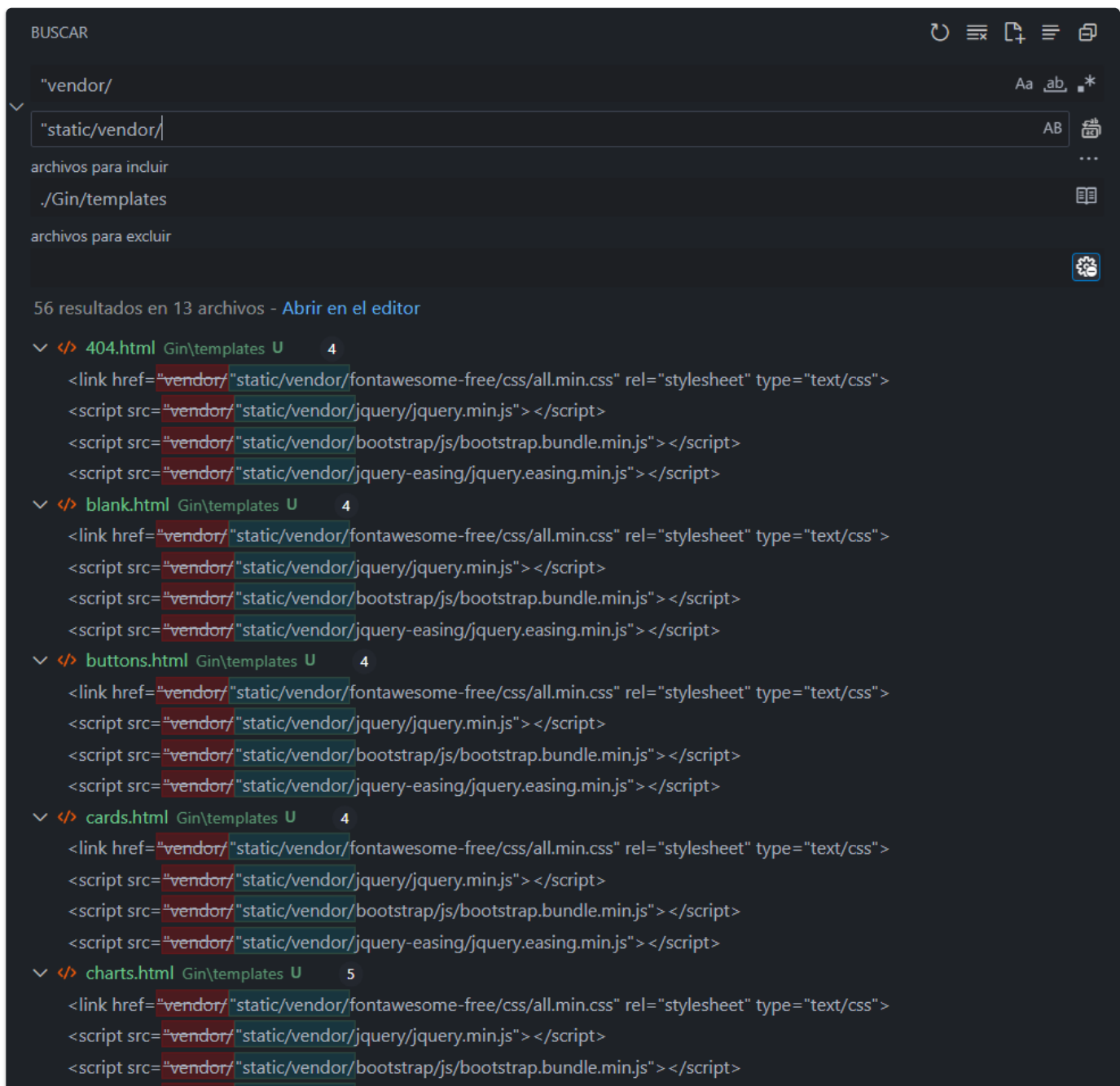
Empezaremos con el `index.html`:Anteponer a todas las direcciones `/static/`, tanto a los `vendor/`, `css/`, `js/`, `img/`.



Ahora hay que hacer lo mismo para los demás.

Buscar en todos los recursos dentro del directorio templates el siguiente patrón:

- `"vendor/` y reemplazar por `"/static/vendor/`.
- `"css/` y reemplazar por `"/static/css/`.
- `"js/` y reemplazar por `"/static/js/`.
- `"img/` y reemplazar por `"/static/img/`.



Finalmente, eliminamos el directorios de recursos, src, ya que no es necesario mantenerlo.

```
rm -rf src
```

```
package routes
```

```
import (  
    "html/template"  
    "net/http"  
    "os"  
    "strings"
```

```

        "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) {
        // Configurar el motor de renderizado de plantillas HTML
        renderer := Renderer()

        // Configurar Fiber para manejar rutas dinámicas
        app.Get("/:page", dynamicPageHandler(renderer))

        // Definir rutas aquí...
        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)
        })
    }

    // Renderer configura el motor de renderizado de plantillas HTML
    func Renderer() *template.Template {
        return template.Must(template.ParseGlob("views/*.html"))
    }

    // Función para manejar páginas dinámicas
    func dynamicPageHandler(_ *template.Template) fiber.Handler {
        return func(c *fiber.Ctx) error {

```

```

        page := c.Params("page")

        // Si el nombre de la página tiene la extensión .html,
eliminarla
        if strings.HasSuffix(page, ".html") {
            page = strings.TrimSuffix(page, ".html")
        }

        // Comprobar si la plantilla solicitada existe
        if _, err := os.Stat("views/" + page + ".html"); err ==
nil {
            // Si la plantilla existe, renderizarla
            return c.Render(page, nil)
        }

        // Si la plantilla no existe, mostrar página de error 404
        return c.Status(http.StatusNotFound).SendString("Página no
encontrada")
    }
}

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 + "!")
}

```

Nuestro `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()
}

```

Comprobamos que todas las páginas estén funcionando correctamente.

Subir los cambios a GitHub:


```
$ git add .  
$ git commit -m "Adaptación de plantilla completa con diferentes páginas  
utilizando 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 adaptar una plantilla HTML completa con diferentes páginas utilizando el framework `Fiber` de Go. Configuramos Fiber para manejar las rutas dinámicas y servir contenido estático, lo que nos permitió crear una aplicación web completa con múltiples vistas. Con este conocimiento, estamos preparados para desarrollar aplicaciones web más complejas y personalizadas en Go utilizando `Fiber`.

Conoce más del autor

¡Encuétrame 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