GO

BOOTCAMP

Clase en vivo

//Go Web

IT BOARDING

BOOTCAMP

Objetivos de la clase:

- Conocer y comprender que son los parámetros

Índice

IT BOARDING
BOOTCAMP

1 | Repaso

IT BOARDING

BOOTCAMP

Context

Agregando valores a nuestro contexto

Podemos pensar el **context** como si fuera un mapa, por lo que puede agregar y recuperar valores por clave. Esto es muy poderoso, ya que nos permite almacenar cualquier tipo de datos dentro del contexto.

```
ctx := context.Background()
context.WithValue(ctx, "api-key", "super-secret-api-key")
```

Es importante tener en cuenta que la función WithValue() devuelve una copia del contexto existente y no modifica el contexto original.

Cancelación del context

Para crear un contexto con cancelación solo tenemos que llamar a la función **context.WithCancel(ctx)** pasando su contexto como parámetro. Esto devolverá un nuevo contexto y una función de cancelación. Para cancelar ese contexto, solo necesita llamar a la función de cancelación.

{}

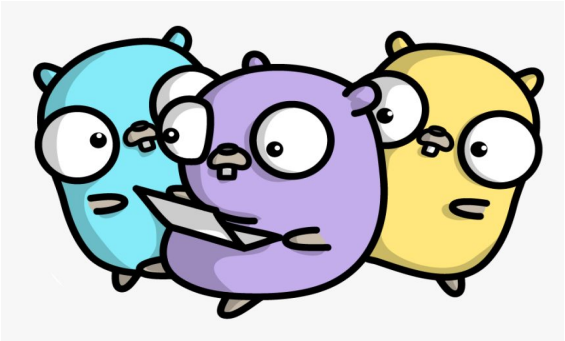
```
ctx, cancel := context.WithCancel(context.Background())
defer cancel()
```

Método GET

// ¿Qué es un GET?

Es un método HTTP que se usa para solicitar una representación de un recurso específico. Las peticiones que usan el método GET sólo deben recuperar datos.

Ejemplo ilustrado



Cliente

Request

GET /hi/greetings.txt HTTP/1.1

Accept: text/*

Host: www.gophers.com

www.gophers.com

2 | Parámetros

IT BOARDING

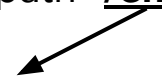
BOOTCAMP

// ¿Qué es un parámetro?

Un parámetro es una variable (en este caso, en la url) utilizada para recibir valores. Estos valores los podemos utilizar dentro de nuestra aplicación.

Fundamentos - Path Params

Una empresa posee una lista de empleados, y queremos consultar la información de empleados de acuerdo a su ID. Si dirigimos la consulta al path “**/employee/:ID**” sería como el siguiente ejemplo:



http://www.employee.com/employees/11

http://www.employee.com/employees/24

Estamos recibiendo del cliente un dato como parámetro sin nombre, y le estamos dando un nombre “ID” al recibirlo en nuestra ruta.

En el ejemplo de arriba, al recibir el valor el *11* nuestro router lo convierte en el parámetro *ID* cuyo valor será *11*.

Fundamentos - Query Params #1

Ahora bien, los parámetros de ruta (*path params*) nos resultan útiles hasta cierto punto. ¿Cómo podemos aclarar qué queremos una información que cumpla determinados requerimientos?

Supongamos que tenemos un servidor de una tienda virtual, y queremos obtener de nuestra colección de artículos aquel ítem cuyo código sea 12742:

http://www.myshop.com/articles?code=12742

Vemos que se parece a una web como las que vimos en ejemplos anteriores, pero, aquí hay un detalle adicional: el signo de pregunta (?)

Fundamentos - Query Params #2

Lo que denota el signo de pregunta (*question mark*) es el componente de **query** de la URL que se pasa al servidor o recurso de gateway.

Este componente se acompaña del path de la URL identificando al recurso.

Toda consulta query se compone siempre de una **llave** y un **valor** (¡sí, como un mapa!) y si deseamos enviar más de una llave/valor debemos separar los pares por **&**,

Ejemplo:
http://www.myshop.com/check-inventari.com?item=28&brand=Amaizing

Ejemplo de Path Params

Ahora bien, vimos qué son las query, y qué son los paths. Veamos cómo podemos hacerlo funcionar con Chi. Let's go!

Primero creamos creamos nuestro Controller con acceso a un storage.

{}

```
package handlers

// ControllerEmployee is the controller for the employee entity that returns handlers
type ControllerEmployee struct {
    // storage
    st map[string]string    // key: id, value: employee name
}
```

Ejemplo de Path Params #2

Creamos nuestra Response para el Cliente.

```
{}
```

```go
package handlers

// Get returns the employee with the given id
type ResponseGetByIdEmployee struct {
    Message string `json:"message"`
    Data    *struct {
        Id   string `json:"id"`
        Name string `json:"name"`
    } `json:"data"`
    Error   bool   `json:"error"`
}
```

Ejemplo de Path Params #2

Creamos nuestra http.HandlerFunc (request - getEmployee).

{}

```go
// Get returns the employee with the given id
func (c *ControllerEmployee) GetById() http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        // request
        id := chi.URLParam(r, "id")

        // process
        // -> get employee
        employee, ok := c.st[id]
        if !ok {
            code := http.StatusNotFound
            body := &ResponseGetByIdEmployee{Message: "Employee not found", Data: nil, Error: true}

            w.WriteHeader(code)
            w.Header().Add("Content-Type", "application/json")
            json.NewEncoder(w).Encode(body)
            return
        }
```

Ejemplo de Path Params #2

Creamos nuestra http.HandlerFunc (response).

{}

```go
// Get returns the employee with the given id
func (c *ControllerEmployee) GetById() http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        // ...
        // response
        code := http.StatusOK
        body := &ResponseGetByIdEmployee{Message: "Employee found", Data: &struct {
            Id   string `json:"id"`
            Name string `json:"name"`
        }{Id: id, Name: employee}, Error: false}

        w.WriteHeader(code)
        w.Header().Add("Content-Type", "application/json")
        json.NewEncoder(w).Encode(body)
    }
}
```

¿Y si mi param no existe en la URL?

¿Cómo sabe Chi cuáles son los parámetros que pasamos en una request? Aquí vemos cómo chi maneja los URL / Path parameters.

http.Request -> Chi guarda una key asociada a un map key-value en el Context ->
http.Request -> handlers -> Chi puede obtener el valor con la key en el Context.

{}

```
var (
    // RouteCtxKey is the context.Context key to store the request context.
    RouteCtxKey = &contextKey{"RouteContext"}
)

// URLParam returns the url parameter from a http.Request object.
func URLParam(r *http.Request, key string) string {
    if rctx := RouteContext(r.Context()); rctx != nil {
        return rctx.URLParam(key)
    }
    return ""
}
```

A codear...

Ahora trabajemos con los query params

Chi no cuenta con una funcionalidad para manejar los Query Params. Debemos utilizar lo que nos provee el type nativo de go ***http.Request.**

{}

```
func (c *ControllerEmployee) GetById() http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        // request
        id := r.URL.Query().Get("id")
        // ...
```

A codear...

Conclusiones

En esta clase terminamos de comprender los conceptos de contexto web, con el que lograremos una comunicación entre las capas de nuestra aplicación.

Además repasamos las Peticiones HTTP GET y aprendimos a aplicar parámetros a nuestros endpoints.

Actividad

Gracias.

IT BOARDING

BOOTCAMP