[00:00] Olá, meu nome é **Guilherme Lima** e eu queria te dar as boas-vindas neste treinamento de **API REST** utilizando a linguagem **GO**. Neste curso vamos criar uma API REST utilizando o GO para guardar **personalidades**.

[00:15] Como assim personalidades? Na minha rua, na minha cidade, existem vários nomes de pessoas que eu não faço a mínima ideia de quem são e eu quero guardar essas informações. Eu coloquei dois nomes aqui que vocês estão vendo na tela de nomes de ruas da minha cidade. Eu coloco o nome dessa personalidade e um pouco da história dela.

**Deodato Petit Wertheimer**: Deodato Petit Wertheimer foi um médico e político brasileiro, seus primeiros anos de vida foram em São Paulo, mas logo mudou para Nova Friburgo no Estado do Rio de Janeiro e com 11 anos de idade ingressou no Colégio Anchieta dos jesuítas.

**Carmela Dutra**: Carmela Teles Leite Dutra foi a primeira-dama do Brasil, de 31 de janeiro de 1946 até a sua morte, tendo sido a esposa de Eurico Gaspar Dutra, 16º Presidente do Brasil. Era, carinhosamente, chamada de Dona Santinha, pela sua forte religiosidade, fazendo seu marido abrir uma capelinha no Palácio Guanabara.

[00:37] Para criar essa aplicação vamos criar um API REST do zero utilizando o GO. Vamos desde os princípios, como retornamos uma **requisição com JSON**, como fazemos o roteamento desses endpoints, como conectamos com banco de dados.

[00:55] Um ponto legal neste curso é que vamos usar o banco de dados no **Docker**, vamos conectar a nossa API REST consumindo um banco de dados no Docker. Vamos usar o RM do GO para conseguir conectar com o banco de dados e manipular mais fácil as nossas requisições sem precisar ficar escrevendo **SQL** a mão.

[01:15] Vamos criar o **CRUD** completo, vamos ser capazes de criar recursos, deletar, editar, atualizar. Serão muitas coisas legais. Além disso, vamos pensar em como deixamos a nossa API com um nível bom de estrutura, informando qual vai ser o Content-Type que estamos devolvendo, vamos criar middlewares também e vamos integrar a nossa API com o **React**.

[01:40] Ao final do curso vamos entender, depois que criamos todo nosso esqueleto, a nossa API como integramos o nosso back-end GO com um front-end React, que é o que vocês estão vendo aqui. Tanto que temos o símbolo do GO e o símbolo do React girando aqui em cima.

[01:56] Esse curso é indicado para as pessoas que já fizeram os treinamentos de GO da Alura e vão dar continuidade. Se você nunca fez nenhum treinamento de GO eu aconselho que você faça primeiro os cursos de GO descritos nessa formação e depois você siga para esse curso de API REST.

[02:14] Se você já tem familiaridade com outra linguagem de programação e você quer desenvolver uma API utilizando o GO, você é bem-vindo nesse curso também. Esse curso não é indicado para as pessoas que já tem uma maturidade muito grande na linguagem GO, se você já trabalha com GO, você já tem uma maturidade muito avançada, você já domina esse tema esse conteúdo, talvez esse curso não seja indicado para você.

[02:36] Sabendo disso vamos começar a nossa API de personalidades.

[00:00] Vamos iniciar os nossos estudos. Antes, para começarmos um projeto GO colocávamos esse projeto na **GOPATH**, eu vou abri-la para vocês verem. Com base nos nossos cursos anteriores eu vou abrir aqui, tínhamos no meu usuário uma pasta chamada “GO” que tinham três arquivos e criamos esses arquivos dentro de GitHub, começamos nosso projeto dentro desse GOPATH.

[00:25] Hoje existe uma forma de conseguirmos iniciar o nosso projeto em qualquer pasta e eu vou mostrar isso para vocês. Como vamos iniciar o projeto do zero eu vou clicar com o botão direito do *mouse* e vou clicar em "Nova pasta". Eu vou chamar essa pasta de “api-go-rest”, essa vai ser a pasta do nosso desenvolvimento.

[00:47] Vou usar o **Visual Studio Code**, como editor de código e vou arrastar essa “api-rest” aqui para o **VS Code**. Ele vai abrir e agora eu vou usar o terminal integrado do VS Code para executarmos o código que vai nos permitir que rodamos qualquer projeto GO sem que estejamos alocados na GOPATH.

[01:13] Vou utilizar as teclas "Ctrl + J" ou “Command + J" se você estiver utilizando MAC. O comando será assim: go mod init e agora vou colocar o comando init e vou dar um nome para esse projeto que vamos criar. Geralmente vinculamos esse projeto a algum repositório que temos lá no GitHub para hospedarmos o nosso código, essa é uma ideia.

[01:39] Você poderia colocar aqui link do GitHub, por exemplo: github.com/guilhermeonrails que é o meu usuário do GitHub, e eu coloco o nome desse projeto, voo colocar: go-rest-api, por exemplo. api-go-rest >>> go mod inint github.com/guilhermeonrails/go-rest-api.

[01:53] Se você não tem um repositório que você queira vincular com o código que vamos desenvolver aqui, se você não tem o GitHub, não tem problema você pode colocar o nome que você quiser. Por exemplo: só go-rest-api.

[02:05] Eu vou deixar esse nome mesmo, vou dar um "Enter" e ele fala que foi criado do go.mod. Vamos ver o que isso significa, quando abrimos a nossa pasta temos esse arquivo “go.mod” e ele fala qual é o módulo que estamos utilizando e onde todos os nossos pacotes e bibliotecas relacionadas ao GO estarão e ele também fala qual a versão GO estamos utilizando.

[02:29] Dessa forma conseguimos iniciar o nosso projeto GO sem que ele seja vinculado com a GOPATH. Para testarmos isso de fato eu vou criar um arquivo chamado “main.go” e nesse arquivo eu vou falar qual é o pacote que ele pertence package main, coisas que nós já vimos nos cursos anteriores de Orientação a Objetos, de GO com Web.

[02:52] Falei qual é o pacote que ele pertence. Vou criar uma função que vou chamar de func main() {. Observe que estou digitando e as coisas já estão acontecendo, já estão aparecendo aqui. Isso acontece porque estou usando uma extensão que é mantida pelo próprio GOogle.

[03:08] Se eu digitar aqui só go e der um "Ente", nesse bloco aqui de extensões ele fala que esse primeiro módulo estou usando, tanto que ele está até habilitado. Com ele conseguimos ter vários atalhos, tudo isso que eu digitei aqui, o package main e a func main(), eu poderia ter feito assim: pkgm e quando eu dou um "Enter" ele coloca qual é o pacote *main* e coloca a função também.

[03:31] Isso vai auxiliar no nosso desenvolvimento, vamos conseguir podar de forma mais rápida também. Se você não tem essa extensão eu recomendo que você instale para o VS Code.

[03:42] Nesse treinamento eu faço bastante menção do VS Code, eu sugiro até que você utilize esse mesmo editor que eu. Se você usa o **Atom** ou outro editor, você pode procurar também essas extensões do GO para o seu editor que você estiver usando.

[03:58] Criei aqui o “main.go” que por enquanto não faz nada e vou exibir uma mensagem simples: fnt.Println e repare que quando eu dei o "Enter" ele já trouxe aqui e fez o *import* para mim. Se não apareceu o *import* você pode digitá-lo import “fnt”. Vou colocar: ("Iniciando o servidor Rest com GO"). Claro que essa é uma mensagem fictícia não estamos iniciando neste caso o servidor com GO, vou rodar esse nosso projeto com o go run main.go e quando eu dou "Enter" ele fala: "Iniciando o servidor Rest com GO". Exibiu a mensagem e deu tudo certo.

package main

import "fmt"

func main() {

fmt.PrintIn("Iniciando o servidor Rest com Go")

}COPIAR CÓDIGO

[04:40] Só que só essa mensagem é um pouco simples, vamos abrir uma página usando outros pacotes e bibliotecas que precisamos. Vou criar uma função que vou chamar de Home() que vai representar a nossa página inicial. func Home(w http.ResponseWritter, r \*http.Request). E como já vimos no curso de Web, sempre quando vamos criar uma nova função com o GO relacionada a abrir uma página, precisamos receber a requisição e responder utilizamos dois atalhos que são esses: *ResponseWriter* e o *Request*. O w para o *responsewriter* e o r apontando para o *request* que vamos utilizar.

[05:24] E vou colocar uma mensagem só não vou fazer nada mais que isso. fnt.Fprint(w, “Home Page”) porque vamos executar uma função que vamos passar o nosso *ResponseWriter*, responsável por escrever na tela, e aqui vou passar uma mensagem vou colocar, por exemplo, "Home Page".

[05:41] Quando eu salvo isso repare que ele trouxe para mim esse outro módulo "net/http", que será responsável por receber as requisições. É um módulo que vai nos auxiliar nessa parte de quando receber um requisição nesse endereço faça tal coisa e mensagem, ele vai auxiliar nessa conversa de cliente servidor também.

package main

import (

"fmt"

"net/http"

)

func Home(w http.ResponseWriter, r \*http.Request) {

fmt.Fprint(w, "Home Page")

}

func main() {

fmt.PrintIn("Iniciando o servidor Rest com Go")

}COPIAR CÓDIGO

[06:06] Vamos executar isso daqui, vamos colocar essa função Home() para funcionar. Para isso eu vou criar outra função que vai ser responsável por todas as rotas que temos na nossa aplicação. Quando navegamos e entramos em um site esse site tem várias páginas, cada página tem sua respectiva função que vai falar: "quando carregar essa página você vai abrir isso, isso e isso".

[06:31] Essa função temos o hábito de chamar no GO de func HandleResquest(). “*Handle*” em inglês significa “lidar”, *HandleResquest* é lidar com as requisições, algo desse tipo. Eu vou colocar aqui usando o módulo que já temos que é o: http.HandleFunc(), eu vou falar assim: "existe uma função que tem esse endereço e vai ser atendida por alguém."

[06:58] Sempre que chegar uma requisição no barra "/", ou seja, quando colocarmos o endereço da nossa página sem nada, sem "/compras" ou "/produtos" eu quero que ele caia nessa página específica e quem vai ser responsável por atendê-lo será a nossa função Home(), só isso, http.HandleFunc(“/”, Home).

[07:15] Agora aqui no nosso arquivo “main” eu vou executar esse nosso HandleResquest() executando ele aqui. Vamos relembrar o que fizemos: criamos uma função chamada Home() e falamos que toda vez que chegar uma requisição em Home() você vai responder exibindo essa mensagem "Home Page". Depois criamos alguém para lidar com essas requisições, quando chegar uma requisição em barra quem vai atender será o controle Home.

[07:41] Na nossa função main() falamos assim: "HandleResquest, executa", ele vai ficar "ouvindo" isso daqui. Só que se ele vai estar ouvindo o nosso servidor precisa estar funcionando, vou subir o nosso servidor aqui também. Vou usar a função log.Fatal(), que significa que qualquer problema que eu tenha quando eu subir o meu servidor ele vai nos apontar, vai mostrar para nós.

[08:02] E vou usar o mesmo pacote log.Fatal(http.ListenAndServe()), ou seja, escuta e sirva, escute e suba e servidor e vou passar aqui qual é a porta que eu quero subir. Vou subir no local *host* na porta (“:8000”, nil) mesmo, vou colocar vírgula aqui porque não temos nenhum *handler* para configurarmos por isso vou deixar como *nil*. log.Fatal(http.ListenAndServe(":8000", nil)). Não temos nenhuma configuração extra que precisaremos fazer.

package main

import (

"fmt"

"net/http"

)

func Home(w http.ResponseWriter, r \*http.Request) {

fmt.print(w, "Home Page")

}

func HandleRequest() {

http.HandleFunc("/", Home)

log.Fatal(http.ListenAndServe(":8000", nil))

}

func main() {

fmt.PrintIn("Iniciando o servidor Rest com Go")

HandleResquest()

}COPIAR CÓDIGO

[08:29] Vou salvar esse nosso projeto, vou abrir aqui o nosso terminal utilizando as teclas "Ctrl + J" ou “Command + J” e vamos executar esse código: go run main.go. E ele vai dar uma mensagem: "Iniciando o servidor Rest com GO" e parou. Eu vou abrir uma aba anônima e vou colocar aqui nesse endereço “localhost:8000” e temos aqui a página *Home Page*.

[08:57] Até aí está tranquilo só que não queremos exibir uma página com um texto específico, queremos que quando chegue uma determinada requisição para um endereço específico queremos responder com dados, de preferência com JSON que é o padrão mais utilizado atualmente. Isso vamos aprender na sequência.

[00:00] Subimos a nossa primeira página Home só que estou preocupado com algo, observe que esse código que fizemos aqui é simples e está com 21 linhas. Vamos relembrar os passos que demos: criamos uma função Home() que quando bater uma requisição para essa função Home() vamos exibir uma mensagem retornando o *ResponseWriter* e a mensagem será *home page*.

[00:26] Depois temos alguém que só toma conta das requisições HandleRequest(), quando chegar alguém aqui nesse *endpoint* específico, nesse endereço específico quem vai atender vai ser o Home.

package main

import (

"fmt"

"net/http"

)

func Home(w http.ResponseWriter, r \*http.Request) {

fmt.print(w, "Home Page")

}

func HandleRequest() {

http.HandleFunc("/", Home)

log.Fatal(http.ListenAndServe(":8000", nil))

}

func main() {

fmt.PrintIn("Iniciando o servidor Rest com Go")

HandleResquest()

}COPIAR CÓDIGO

[00:38] Já conseguimos quebrar isso, por que? Podíamos colocar todo esse conteúdo aqui em um arquivo específico, em uma pasta específica porque a medida que o nosso projeto for aumentando temos as divisões certinhas, já sabemos exatamente onde precisamos alterar no nosso código.

[00:54] Imagine que estamos com uma página só exibindo um texto *home page* e estamos com 21 linhas, à medida que o nosso projeto for aumentar muito, tivermos diversos *endpoints*, rotas, *controllers* e depois modelos isso vai ficar muito difícil manter. Por isso vamos modularizar o nosso código.

[01:12] A primeira coisa que vou fazer vai ser criar uma pasta chamada *routes*. Vou criar aqui clicando nesse ícone aqui, um novo arquivo, um *new file* que vou chamar de “routes”. Aqui dentro eu vou criar um arquivo chamado “routes.go”, esse arquivo eu vou falar qual é o pacote que ele pertence que é o pacote de rotas.

[01:32] E eu vou manter nesse código de rotas esse nossa HandleResquest(). Vou tirar HandleResquest() da pasta “main.go” e vou colocá-lo aqui na pasta “routes.go” utilizando as teclas "Ctrl + C" e "Ctrl + V" e quando eu salvo ele já faz os *imports* certinhos para mim.

[01:45] Só que existe algo estranho aqui, ele fala que fiz os *imports* só que existe uma variável que está declarada e não sei o que ela é, que é essa Home. O que é Home? Ele não consegue enxergar o que está escrito aqui no nosso código “main.go”, que é a nossa função Home(). O que vamos fazer?

[02:02] Esse Home a funcionalidade dele é quando chegar uma requisição aí que lá no nosso arquivo de “routes.go” vai ser o responsável, quem vai controlar essa página vai ser essa nossa função.

[02:15] Podemos colocar isso em uma pasta chamada "controllers". Vou colocar aqui uma pasta chama “controllers” e vou criar um arquivo chamado “controllers.go”. Esse arquivo *controllers* eu vou colocar justamente esse nosso código Home, utilizo as teclas “Ctrl + X" para tirá-lo daqui e antes de colocá-lo aqui no “controller.go” vamos falar que package controllers, coloca aqui a nossa função, salvo e ela já está certinha.

package controllers

import (

"fmt"

"net/http"

)

func Home(w http.ResponseWriter, r \*http.Request) {

fmt.Fprint(w, "Home Page")

}COPIAR CÓDIGO

[02:43] Lá nas rotas eu vou falar que quando chegar uma requisição para barra quem vai atender vai ser alguém que está lá nos *controllers* e aqui eu tenho várias opções. Como a nossa função chama-se Home(), será http.HandlerFunc(“/”, controllers.Home e quando eu salvo esse projeto ele já traz o *import* certinho.

package routes

import (

"log"

"net/http"

"github.com/guilhermeonrails/go-rest-api/controllers"

)

func HandleResquest() {

http.HandleFunc("/", controllers.Home)

log.Fatal(http.ListenAndServe(":8000", nil))

}COPIAR CÓDIGO

[03:13] Esse *import* é único, é o nome que vai estar linkado com o projeto que você iniciou lá no GoMod. Vamos supor que você colocou o nome do seu projeto só go-rest-api ele vai trazer só: go-rest-api/controllers. Se você colocou um repositório do GitHub ele vai trazer o nome completo para você também.

[03:33] Colocamos o Home() no “controllers.go”, ele fez os *imports* necessários do Home(), de todos os modos que precisamos para esse nosso projeto funcionar e ele já consegue identificar. No “main.go” temos um único problema, ele sabe o que é HandleResquest(), vamos falar para ele que o HandleResquest() está no nosso arquivo de rotas, é ele quem vai lidar com todas as nossas requisições.

[03:57] Ao colocar routes.HandleResquest() ele já aparece o que eu quero. Eu salvo e ele já está tudo funcionando. Ele sabe que ele precisa executar essa função HandleResquest() lá do nosso módulo de rotas.

[04:16] Vamos ver se isso está funcionando. Eu vou abrir o nosso terminal e rodar o nosso servidor, vou colocar aqui go run main.go ele vai falar: "iniciando o servidor rest com go", vou atualizar e está funcionando.

[04:33] Olha o que fizemos agora: temos um arquivo *main* com 13 linhas.

package main

import (

"fmt"

"github.com/guilhermeonrails/go-rest-api/routes"

)

func main() {

fmt.Println("Iniciando o servidor Rest com Go")

routes.HandleResquest()

}COPIAR CÓDIGO

[04:41] Temos um arquivo só para *controllers*.

package controllers

import (

"fmt"

"net/http"

)

func Home(w http.ResponseWriter, r \*http.Request) {

fmt.Fprint(w, "Home Page")

}COPIAR CÓDIGO

[04:43] E um arquivo para manter só as nossas rotas.

package routes

import (

"log"

"net/http"

"github.com/guilhermeonrails/go-rest-api/controllers"

)

func HandleResquest() {

http.HandleFunc("/", controllers.Home)

log.Fatal(http.ListenAndServe(":8000", nil))

}COPIAR CÓDIGO

[04:45] Se surgir uma nova rota colocamos qual será o *endpoint* dessa rota. Só colocamos aqui o *endpoint* dessa nova rota e depois vincula com o *controller* que vamos criar e o nosso código ficará mais fácil para manter e organizar.