Golang APIRest

Tags Golang

De primeira mão, foi criado a função

```
func HandleRequest() {
    http.HandleFunc("/",nomeFuncao)
    http.ListenAndServe(":8000", nil)
}
```

> → Faça de exercicio, algo bem parecido ao padrao MVC do Golang Aplicação Web do zero.
>
> → Faça um tipo personalidades e declare ela em main.go mocando algumas personalidades.

Quando for usar uma struct que va utilizar json, na declaração da struct,

... Nome string `json:"nome"` ...

🦍**Gorilla Mux: Melhor manipulação de rotas.**

Instalação: procurar no site do goc.

Para usar:

```
import "github.com/gorilla/mux"

func HandleRequest() {
    r := mux.NewRouter()
    r.http.HandleFunc("/api/personalidade/{id}",nomeFunca
o).Methods("Get")
    r.http.ListenAndServe(":8000", r)
}
```

```
----
func RetornaUmaPersonalidade(w http.ResponseWriter, r *htt
p.Request) {
    vars:= mux.Vars(r)
    id := vars["id"]

    for _, personalidade.Id := range models.Personalidades
{
        if strconv.Itoa(peprsonalidade.Id) == id{
            json.NewEncoder(w).Encode(personalidade)
        }
    }
}
```

Modulo JSON:

`json.NewEncoder(w).Encode(models.Personalidades)`
→ Vai encodar um novo Json, e encodar as personalidades.

> Até aqui, deve retornar apenas uma das personalidades baseada no ID. e uma que retorne tudo!

💡 Lembrando: Para mocar os dados:

Crie uma variavel do tipo da struct do model. Exemplo: dentro de personalidade tem o tipo Personalidade, com isso crio uma variavel Personalidades []Personalidade

Conexão com o banco.

→ Criar pasta migration e o arquivo docker-database-inital.sql

→ docker-compose.yml e docker compose up, para subir.

Configurando o PgAdmin:

→ Conectar no pgadmin local host e criar um novo server, para colocar o docker e tals

→
Para entrar dentro do conteiner: `docker-compose exec postgres sh` , e depois de `hostname -i`

> 💡 Até aqui, deve ser capaz de o banco e docker estarem rodando, com os dados la

ORM - GORM

Faça os downloads necessários, tanto do GORM, como da sua versão do postgree.

Crie um arquivo db.go

```
package database

import "gorm.io/gorm"

var (
    DB *gorm.DB
    err error
)

func ConectaComBancoDeDados() {
        stringDeConexao := "host=localhost user=root passwo
rd=root dbname=root port=5432 sslmode=disable "
        DB, err = gorm.Open(postgres.Open(stringDeConexao))
        if err != nil{
            log.panic(err)
        }
}
```

```
Utilizacao do GORM:
```

```
database.DB.Find(&p) //Retorna um select *
database.DB.Find(&p,id) //Retorna um select * where, basead
o no valor de id

// --- INSERT através do Post, para isso não esquecer de na
rota definir o method como
//Post

func insertDado (w http.ResponseWriter , r *http.Request) {
    var varaivel tipoVariavel
    json.NewDecoder(r.Body).Decode(&variavel) //Ou seja uti
lizamos o Decoder para receber coisas, para isso usando as
informações da request
    database.DB.Create(&variavel)
    json.NewEncoder(w).Encode(variavel)
}

// --- DELETE, usar o metodo Delete !!
database.DB.Delete(&personalidade, id)

// --- UPDATE, usar metodo PUT
func updateDado (w http.ResponseWriter , r *http.Request) {
    //Codigo para pegar o id da URL
    //importante primeiro utilizar o first, para encontrar
o que quer alterar
    database.DB.Fist(&variavel, id)
    json.NewDecoder(r).Decoder(&variavel)
    database.DB.Save(&variavel)
    json.NewEncoder(w).Encode(variavel)
}
```

Para adicionar um header na resposta da requisição basta utilizar o **hearder,** utilizando um método do ResponseWriter → `w.Header().Set("Content-type","application/json")`

MiddleWare

É criado uma função ContentTypeMiddleware e um novo arquivo no pacote middleware, esta função irá ter a seguinte assinatura:

```
package middleware

import "net/http"

    func ContentTypeMiddleware(next http.Handler) http.Hand
ler {
        return http.HandlerFunc(func(w ResponseWriter, r *h
ttp.Request){
            w.Header().Set("Content-Type","application/jso
n")
            next.ServeHTTP(w,r)
        })

    }
```

Após a criação do middleware, é preciso utilizar ele em main, assim que declara o router

`r.Use(middleware.ContentTypeMiddleWare)`