

Lección 3 - Crear un servidor web

Tabla de contenido

- [Introducción](#)
- [Objetivos](#)
- [Desarrollo](#)
 - [Configuración de un Servidor Básico con Fiber](#)
 - [Implementación de Middleware en Fiber](#)
 - [Manejo de Rutas y Solicitudes HTTP](#)
- [Conclusión](#)
- [Conoce más del autor](#)

Introducción

En esta lección, exploraremos cómo configurar un servidor básico con `Fiber` y cómo manejar rutas y solicitudes HTTP de manera efectiva.

Objetivos

1. **Configuración de un Servidor Básico con Fiber:**
 - Aprender a inicializar un servidor web básico con `Fiber` y configurarlo para escuchar en un puerto específico.
 2. **Manejo de Rutas y Solicitudes HTTP:**
 - Explorar cómo definir rutas y manejar solicitudes HTTP utilizando `Fiber` para responder con diferentes tipos de contenido.
 3. **Implementación de Middleware en Fiber:**
 - Entender cómo agregar middleware en `Fiber` para realizar operaciones adicionales, como registro de solicitudes, manejo de autenticación, entre otros.
-

Desarrollo

Configuración de un Servidor Básico con Fiber

Para comenzar, crearemos un servidor web básico utilizando `Fiber`. Aquí tienes un ejemplo básico de cómo configurar un servidor y manejar una solicitud:

```
package main

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

func main() {
    // Crea una instancia de la aplicación Fiber
    app := fiber.New()

    // Define una ruta principal
    app.Get("/", func(c *fiber.Ctx) error {
        return c.SendString("¡Hola, mundo!")
    })

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

- En este código, creamos una instancia de la aplicación `Fiber`, definimos una ruta para la raíz ("/") y manejamos la solicitud enviando un saludo.
- La función `func(c *fiber.Ctx) error` recibe un contexto de Fiber (`c`) que contiene información sobre la solicitud entrante y proporciona métodos para enviar respuestas. En este caso, utilizamos `c.SendString` para enviar la respuesta.

Implementación de Middleware en Fiber

Fiber nos permite agregar middleware para realizar operaciones adicionales antes o después de manejar una solicitud. Aquí hay un ejemplo de cómo agregar middleware para el registro de solicitudes:

```
// 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()
}
```

```
}

// Agregar middleware de registro a todas las rutas
app.Use(loggingMiddleware)
```

Hacer la llamada del `Middleware`:

```
package main

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

func main() {
    // Crea una instancia de la aplicación Fiber
    app := fiber.New()

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

    // Define una ruta principal
    app.Get("/", func(c *fiber.Ctx) error {
        return c.SendString("¡Hola, mundo!")
    })

    // Inicia el servidor en el puerto 3000
    log.Println("Servidor Fiber en ejecución en http://localhost:3000")
    log.Fatal(app.Listen(":3000"))
}
```

Manejo de Rutas y Solicitudes HTTP

`Fiber` nos permite definir rutas y manejar solicitudes HTTP de forma clara y estructurada. Vamos a trabajar una solicitud POST, donde enviamos un JSON.

```
package main

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

)

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

func main() {
    // Crea una instancia de la aplicación Fiber
    app := fiber.New()

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

    // Define una ruta principal
    app.Get("/", func(c *fiber.Ctx) error {
        return c.SendString("¡Hola, mundo!")
    })

    // 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
        }

        // Procesa los datos del usuario y realiza operaciones
        adicionales

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

    // Inicia el servidor en el puerto 3000
    log.Println("Servidor Fiber en ejecución en
http://localhost: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()
}

```

Cuando se realiza una solicitud POST a la ruta `/api/usuarios` en el servidor `Fiber`, se espera que el cuerpo de la solicitud contenga datos en formato JSON que representen un usuario. Estos datos deben incluir al menos tres campos: `id`, `username` y `email`, como se define en la estructura `User` en el código.

```

{
  "id": 1,
  "username": "jersonmartinez",
  "email": "jerson@crashell.com"
}

```

The screenshot displays a development environment with the following components:

- Code Editor:** Shows the `main.go` file for the Fiber server. It includes a main function that sets up a new Fiber app, defines a GET route for the root path, and a POST route for `/api/usuarios`. The POST handler parses the request body as JSON, creates a `User` struct, and returns it as a JSON response. The server is configured to run on `localhost:3000`.
- REST Client:** A "New Request" window is open, showing a POST request to `http://127.0.0.1:3000/api/usuarios`. The request body is a JSON object: `{ "id": 1, "username": "jersonmartinez", "email": "jerson@crashell.com" }`. The response status is `200 OK` with a size of `66 Bytes` and a time of `1 ms`.
- Terminal:** Shows the output of the `go run main.go` command. It indicates that the Fiber v2.52.4 server is running on `http://127.0.0.1:3000`. Below this, there is a summary of server statistics: `Handlers: 4, Processes: 1, Prefork: Disabled, PID: 39132`. The terminal also shows a series of log messages indicating that the server received several POST requests to `/api/usuarios`.

El servidor `Fiber` recibirá estos datos y los analizará para convertirlos en una estructura `User`. Luego, el servidor puede realizar cualquier operación adicional necesaria con estos datos y, finalmente, responderá con los mismos datos del usuario en formato JSON como confirmación de que la solicitud fue procesada correctamente.

Conclusión

¡Con estos pasos, hemos creado con éxito un servidor web funcional con **Fiber**! La definición de rutas, la implementación de controladores y la configuración inicial del proyecto son fundamentales para comenzar a construir aplicaciones web emocionantes.

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