Instituto Federal de Educação, Ciência e Tecnologia de São Paulo

Golang: a linguagem do futuro

Cubatão

2021

Arthur Carvalho Simões
Alisson de Sousa Vieira
Everson Pereira da Silva

Golang: a linguagem do futuro

TCC apresentado no ensino médio integrado ao ensino Técnico em Informática do IFSP.

Cubatão
2021

SUMÁRIO

História

A Linguagem Go, começou a ser desenvolvida dentro dos escritórios do Google, com a finalidade de ser algo mais simples de se programar e também por conta do descontentamento com os recursos de sistemas que existiam no dado momento. Ken Thompson, Rob Pike e Robert Griessemer foram os engenheiros principais responsáveis pela criação e pelo desenvolvimento da linguagem.

Ken Thompson

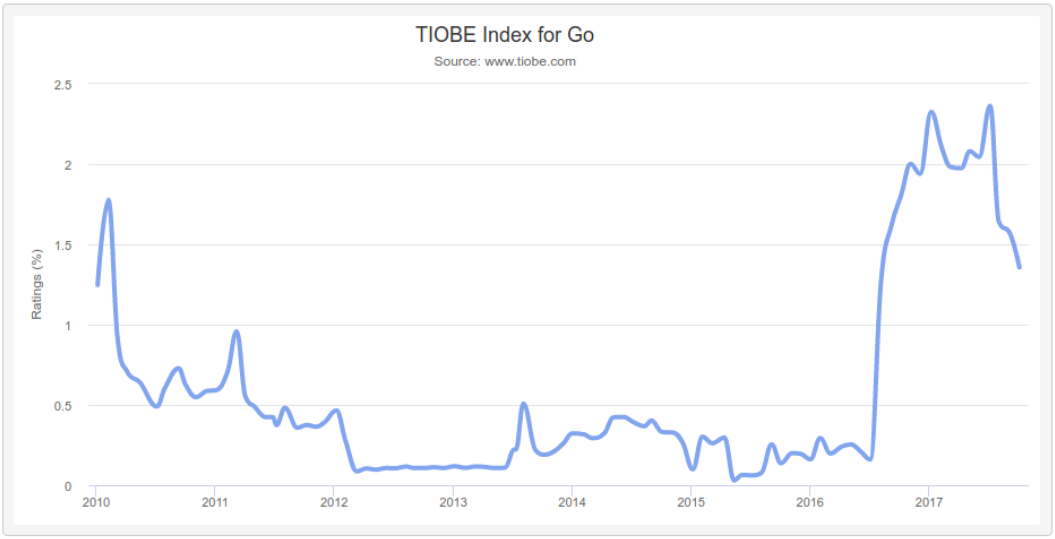
Rob Pike

Robert Griessemer

Com os seus trabalhos se iniciando em 2007, como um projeto interno da empresa, logo em Novembro de 2009, a linguagem já foi lançada em código aberto. Uma linguagem que logo fez muito sucesso e se tornou referência no mercado, pela sua popularidade do Google e em grandes projetos como Uber, Twitter e Dropbox, entre outros no período de 2016. Além de tudo isso, trata-se de uma linguagem leve e rápida, uma linguagem super otimizada para diversas tarefas, desde as simples até as complexas do cotidiano.

A linguagem de programação Go, também conhecida como "golang do Google", tem cada vez mais aumentado sua popularidade nos últimos anos. Isso se deve, principalmente, pela linguagem ser de código aberto e leve, adequando-se a diversas situações, como por exemplo em serviços de portes menores.

É perceptível o crescimento da linguagem, principalmente desde 2016, tendo como base o índice TIOBE, onde hoje em dia a linguagem Go ocupa a posição número 16 na lista mensal, bem diferente da posição 65, que era a ocupada há 5 anos atrás.



Principais características da linguagem Go:

Uma das características que mais chamam a atenção nesta linguagem é a sua performance, pois, por ela ser compilada em linguagem de máquina, torna o processo muito mais rápido.

Ele aceita programação funcional, suporta funções anônimas e as de primeira classe. Tem uma interface legível e fácil de se programar, apresenta diversos recursos e funcionalidades e é, como dito anteriormente, uma linguagem em código aberto.

Além de todas essas características, a linguagem Go está crescendo cada vez mais no mercado, e com certeza você não vai querer ficar de fora. Então vamos aprender Go?!

As IDEs mais populares para programar em Go:

- Lite IDE

LiteIDE é um Go IDE simples e de código aberto. É notável por ser o primeiro IDE a ser direcionado diretamente ao voltar em 2012. É um C ++ Qt, o que significa que se parece e se sente semelhante a outros compiladores como Visual Studio e GCC C ++.

Uma vez que foi projetado diretamente para Golang, LiteIDE tem uma série de recursos úteis para desenvolvedores diretamente da caixa, incluindo comandos de construção configuráveis, um editor de código avançado e amplo suporte Golang. Outros recursos incluem gerenciamento de código, um depurador gdb e Delve, preenchimento automático e temas com WordApi, sistema baseado em tipo MIME e muito mais.

- Visual Studio Code

O IDE de código aberto sempre popular da Microsoft tem uma extensão Go disponível para Visual Studio Code. O plug-in vscode-go oferece aos desenvolvedores ainda mais recursos, incluindo integração com várias ferramentas Go.

O VS Code oferece preenchimento inteligente com IntelliSense, integração Git embutida, a capacidade de depurar código direto do editor e muito mais. O VS Code é altamente extensível, com várias opções de personalização por meio de suas muitas extensões. Ele também oferece suporte em dezenas de idiomas.

- Eclipse

Com o plug-in GoClipse, os desenvolvedores podem utilizar o popular Eclipse IDE para sua programação. Tanto o IDE Eclipse quanto o plug-in GoClipse são gratuitos e de código aberto.

Os editores GoClipse fornecem aos desenvolvedores uma ampla gama de recursos, incluindo um editor de código-fonte, um assistente de projeto e construtor para ajudar a relatar erros de construção no editor e um suporte a depurador GDB completo.

- IDE Atom

Os desenvolvedores podem aproveitar as vantagens da integração de linguagem aprimorada deste IDE com um editor mais inteligente. O pacote go-plus de código aberto torna ainda mais fácil para os desenvolvedores codificar em Go.

O Atom e o pacote go-plus oferecem suporte Golang para as ferramentas, fluxos de construção, linters, veterinário e ferramentas de cobertura. Outras funcionalidades incluem preenchimento automático, formatação, teste e documentação. Funcionalidades de depuração adicionais podem ser adicionadas com o pacote go-debug com delve.

- VIM

VI Improved possui vários plug-ins para ajudar os desenvolvedores a editar seu código Go ainda mais facilmente. O plugin vim-go instala automaticamente todos os bits e bobs necessários para fornecer uma integração mais suave para desenvolvedores Go no Vim.

O Vim-go apresenta uma série de bits úteis, incluindo um compilador, destaque de sintaxe aprimorado e dobramento, suporte de conclusão e um monte de programas de depuração com suporte delve integrado. Há também uma série de ferramentas de análise de fonte avançados que utilizam guru, incluindo :GoImplements, :GoCalleese e :GoReferrers.

Outros plug-ins do vim incluem um plug- in Syntastic para feedback sobre erros do compilador, um plug- in tagbar para Gotags, um plug-in do compilador vim para verificação de sintaxe e até mesmo um vim-bootstrap para gerar uma configuração .virmrc.

Essas estão entre as principais IDEs para a programação em Go. Para o desenvolvimento da nossa apostila, utilizaremos o Visual Studio Code, porque além de estar entre as IDEs mais populares, temos mais facilidade com ela por já termos a usado em outras situações.

A seguir aprenda e confira o passo a passo para a instalação do VS Code.

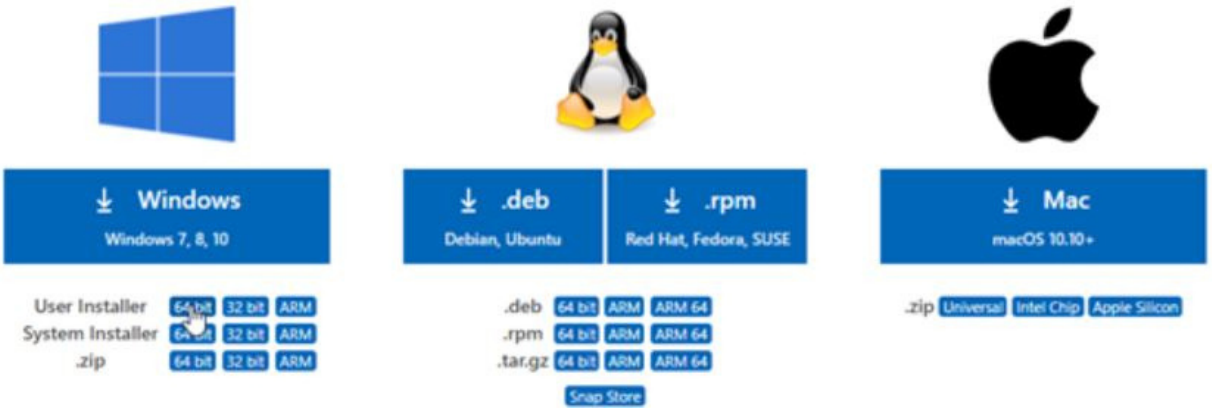
Baixando e instalando a IDE
passo a passo

Nesta apostila utilizaremos a IDE Visual Studio Code, dada sua grande qualidade. É um processo extremamente simples e você fará com facilidade, siga os passos!

Primeiro, iremos no link de download da IDE.
Clique aqui → https://code.visualstudio.com/Download

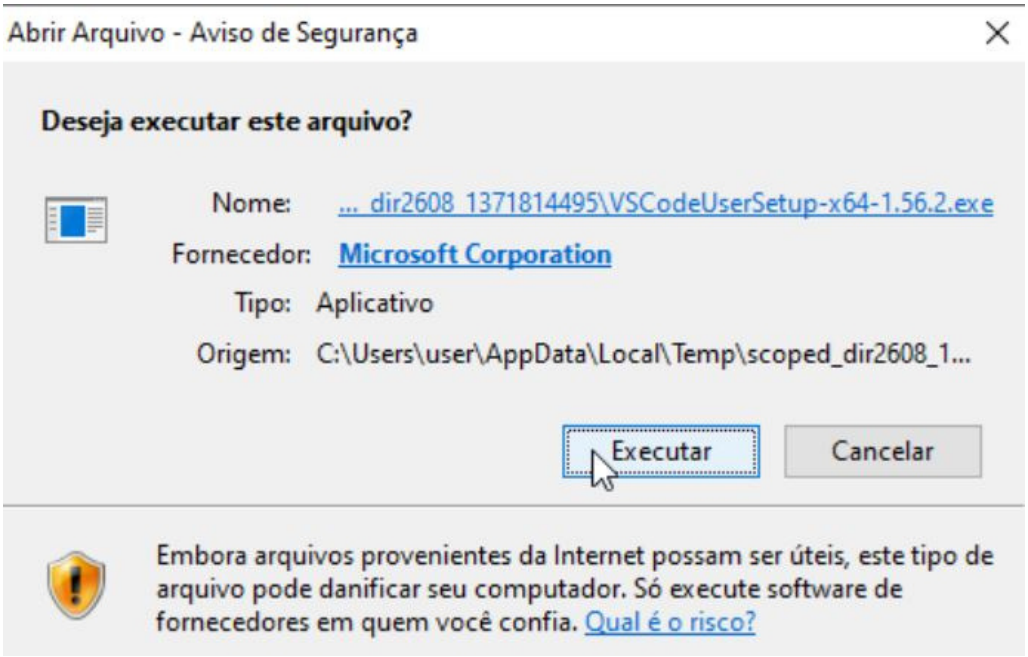
Após isso, você escolherá sua opção de download, conforme seu sistema operacional e a arquitetura (64 bits, 32 bits, ARM...)

No nosso caso de exemplo, estamos usando uma máquina com o Windows 10 e arquitetura 64 bits.



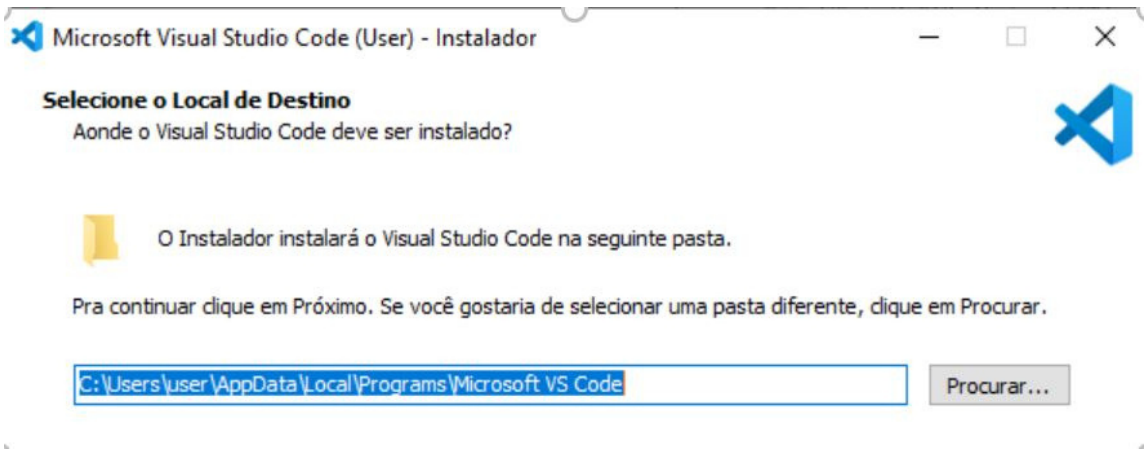
Depois de concluir o download, nós abriremos o arquivo.

Aparecerá esta janela e você deve clicar em "executar".

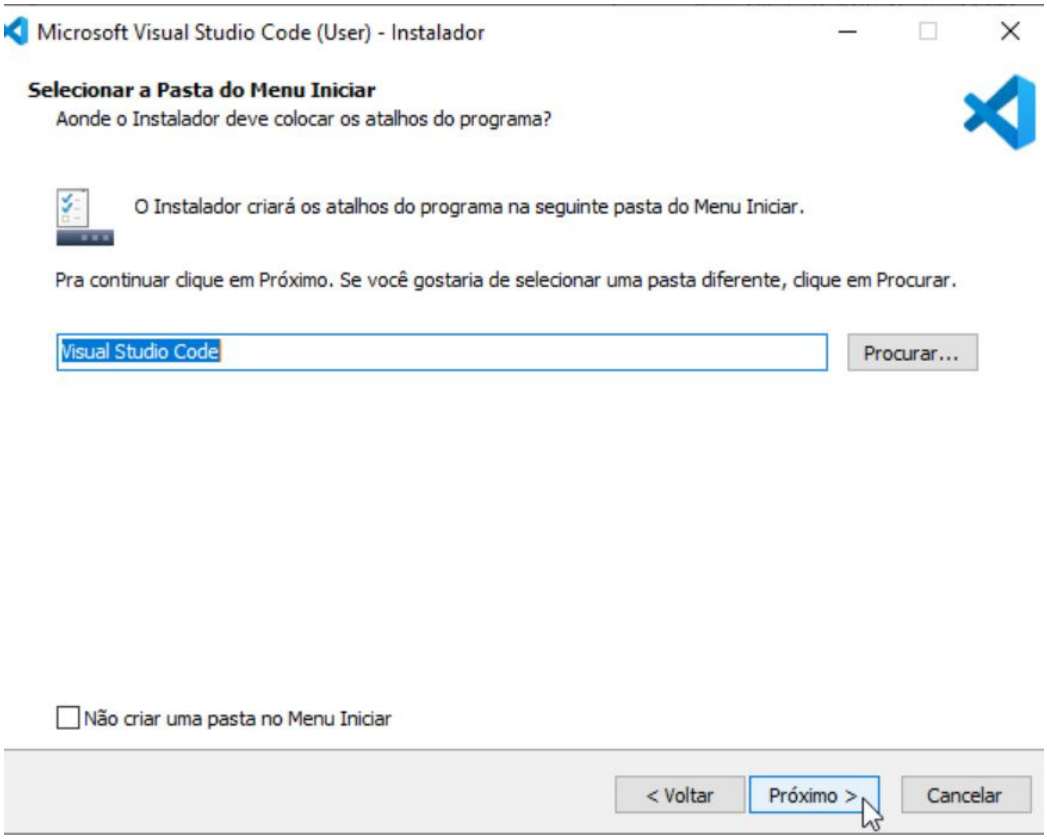


Aqui, para prosseguir, será necessário que você concorde com os termos de licença. Assim que aceitos, você poderá clicar em "próximo" para continuar.

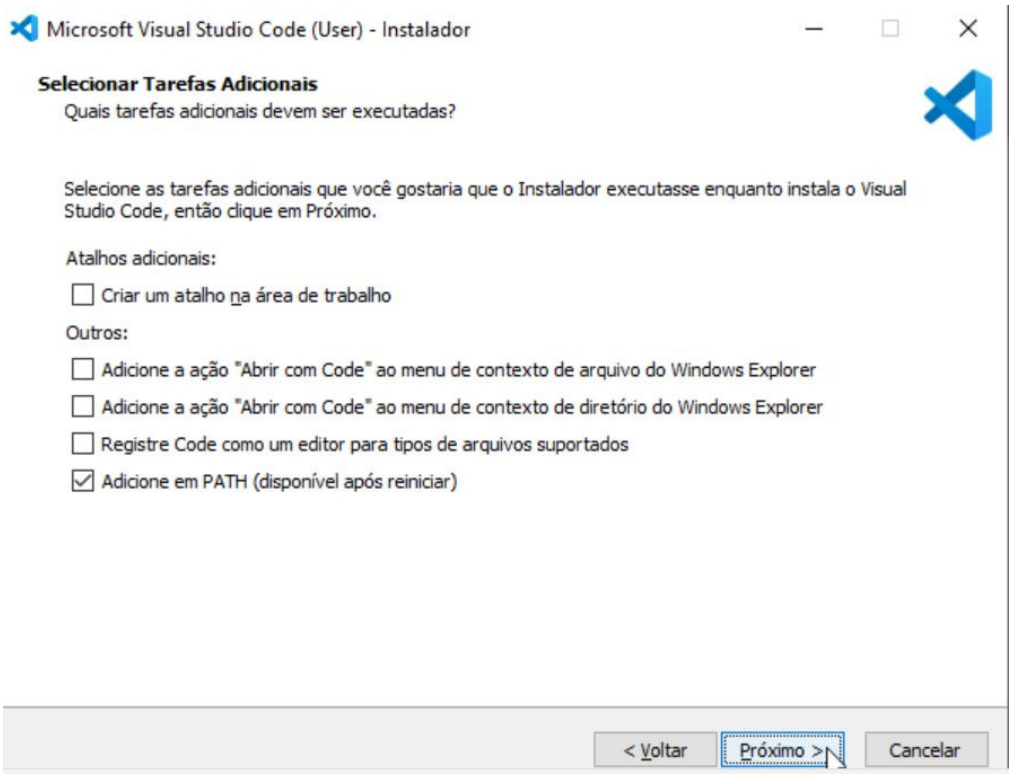
Agora você deverá escolher o local a ser instalado a IDE, ou se preferir, deixar o local padrão de instalação. No nosso exemplo, deixamos no local padrão. Depois, clique em "próximo".



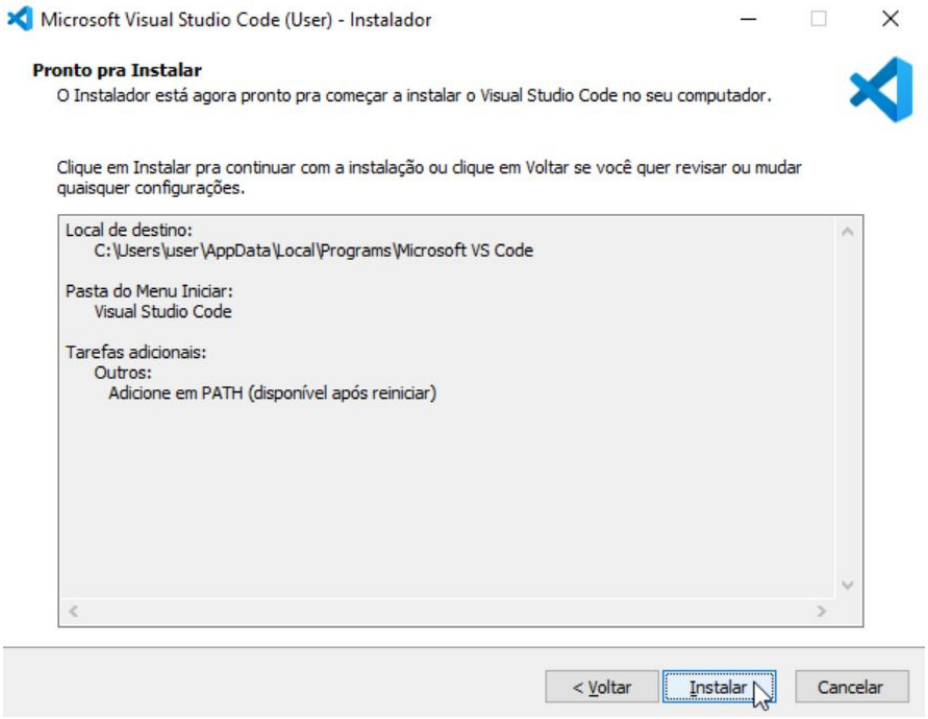
Caso você queira escolher um local específico do Menu Iniciar para os atalhos do programa serem alocados, você pode alterar clicando em "procurar…" e selecionando o local. Se não, apenas clique em "próximo". Obs: há, também, a opção de não criar uma pasta no Menu Iniciar, você só deve clicar no quadrado ao lado da opção e marcá-lo.



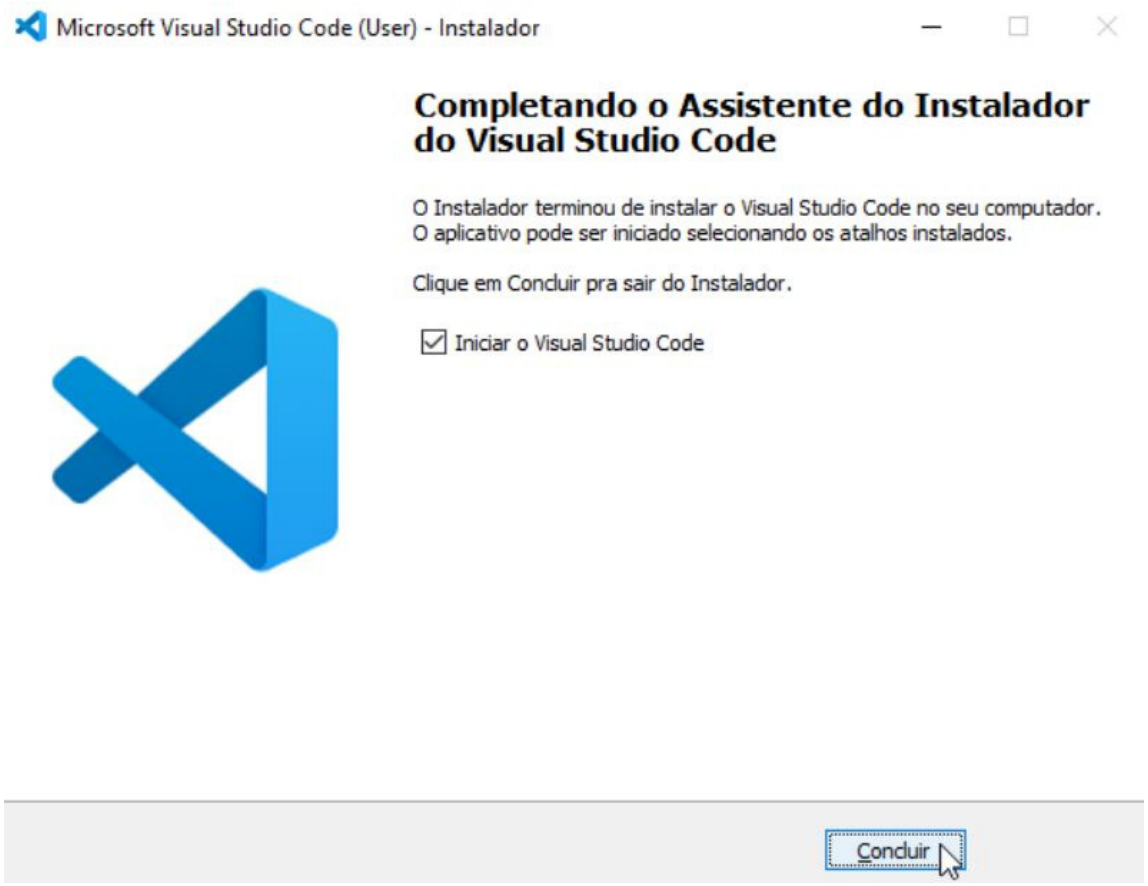
O Visual Studio Code na sua instalação nos permite algumas opções adicionais, que vão do gosto de cada usuário. Você pode apenas prosseguir aqui, clicando em "próximo".



Agora, clique em "instalar" e a instalação se iniciará.



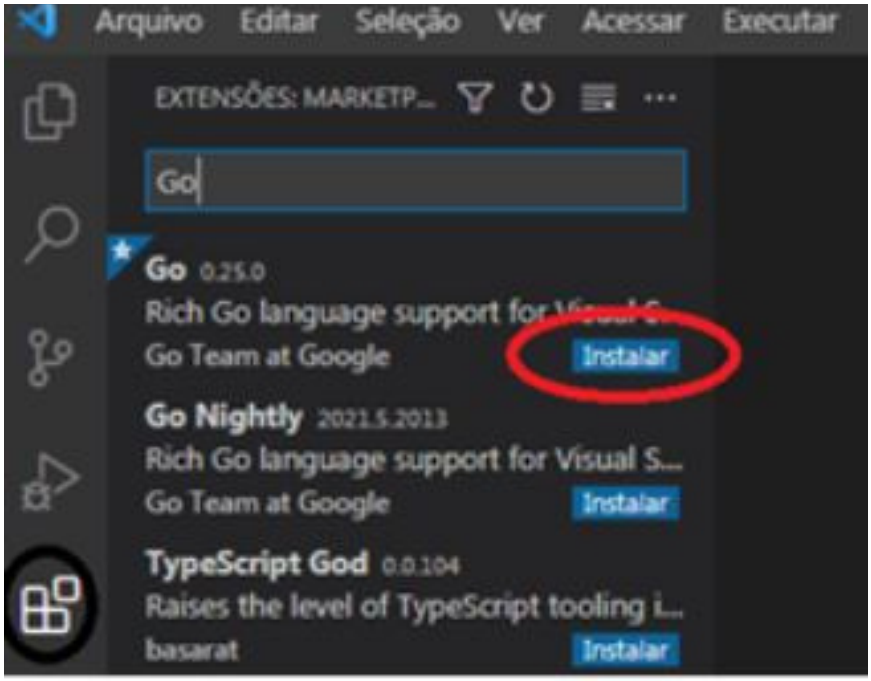
Logo após a instalação, essa janela aparecerá, e você pode marcar a opção "Iniciar o Visual Studio Code", clicar em "concluir" e já começar a programar! Ou, caso preferir, pode deixar a opção desmarcada.



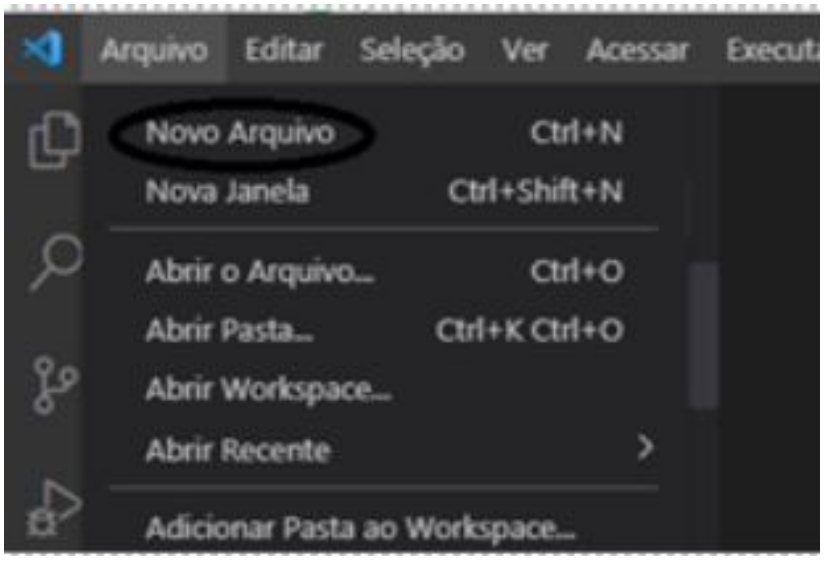
Fazendo o nosso primeiro programa "Hello World"

Após instalar o Visual Studio Code, para fazer o seu primeiro programa, siga os seguintes passos. Primeiro, instale o pacote de extensão para a linguagem Go.

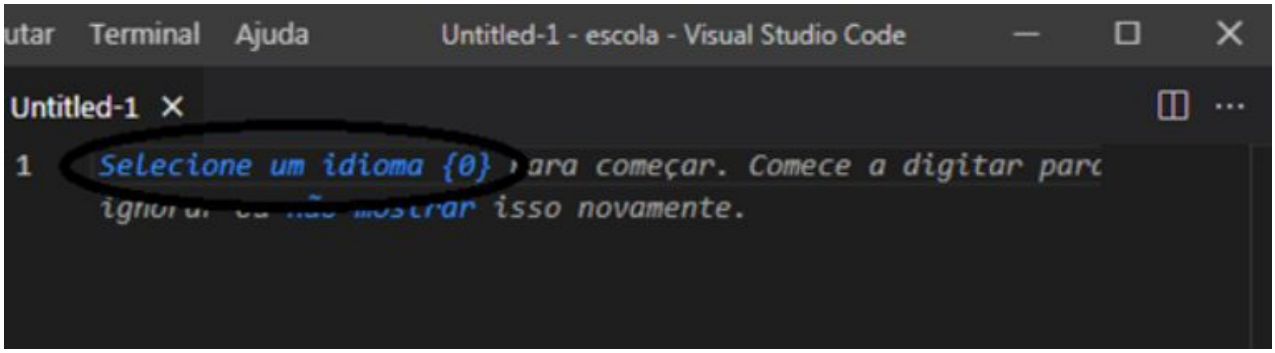
Para isso, execute o VSCODE. Após executá-lo, vá na aba de extensões, depois clique na barra de pesquisa e digite "go", após isso, clique em instalar.



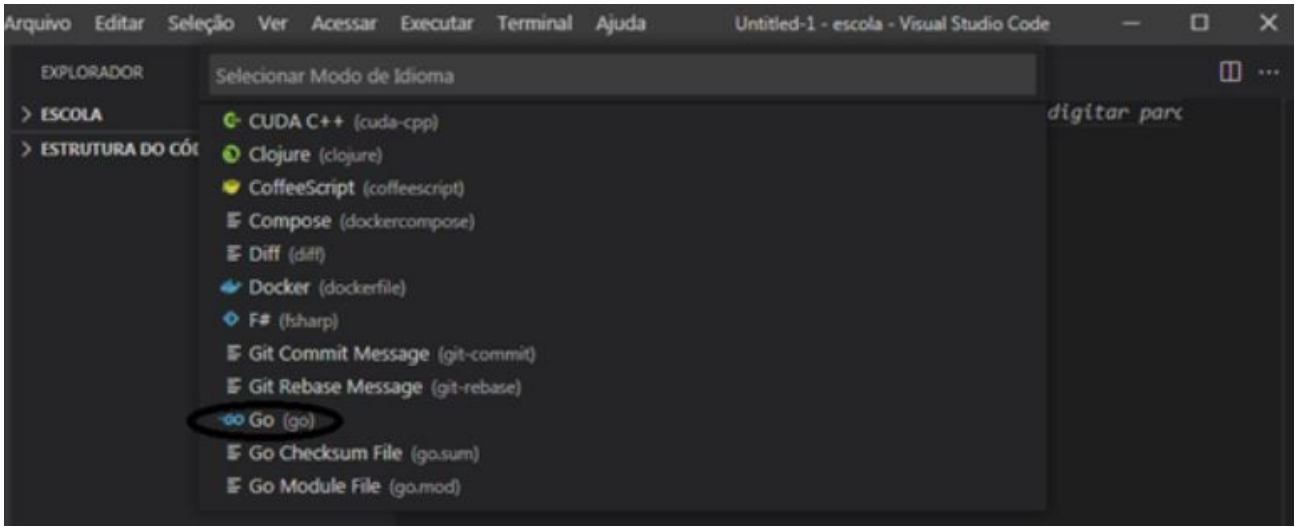
Após instalar a extensão necessária para programar na linguagem Go, crie um novo arquivo, conforme a imagem.



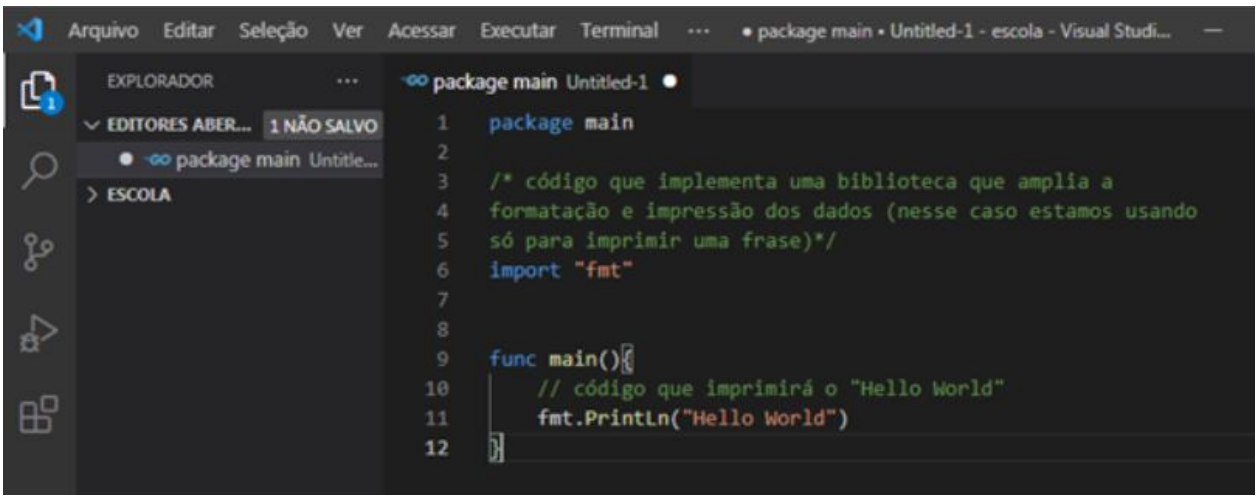
Vá em "Selecione um idioma".



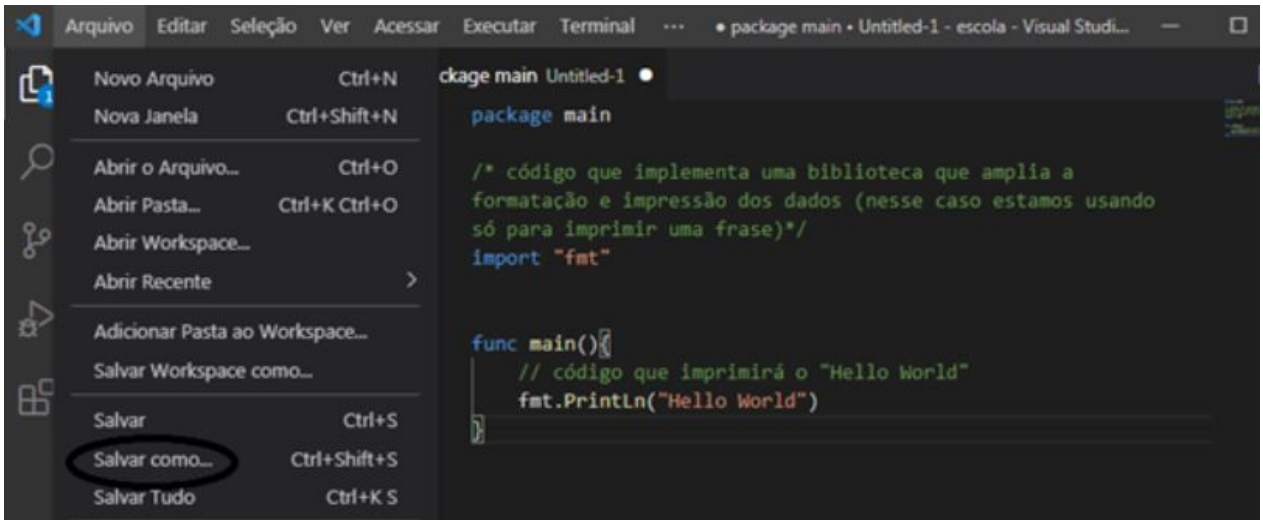
Procure por "Go" na barra de pesquisa ou apenas encontre e clique na linguagem Go, como fizemos.



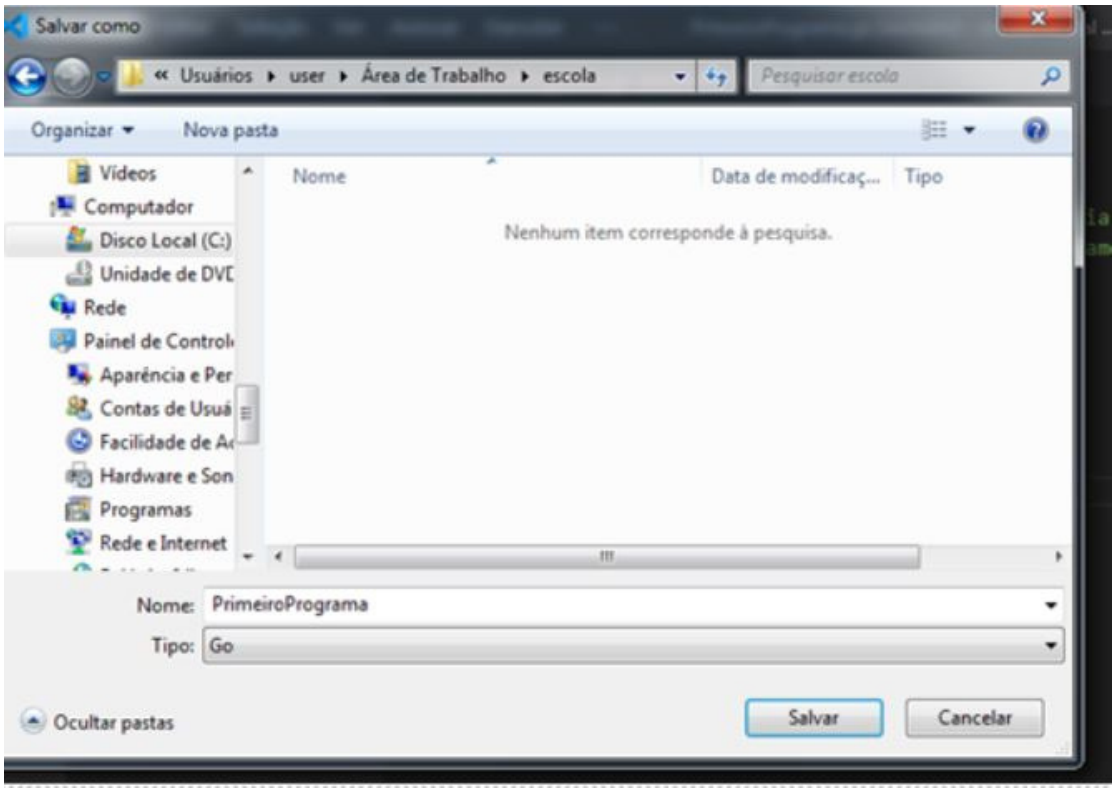
Depois é só inserir o código. Para nosso o primeiro programa, como de costume quando se está aprendendo uma nova linguagem, vamos exibir o famoso "Hello World". Confira logo abaixo a implementação do código.



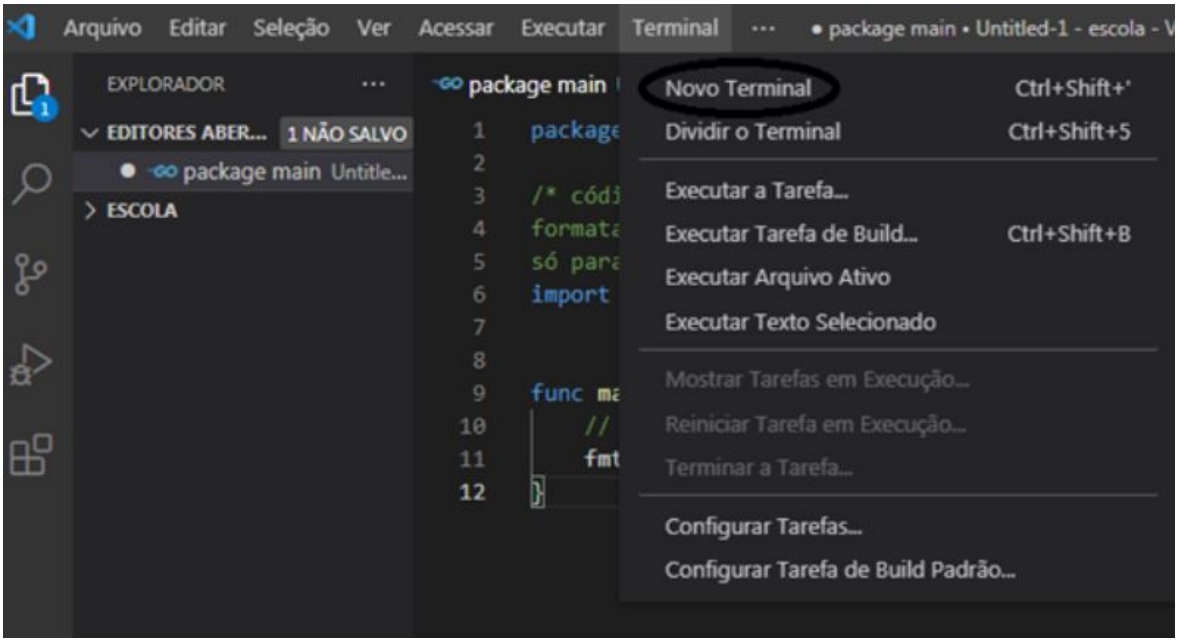
Depois de escrever o programa, agora precisamos salvar, para isso clicamos em "arquivo" e logo, vamos em "salvar como".



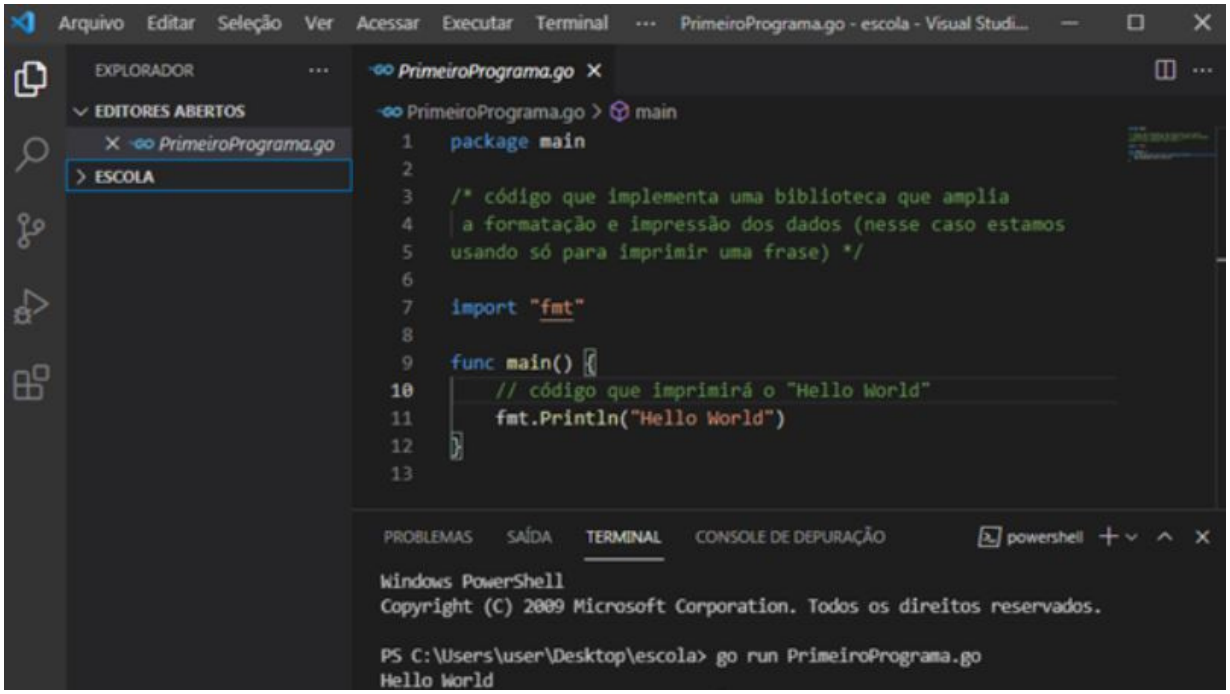
Especificamos o caminho que queremos salvar o arquivo e o nome dele, no nosso caso salvamos em "C:user/users/desktop/escola" com o nome de "PrimeiroPrograma".



Após o salvamento, o próximo passo é executar o programa, então clicamos em "novo terminal"



Agora é só digitar no terminal "run go + 'caminho do arquivo onde foi salvo/NomeDoArquivo'.go". Como no terminal já estamos em C:user/users/escola, basta digitar "run go PrimeiroPrograma.go". E finalmente, "Hello World" imprimido!

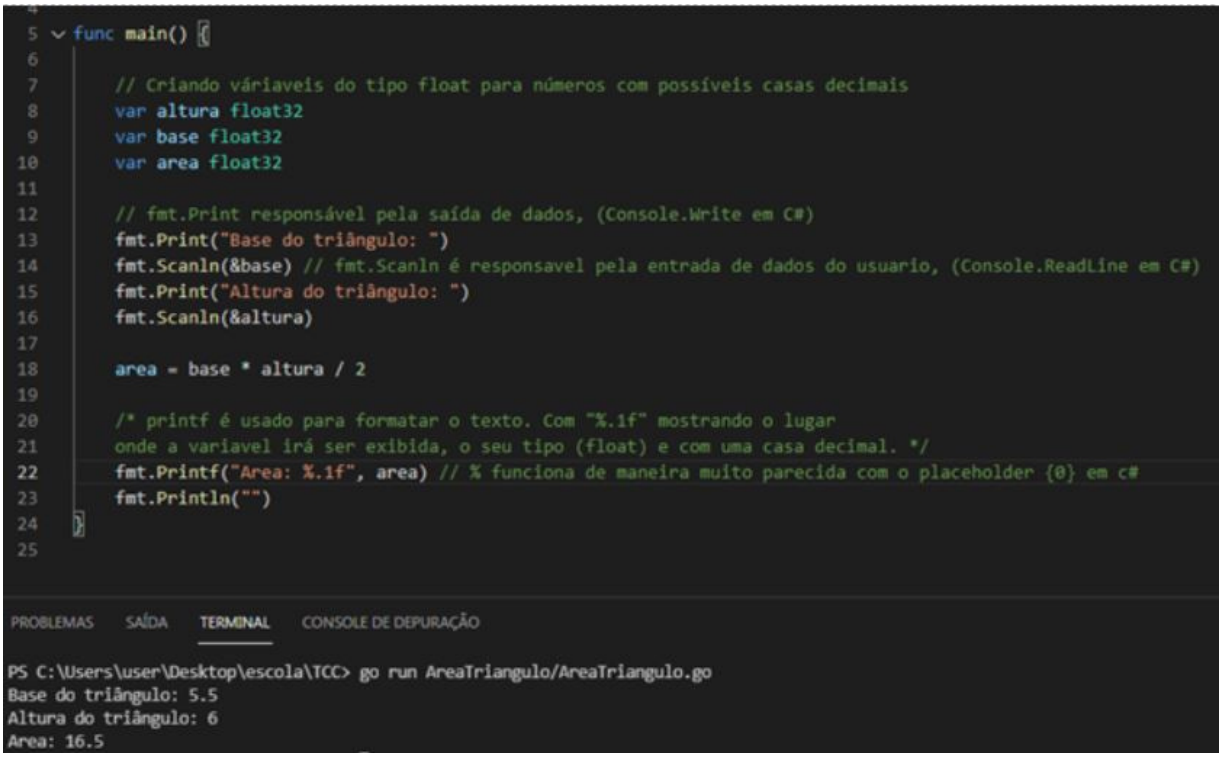


Golang em prática

As primeiras aplicações: entrada e saída, operações aritméticas, funções matemáticas, decisão lógica, laços de repetição e arrays.

- Calcular a área do triângulo

No nosso primeiro programa, iremos, a partir da digitação da base e altura de um triângulo, calcular sua área e exibi-la no monitor.



Além do comando de entrada de dados fmt.Scanln, também utiliza-se o fmt.Scan e o fmt.Scanf. A diferença do fmt.Scanln para o fmt.Scan é que, o último citado, irá receber o que o usuário digitar e não pulará uma linha, portanto a próxima instrução será exibida na mesma linha onde o usuário digitou. Já o fmt.Scanf possibilita a formatação do que será digitado, seguindo a mesma lógica do printf. Este comando não funciona no sistema operacional windows.

Quanto aos comandos de saída de dados, o único que não foi explicado foi o fmt.Println. Este exibirá um texto e pulará uma linha, seguindo a mesma lógica do fmt.Scanln.

- Calcular seno e cosseno de um ângulo

Nesse programa será calculado o seno e o cosseno de um ângulo em graus, utilizando funções matemáticas que o pacote “math” nos disponibiliza.

CossenoeSeno > CossenoSeno.go

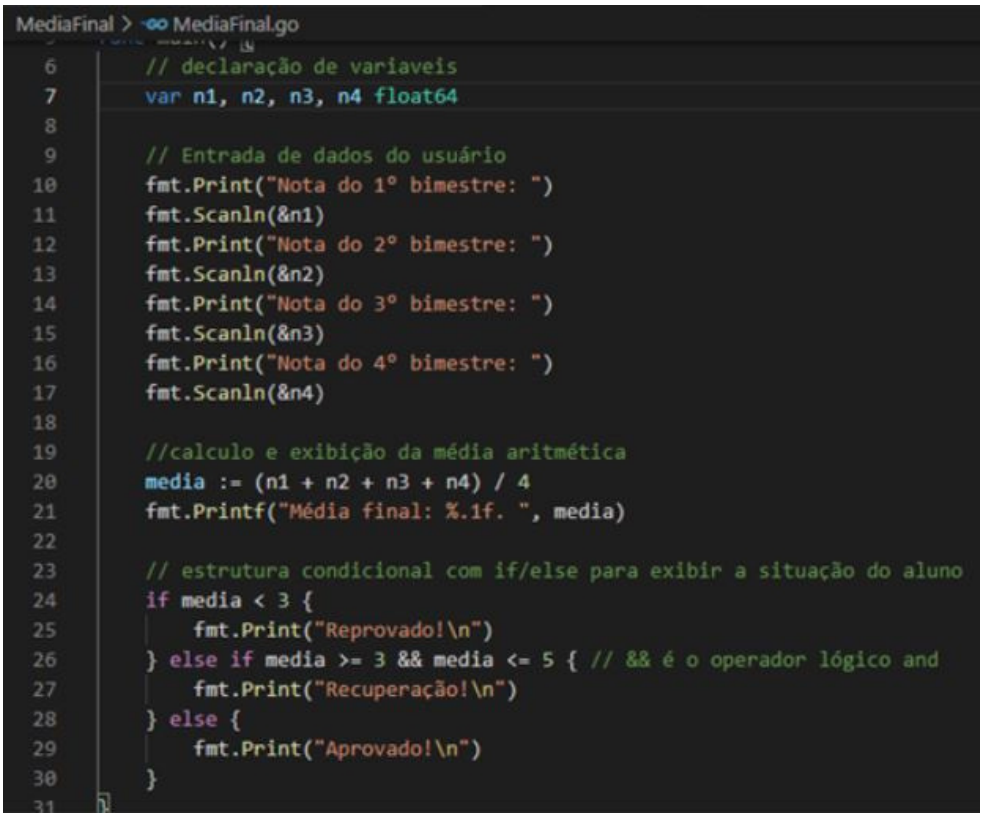
```go
import (
    "fmt"
    "math"
)

func main() {
    // definição de variavel inteira (int)
    var grau int

    var rad float64

    fmt.Print("Angulo em º: ")
    fmt.Scanln(&grau)

    /* math.PI, math.sin e math.cos são funções matemáticas
    que retornam repectivamente o PI(3.14), seno em rads e cosseno rads. */
    rad = float64(grau) * math.Pi / 180
    cosseno := math.Cos(float64(rad))
    seno := math.Sin(float64(rad))

    fmt.Printf("cos: %.2f\nsen: %.2f\n", cosseno, seno)
}
```

As funções math.Sin e math.Cos retornam valores em radianos, por isso foi preciso a conversão para graus. Além do seno, cosseno e PI, com a função math também é possível calcular logaritmos (math.Log), raiz quadrada (math.Sqrt), potência (math.Pow) e muitas outras coisas ligadas à matemática.

Nesses programas já é notável a presença de operadores aritméticos, mas não todos, portanto é preciso ressaltar-los. Na linguagem GO utilizamos 7, estes são:
+ (soma), - (subtração), * (multiplicação), / (divisão), % (resto da divisão), ++ (incremento) e -- (decremento).

- Calcular média aritmética final

Nesse programa será calculado a média aritmética final de um aluno considerando as 4 notas bimestrais. Após o cálculo, o programa deverá exibir se o aluno está aprovado, reprovado ou de recuperação. Para aprovado sua nota terá que ser maior ou igual à 6. De recuperação, a nota terá que estar entre 3 e 6. E será reprovado quem estiver com a nota menor que 3.



Aqui observamos a presença de operadores relacionais, que são importantes para as estruturas de condição e repetição, esses verificam se tal condição é verdadeira ou falsa. A linguagem GO conta com 6 operadores relacionais, estes: > (maior), < (menor), >= (maior ou igual), <= (menor ou igual), == (comparação) e != (diferente).

Também vemos a presença do && no código, este é um operador lógico, que atua no controle para dizer se a condição ao todo é verdadeira ou falsa. Contamos com && (and), || (or), != (negação).

- Calcular com base no operador aritmético digitado

O programa solicita a digitação de dois números e um caractere, sendo que este poderá ser “+”, “-”, “*” ou “/”. Mediante o caractere digitado o programa realizará o respectivo cálculo e exibirá o resultado, se o caractere não corresponder a nenhum dos 4 caracteres em questão, será exibido uma mensagem de erro. Este programa foi produzido usando switch-case e if/else para mostrar à vocês como usar estrutura condicional usando GO.

Com if/else:

```go
func main() {
    // Declaração de variaveis.
    var vlr1, vlr2 float64
    var op string

    // Entrada do usuario com os 2 valores.
    fmt.Print("1° Valor: ")
    fmt.Scanln(&vlr1)
    fmt.Print("2° Valor: ")
    fmt.Scanln(&vlr2)
    println("")

    // Entrada do usuario com um dos operadores matemáticos (+, -, *, /).
    fmt.Print(" *  ->  multiplicar\n /  ->  dividir\n +  ->  somar\n -  ->  subtrair\n\nInsira o operador lógico: ")
    fmt.Scanln(&op)

    // Estrutura condicional com if/else para executar o calculo com base no operador matemático escolhido.
    if op == "+" {
        fmt.Print(vlr1, " + ", vlr2, " = ", vlr1+vlr2, "\n")
    } else if op == "-" {
        fmt.Print(vlr1, " - ", vlr2, " = ", vlr1-vlr2, "\n")
    } else if op == "*" {
        fmt.Print(vlr1, " * ", vlr2, " = ", vlr1*vlr2, "\n")
    } else if op == "/" {
        fmt.Print(vlr1, " / ", vlr2, " = ", vlr1/vlr2, "\n")
    } else {
        fmt.Print("Erro! Você digitou o caracter incorreto.\n")
```

Com switch-case:

```go
func main() {
    // Declaração de variaveis.
    var vlr1, vlr2 float64
    var op string

    // Entrada do usuario com os 2 valores.
    fmt.Print("1° Valor: ")
    fmt.Scanln(&vlr1)
    fmt.Print("2° Valor: ")
    fmt.Scanln(&vlr2)
    println("")

    // Entrada do usuario com um dos operadores matemáticos (+, -, *, /).
    fmt.Print(" *  ->  multiplicar\n /  ->  dividir\n +  ->  somar\n -  ->  subtrair\n\nInsira o operador lógico: ")
    fmt.Scanln(&op)

    // Estrutura condicional com switch/case para executar o calculo com base no operador matemático escolhido.
    switch op {
    case "+":
        fmt.Print(vlr1, " + ", vlr2, " = ", vlr1+vlr2, "\n")
    case "-":
        fmt.Print(vlr1, " - ", vlr2, " = ", vlr1-vlr2, "\n")
    case "*":
        fmt.Print(vlr1, " * ", vlr2, " = ", vlr1*vlr2, "\n")
    case "/":
        fmt.Print(vlr1, " / ", vlr2, " = ", vlr1/vlr2, "\n")
    default:
        fmt.Print("Erro! Você digitou o caracter incorreto.\n")
    }
}
```

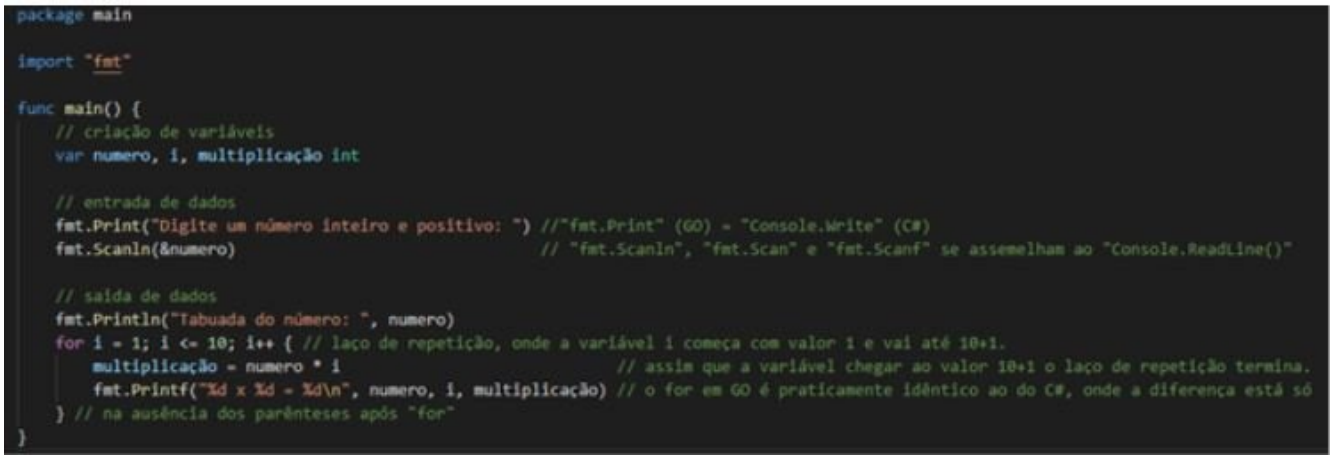
Por conter 5 situações, o mais indicado é usar o switch-case, porém vai de sua preferência, pois o resultado será o mesmo.

```
1° Valor: 4
2° Valor: 15

 *  ->  multiplicar
 /  ->  dividir
 +  ->  somar
 -  ->  subtrair

Insira o operador lógico: *
4 * 15 = 60
```

Laços de repetição e arrays.

- Exibir a tabuada do número digitado

Neste programa iremos inserir um novo conceito: laços de repetição. A ideia é digitar um número inteiro e positivo e o programa nos retornará a tabuada do mesmo.



```go
package main

import "fmt"

func main() {
    // criação de variáveis
    var numero, i, multiplicação int

    // entrada de dados
    fmt.Print("Digite um número inteiro e positivo: ") //"fmt.Print" (GO) - "Console.Write" (C#)
    fmt.Scanln(&numero)                               // "fmt.Scanln", "fmt.Scan" e "fmt.Scanf" se assemelham ao "Console.ReadLine()"

    // saída de dados
    fmt.Println("Tabuada do número: ", numero)
    for i = 1; i <= 10; i++ { // laço de repetição, onde a variável i começa com valor 1 e vai até 10+1.
        multiplicação = numero * i                    // assim que a variável chegar ao valor 10+1 o laço de repetição termina.
        fmt.Printf("%d x %d = %d\n", numero, i, multiplicação) // o for em GO é praticamente idêntico ao do C#, onde a diferença está só
    } // na ausência dos parênteses após "for"
}
```

Neste programa nos deparamos com o for, o único construtor de looping de Go. No exemplo acima utilizamos a forma mais clássica, onde temos o início de uma variável (i = 1), uma condição (i <= 10) e o final, que seria o código entre as chaves ({ }) e, também a incrementação (i++). Utilizamos dessa incrementação para variar o número utilizado como condição e tornar possível o fim desse laço.

- Exibir os "n" primeiros termos de uma série

Neste programa o usuário digitará um número que determinará a quantidade de termos que aparecerão referentes à série ($x^2 + 1$), onde x = {1, 2, 3, 4, 5…}.

```go
package main

import (
    "fmt"
    "math"
)

func main() {
    // criação de variáveis
    var n, i int
    var x float64 = 1

    // entrada de dados
    fmt.Print("Digite os n primeiros termos da série: ") //"fmt.Print" (GO) - "Console.Write" (C#)
    fmt.Scan(&n)                                         // "fmt.Scanln", "fmt.Scan" e "fmt.Scanf" se assemelham ao "Console.ReadLine()"

    // saída de dados
    for i = 1; i <= n; i++ {
        if i != n {
            fmt.Printf("%0.f, ", math.Pow(x, 2)+1) // aqui, usamos o Printf, onde tudo é formatado para string.
        } else { //  "%0.f" se refere à variável float que é retornada de "math.Pow(x, 2)+1".
            fmt.Printf("%0.f", math.Pow(x, 2)+1) //   "0." é uma limitação de casas depois da vírgula.
        }
        x++

    }
}
```

Neste programa começamos importando a biblioteca "math", para podermos, posteriormente, usar a função "Math.Pow(x,y)", que retorna x**y (x elevado a y).
Após digitarmos o número, o programa usará esse valor como base para criar um laço de repetição, onde uma variável i começará valendo 1 e vai até o valor n + 1, para ser possível o fim do laço. Dentro desse laço, fazemos um if, onde enquanto i for diferente de n, será impresso ao lado do número uma vírgula e um espaço. Quando i for igual a n, o número será impresso sem o espaço e a vírgula, pois será o último número da série.

Execução:

```
Digite os n primeiros termos da série: 4
2, 5, 10, 17
```

- Listar os números da série de Fibonacci menores que 1000

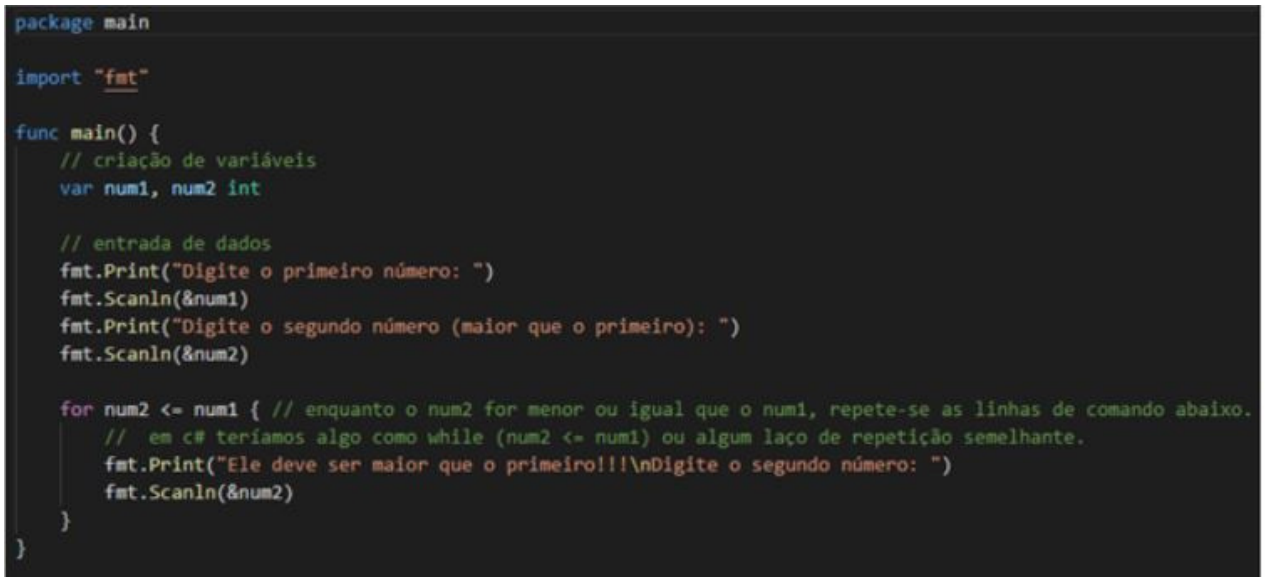
Série de Fibonacci é uma sequência de números na qual 1 é o primeiro e o segundo termo, e os demais são resultados da soma de seus dois antecessores.

```go
package main

import "fmt"

func main() {
    // criação de variáveis
    var num1, num2, aux int
    num1 = 0
    num2 = 1

    // laço de repetição, onde enquanto num2 for menor ou igual a 1000, continuará ocorrendo.
    for num2 <= 1000 {
        aux = num1
        num1 = num2
        num2 = num1 + aux
        if num2 < 1000 { // para assegurarmos que só teremos os números menores que 1000, criamos este if.
            fmt.Println(num2)
        }
    }
}
```

Neste programa usamos o for de uma maneira diferente: usando apenas uma condição. O "for num2 <= 1000" significa que enquanto a variável num2 for menor ou igual a 1000, as linhas de códigos dentro das chaves serão executadas.

- Entrar com dois valores, onde o segundo deverá obrigatoriamente ser maior que o primeiro.

Como já diz o título, faremos um programa que o segundo valor inserido deverá obrigatoriamente ser maior que o primeiro!



```go
package main

import "fmt"

func main() {
    // criação de variáveis
    var num1, num2 int

    // entrada de dados
    fmt.Print("Digite o primeiro número: ")
    fmt.Scanln(&num1)
    fmt.Print("Digite o segundo número (maior que o primeiro): ")
    fmt.Scanln(&num2)

    for num2 <= num1 { // enquanto o num2 for menor ou igual que o num1, repete-se as linhas de comando abaixo.
        //  em c# teríamos algo como while (num2 <= num1) ou algum laço de repetição semelhante.
        fmt.Print("Ele deve ser maior que o primeiro!!!\nDigite o segundo número: ")
        fmt.Scanln(&num2)
    }
}
```

É um programa muito simples de ser feito, seguimos com o mesmo uso de for do programa anterior. Entramos com dois valores e, caso o segundo for menor ou igual ao primeiro, você deverá inserir novamente o valor.

- Entrar com seu sexo, F ou M, obrigatoriamente

Outro programa com um modelo muito similar ao anterior, mas agora tendo limitações mais específicas, sendo estas "f", "F", "m" e "M".

```go
import "fmt"

func main() {
    // criação de variáveis
    var sexo string

    // entrada de dados
    fmt.Print("Digite o seu sexo (F ou M): ")
    fmt.Scanln(&sexo)

    for sexo != "f" && sexo != "F" && sexo != "m" && sexo != "M" { // enquanto valor de sexo for diferente de "f", "F", "m" e "M", o laço
        // se repetirá.
        fmt.Print("Sexo inválido!\nDigite o seu sexo: ")
        fmt.Scanln(&sexo)
    }
```

Utilizamos "&&" com o sentido de "e", ou seja, deve-se cumprir todas as exigências requeridas.

- Fatorial de um número obrigatoriamente inteiro e positivo

Neste programa será requerido um número inteiro e positivo, e, sendo este inteiro e positivo, será exibido seu fatorial. O fatorial de um número é calculado pela multiplicação desse número por todos os seus antecessores até chegar ao número 1. Note que nesses produtos, o zero (0) é excluído.

```go
package main

import "fmt"

func main() {
    // criação de variáveis
    var resposta string = "S"
    var numero, fatorial, i int

    for resposta == "S" { // laço de repetição

        // entrada de dados
        fmt.Print("\nDigite um número inteiro e positivo: ")
        fmt.Scanln(&numero)

        for numero < 0 { // enquanto o valor do número dado for menor que zero, o usuário terá de digitar novamente.
            fmt.Print("Erro!\nDigite um número inteiro e positivo: ")
            fmt.Scanln(&numero)
        }

        if numero == 0 { // o fatorial de "0" é 1.
            fmt.Printf("O fatorial deste número é: 1")
        }

        fatorial = numero
        //Saída de dados
        for i = numero - 1; i >= 1; i-- { // este laço fará com que o fatorial do número será numero * (numero-1)... até numero * 1.
            fatorial = fatorial * i // este laço só ocorrerá caso o número seja inteiro, positivo e diferente de 0.
            // caso o número seja 1, será numero * numero, pois será numero * 1.
        }
        fmt.Printf("O fatorial de %d é %d", numero, fatorial)

        // Verificação se deseja continuar
        fmt.Print("\n\n Você deseja continuar? (Apenas S ou N serão aceitos como resposta!): ")
        fmt.Scanln(&resposta) //se a resposta for "S", o laço de repetição que inicia o programa é reiniciado.
        // se for "N", o programa acaba.

        for resposta != "S" && resposta != "N" { //se a resposta for diferente de "S" e "N", aparecerá uma mensagem de erro e
            fmt.Print("\n\n Erro!\nVocê deseja continuar?(Apenas S ou N serão aceitos como resposta!) ") // pedirá novamente.
            fmt.Scanln(&resposta)
        }
    }
```

Neste programa utilizamos uma variável para tornar possível a saída dele. Note que foi criada uma variável chamada resposta com o valor de "S". Enquanto este valor for "S", o programa continuará rodando.

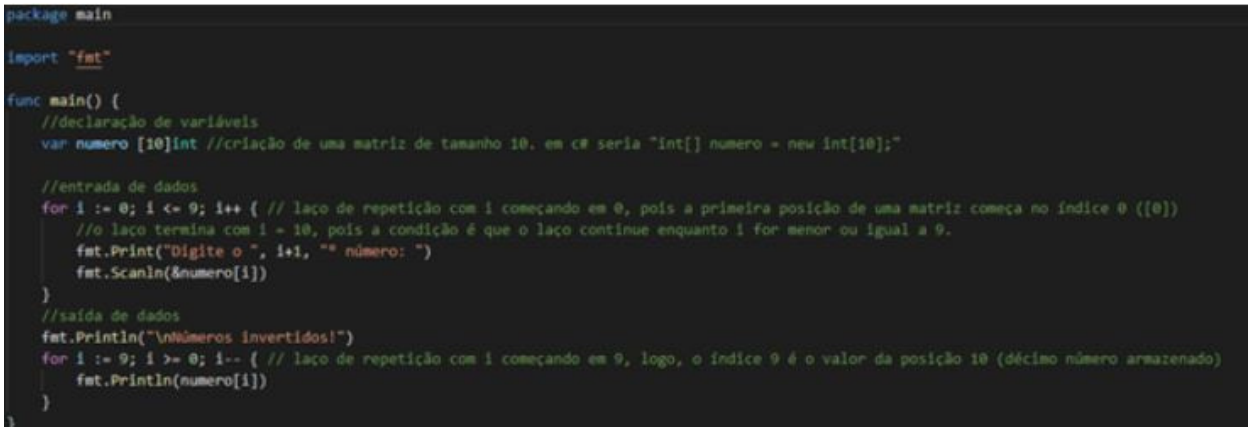
Fazemos também o uso de ifs para quando o número digitado for 0, retornar automaticamente 1, visto que o fatorial de 0 é 1, mas é uma excessão à regra.

Para termos o fatorial de um número, fazemos um laço onde temos uma variável chamada i, que tem o valor do número digitado - 1 e ela vai até 1. Por exemplo: Número digitado = 5 e i = 4. O laço fará 5*4*3*2*1 e retornará o fatorial de 5.

```
Digite um número inteiro e positivo: 5
O fatorial de 5 é 120

 Você deseja continuar? (Apenas S ou N serão aceitos como resposta!):
```

- Armazenar 10 números na memória e exibi-los na ordem inversa.

Agora começaremos a trabalhar com variáveis indexadas, também conhecidos como arrays, vetores e também matrizes. A ideia é que a variável tenha espaços dentro dela, onde por exemplo uma matriz " var numero [10]int " possa armazenar 10 valores nela. Neste programa digitaremos 10 números e eles serão impressos na ordem inversa.



```go
package main

import "fmt"

func main() {
    //declaração de variáveis
    var numero [10]int //criação de uma matriz de tamanho 10. em c# seria "int[] numero = new int[10];"

    //entrada de dados
    for i := 0; i <= 9; i++ { // laço de repetição com i começando em 0, pois a primeira posição de uma matriz começa no índice 0 ([0])
        //o laço termina com i = 10, pois a condição é que o laço continue enquanto i for menor ou igual a 9.
        fmt.Print("Digite o ", i+1, "º número: ")
        fmt.Scanln(&numero[i])
    }
    //saída de dados
    fmt.Println("\nNúmeros invertidos!")
    for i := 9; i >= 0; i-- { // laço de repetição com i começando em 9, logo, o índice 9 é o valor da posição 10 (décimo número armazenado)
        fmt.Println(numero[i])
    }
}
```

Tudo é efetuado de maneira simples. A única coisa a se atentar é que uma matriz começa por [0] e não por [1]. Por isso nesse exemplo o for começa com i = 0. Pois quando i for 0, o número digitado será armazenado no índice [0], ou seja, numero[i]. Quando i for 1, numero[i] será numero[1].

- Armazenar 10 números na memória, somá-los e exibir a média do valor.

Muito similar ao anterior, mas agora teremos o segundo for para somar os números percorrendo todo o array numero. Digitaremos 10 números e o programa somará eles e imprimirá a média da soma.

```go
package main

import "fmt"

func main() {
    //declaração de variáveis
    var numero [10]float64 //criação de uma matriz de tamanho 10. em c# seria "int[] numero = new float[10];"
    var soma float64 = 0
    //entrada de dados
    for i := 0; i <= 9; i++ { // laço de repetição com i começando em 0, pois a primeira posição de uma matriz começa no índice 0 ([0])
        // laço termina com i = 10, pois a condição é que o laço continue enquanto i for menor ou igual a 9.
        fmt.Print("Digite o ", i+1, "º número: ")
        fmt.Scanln(&numero[i])
    }

    //saída de dados
    for i := 9; i >= 0; i-- { // laço de repetição com i começando em 9, logo, o índice 9 é o valor da posição 10 (décimo número armazenado)
        soma += numero[i] // então somaremos à variável soma (que vale 0) os valores de cada posição da matriz numero.
    }
    media := soma / 10
    fmt.Printf("\nA média dos números inseridos é %.2f!!\n", media)

}
```

Execução:

```
Digite o 1º número: 1
Digite o 2º número: 2
Digite o 3º número: 3
Digite o 4º número: 4
Digite o 5º número: 5
Digite o 6º número: 6
Digite o 7º número: 7
Digite o 8º número: 8
Digite o 9º número: 9
Digite o 10º número: 10

A média dos números inseridos é 5.50!!
```

- Armazenar 10 números na memória e identificar o maior e o menor número

Neste programa precisaremos de um array com 10 posições para armazenar os 10 números e duas variáveis: maior e menor. Ambas começam valendo 0.

No laço de repetição teremos dois ifs: um para decidir se o valor inserido é a maior dos inseridos e o outro para decidir se o valor inserido é o menor dos inseridos.

Esses if's funcionam juntamente ao laço de repetição, pois se compara cada valor de cada posição do array número.

```go
package main

import (
    "fmt"
)

func main() {
    //declaração de variáveis
    var numero [10]int //criação de uma matriz de tamanho 10. em c# seria "int[] numero = new float[10];"
    maior := 0
    menor := 0
    //entrada de dados
    for i := 0; i <= 9; i++ { // laço de repetição com i começando em 0, pois a primeira posição de uma matriz começa no índice 0 ([0])
        //o laço termina com i = 10, pois a condição é que o laço continue enquanto i for menor ou igual a 9.
        fmt.Print("Digite o ", i+1, "º número: ")
        fmt.Scanln(&numero[i])

        if numero[i] > maior {
            maior = numero[i]
            if i == 0 {
                menor = maior
            }
        }
        if numero[i] < menor {
            menor = numero[i]
        }
    }
    //saída de dados
    fmt.Print("\n\nO menor número digitado é: ", menor)
    fmt.Print("\nO maior número digitado é: ", maior, "\n")
}
```

- Informar idade com base no nome da pessoa

Nesse programa será digitado e armazenado o nome e a idade de dez pessoas. Feito isso será solicitado a digitação de um nome, e então a pesquisa. Caso o sujeito pesquisado tenha sido encontrado, será exibido a sua idade, caso contrário será informado que a pessoa não foi localizada. Logo após, será perguntado se ele quer ou fazer uma nova consulta, para isso terá que responder “S” ou “N” (sim ou não). Acompanhe o código a seguir

```go
package main

import "fmt"

func main() {
    // Declaração de Variáveis
    var nome [10]string
    var idade [10]string
    var nomePesq string
    resp := "S"

    // Entrada de dados
    for i := 0; i <= 9; i++ {
        fmt.Print("\nDigite o nome da ", i+1, "ª pessoa: ")
        fmt.Scanln(&nome[i])
        fmt.Print("Digite a idade da ", i+1, "ª pessoa: ")
        fmt.Scanln(&idade[i])
    }

    for resp == "S" {
        a := false
        fmt.Print("\nDigite o nome da pessoa que deseja saber a idade: ")
        fmt.Scanln(&nomePesq)

        // Saída de dados
        for i := 0; i <= 9; i++ {
            // Comparação para ver se o nome pesquisado foi um dos 10 nomes digitados anteriormente
            if nome[i] == nomePesq {
                fmt.Println("A idade do ", nomePesq, "é de: ", idade[i], "anos")
                a = true
            }
        }
```

```
for resp == "S" {
    a := false
    fmt.Print("\nDigite o nome da pessoa que deseja saber a idade: ")
    fmt.Scanln(&nomePesq)

    // Saída de dados
    for i := 0; i <= 9; i++ {
        // Comparação para ver se o nome pesquisado foi um dos 10 nomes digitados anteriormente
        if nome[i] == nomePesq {
            fmt.Println("A idade do ", nomePesq, "é de: ", idade[i], "anos")
            a = true
        }
    }

    /* se a for falso, significa que não entrou na estrutura de condição,
    portanto o nome não foi cadastrado */
    if a == false {
        fmt.Println("Pessoa não localizada!!")
    }

    // Verificação para uma nova consulta
    fmt.Print("\nDeseja fazer uma nova consulta? (digite apenas S ou N): ")
    fmt.Scanln(&resp)
    for resp != "S" && resp != "N" {
        fmt.Print("\nErro!\nDeseja fazer uma nova consulta? (digite apenas S ou N): ")
        fmt.Scanln(&resp)
    }
}
```

Execução:

```
Digite o nome da 1º pessoa: Everson
Digite a idade da 1º pessoa: 12

Digite o nome da 2º pessoa: Alisson
Digite a idade da 2º pessoa: 65

Digite o nome da 3º pessoa: Jeremias
Digite a idade da 3º pessoa: 11

Digite o nome da 4º pessoa: Maria
Digite a idade da 4º pessoa: 5

Digite o nome da 5º pessoa: Lucas
Digite a idade da 5º pessoa: 7

Digite o nome da 6º pessoa: Livia
Digite a idade da 6º pessoa: 3

Digite o nome da 7º pessoa: Paulo
Digite a idade da 7º pessoa: 2

Digite o nome da 8º pessoa: Irineu
Digite a idade da 8º pessoa: 1

Digite o nome da 9º pessoa: Leticia
Digite a idade da 9º pessoa: 87

Digite o nome da 10º pessoa: Mateus
Digite a idade da 10º pessoa: 43

Digite o nome da pessoa que deseja saber a idade: Lucas
A idade do  Lucas é de:  7 anos

Deseja fazer uma nova consulta? (digite apenas S ou N): N
PS C:\Users\user\Desktop\escola\TCC\Capitulo 2>
```

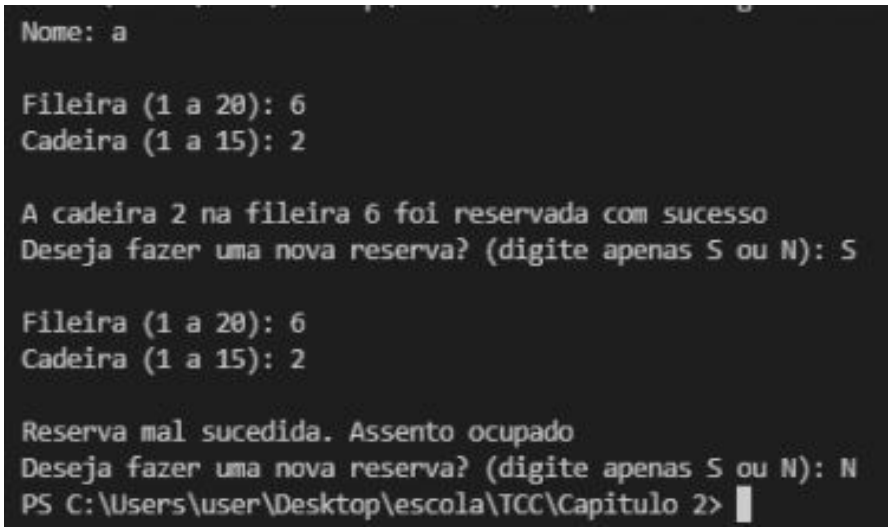
- Registrar cadeira no cinema

O programa começará solicitando ao usuário a digitação de seu nome, o número da fileira e cadeira em que deseja se sentar. Se o assento estiver vazio, será registrado na matriz o seu nome, caso contrário será informado que o assento está ocupado. Feito isso o programa questionará se desejamos nova reserva, validando nossa resposta e repetindo todo o processo.
Para alimentar uma matriz é preciso de dois for, o primeiro vai iterar na linha e o segundo vai iterar na coluna, preenchendo assim todas as lacunas da matriz. Veja o código no a seguir

```go
func main() {
    // declaração de variáveis
    var registro [20][15]string // declaração da matriz registro
    var nome string
    fileira := 0
    cadeira := 0
    a := false
    resp := "S"

    // entrada de dados (Nome, Fileira e Cadeira)
    fmt.Print("Nome: ")
    fmt.Scanln(&nome)

    for resp == "S" {
        a = false
        fmt.Print("\nFileira (1 a 20): ")
        fmt.Scanln(&fileira)
        for fileira <= 0 && fileira > 20 {
            fmt.Print("\nErro!\nFileira (1 a 20): ")
            fmt.Scanln(&fileira)
        }

        fmt.Print("Cadeira (1 a 15): ")
        fmt.Scanln(&cadeira)
        for cadeira <= 0 && cadeira > 15 {
            fmt.Print("\nErro!\nCadeira (1 a 15): ")
            fmt.Scanln(&cadeira)
        }

        fileira = fileira - 1
        cadeira = cadeira - 1
```

```go
        fileira = fileira - 1
        cadeira = cadeira - 1

        // Verificando se a cadeira e fileira do cinema já está reservada
        for i := 0; i < 20; i++ {
            for j := 0; j < 15; j++ {
                // Se o local não estiver ocupado, será ser reservada com o bloco de instruções abaixo
                if registro[fileira][cadeira] == "" {
                    registro[fileira][cadeira] = nome
                    fmt.Print("\nA cadeira ", cadeira+1, " na fileira ", fileira+1, " foi reservada com sucesso")
                    a = true
                }
            }
        }

        // Se já estiver ocupada, a seguinte mensagem será exibida
        if a == false {
            fmt.Print("\nReserva mal sucedida. Assento ocupado")
        }

        // Por fim, será perguntado se quer ou não reservar mais uma cadeira do cinema, se sim ("S"), se não ("N")
        fmt.Print("\nDeseja fazer uma nova reserva? (digite apenas S ou N): ")
        fmt.Scanln(&resp)
        for resp != "S" && resp != "N" {
            fmt.Print("\nErro!\nDeseja fazer uma nova reserva? (digite apenas S ou N): ")
            fmt.Scanln(&resp)
        }
    }
}
```

Execução:


Nome: a
Fileira (1 a 20): 6
Cadeira (1 a 15): 2
A cadeira 2 na fileira 6 foi reservada com sucesso
Deseja fazer uma nova reserva? (digite apenas S ou N): S
Fileira (1 a 20): 6
Cadeira (1 a 15): 2
Reserva mal sucedida. Assento ocupado
Deseja fazer uma nova reserva? (digite apenas S ou N): N
PS C:\Users\user\Desktop\escola\TCC\Capitulo 2>

Modo gráfico (GUI)

Neste capítulo estudaremos o desenvolvimento de aplicações em modo gráfico (GUI), utilizaremos o walk como framework . Para tal, precisaremos seguir alguns passos:

Instalando o git

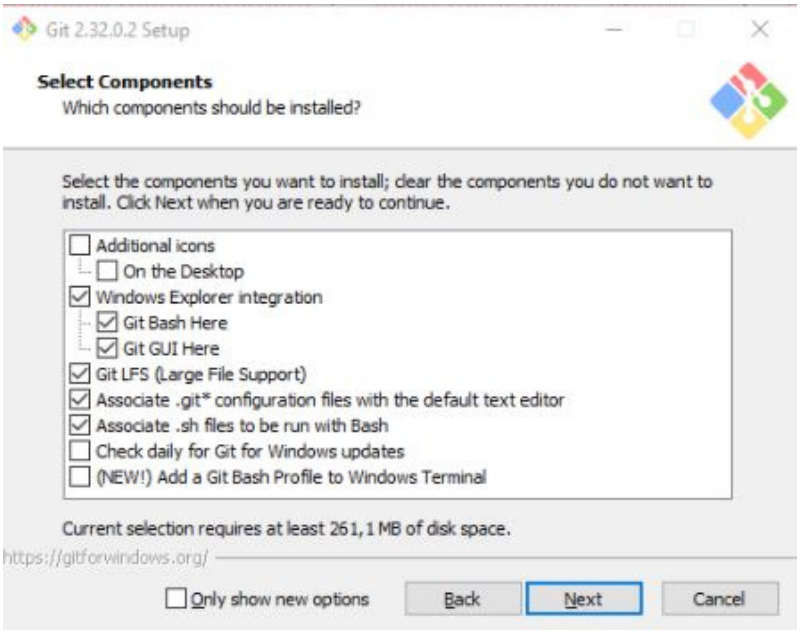
Primeiramente teremos que instalar o git-scm, um sistema de controle de versão de arquivos. Através dele, é possível desenvolver projetos onde pessoas podem, simultaneamente, contribuir nele.

O link para download é: https://git-scm.com/downloads

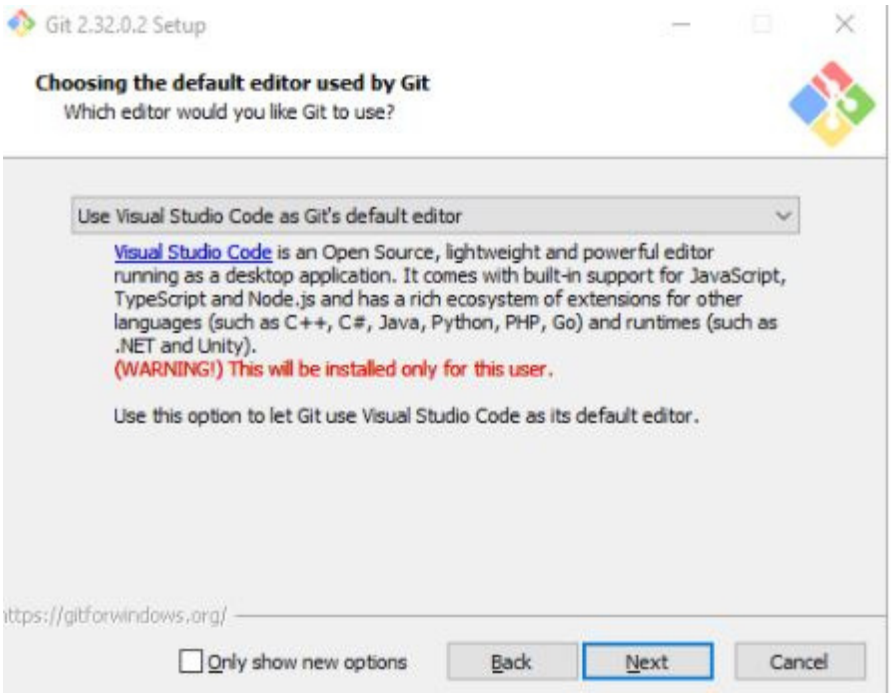
Após clicar no link, abaixo de "Downloads", você irá escolher seu sistema operacional. No nosso exemplo estaremos clicando em "Windows", pois estamos utilizando uma máquina com Windows 10.

Após abrir o setup do git, você se deparará com essa tela. Quando estiver pronto, clique em "Next".

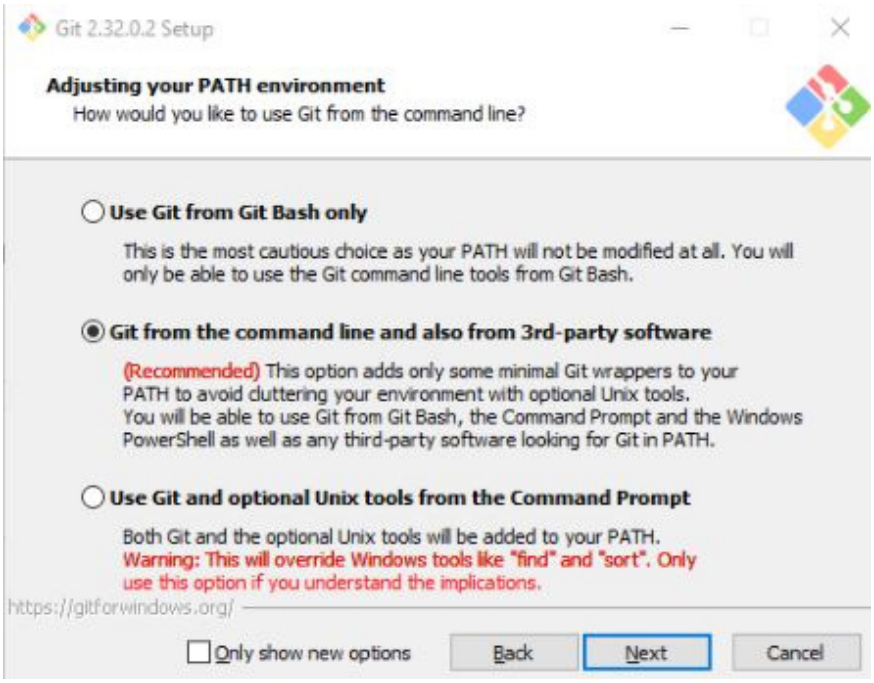
Aqui você escolherá os componentes que deseja instalar. Se você for um iniciante, recomendamos que deixe como está e apenas clique em "Next".



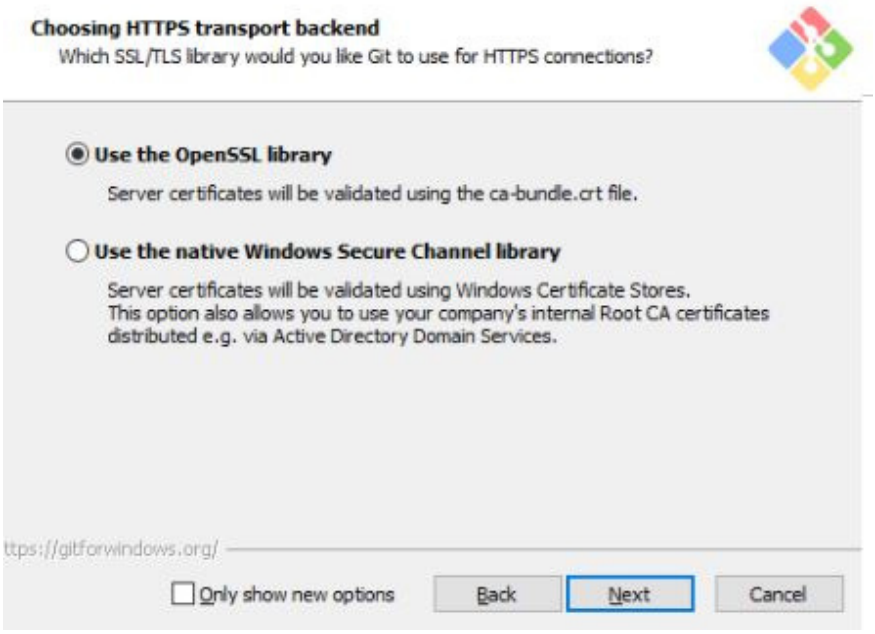
Agora você escolherá o editor de texto que você utiliza para programar. No nosso caso, estamos usando o Visual Studio Code, então estaremos usando a opção selecionada por padrão. Clique em "Next" após isso.



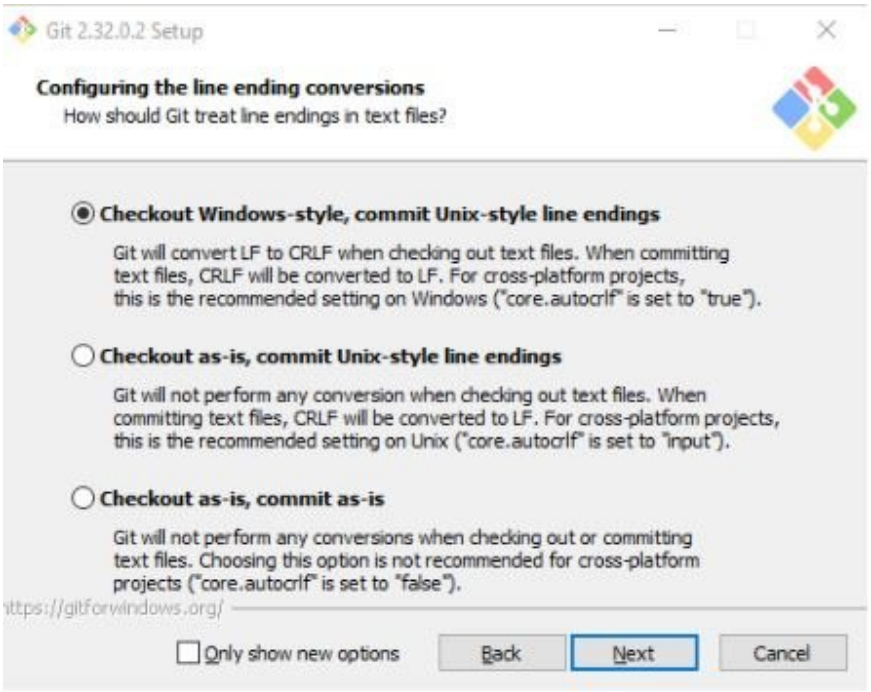
Agora você terá que selecionar como você irá usar o git pelo prompt de comando. Recomendamos escolher a opção do meio que vem por padrão.



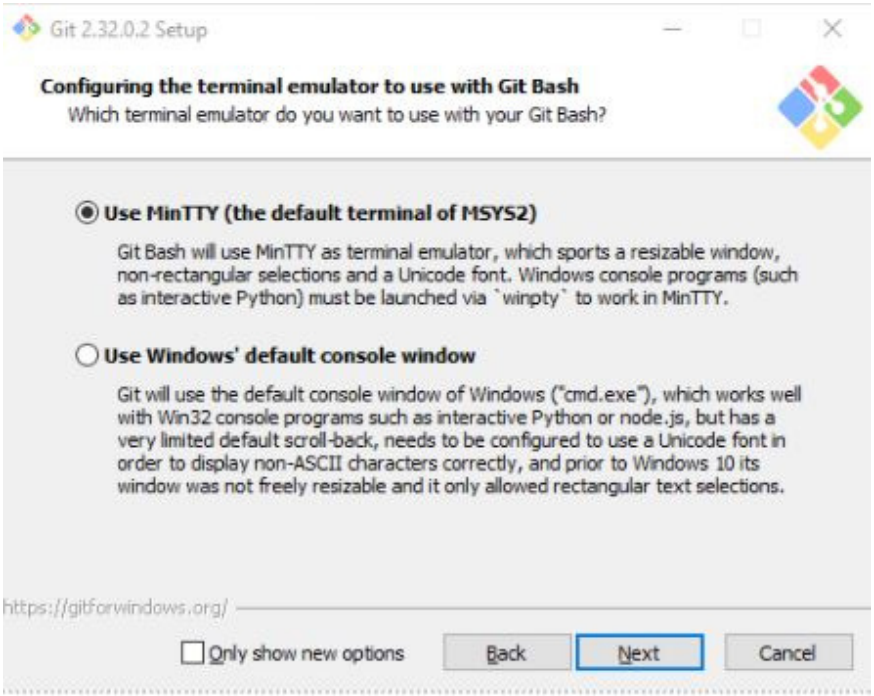
Agora, resumidamente, você selecionará a biblioteca que será usada quando conexões https forem criadas usando git. OpenSSL é selecionada por padrão, deixaremos ela e clicamos em "Next".



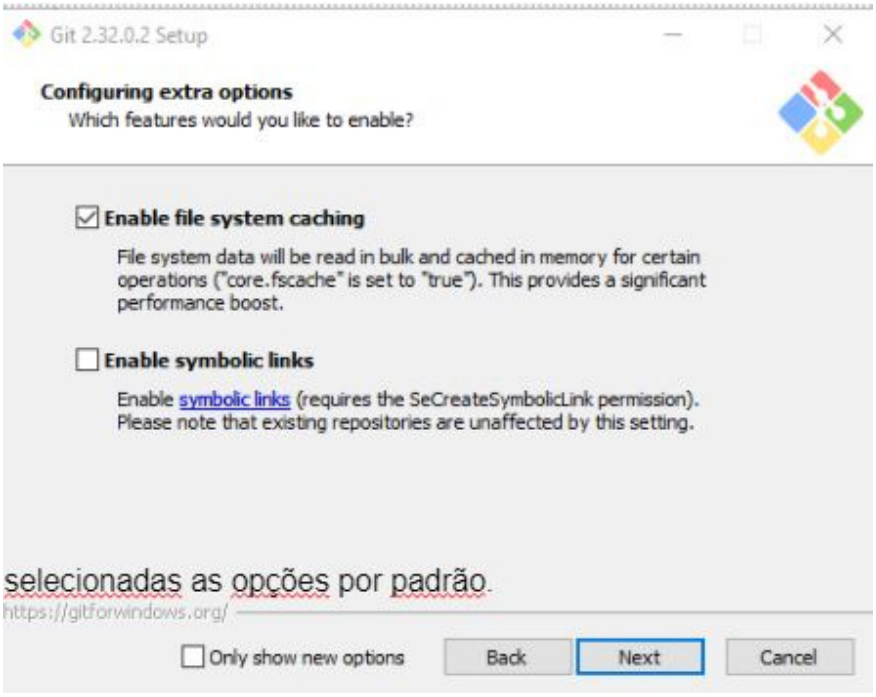
Aqui selecionamos opções de line ending. Deixaremos na opção padrão e clicamos em "Next".



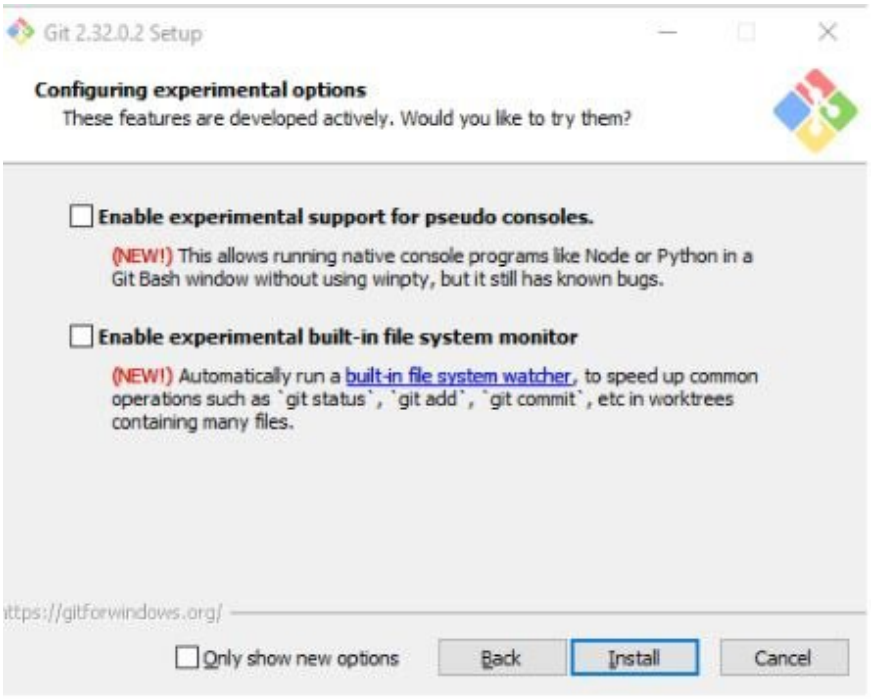
Agora selecione o emulador do terminal que será utilizado. MinTTY é a opção selecionada por padrão. Clique "Next".



Clique "next" novamente, deixando selecionadas as opções por padrão.



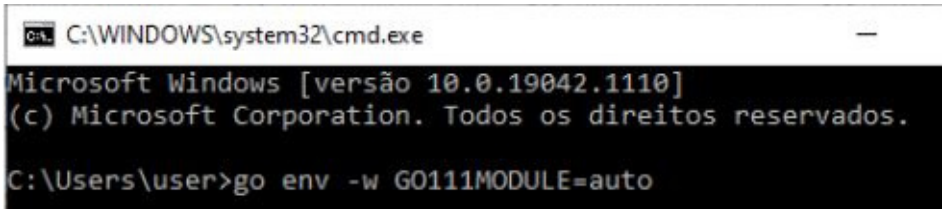
Configurações experimentais não são selecionadas por padrão. Você pode clicar em "Install" e, finalmente, terá instalado o git!



Comandos no terminal - GO111MODULE e bibliotecas

Primeiramente abriremos nosso cmd e iremos inserir três comandos:

1º) "go env -w GO111MODULE=auto" - Para podermos trabalhar com bibliotecas sem problema algum!



2º) "go get github.com/lxn/walk" - Para baixarmos uma biblioteca que nos possibilitará de trabalhar com desenvolvimento de aplicações em modo gráfico! Você perceberá que você ficará impossibilitado de digitar por um tempo, e é normal! Isso significa que o download está sendo feito. Caso você queira visitar o link da biblioteca, acesse aqui ~> github.com/lxn/walk.

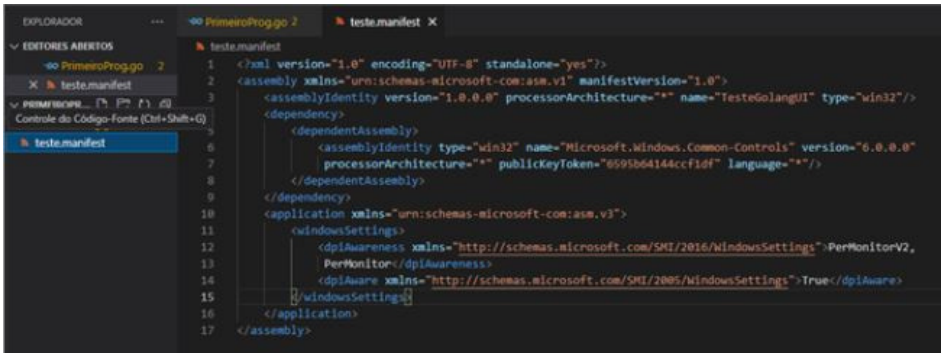
3º) "go get github.com/akavel/rsrc" - Isso irá permitir que consigamos criar o arquivo .manifest que será muito importante para o funcionamento da nossa GUI.

Pronto! Agora vamos pôr a mão na massa e criar nosso primeiro programa!

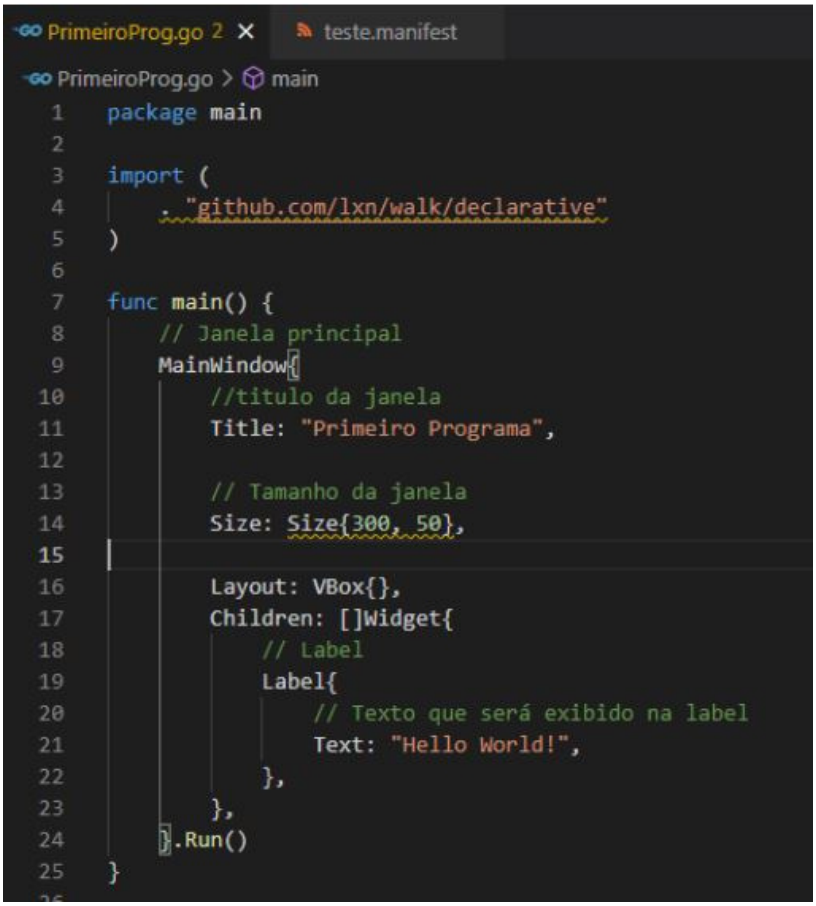
Golang em prática

Hello World

Para exibir nosso primeiro "hello world" devemos primeiro criar o arquivo .manifest e digitar o código a seguir:



Logo após fazer isso, vamos criar o nosso arquivo .go e digitar o código que imprimirá na tela o hello world.

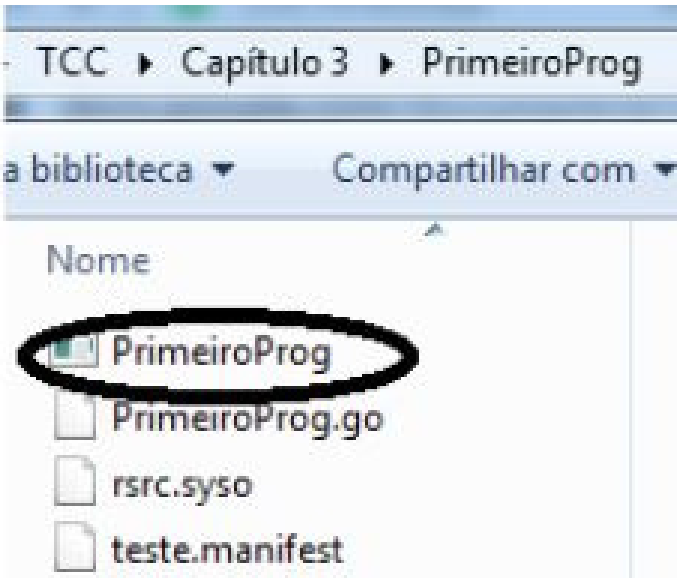


Antes de executar é necessário compilar o arquivo .manifest. Como o nome do nosso arquivo é teste.manifest compilamos da seguinte maneira:

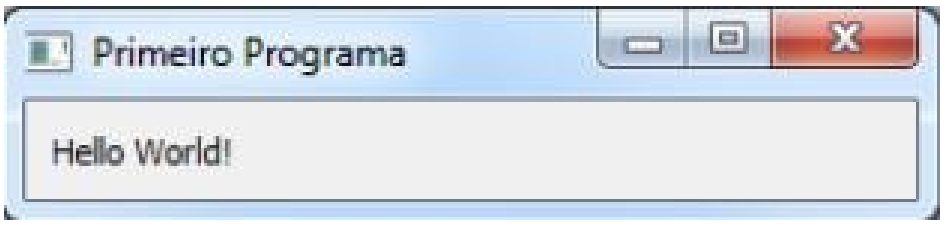
Nessa mesma pasta será gerado um arquivo rsrc.syso. Feito isso, agora é só executar o nosso programa. Para isso usamos o comando "go build" no terminal dentro da pasta do nosso projeto.

```
PS C:\Users\user\Desktop\escola\TCC\Capítulo 3\PrimeiroProg> rsrc -manifest teste.manifest -o rsrc.syso
PS C:\Users\user\Desktop\escola\TCC\Capítulo 3\PrimeiroProg> go build
PS C:\Users\user\Desktop\escola\TCC\Capítulo 3\PrimeiroProg>
```

Agora é só abrir a pasta do projeto e dar dois cliques no .exe



Resultado da execução:

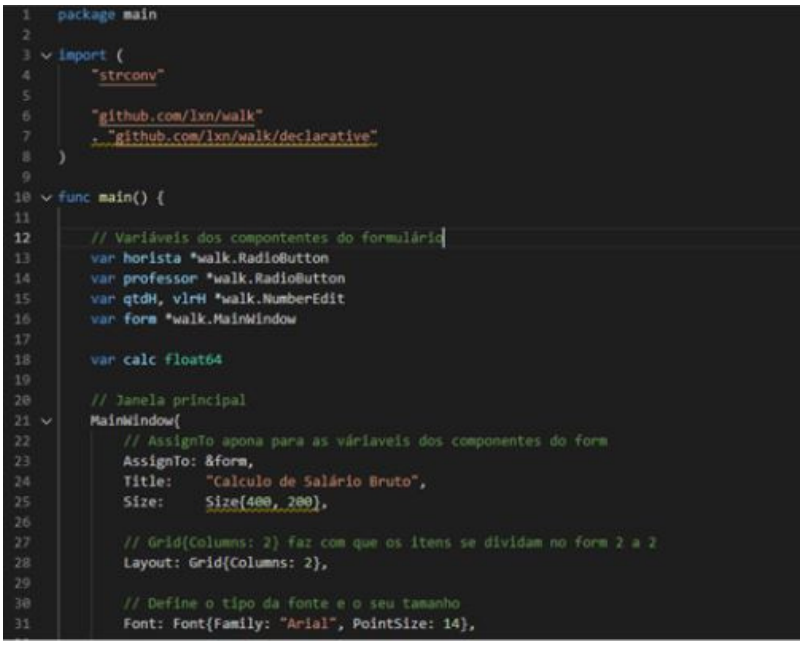


Agora aprofundaremos um pouco mais o nosso conhecimento em interface gráfica usando GO com alguns exercícios mais complexos. Lembre-se que o arquivo .manifest e o rsrc.syso podem ser copiados para os outros programas, assim tendo que fazer e executar somente o arquivo ".go".

- Cálculo do Salário Bruto

Faremos dois programas onde o usuário entrará com a quantidade de horas trabalhadas e o valor que ele ganha por hora, para esses campos utilizamos o NumberEdit, ele é importante para entrada com valores numéricos. O usuário também terá que dizer se ele é horista ou professor. Para isso faremos dois programas, um usando radio buttons e outro usando uma comboBox. O salário bruto do professor é valorhora * qtdhora * 1.25. Enquanto o do horista é valor hora * qtdhora. Veja a seguir o código comentado de ambos programas e as suas respectivas execuções:

Utilizando Radio Button



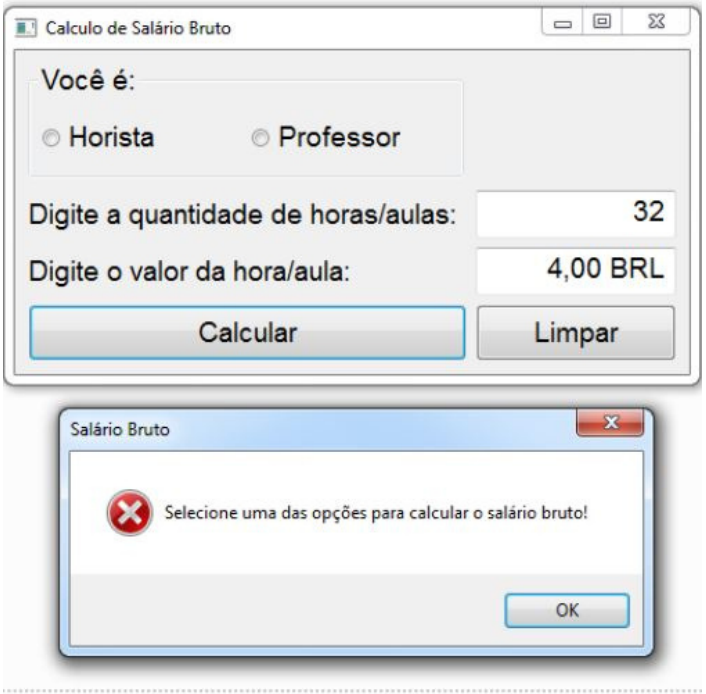
```go
package main

import (
    "strconv"

    "github.com/lxn/walk"
    . "github.com/lxn/walk/declarative"
)

func main() {

    // Variáveis dos compontentes do formulário
    var horista *walk.RadioButton
    var professor *walk.RadioButton
    var qtdH, vlrH *walk.NumberEdit
    var form *walk.MainWindow

    var calc float64

    // Janela principal
    MainWindow{
        // AssignTo apona para as váriaveis dos componentes do form
        AssignTo: &form,
        Title:    "Calculo de Salário Bruto",
        Size:     Size{400, 200},

        // Grid{Columns: 2} faz com que os itens se dividam no form 2 a 2
        Layout: Grid{Columns: 2},

        // Define o tipo da fonte e o seu tamanho
        Font: Font{Family: "Arial", PointSize: 14},
```

```go
// Define que todo o seu conteúdo estará dentro da janela principal
Children: []Widget{

    // Grupo de Radio Button.
    RadioButtonGroupBox{

        // Título do grupo de radios buttons.
        Title:  "Você é:",
        Layout: Grid{Columns: 2},
        Buttons: []RadioButton{
            // Definindo dois radios buttons (Horista e Professor).
            {Text: "Horista", Value: "Hor", AssignTo: &horista},
            {Text: "Professor", Value: "Prof", AssignTo: &professor},
        },
    },

    Label{
        Text: "Digite a quantidade de horas/aulas:",
    },

    // Caixa de Texto numérico onde será digitado a quantidade de horas.
    NumberEdit{
        AssignTo: &qtdH,
    },

    Label{
        Text: "Digite o valor da hora/aula:",
    },
```

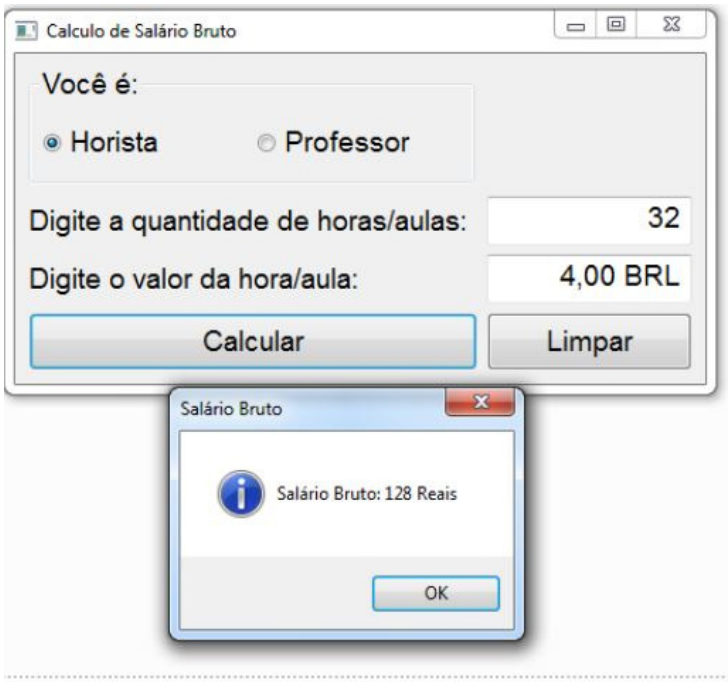
```
            // Caixa de Texto numérico onde será digitado o valor de horas trabalhadas.
            NumberEdit{
                AssignTo: &vlrH,

                // Define a quantidade de casas decimais e adiciona um texto no final da caixa.
                Suffix:   " BRL",
                Decimals: 2,
            },

            // Botão responsável por calcular o salário bruto
            PushButton{
                // Texto do botão
                Text: "Calcular",

                // Função quando o botão for clicado
                OnClicked: func() {

                    // Checando se o radio button "professor" foi marcado
                    if professor.Checked() == true {
                        // Calculo do salario para professor
                        calc = vlrH.Value() * qtdH.Value() * 1.25

                        // Exibindo salário em uma caixa de mensagem
                        walk.MsgBox(form, "Salário Bruto", "Salário Bruto: "+strconv.FormatFloat(calc, 'f', -1,
                        64)+" Reais", walk.MsgBoxIconInformation)
                    }
```

```
                    // Verificando se o radio button "horista" foi marcado
                    if horista.Checked() == true {
                        // Calculo do salario para horista
                        calc = vlrH.Value() * qtdH.Value()
                        walk.MsgBox(form, "Salário Bruto", "Salário Bruto: "+strconv.FormatFloat(calc,
                        'f', -1, 64)+" Reais", walk.MsgBoxIconInformation)
                    }

                    /* Checando se nenhum dos radios buttons foram marcados,
                       se nao forem será exibido uma caixa de mensagem exibindo o erro */
                    if horista.Checked() == false && professor.Checked() == false {
                        walk.MsgBox(form, "Salário Bruto", "Selecione uma das opções para calcular o salário bruto!",
                        walk.MsgBoxIconError)
                    }
                },
            },

            // Botão Limpar, caso seja clicado limpará todos os campos do form
            PushButton{
                Text: "Limpar",
                OnClicked: func() {
                    qtdH.SetValue(0)
                    vlrH.SetValue(0)
                    horista.SetChecked(false)
                    professor.SetChecked(false)
                },
            },
        },
    }.Run()
}
```

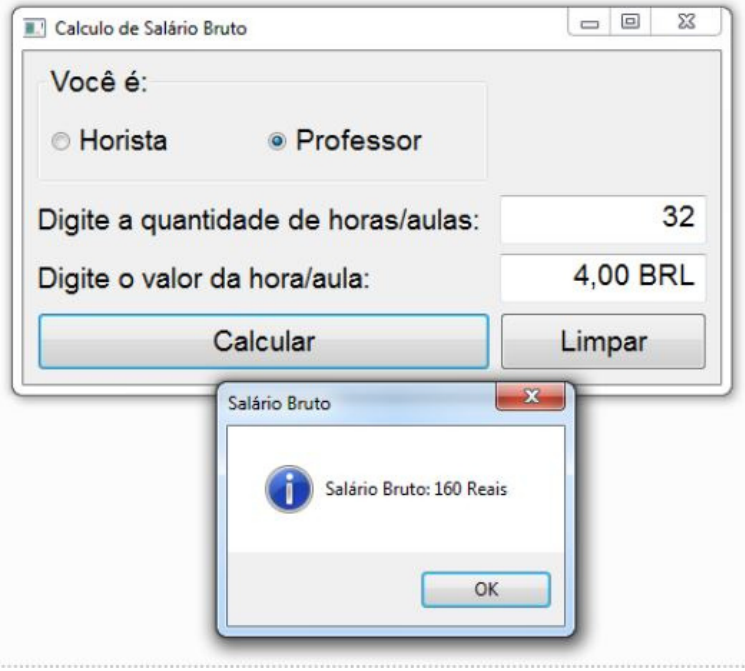
Execução e comportamentos: Quando nenhuma opção é selecionada



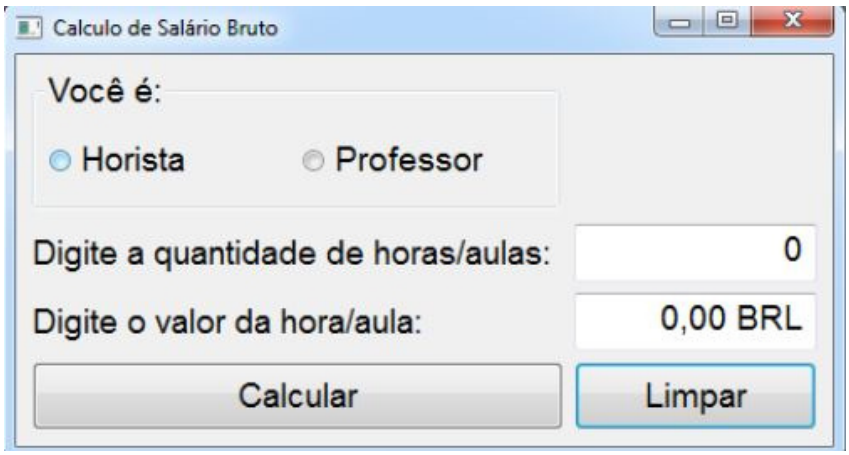
Quando a opção horista é selecionada



Quando a opção "professor" é selecionada



Quando o botão limpar é acionado



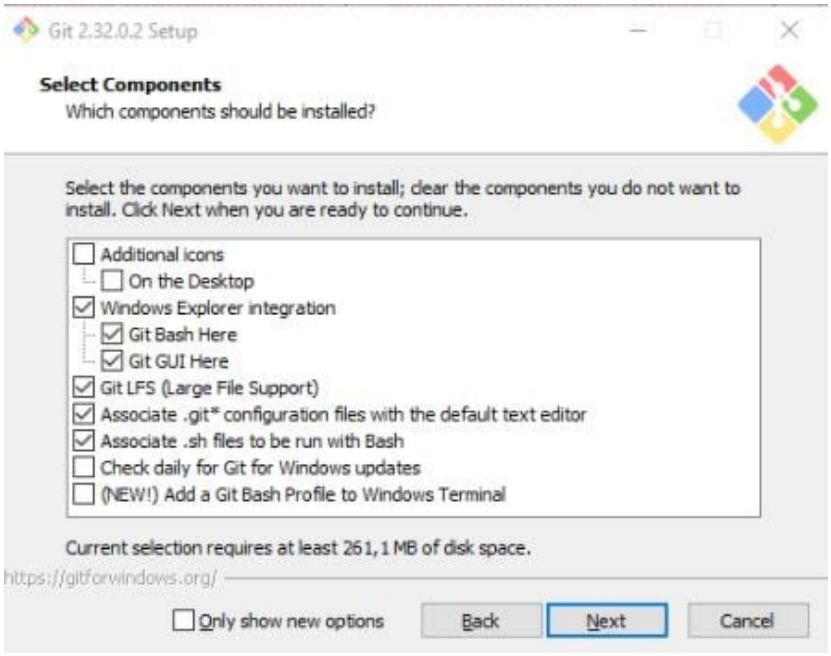
Utilizando ComboBox

```go
package main

import (
    "strconv"

    "github.com/lxn/walk"
    . "github.com/lxn/walk/declarative"
)

func main() {
    var qtdH, vlrH *walk.NumberEdit
    var form *walk.MainWindow
    var combo *walk.ComboBox
    var calc float64

    MainWindow{
        AssignTo: &form,
        Title:    "Calculo de Salário Bruto",
        Size:     Size{400, 200},
        Layout:   Grid{Columns: 2},
        Font:     Font{Family: "Arial", PointSize: 14},
        Children: []Widget{
            Label{
                Text: "Insira: ",
            },
            ComboBox{
                AssignTo: &combo,

                // Não deixará o usuário digitar o que quiser
                Editable: false,
```

```go
                // o que aparecerá dentro da combobox
                Model:    []string{"Horista", "Professor"},
            },

            Label{
                Text: "Digite a quantidade de horas/aulas:",
            },
            NumberEdit{
                AssignTo: &qtdH,
            },

            Label{
                Text: "Digite o valor da hora/aula:",
            },
            NumberEdit{
                AssignTo: &vlrH,
                Suffix:   " BRL",
                Decimals: 2,
            },

            PushButton{
                Text: "Calcular",
                OnClicked: func() {
                    // Checando qual opção foi marcada no checkbox
                    if combo.Text() == "Horista" {
                        calc = vlrH.Value() * qtdH.Value()
                        walk.MsgBox(form, "Salário Bruto", "Salário Bruto: "+strconv.FormatFloat(calc,
                            'f', -1, 64)+" Reais", walk.MsgBoxIconInformation)
                    }
```

```go
                    if combo.Text() == "Professor" {
                        calc = vlrH.Value() * qtdH.Value() * 1.25
                        walk.MsgBox(form, "Salário Bruto", "Salário Bruto: "+strconv.FormatFloat(calc,
                            'f', -1, 64)+" Reais", walk.MsgBoxIconInformation)
                    }

                    if combo.Text() != "Professor" && combo.Text() != "Horista" {
                        walk.MsgBox(form, "Salário Bruto", "Selecione uma das opções para calcular o salário bruto!",
                            walk.MsgBoxIconError)
                    }
                },
            },

            PushButton{
                Text: "Limpar",
                OnClicked: func() {
                    qtdH.SetValue(0)
                    vlrH.SetValue(0)

                    // Limpa a comboBox
                    combo.SetCurrentIndex(-1)
                },
            },
        },
    }.Run()
```

Execução e comportamentos (como o comportamento é quase o mesmo só que usando comboBox, mostraremos apenas uma tela de execução, esta sendo quando a opção horista é selecionada e clicamos no botão calcular):



- Cálculo da tabuada

Esse programa receberá um valor dentro de um numberEdit que será o valor base para o cálculo da tabuada. Quando o usuário clicar no botão “Calcular” a tabuada deverá ser impressa na TextEdit, um campo que armazena textos, também temos o LineEdit, porém como o próprio nome já diz, ele suporta só uma linha. Vejamos o código comentado e a execução desse programa a seguir:

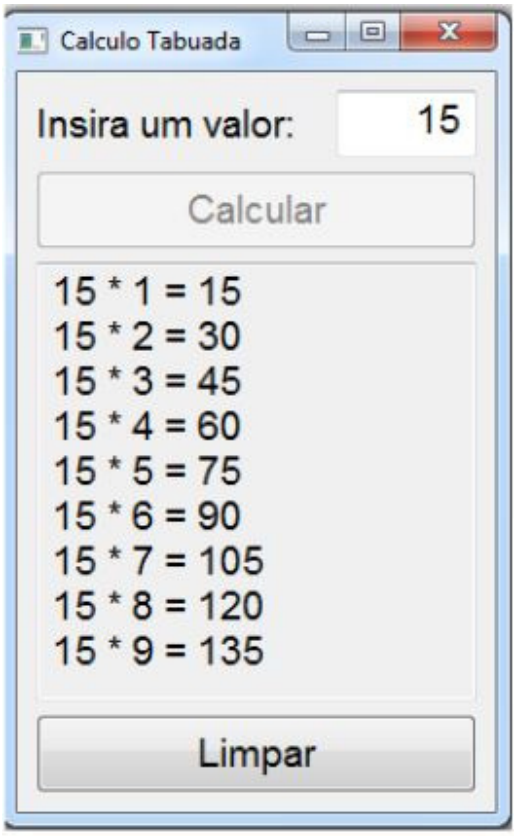
```go
package main

import (
    "strconv"

    "github.com/lxn/walk"
    . "github.com/lxn/walk/declarative"
)

func main() {
    var cons *walk.NumberEdit
    var form *walk.MainWindow
    var list *walk.TextEdit
    var calcular *walk.PushButton

    var calc, i float64

    MainWindow{
        AssignTo: &form,
        Title:    "Calculo Tabuada",
        Size:     Size{250, 400},
        Layout:   Grid{Columns: 2},
        Font:     Font{Family: "Arial", PointSize: 14},
        Children: []Widget{
            Label{
                Text: "Insira um valor: ",
            },

            // Campo que armazenará o valor da tabuda
            NumberEdit{
                AssignTo: &cons,
            },
```

```go
            // Botão Calcular
            PushButton{
                AssignTo:   &calcular,
                ColumnSpan: 2,
                Text:       "Calcular",
                OnClicked: func() {
                    // desabilitando o botão "calcular" até que o usuário clique no botão "limpar"
                    calcular.SetEnabled(false)

                    // Código gerador da tabuada
                    for i = 1; i <= 10; i++ {
                        // Calculo do valor recebido pelo usuario multiplicado pelas iterações do for
                        calc = cons.Value() * i

                        // Setando o texto na TextEdit onde será exibido a tabuada e pulando uma linha à cada iteração
                        list.SetText(list.Text() + strconv.FormatFloat(cons.Value(), 'f', -1,
                            64) + " * " + strconv.FormatFloat(i, 'f', -1, 64) + " = " + strconv.FormatFloat(calc,
                            'f', -1, 64) + "\r\n")
                    }
                },
            },

            // Text edit onde será exibida a tabuada
            TextEdit{
                ColumnSpan: 2,
                AssignTo:   &list,
                Text:       "",

                // Código que deixa a text edit apenas para leitura impossibilitando que o usuario insira texto
                ReadOnly: true,
            },
```

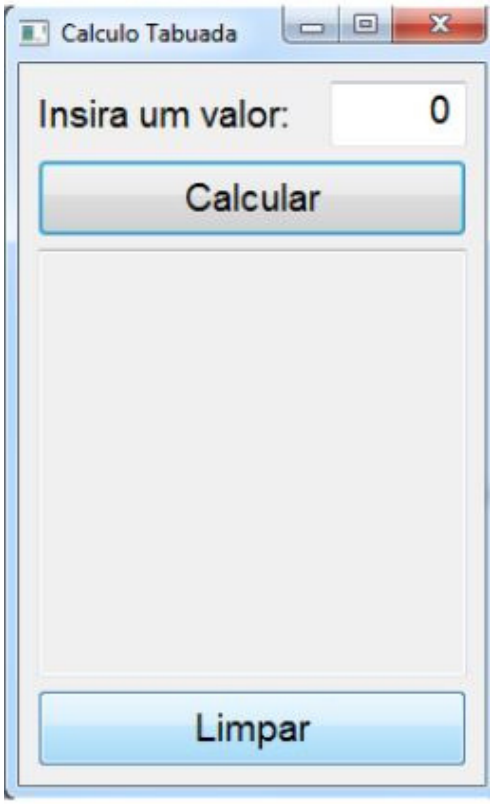
```go
            // Limpando os campos do form
            PushButton{
                ColumnSpan: 2,
                Text:       "Limpar",
                OnClicked: func() {
                    list.SetText("")
                    cons.SetValue(0)

                    // Após a limpeza habilitando o botão "calcular"
                    calcular.SetEnabled(true)
                },
            },
        },
    }.Run()
}
```

Execução e Comportamentos:

Quando o botão “Calcular” é acionado



Quando o botão “Limpar” é acionado



Relacionando conceitos de POO (programação orientada a objetos) com golang

GO é uma linguagem orientada a objetos?

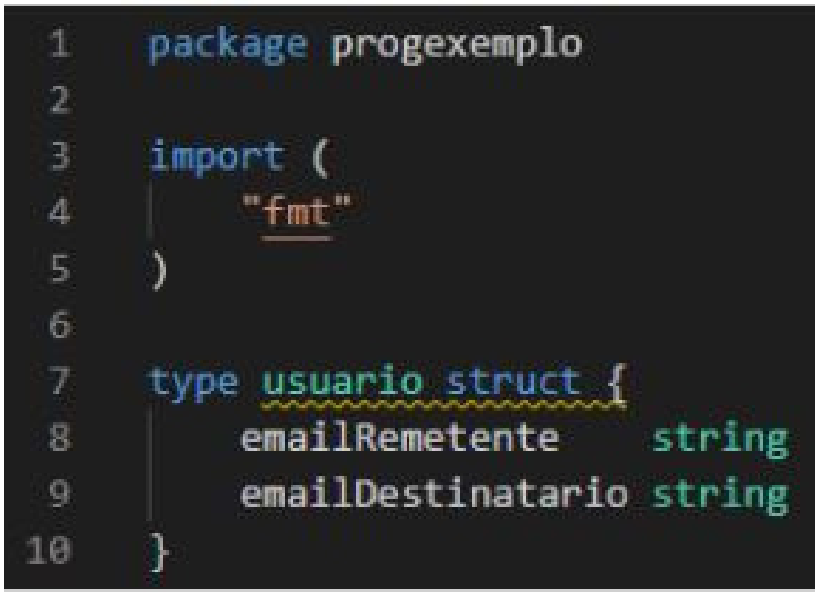
Go nos permite desenvolver um estilo similar à programação orientada a objetos e, a partir disso, construir tipos e métodos. Contudo, não há hierarquia de tipos, é um estilo bem diferente da implementação que conhecemos em outras linguagens de programação, pois comparado às outras linguagens, Go possui poucos recursos nessa área. O que temos nessa área é uma abstração que nos permite utilizar diferentes tipos de structs sem nos preocupar com qual struct exatamente está sendo trabalhada.

Agora vamos ver e aprender conceitos importantes com um programa de exemplo que ajudará a entender os outros exercícios que serão feitos neste capítulo.

Exercício Exemplo:

Struct

Go não possui classes, entretanto, existem tipos que são chamados de structs, estes possuem uma estrutura muito similar às classes. Veja a seguir como structs se comportam:



Essas variáveis que a struct do tipo "usuario" possui, são como os atributos de uma classe, guardando apenas estado. Na declaração das variáveis só é preciso colocar o nome do campo e ao lado o seu tipo.

Método

Após entender como funcionam as structs e criar o tipo usuário, vamos falar sobre o método. Assim como em outras linguagens de programação, o método é responsável por guardar os comportamentos. Veja a seguir:

```
1  package progexemplo
2
3  import (
4      "fmt"
5  )
6
7  type usuario struct {
8      emailRemetente    string
9      emailDestinatario string
10 }
11
12 func (u usuario) enviarEmail() {
13     fmt.Println("Foi enviado um email de: '", u.emailRemetente, "' para: '", u.emailDestinatario, "'")
14 }
15
```

Os métodos são referenciados às structs por meio de uma cláusula receptadora, no exemplo acima, "(u usuario)", com “u” sendo uma instância da struct do tipo “usuário”.

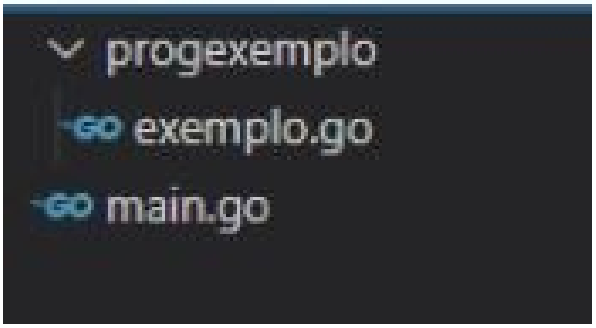
Agora, para começar a brincar com essa struct, iremos para a função principal que será responsável por manipulá-la. Porém, para isso é preciso entender que se quisermos criar a função principal em um arquivo separado do que seria a nossa “classe” em outras linguagens, devemos entender como funcionam os packages (pacotes) em golang.

Packages

Os packages (pacotes) são usados para organizar o código-fonte a fim de uma melhor reutilização e legibilidade. Os pacotes são uma coleção de arquivos que residem no mesmo diretório. Os pacotes oferecem compartimentação de código e, assim, torna-se fácil manter os projetos em golang.

Os arquivos fonte pertencentes a um pacote devem ser colocados em pastas separadas por conta própria. É uma convenção em Go nomear esta pasta com o mesmo nome do pacote.

Apenas o arquivo principal não deverá ser colocado em uma pasta separada, ficando apenas dentro do diretório que estamos. Por enquanto pode parecer um pouco confuso, porém logo mais tudo será esclarecido. Com isso, dando continuidade ao nosso programa de exemplo, iremos organizar ele da seguinte maneira:



Dentro da pasta pro exemplo iremos guardar o arquivo exemplo.go que produzimos até aqui e fora deixaremos o nosso arquivo principal que será responsável por se conectar ao outro arquivo. Vejamos a seguir como fazer isso:

Para isso é preciso de dentro do nosso diretório, executar o seguinte comando no terminal: “go mod init nomearq”. Onde o “nomearq” é o nome que será usado para definir uma pasta imaginária que será pai das demais pastas, onde o intuito é de conectar o package principal (main) ao package exemplo, e assim podermos usar a struct “usuario” do arquivo exemplo.go.

```
PS C:\Users\user\Desktop\escola\TCC\progexemplo> go mod init POO
```

Colocamos o nosso “nomearq” como “POO”, porém você pode escolher o nome que preferir. Este comando irá gerar um arquivo go.mod dentro do nosso diretório, ele vai ser responsável por conectar as packages e fazer o programa entender o que irá ser importado.

Agora basta importar o package progexemplo com o seguinte código no arquivo main.go

Caso você tenha colocado outro nome quando executou o comando “go mod init nomearq”, basta substituir “POO” pelo nome que escolheu.
Pronto, agora é só codificar o arquivo principal!

Dados públicos e privados de uma struct

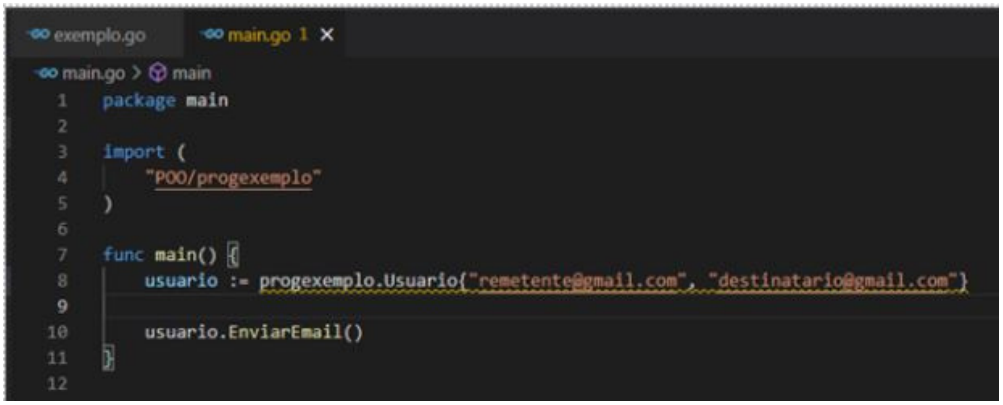
Para criar uma instância de alguma struct é preciso criar a seguinte linha de código “instância := package.struct{valorescampos}”. Dito isso, no nosso caso executamos o código abaixo:

```go
package main

import (
    "POO/progexemplo"
)

func main() {
    usuario := progexemplo.usuario{"remetente@gmail.com", "destinatario@gmail.com"}
}
```

Este código era para estar funcionando, porém há um problema. Acontece que quando criamos uma struct com a primeira letra minúscula, golang entende que a struct é privada. Portanto devemos colocar tudo que iremos acessar no arquivo principal com a primeira letra maiúscula. Corrigindo o código do arquivo exemplo.go:

```go
package progexemplo

import (
    "fmt"
)

type Usuario struct {
    EmailRemetente    string
    EmailDestinatario string
}

func (u Usuario) EnviarEmail() {
    fmt.Println("Foi enviado um email de: '", u.EmailRemetente, "' para: '", u.EmailDestinatario, "'")
}
```

Agora o método, a struct e seus campos estão públicos e livres para serem acessados de outros arquivos. Após corrigir o problema, agora é só chamar o método. E para isso utilizamos “instancia.Metodo()”. No nosso caso:



```go
package main

import (
    "POO/progexemplo"
)

func main() {
    usuario := progexemplo.Usuario{"remetente@gmail.com", "destinatario@gmail.com"}

    usuario.EnviarEmail()
}
```

Executando o programa:

```
PS C:\Users\user\Desktop\escola\TCC\progexemplo> go run main.go
Foi enviado um email de: ' remetente@gmail.com ' para: ' destinatario@gmail.com '
```

Encapsulamento

Um dos pilares da programação orientada à objetos é o encapsulamento, que como sabemos não permite que possa se alterar diretamente os atributos de um objeto. Relacionando com golang, o nosso código quebra essa regra, pois estamos setando valores diretamente nas structs. Para resolver isso é preciso criar métodos get e set deixando os campos da struct minúsculo para não poderem serem acessadas novamente. Vejamos a seguir:

```go
package progexemplo

import (
    "fmt"
)

type Usuario struct {
    emailRemetente    string
    emailDestinatario string
}

// parametro
func (u *Usuario) SetEmailRemetente(emailr string) {
    u.emailRemetente = emailr
}

// tipo do retorno da função
func (u Usuario) GetEmailRemetente() string {
    return u.emailRemetente
}

func (u *Usuario) SetEmailDestinatario(emaild string) {
    u.emailDestinatario = emaild
}

func (u Usuario) GetEmailDestinatario() string {
    return u.emailDestinatario
}

func (u Usuario) EnviarEmail() {
    fmt.Println("Foi enviado um email de: '", u.emailRemetente, "' para: '", u.emailDestinatario, "'")
}
```

As funções set sempre recebem um parâmetro e é preciso especificar o tipo do que está sendo recebido, além disso, antes do tipo da struct é preciso passar um ponteiro representado por “*”, ele permite que ele altere o valor do campo e este valor seja exibido independente do escopo que for chamado. Já a função get sempre retorna um valor e é preciso especificar o tipo do que está sendo retornado. Agora na função principal é só chamar as funções set e executar o programa. Veja a seguir:

```go
package main

import (
    "POO/progexemplo"
)

func main() {
    usuario := new(progexemplo.Usuario)

    usuario.SetEmailDestinatario("destinatario@gmail.com")
    usuario.SetEmailRemetente("remetente@gmail.com")

    usuario.EnviarEmail()
}
```

```
PS C:\Users\user\Desktop\escola\TCC\progexemplo> go run main.go
Foi enviado um email de: ' remetente@gmail.com ' para: ' destinatario@gmail.com '
```

Visto isso, já podemos apresentar os próximos 4 projetos que vamos desenvolver durante este capítulo. Utilizaremos a interface gráfica para exibir as informações na tela, portanto iremos misturar conceitos que vimos no capítulo passado.

Cada exercício terá:
Uma package bll que irá ser responsável pela validação dos dados

Uma package erro que irá tratar se há algum erro na entrada de dados e se caso existir, retornará uma mensagem de erro.

Uma package que guardará a struct que desenvolveremos variando de projeto para projeto

Um package main que será responsável por importar e manipular todas as informações das outras packages e exibir na tela o que for proposto.

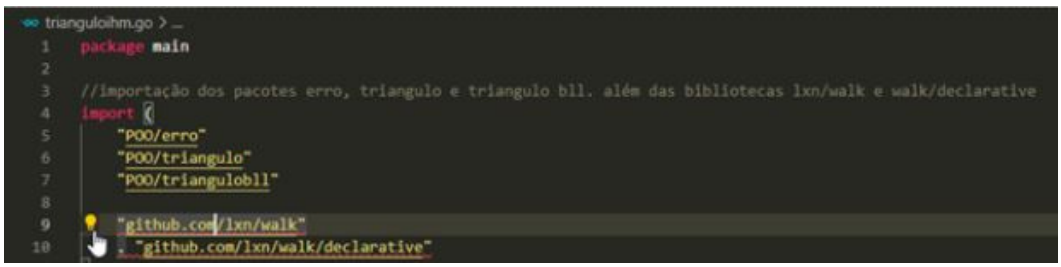
- Calcular a área de um triângulo

Nesse programa o usuário deverá entrar com o valor da base e da altura e quando ele clicar no botão “calcular” terá que ser exibido a área do triângulo. Vejamos como fazer isto:

Primeiramente iremos dividir o programa com 3 pastas distintas para definirmos seus packages e tornarmos o código reutilizável e mais legível. Teremos 3 packages além do main: erro, triangulo e triangulobll.

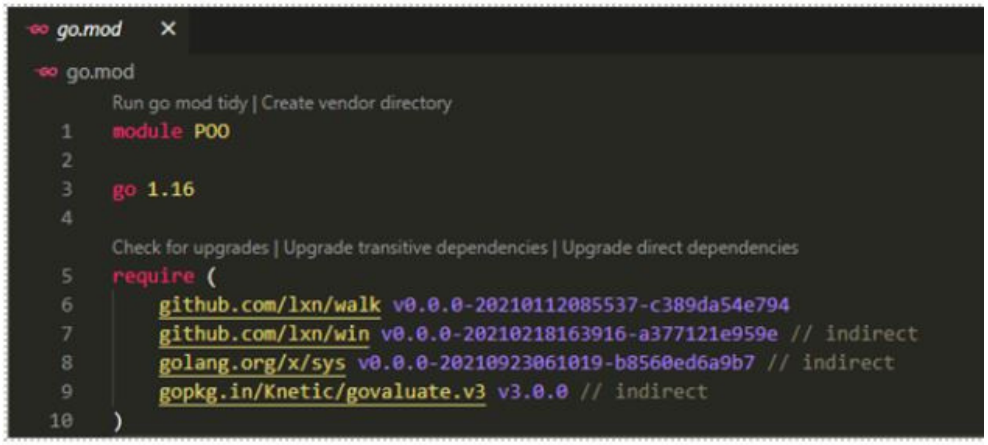
Após isso, precisaremos escrever no terminal "go mod init nomeDaPastaImaginária" para podermos conectar os packages. No nosso caso daremos o nome da pasta imaginária de "POO".

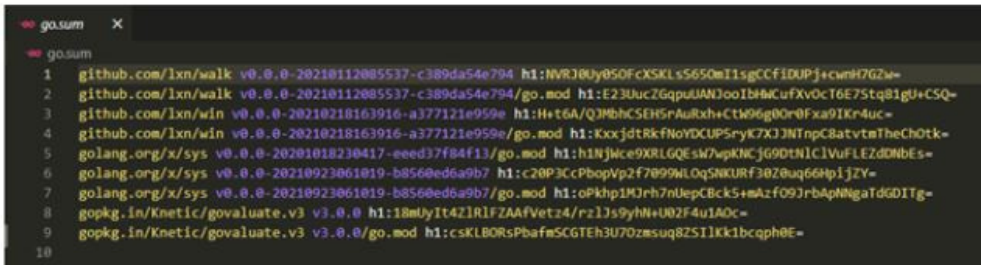
Mas, você perceberá que mesmo assim ainda haverá um problema: o código referente a parte gráfica estará dando erro. Isso se dá pela falta de um arquivo .sum e de um código extra no nosso arquivo mod. Caso você esteja utilizando o Visual Studio Code, perceberá que há uma lâmpada amarela ao lado da importação da biblioteca lxn/walk. Clicaremos nela



Após clicarmos na lâmpada, teremos uma opção "go get package github.com/lxn/walk". Clicaremos nela.

Perceberemos que houve uma alteração no arquivo .mod e que um arquivo .sum foi criado.





erro.go

Agora editaremos o arquivo erro.go: criaremos uma struct Erro e 4 funções públicas: GetErro, GetMens, SetErro e SetMens. Esse package será usado em todos os programas do capítulo, pois o código será o mesmo para todos, pois irá armazenar se houve erro e a sua mensagem, sendo o triangulobll responsável por esta tarefa.

As funções SetErro e SetMens têm como parâmetro um ponteiro da struct Erro para possibilitar a alteração direta na memória, independente do escopo: possibilitando as variáveis "mens" e "erro" serem alteradas através de outros arquivos/pacotes.

```go
package erro

type Erro struct {
    mens string
    erro bool
}

func (e *Erro) SetErro(erro bool) {
    e.erro = erro
}

func (e Erro) GetErro() bool {
    return e.erro
}

func (e *Erro) SetMens(mens string) {
    e.mens = mens
}

func (e Erro) GetMens() string {
    return e.mens
}
```

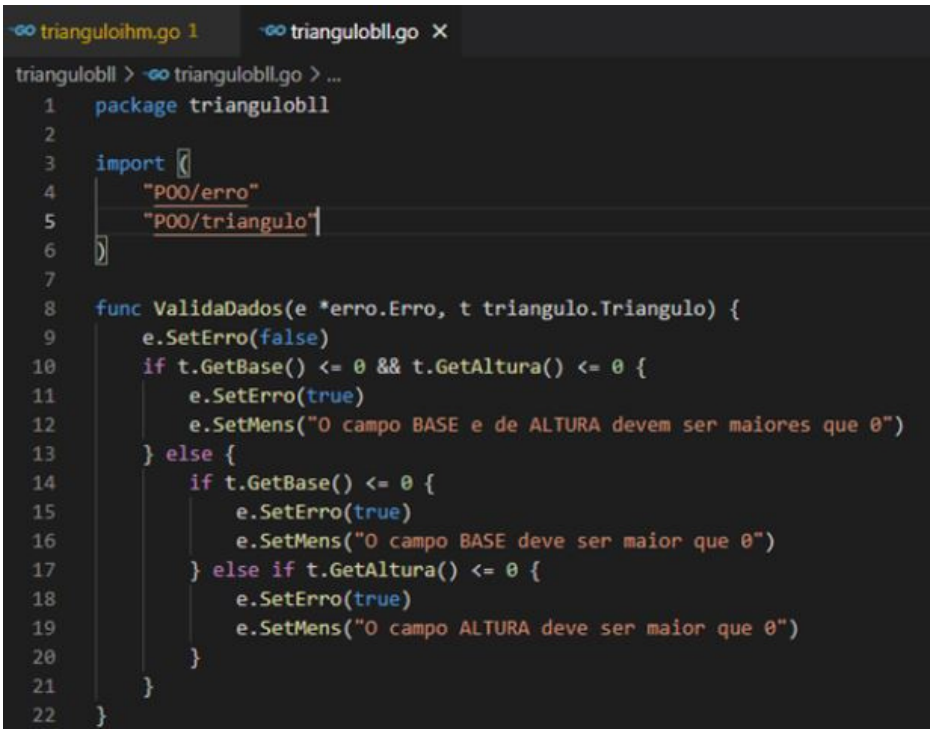
triangulo.go

Já no arquivo triangulo.go teremos uma struct Triangulo com dois campos float: altura e base.
Seguindo a regra do encapsulamento, nas funções teremos o: SetAltura, GetAltura, SetBase, GetBase, todas públicas, para que o usuário não altere e nem pegue diretamente os valores dos campos. Além claro do getArea, responsável pelo cálculo da área. Todas tendo como parâmetro uma variável do tipo struct Triangulo, mas as funções "Set" terão como parâmetro o ponteiro dessa struct para poderem alterar o valor sendo chamada em qualquer arquivo.

```go
package triangulo

type Triangulo struct {
    altura float64
    base   float64
}

func (t *Triangulo) SetAltura(altura float64) {
    t.altura = altura
}

func (t Triangulo) GetAltura() float64 {
    return t.altura
}

func (t *Triangulo) SetBase(base float64) {
    t.base = base
}

func (t Triangulo) GetBase() float64 {
    return t.base
}

func (t Triangulo) GetArea() float64 {
    return t.base * t.altura / 2
}
```

triangulobll.go

Agora iremos para o arquivo triangulobll.go. Nele teremos de importar os packages erro e triangulo para termos suas funções funcionando perfeitamente.
Uma função para validar os dados recebidos, que recebe como parâmetro uma variável da struct Erro que tem sua origem no package erro e uma variável da struct Triangulo que tem sua origem no package triangulo. Percebemos a relação de funções de diferentes packages em um mesmo arquivo, mesmo este não sendo o arquivo principal. Basicamente nessa função obrigamos o usuário a enviar dados da base e altura do triângulo sendo maiores que 0, caso contrário ele setará nos campos da struct Erro uma mensagem de erro e dirá que houve erro passando o valor “true”.



trianguloihm.go

Agora iremos para o arquivo principal: trianguloihm.go. Onde utilizaremos tudo que foi feito nos outros packages, apenas manipulando os dados que foram inseridos pelo usuário. Começamos importando todos os packages da pasta e também as bibliotecas para utilização de interface gráfica.

```go
package main

//importação dos pacotes erro, triangulo e triangulobll, além das bibliotecas lxn/walk e walk/declarative
import (
    "POO/erro"
    "POO/triangulo"
    "POO/triangulobll"

    "github.com/lxn/walk"
    . "github.com/lxn/walk/declarative"
)

func main() {
    //definindo valores para os elementos gráficos do programa
    var altura *walk.NumberEdit
    var base *walk.NumberEdit
    var resultado *walk.NumberEdit
    var calcular *walk.PushButton
    var form *walk.MainWindow

    MainWindow{
        AssignTo: &form,
        Title:    "Calculo Área",
        Size:     Size{250, 200},
        Layout:   Grid{Columns: 2},
        Font:     Font{Family: "Arial", PointSize: 14},
```

```go
        Children: []Widget{
            Label{
                Text: "Base: ",
            },

            NumberEdit{
                AssignTo: &base,
            },

            Label{
                Text: "Altura: ",
            },

            NumberEdit{
                AssignTo: &altura,
            },

            //botão calcular
            PushButton{
                AssignTo: &calcular,
                Text:     "Calcular",
                // assim que for clicado, a função será acionada...
                OnClicked: func() {
                    // instanciando uma variável do tipo Triangulo e do tipo Erro.
                    // ambas sendo referenciadas por seus packages de origem.
                    triangulo := new(triangulo.Triangulo)
                    erro := new(erro.Erro)
```

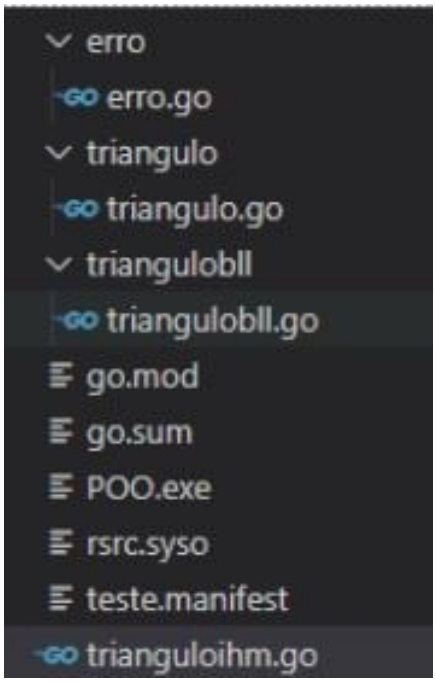
Após a construção gráfica, teremos pouco trabalho escrito e um código muito legível por conta da utilização de packages distintos junto de suas funções.

```go
                    // chamando as funções SetBase e SetAltura da variável tipo Triangulo do package triangulo
                    // e definindo os valores da base e da altura do triângulo
                    // tendo como parâmetro o que foi escrito
                    // nas caixas de número que são referenciadas pelas variáveis "base" e "altura"
                    triangulo.SetBase(base.Value())
                    triangulo.SetAltura(altura.Value())

                    // chamando a função ValidaDados do package triangulobll e usando como parâmetro
                    // os ponteiros das variáveis tipo Erro e Triangulo, respectivamente.
                    triangulobll.ValidaDados(*erro, *triangulo)

                    // se a função GetErro retornar true, uma mensagem de erro será exibida na tela.
                    // a mensagem exibida vem da função GetMens
                    // função esta que recebe essa mensagem através do pacote triangulobll
                    // por meio da função SetMens
                    if erro.GetErro() {
                        walk.MsgBox(form, "Erro!", erro.GetMens(), walk.MsgBoxIconError)
                        // caso a função GetErro retorne false, o valor da caixa de número referente à
                        // área do triângulo terá como valor o resultado da função GetArea que retorna a área do triângulo criado.
                        // além disso, os campos de base e altura serão desativados.
                    } else {
                        resultado.SetValue(triangulo.GetArea())
                        base.SetEnabled(false)
                        altura.SetEnabled(false)
                    }
                },
```

```
            // botão limpar
            PushButton{
                Text: "Limpar",
                // após ser clicado, a função será acionada...
                OnClicked: func() {
                    // os campos referentes à altura, base, resultado, e altura terão alterações: valores resetados
                    // e botões reativados.
                    altura.SetValue(0)
                    base.SetValue(0)
                    resultado.SetValue(0)
                    base.SetEnabled(true)
                    altura.SetEnabled(true)
                },
            },

            Label{
                Text: "Área: ",
            },

            // campo onde aparece a área do triângulo.
            // nunca poderá editado e/ou clicado.
            NumberEdit{
                AssignTo: &resultado,
                ReadOnly: true,
                Enabled:  false,
            },
        },
    }.Run()
}
```

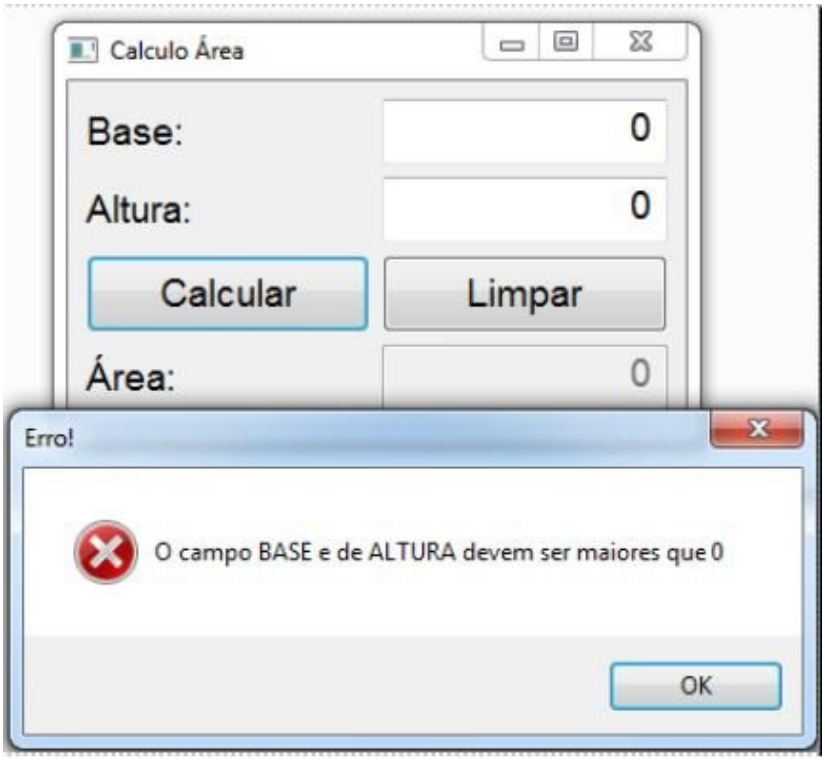
Não se esqueça de gerar o arquivo .manifest e o rsrc.syso. Caso não se lembre é só voltar no capítulo 3 onde ensinamos detalhadamente como criar.

Entre no terminal e execute o comando “go build”. Será gerado um arquivo .exe com o nome que digitamos no comando “go mod init”. A estrutura final do nosso diretório no final deverá se apresentar da seguinte maneira:

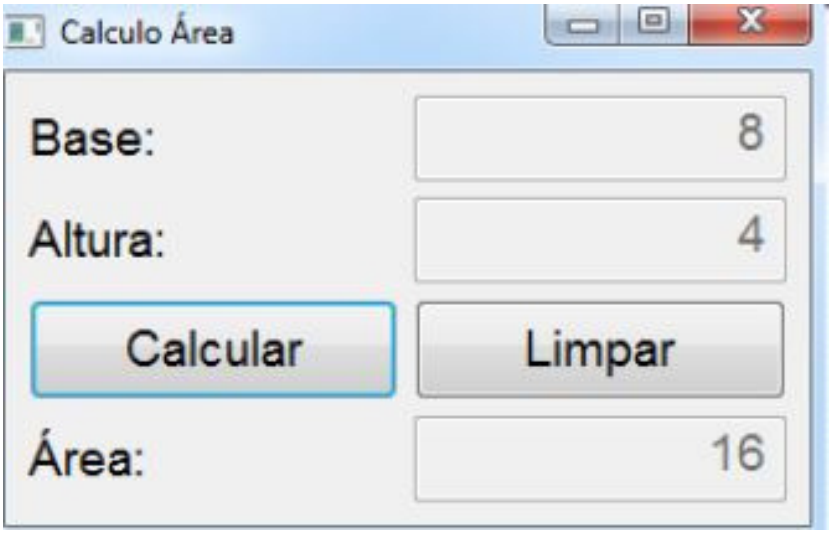


Agora vamos para as telas de execução!!!

Execução da aplicação quando deve resultar em erro:



Execução quando o usuário insere os valores da base e da altura corretamente:



Importante!!!

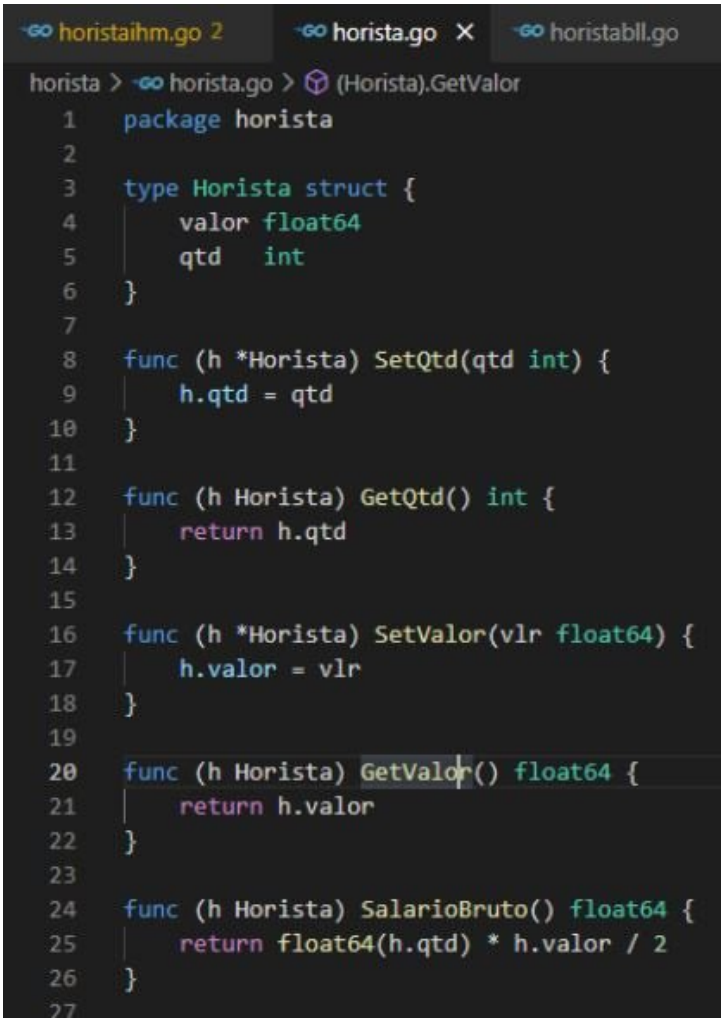
A partir daqui não entraremos em tantos detalhes, pois muita coisa a partir de agora se torna repetitiva. Os programas apesar de serem diferentes possuem muitas coisas em comum. Não esqueça de criar o arquivo go.mod e go.sum para os próximos programas, pois são muito importantes para o funcionamento das aplicações, além disso não esqueça de reutilizar o arquivo erro.go, pois não o mostraremos novamente.

- Calcular o salário de um professor horista

Nesse programa o usuário deverá entrar com a quantidade de horas e o valor recebido para cada hora. Quando o botão calcular for clicado, o salário deve ser exibido. Veja a seguir o código de cada arquivo:

horista.go

Aqui teremos uma struct “Horista” pública, com os campos privados “valor” (correspondente ao valor da hora) e qtd (correspondente à quantidade de horas trabalhadas). Os métodos get e set, seguindo as regras do encapsulamento. Além do método responsável por retornar o salário bruto.



horistabll.go

Neste arquivo temos uma função que será responsável por receber a instância da struct Horista com todos os valores definidos, e verificar se os valores digitados pelo usuário estão corretos, enviando informações ao package erro a fim de definir uma mensagem de erro, caso exista uma. Neste caso a mensagem de erro será definida somente se os valores forem maiores que 0.

```go
package horistabll

import (
    "POO/erro"
    "POO/horista"
)

func ValidaDados(e *erro.Erro, h horista.Horista) {
    e.SetErro(false)

    if h.GetQtd() <= 0 && h.GetValor() <= 0 {
        e.SetErro(true)
        e.SetMens("O campo QUANTIDADE DE HORAS e VALOR DA HORA devem ser maiores que 0")
    } else {
        if h.GetQtd() <= 0 {
            e.SetErro(true)
            e.SetMens("O campo QUANTIDADE DE HORAS deve ser maior que 0")
        } else if h.GetValor() <= 0 {
            e.SetErro(true)
            e.SetMens("O campo VALOR DA HORA deve ser maior que 0")
        }
    }
}
```

trianguloihm.go

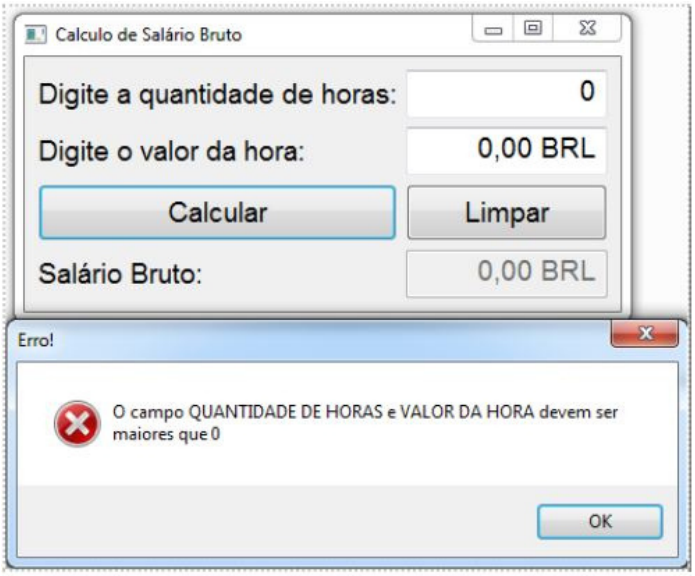
O trianguloihm.go será responsável por manipular as outras packages e exibir na tela o resultado caso esteja tudo certo a área do triângulo

```go
package main

import (
    "POO/erro"
    "POO/horista"
    "POO/horistabll"

    "github.com/lxn/walk"
    . "github.com/lxn/walk/declarative"
)

func main() {
    var qtdH, vlrH, Salario *walk.NumberEdit
    var form *walk.MainWindow

    MainWindow{
        AssignTo: &form,
        Title:    "Calculo de Salário Bruto",
        Size:     Size{400, 200},
        Layout:   Grid{Columns: 2},
        Font:     Font{Family: "Arial", PointSize: 14},
        Children: []Widget{
            Label{
                Text: "Digite a quantidade de horas:",
            },
            NumberEdit{
                AssignTo: &qtdH,
            },

            Label{
                Text: "Digite o valor da hora:",
            },
```

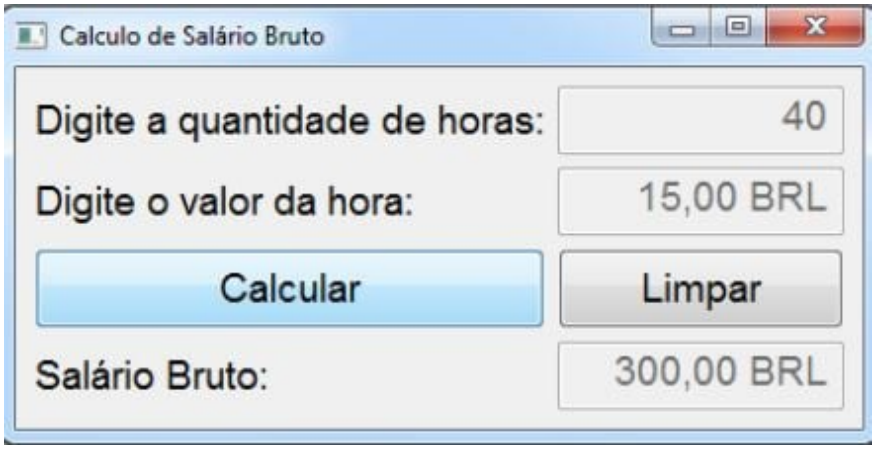
```
        NumberEdit{
            AssignTo: &vlrH,
            Suffix:   " BRL",
            Decimals: 2,
        },

        PushButton{
            Text: "Calcular",
            OnClicked: func() {

                // Criando instâncias das structs Erro e Horista
                erro := new(erro.Erro)
                horista := new(horista.Horista)

                // Setando o que os valores digitados pelo usuário nos campos da struct Horista
                horista.SetQtd(int(qtdH.Value()))
                horista.SetValor(vlrH.Value())

                // Validando se o que o usuário digitou está correto
                horistabll.ValidaDados(erro, *horista)

                // Caso esteja errado será exibido uma mensagem de erro, caso contrário será exibido o salário na tela
                if erro.GetErro() {
                    walk.MsgBox(form, "Erro!", erro.GetMens(), walk.MsgBoxStyle(walk.MsgBoxIconError))
                } else {
                    Salario.SetValue(horista.SalarioBruto())
                    qtdH.SetEnabled(false)
                    vlrH.SetEnabled(false)
                }
            },
        },
```

```
        PushButton{
            Text: "Limpar",
            OnClicked: func() {
                qtdH.SetValue(0)
                vlrH.SetValue(0)
                Salario.SetValue(0)
                vlrH.SetEnabled(true)
                qtdH.SetEnabled(true)
            },
        },

        Label{
            Text: "Salário Bruto: ",
        },
        NumberEdit{
            AssignTo: &Salario,
            Suffix:   " BRL",
            Decimals: 2,
            Enabled:  false,
        },
    },
}.Run()
```

Execução quando o usuário insere dados incorretamente:



Execução quando os dados são inseridos corretamente:

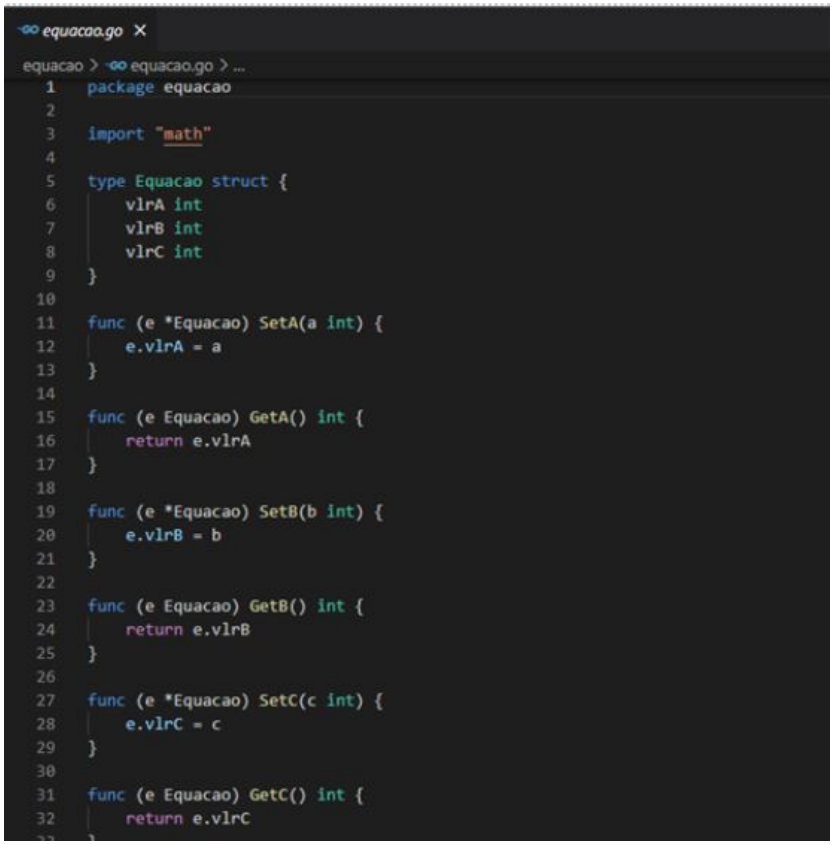


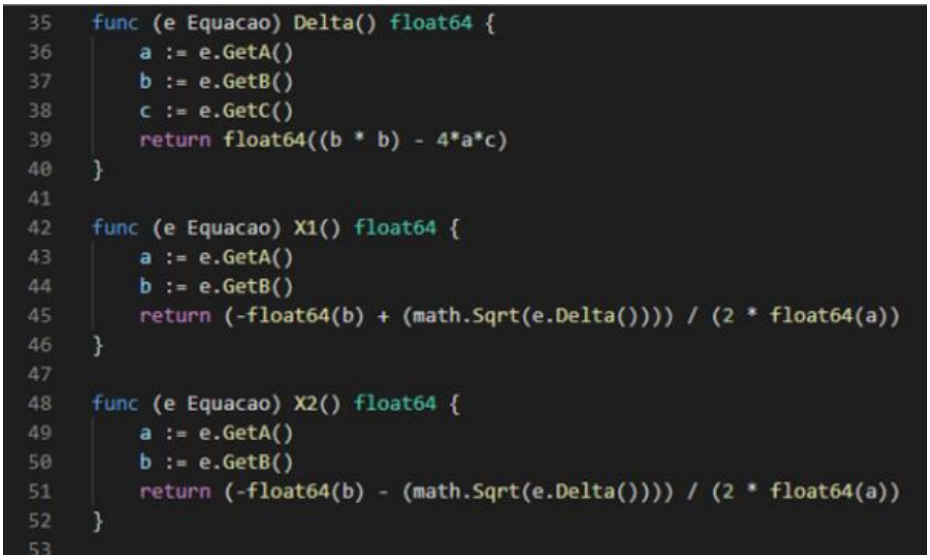
- Equação de segundo grau

Nesse programa o usuário deverá entrar com o valor de a, b e c de uma equação de segundo grau. Quando o usuário clicar no botão calcular deverá ser exibido na tela o valor do x1 e do x2. Veja a seguir os programas:

equação.go

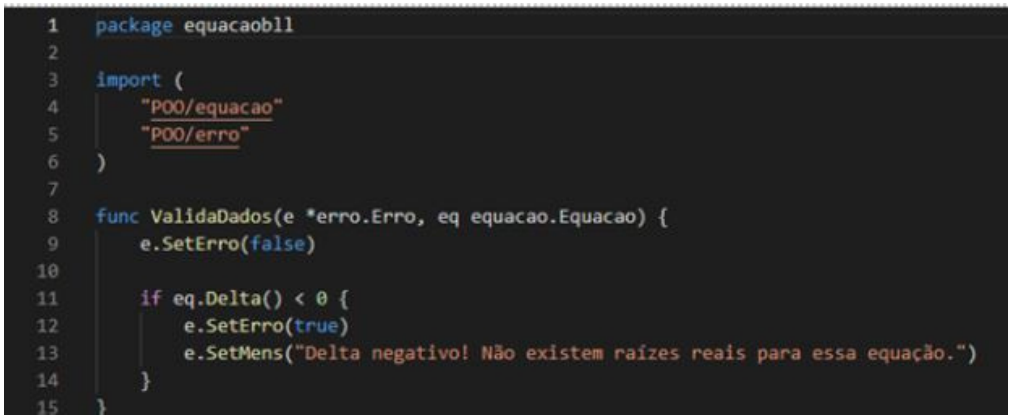
Aqui teremos uma struct “Equacao” pública, com os campos privados “a”, “b” e “c”. Contendo também os métodos get e set, seguindo as regras do encapsulamento, para que não seja possível acessar diretamente os campos presentes nas structs. Além do método responsável por retornar o delta, x1 e o x2. Cabe ressaltar é o uso do retorno do delta presente nos métodos x1 e x2, algo que ainda não tínhamos visto.





equacaobll.go

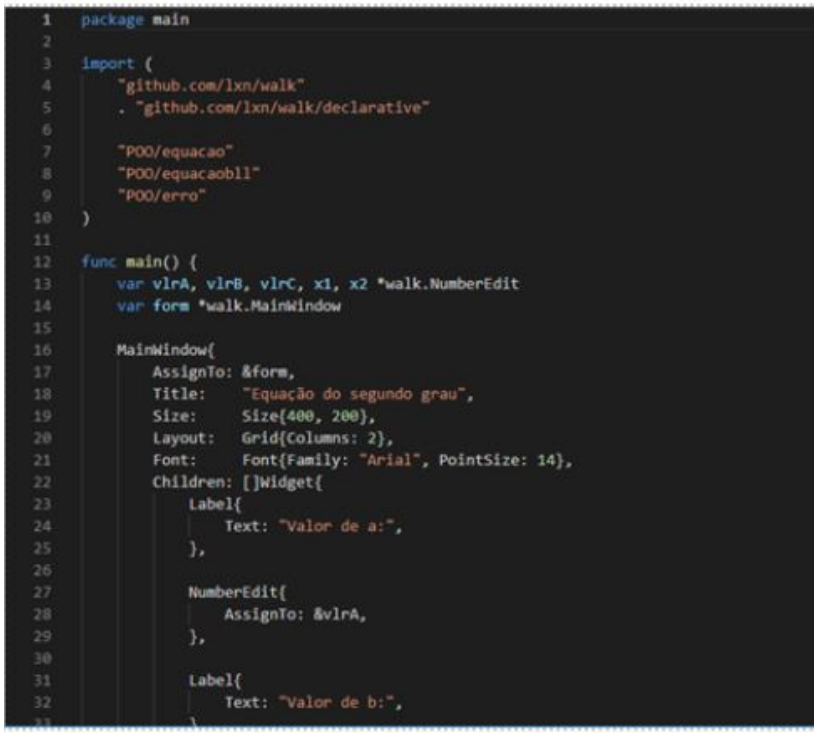
No arquivo equacaobll.go setaremos uma mensagem de erro na struct Erro, apenas se o delta for negativo, pois os valores de “a”, “b” e “c” neste caso podem ser menores que 0.



```go
package equacaobll

import (
    "POO/equacao"
    "POO/erro"
)

func ValidaDados(e *erro.Erro, eq equacao.Equacao) {
    e.SetErro(false)

    if eq.Delta() < 0 {
        e.SetErro(true)
        e.SetMens("Delta negativo! Não existem raízes reais para essa equação.")
    }
}
```

equacaoihm.go

O equacaoihm.go será responsável por manipular os métodos das outras packages e exibir na tela o resultado, caso esteja tudo certo com o x1 e o x2.



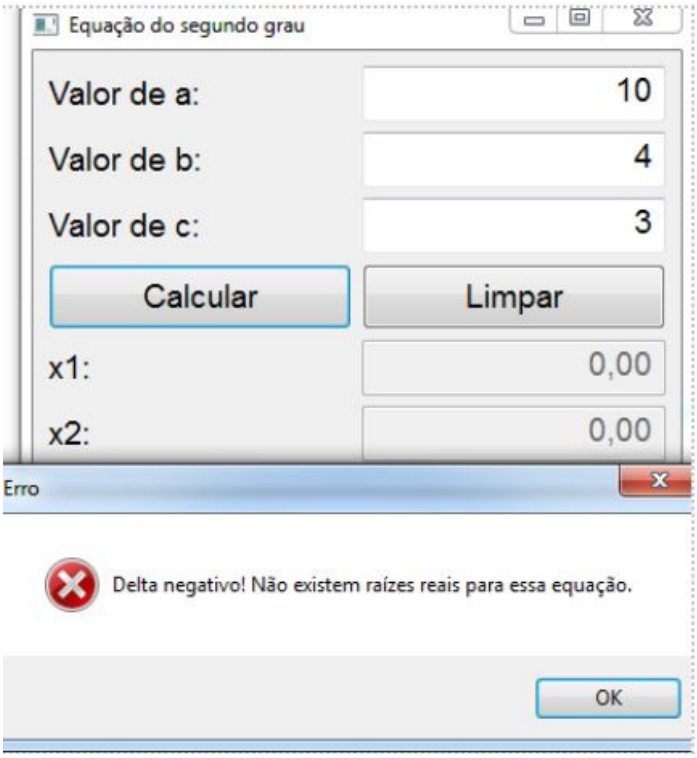
```go
package main

import (
    "github.com/lxn/walk"
    . "github.com/lxn/walk/declarative"

    "POO/equacao"
    "POO/equacaobll"
    "POO/erro"
)

func main() {
    var vlrA, vlrB, vlrC, x1, x2 *walk.NumberEdit
    var form *walk.MainWindow

    MainWindow{
        AssignTo: &form,
        Title:    "Equação do segundo grau",
        Size:     Size{400, 200},
        Layout:   Grid{Columns: 2},
        Font:     Font{Family: "Arial", PointSize: 14},
        Children: []Widget{
            Label{
                Text: "Valor de a:",
            },

            NumberEdit{
                AssignTo: &vlrA,
            },

            Label{
                Text: "Valor de b:",
```

```
equacaoihm.go 2
equacaoihm.go > main
34
35              NumberEdit{
36                  AssignTo: &vlrB,
37              },
38
39              Label{
40                  Text: "Valor de c:",
41              },
42
43              NumberEdit{
44                  AssignTo: &vlrC,
45              },
46
47              PushButton{
48                  Text: "Calcular",
49                  OnClicked: func() {
50                      // Instanciando structs
51                      erro := new(erro.Erro)
52                      equacao := new(equacao.Equacao)
53
54                      // Setando valores nos campos da struct Equacao
55                      equacao.SetA(int(vlrA.Value()))
56                      equacao.SetB(int(vlrB.Value()))
57                      equacao.SetC(int(vlrC.Value()))
58
59                      // Validando dados inseridos pelo usuário
60                      equacaobll.ValidaDados(erro, *equacao)
```

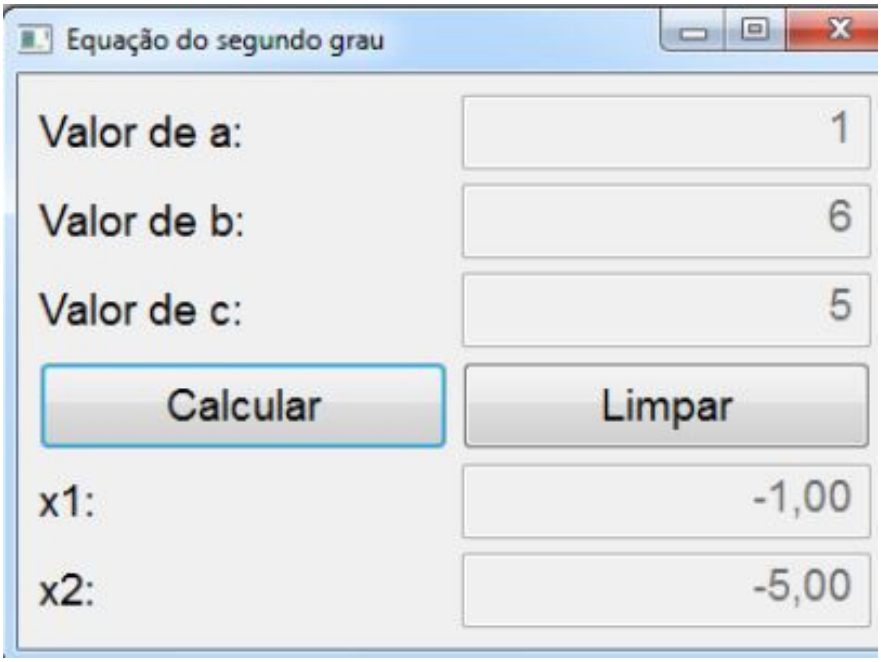
```
equacaoihm.go 2
equacaoihm.go > main
62                  /* Caso os dados inseridos estejam corretos será exibido o valor de x1 e x2,
63                  caso contrário ele mostrará uma mensagem de erro que foi setado na struct Erro
64                  pela método ValidaDados presente no package equacaobll */
65                  if erro.GetErro() {
66                      walk.MsgBox(form, "Erro", erro.GetMens(), walk.MsgBoxStyle(walk.MsgBoxIconError))
67                  } else {
68                      x1.SetValue(equacao.X1())
69                      x2.SetValue(equacao.X2())
70                      vlrA.SetEnabled(false)
71                      vlrB.SetEnabled(false)
72                      vlrC.SetEnabled(false)
73                  }
74              },
75          },
76
77          PushButton{
78              Text: "Limpar",
79              OnClicked: func() {
80                  vlrA.SetValue(0)
81                  vlrB.SetValue(0)
82                  vlrC.SetValue(0)
83                  x1.SetValue(0)
84                  x2.SetValue(0)
85                  vlrA.SetEnabled(true)
86                  vlrB.SetEnabled(true)
87                  vlrC.SetEnabled(true)
88              },
89          },
90
```

```
91              Label{
92                  Text: "x1: ",
93              },
94              NumberEdit{
95                  AssignTo: &x1,
96                  Enabled:  false,
97                  Decimals: 2,
98              },
99
100             Label{
101                 Text: "x2: ",
102             },
103             NumberEdit{
104                 AssignTo: &x2,
105                 Enabled:  false,
106                 Decimals: 2,
107             },
108         },
```

Tela de execução quando os dados são inseridos incorretamente:



Tela de execução quando os dados são inseridos corretamente:

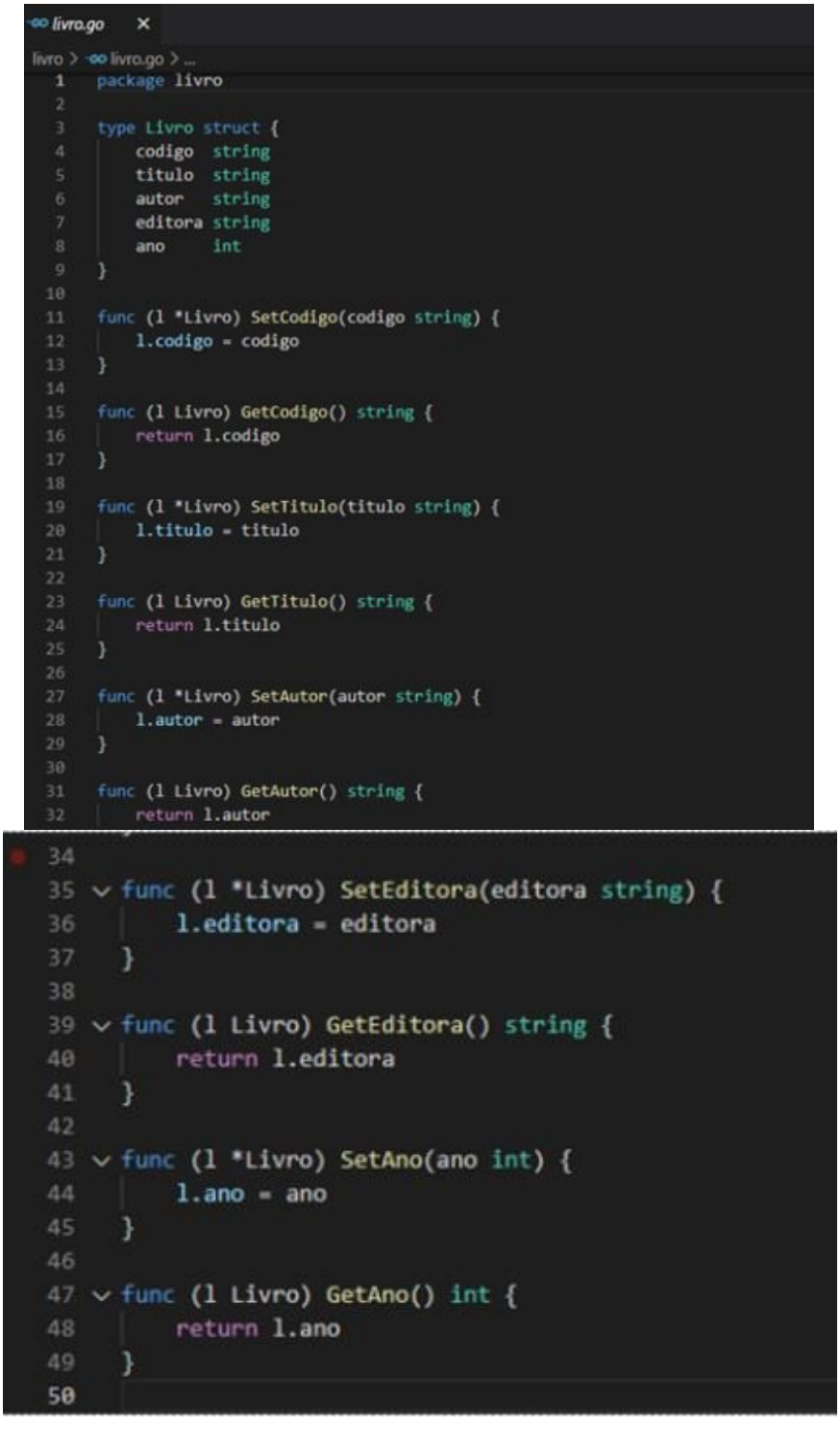


- Cadastrar livros

Nesse programa, o usuário deverá entrar com titulo, código, autor, editora e o ano. Após a entrada quando ele clicar no botão cadastrar, os dados do livro serão guardados. Também terá um botão consultar, este quando for clicado e o código inserido for igual ao do livro cadastrado, será exibido todos os dados dos livros cadastrados. Veja a seguir os arquivos:

livro.go

Aqui teremos uma struct “Livro” pública, com os campos privados “código”, “titulo” e “autor”, “editora” e “ano”. Contendo também os métodos get e set, seguindo as regras do encapsulamento, para que não seja possível acessar diretamente os campos presentes nas structs. Este programa não terá nenhum método adicional, pois a intenção é apenas usar as funções get para preencher os campos de texto quando o botão “consultar” for clicado.



```go
package livro

type Livro struct {
    codigo  string
    titulo  string
    autor   string
    editora string
    ano     int
}

func (l *Livro) SetCodigo(codigo string) {
    l.codigo = codigo
}

func (l Livro) GetCodigo() string {
    return l.codigo
}

func (l *Livro) SetTitulo(titulo string) {
    l.titulo = titulo
}

func (l Livro) GetTitulo() string {
    return l.titulo
}

func (l *Livro) SetAutor(autor string) {
    l.autor = autor
}

func (l Livro) GetAutor() string {
    return l.autor
}

func (l *Livro) SetEditora(editora string) {
    l.editora = editora
}

func (l Livro) GetEditora() string {
    return l.editora
}

func (l *Livro) SetAno(ano int) {
    l.ano = ano
}

func (l Livro) GetAno() int {
    return l.ano
}
```

livrobll.go

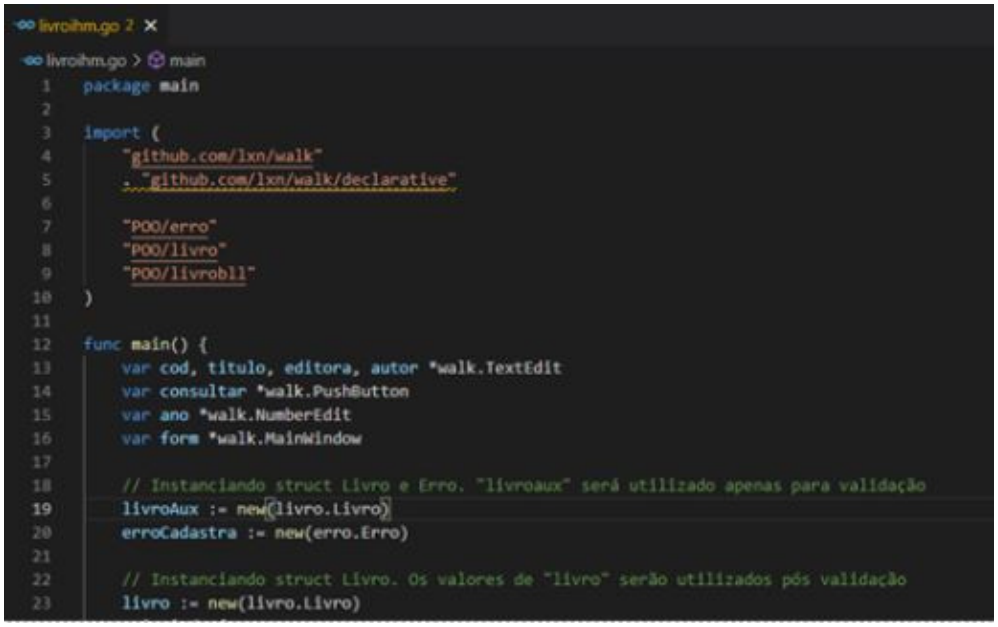
Aqui diferentemente dos outros programas teremos duas funções. Uma será responsável por validar os dados inseridos quando o botão cadastrar for clicado. A outra vai validar o código digitado, assim verificando se é igual ou não ao código cadastrado.

```go
package livrobll

import (
    "POO/erro"
    "POO/livro"
)

func ValidaDados(e *erro.Erro, l livro.Livro) {
    e.SetErro(false)

    if l.GetCodigo() == "" {
        e.SetMens("O campo código é de preenchimento obrigatório")
        e.SetErro(true)
        return
    }

    if l.GetTitulo() == "" {
        e.SetMens("O campo titulo é de preenchimento obrigatório")
        e.SetErro(true)
        return
    }

    if l.GetAutor() == "" {
        e.SetMens("O campo autor é de preenchimento obrigatório")
        e.SetErro(true)
        return
    }

    if l.GetEditora() == "" {
        e.SetMens("O campo editora é de preenchimento obrigatório")
        e.SetErro(true)
        return
```

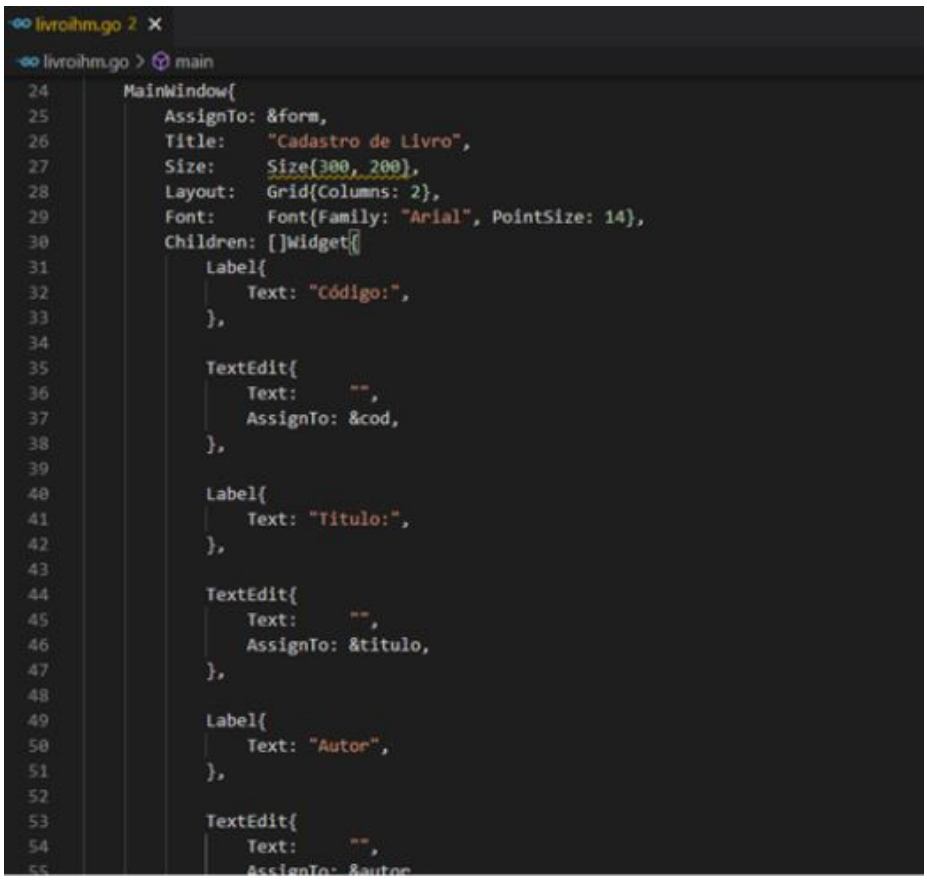
```go

    if l.GetAno() <= 0 {
        e.SetMens("O campo ano tem de ser maior do que 0")
        e.SetErro(true)
        return
    }
}

func ValidaCodigo(e *erro.Erro, l livro.Livro, cod string) {
    e.SetErro(false)
    if cod == "" {
        e.SetMens("O campo código é de preenchimento obrigatório")
        e.SetErro(true)
    } else {
        if l.GetCodigo() != cod {
            e.SetMens("Não há nenhum livro cadastrado com este código")
            e.SetErro(true)
        }
    }
}
```

livroihm.go

O equacaoihm.go será responsável por manipular os métodos das outras packages e exibir na tela conforme o que for digitado estiver corretamente ou erroneamente.



```go
package main

import (
    "github.com/lxn/walk"
    . "github.com/lxn/walk/declarative"

    "POO/erro"
    "POO/livro"
    "POO/livrobll"
)

func main() {
    var cod, titulo, editora, autor *walk.TextEdit
    var consultar *walk.PushButton
    var ano *walk.NumberEdit
    var form *walk.MainWindow

    // Instanciando struct Livro e Erro. "livroaux" será utilizado apenas para validação
    livroAux := new(livro.Livro)
    erroCadastra := new(erro.Erro)

    // Instanciando struct Livro. Os valores de "livro" serão utilizados pós validação
    livro := new(livro.Livro)
```



```go
    MainWindow{
        AssignTo: &form,
        Title:    "Cadastro de Livro",
        Size:     Size{300, 200},
        Layout:   Grid{Columns: 2},
        Font:     Font{Family: "Arial", PointSize: 14},
        Children: []Widget{
            Label{
                Text: "Código:",
            },

            TextEdit{
                Text:     "",
                AssignTo: &cod,
            },

            Label{
                Text: "Titulo:",
            },

            TextEdit{
                Text:     "",
                AssignTo: &titulo,
            },

            Label{
                Text: "Autor",
            },

            TextEdit{
                Text:     "",
```

```
58      Label{
59          Text: "Editora",
60      },
61
62      TextEdit{
63          Text:      "",
64          AssignTo: &editora,
65      },
66
67      Label{
68          Text: "Ano",
69      },
70
71      NumberEdit{
72          AssignTo:           &ano,
73          RightToLeftReading: false,
74      },
75
76      PushButton{
77          ColumnSpan: 2,
78          Text:       "Cadastrar",
79          OnClicked: func() {
80              // Setando valores no "livroaux"
81              livroAux.SetTitulo(titulo.Text())
82              livroAux.SetAutor(autor.Text())
83              livroAux.SetCodigo(cod.Text())
84              livroAux.SetEditora(editora.Text())
85              livroAux.SetAno(int(ano.Value()))
```

livroihm.go 2 ×

livroihm.go > main

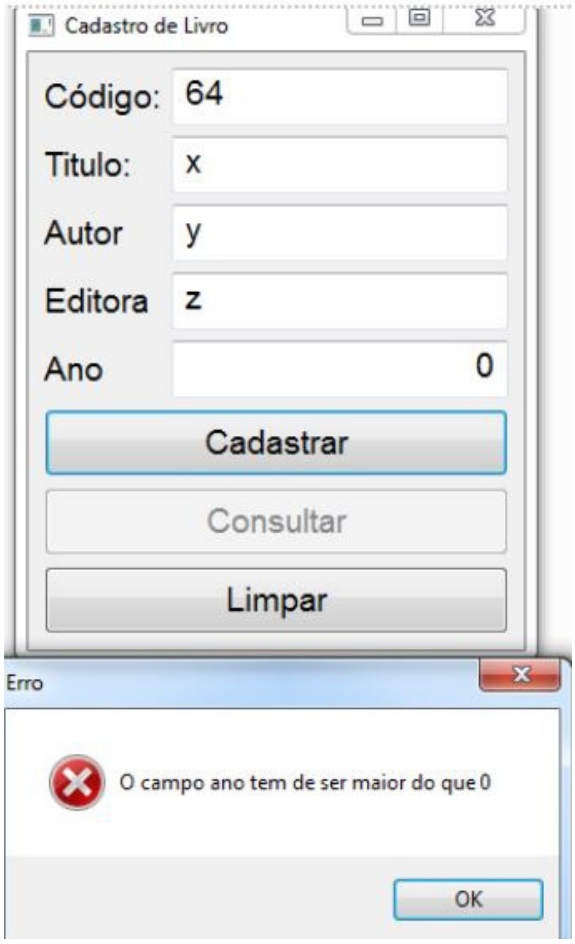
```
87              // Verificando se o que foi digitado está correto ou não
88              livrobll.ValidaDados(erroCadastra, *livroAux)
89
90              /* Caso esteja errado, será exibido uma mensagem de erro.
91              Caso contrário além de informar que os dados estão corretos, os dados de "livroaux" serão setados
92              na instância "livro" */
93              if erroCadastra.GetErro() {
94                  walk.MsgBox(form, "Erro", erroCadastra.GetMens(), walk.MsgBoxStyle(walk.MsgBoxIconError))
95              } else {
96                  consultar.SetEnabled(true)
97                  walk.MsgBox(form, "Boa!", "Os dados inseridos com sucesso!", walk.MsgBoxStyle(walk.MsgBoxIconInformation))
98                  livro.SetTitulo(livroAux.GetTitulo())
99                  livro.SetAutor(livroAux.GetAutor())
100                 livro.SetCodigo(livroAux.GetCodigo())
101                 livro.SetEditora(livroAux.GetEditora())
102                 livro.SetAno(livroAux.GetAno())
103             }
104         },
105     },
106
107     PushButton{
108         AssignTo:   &consultar,
109         ColumnSpan: 2,
110         Text:       "Consultar",
111         Enabled:    false,
112         OnClicked: func() {
113             // Criando nova instância da struct Erro
114             erro := new(erro.Erro)
115
116             // Verificará se o código digitado na caixa de texto é igual ao código do livro cadastrado.
117             livrobll.ValidaCodigo(erro, *livro, cod.Text())
118
```

livroihm.go 2 ×

livroihm.go > main

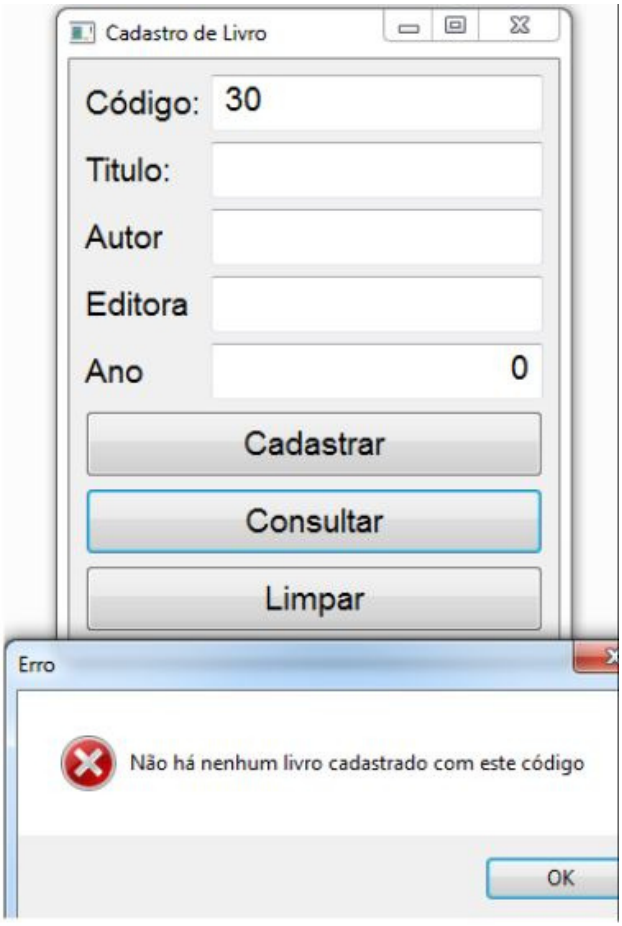
```
119             /* Se der tudo certo será mostrado os dados do livro cadastrado, senão será mostrado uma
120             mensagem de erro */
121
122             if erro.GetErro() {
123                 walk.MsgBox(form, "Erro", erro.GetMens(), walk.MsgBoxStyle(walk.MsgBoxIconError))
124             } else {
125                 cod.SetText(livro.GetCodigo())
126                 titulo.SetText(livro.GetTitulo())
127                 autor.SetText(livro.GetAutor())
128                 editora.SetText(livro.GetEditora())
129                 ano.SetValue(float64(livro.GetAno()))
130             }
131         },
132     },
133
134     PushButton{
135         ColumnSpan: 2,
136         Text:       "Limpar",
137         OnClicked: func() {
138             cod.SetText("")
139             titulo.SetText("")
140             autor.SetText("")
141             editora.SetText("")
142             ano.SetValue(0)
143         },
144     },
```

Tela de execução quando o botão cadastrar é clicado e os dados são inseridos incorretamente:

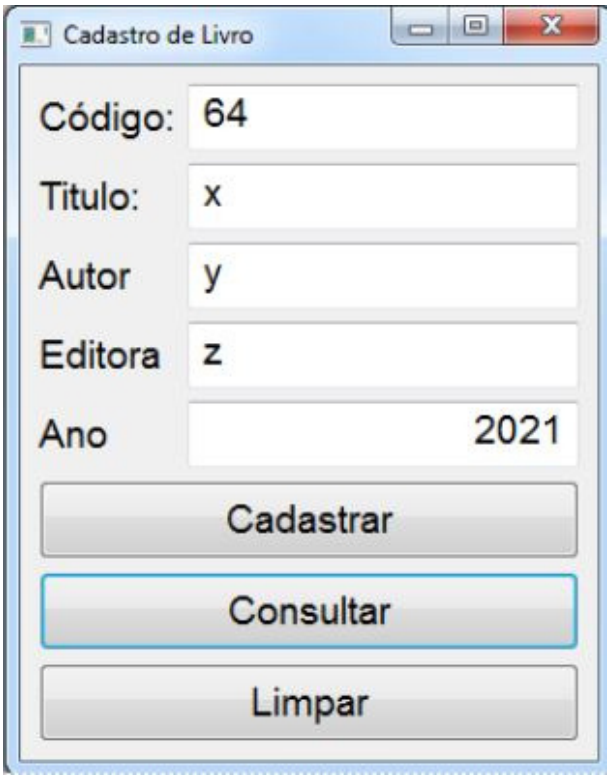


Tela de execução quando o botão cadastrar é clicado e os dados são inseridos corretamente:

Após cadastrar o livro com o código 64. Veja a tela de execução quando digitamos um código diferente do valor 64 e clicamos no botão “consultar”:



Tela de execução quando digitamos o código do livro cadastrado:



Acesso a banco de dados em golang

Neste capítulo iremos aprender a manipular um banco de dados usando golang através de um projeto que utilizará CRUD. Antes iremos abordar as ferramentas e as bibliotecas que iremos utilizar.

WampServer

Para começarmos, é necessário ter um aplicativo para simular um servidor virtual dentro de seu computador para que seja possível a conexão ao banco de dados.

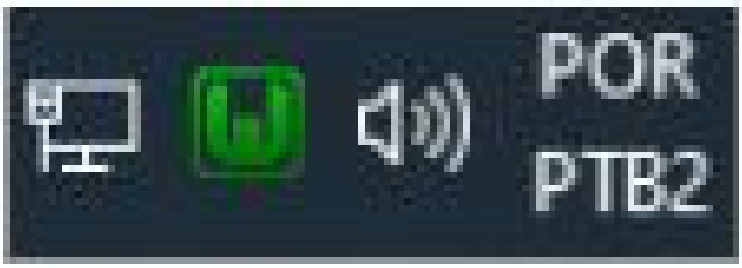
Utilizaremos o WampServer.
Link para download: https://www.wampserver.com

MySQL

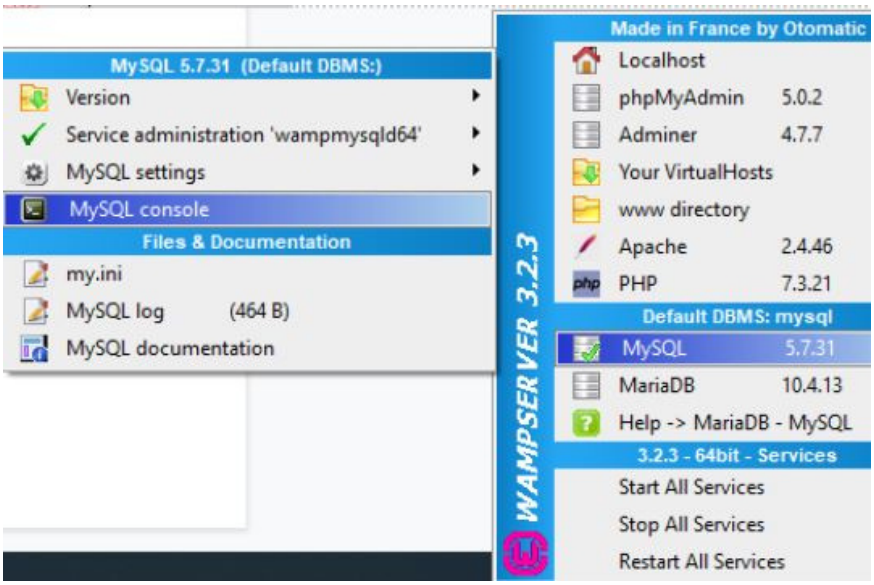
Após a instalação do WampServer, é necessário criar o banco de dados onde iremos guardar os dados. Para isso, usaremos comandos da linguagem SQL no MySQL.

Link para download: https://dev.mysql.com/downloads/installer/
Criando um banco de dados e deixando ele ativo em um servidor

Primeiramente clique no ícone dele que aparecerá no canto inferior direito após a instalação.

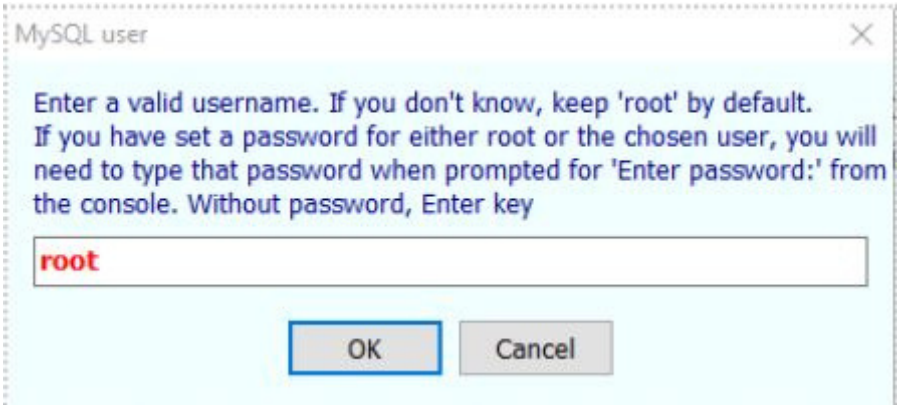


Clique com o botão esquerdo no ícone, passe o mouse sobre "MySQL" e clique em "MySQL console".



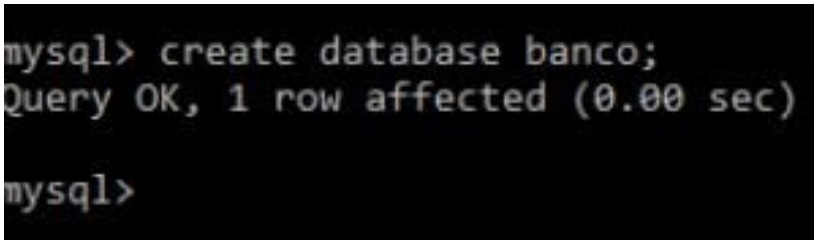
A depender da sua configuração, coloque seu login e senha de conexão. Caso não tenha feito nenhuma alteração, o login padrão é root e a senha é vazia (não há uma senha).

Entrando com o login:

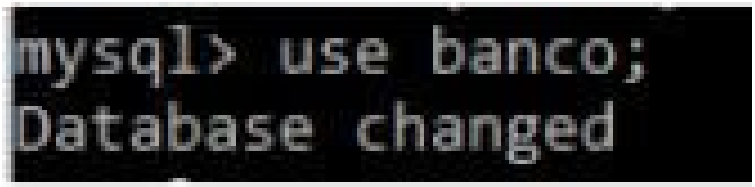


Entrando com a senha (no nosso caso vazia):

Caso queira iniciar um servidor já existente, é só executar o comando use database nome_do_banco. No nosso caso criaremos um banco de dados com o comando create database banco, pois o usaremos mais para frente a fim de mostrar como abrir uma conexão com ele.



Após a criação do banco, é necessário informar que queremos deixar ele ativo no servidor. Para isso usamos o comando dito ali em cima: use database nome_do_banco;



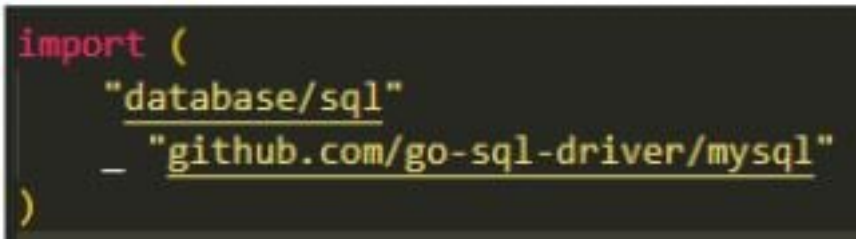
Agora, partiremos para o VSCode! Mostraremos como nos conectar a este banco de dados que criamos e já deixamos ativo.

Conexão com o banco através de GO
Instalando uma nova biblioteca

Para prosseguirmos, teremos que instalar mais uma biblioteca:
github.com/go-sql-driver/mysql

Como demonstramos em outro capítulo, entre no seu terminal e digite o comando "go get -u github.com/go-sql-driver/mysql" para poder instalar em sua máquina.
Após instalarmos a biblioteca, no arquivo em que queremos nos conectar teremos que importar dois pacotes:
database/sql

github.com/go-sql-driver/mysql (previamente instalado).



Abrindo conexão com o banco de dados criado

Depois de importarmos esses pacotes, agora é só ir para a função main e abrir uma conexão com o nosso “banco”. Para conectar-se ao banco previamente criado, teremos que utilizar uma função de um dos pacotes importados chamada "sql.Open".
Iniciaremos duas variáveis, onde uma referenciará o banco de dados e a outra receberá um erro, caso a conexão seja mal-sucedida.

```go
db, err := sql.Open("mysql", "root:@tcp(localhost:3306)/banco")
if err != nil {
    panic(err.Error())
}
```

Nesse caso, usamos uma variável com o nome "db" para referenciar o banco de dados e uma variável chamada "erroBD" para receber um possível erro de conexão. O modelo de conexão consiste em iniciar as duas variáveis e enviar o valor da função sql.Open("mysql", loginDoBancoDeDados:senhaDoBancoDeDados(portaDoBanco)/nomeDoBanco")

Após instalar as bibliotecas e entender como abrir uma conexão com o banco de dados, acompanhe a seguir o nosso projeto utilizando o banco de dados.

Golang em prática

Cadastro de Livros usando CRUD

O projeto CRUD consistirá em uma aplicação de cadastro de livros parecida com aquela que vimos no capítulo IV, porém este usará banco de dados. Teremos a opção de salvar (cadastrar) o livro, procurá-lo no sistema (ler) através do seu código, alterar informações (atualizar) e excluir do sistema (excluir). CRUD é o acrônimo da expressão do idioma Inglês, Create (Criação), Read (Consulta), Update (Atualização) e Delete (Destruição). Este acrônimo é comumente utilizado para definir as quatro operações básicas usadas em Banco de Dados Relacionais.

Criando o banco de dados e a tabela no MySQL Workbench

Dentro do MySQL Workbench criaremos um banco de dados chamado “cadastrolivro” e uma tabela chamada “livros” que será utilizada para armazenar os valores. Iremos inserir também uns li

```sql
create database cadastrolivro;

use cadastrolivro;

create table livros(
codigo varchar(10) unique,
titulo varchar(30) not null,
autor varchar(30) not null,
editora varchar(30) not null,
ano int not null
);

insert into livros values
('1', 'Dom Quixote', 'Miguel de Cervantes', 'Pinguim Companhia', '1605'),
('2', 'Guerra e Paz', 'Liev Tolstói', 'Companhia das Letras', '1869'),
('3', 'A montanha mágica', ' Thomas Mann', 'Companhia das Letras', '1924');
```

Após inserir os livros na tabela veja como ela ficou logo abaixo

```
select * from livros;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

codigo	titulo	autor	editora	ano
1	Dom Quixote	Miguel de Cervantes	Pinguim Companhia	1605
2	Guerra e Paz	Liev Tolstói	Companhia das Letras	1869
3	A montanha mágica	Thomas Mann	Companhia das Letras	1924

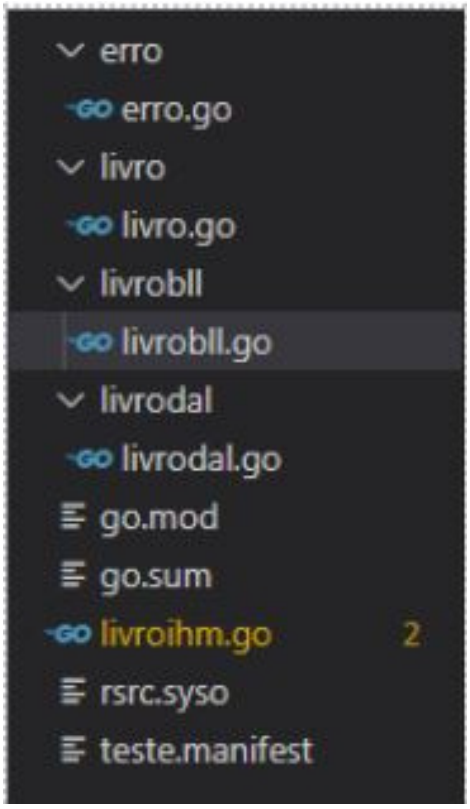
Após criarmos o banco de dados e a tabela, iremos deixar ele ativo em um servidor. Já vimos como fazer isso, logue no wampServer e digite o código a seguir, que irá deixar o nosso banco “cadastrolivro” ativo.

```
mysql> use cadastrolivro;
Database changed
```

Agora iremos ao VSCode criar a nossa aplicação, comentando cada parte do código, como de costume.

Codificando a nossa aplicação

A estrutura da nossa pasta ficará com a estrutura a seguir:



Lembre se que:

erro.go -> é o mesmo arquivo que utilizamos no projeto passado, portanto não mostraremos o código novamente. Ele continuará guardando a mensagem e o valor do erro em uma struct.

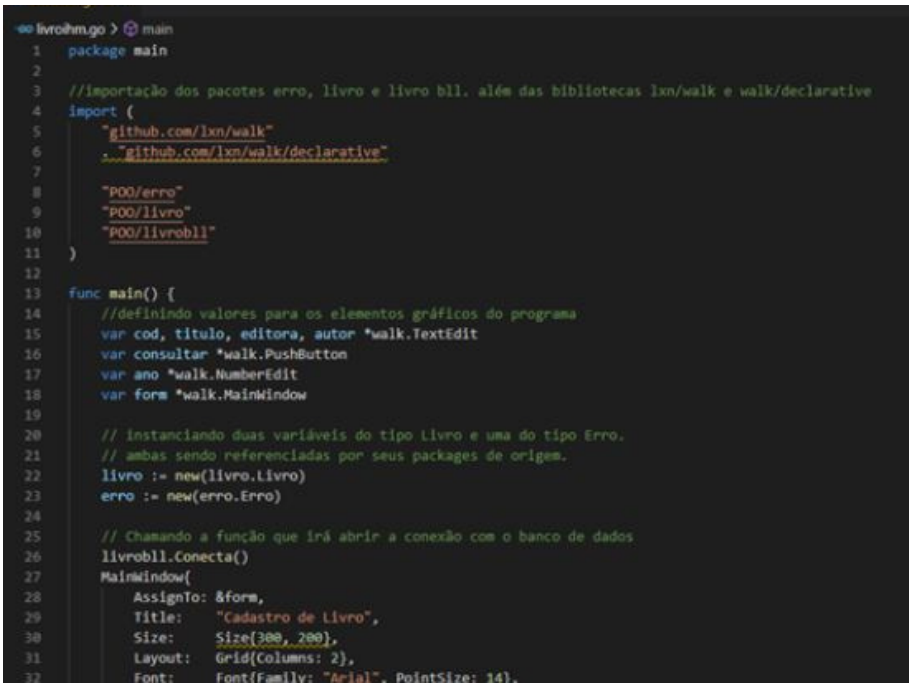
livro.go -> também é o mesmo arquivo que utilizamos no projeto passado, então não o mostraremos novamente. Ele continuará guardando o código, titulo, autor, editora e o ano do livro em uma struct.

Livrobll.go -> Ainda será responsável pela validação dos dados, além disso servirá como um intermediário entre o livroihm.go e o livrodal.go.

livroihm.go -> Continuará sendo a nossa parte gráfica do nosso programa, porém com alguns novos botões e diferentes funcionalidades. A única novidade dentro da nossa pasta raiz será a presença do arquivo “livrodal.go”, quando chamada será responsável por fazer qualquer atividade relacionada ao banco de dados. Ela guardará as funções responsáveis por inserir, atualizar, consultar e apagar um livro do nosso banco de dados. Além também de possuir uma função de conectar e desconectar a conexão com o banco de dados.
Vamos ao código!!!

livroihm.go

A grande novidade nesse arquivo é que quando chamamos a função validaCodigo() e validaDados(), é que estamos passando uma string para identificar cada operação. Por exemplo, passamos a string “i” para identificar que a operação a ser validada é a de inserir. Veja o código comentado do arquivo a seguir:



```
                    // Validando o código digitado
                    livrobll.ValidaCodigo(erro, livro, "e")

                    if erro.GetErro() {
                        walk.MsgBox(form, "Erro", erro.GetMens(), walk.MsgBoxStyle(walk.MsgBoxIconError))
                    } else {
                        walk.MsgBox(form, "Excluído!", erro.GetMens(), walk.MsgBoxStyle(walk.MsgBoxIconInformation))
                    }
                },
            },

            PushButton{
                ColumnSpan: 2,
                Text:       "Limpar",
                OnClicked: func() {
                    cod.SetText("")
                    titulo.SetText("")
                    autor.SetText("")
                    editora.SetText("")
                    ano.SetValue(0)
                },
            },
        },
    }.Run()
}
```

livroihm.go 2 ×

livroihm.go > main

```
            PushButton{
                ColumnSpan: 2,
                Text:       "Alterar",
                OnClicked: func() {

                    // Setando os valores no livro instanciado
                    livro.SetCodigo(cod.Text())
                    livro.SetTitulo(titulo.Text())
                    livro.SetAutor(autor.Text())
                    livro.SetEditora(editora.Text())
                    livro.SetAno(int(ano.Value()))

                    // Validando os dados
                    livrobll.ValidaDados(erro, *livro, "a")

                    if erro.GetErro() {
                        walk.MsgBox(form, "Erro!", erro.GetMens(), walk.MsgBoxStyle(walk.MsgBoxIconError))
                    } else {
                        walk.MsgBox(form, "Alterado!", erro.GetMens(), walk.MsgBoxStyle(walk.MsgBoxIconInformation))
                    }

                },
            },

            PushButton{
                ColumnSpan: 2,
                Text:       "Excluir",
                OnClicked: func() {
                    // Setando um código no livro instanciado
                    livro.SetCodigo(cod.Text())
```

Livrobll.go

A grande novidade nesse arquivo é a presença das funções Conecta() e Desconecta() que irão chamar a função Conecta() e Desconecta() do pacote livrodal.go. Tivemos que criar estas funções, pois o livrobll.go serve como intermediário entre o livroihm.go e livrodal.go, portanto seria uma má prática chamar a função Conecta() e Desconecta do pacote livrodal.go diretamente do livroihm.go.

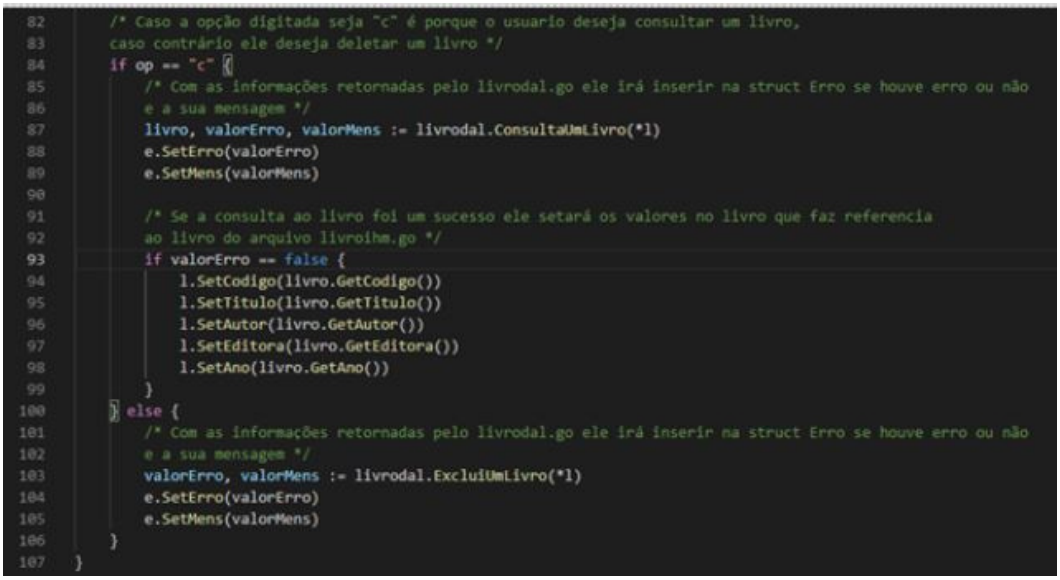
Além disso, há uma verificação com o foco em saber qual é a operação a ser feita. Se por exemplo a opção for “i”, será chamada a função do livrodal.go responsável por inserir um livro na tabela do banco de dados. A função retornará o valor do erro e a mensagem, para que o livrobll.go possa setar essas informações na struct Erro do pacote erro.go.

Veja o código a seguir:

livrobll.go

livrobll > livrobll.go > ValidaCodigo

```go
package livrobll

import (
    "POO/erro"
    "POO/livro"
    "POO/livrodal"
)

/* Quando chamada pelo livrolhm.go irá chamar a função pertencente
ao pacote livrodal que abrirá uma conexão ao banco de dados */

func Conecta() {
    livrodal.Conecta()
}

/* Quando chamada irá executar a função do pacote livrodal.go
que irá desfazer a conexão com o banco de dados */
func Desconecta() {
    livrodal.Desconecta()
}

func ValidaDados(e *erro.Erro, l livro.Livro, op string) {
    e.SetErro(false)

    if l.GetCodigo() == "" {
        e.SetMens("O campo código é de preenchimento obrigatório")
        e.SetErro(true)
        return
    }

    if l.GetTitulo() == "" {
        e.SetMens("O campo titulo é de preenchimento obrigatório")
        e.SetErro(true)
        return
    }

    if l.GetAutor() == "" {
        e.SetMens("O campo autor é de preenchimento obrigatório")
        e.SetErro(true)
        return
    }

    if l.GetEditora() == "" {
        e.SetMens("O campo editora é de preenchimento obrigatório")
        e.SetErro(true)
        return
    }

    if l.GetAno() <= 0 {
        e.SetMens("O campo ano tem de ser maior do que 0")
        e.SetErro(true)
        return
    }

    /* Caso a opção digitada seja "i" é porque o usuario deseja inserir um livro,
    caso contrário ele deseja atualizar um livro */
    if op == "i" {
        /* Com as informações retornadas pelo livrodal.go ele irá inserir na struct Erro se houve erro ou não
        e a sua mensagem */
        valorErro, valorMens := livrodal.InserirLivro(l)
        e.SetErro(valorErro)
        e.SetMens(valorMens)
    } else {
        /* Com as informações retornadas pelo livrodal.go ele irá inserir na struct Erro se houve erro ou não
        e a sua mensagem */
        valorErro, valorMens := livrodal.AtualizaUmLivro(l)
        e.SetErro(valorErro)
        e.SetMens(valorMens)
    }
}

func ValidaCodigo(e *erro.Erro, l *livro.Livro, op string) {
    e.SetErro(false)
    if l.GetCodigo() == "" {
        /* Com as informações retornadas pelo livrodal.go ele irá inserir na struct Erro se houve erro ou não
        e a sua mensagem */
        e.SetMens("O campo código é de preenchimento obrigatório")
        e.SetErro(true)
        return
    }
```



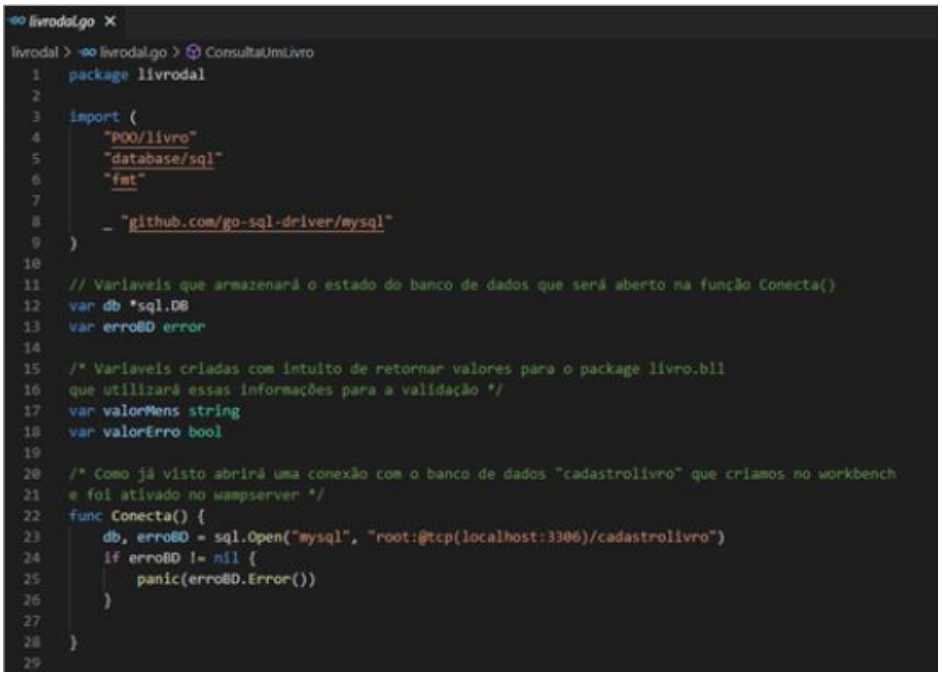
```
    /* Caso a opção digitada seja "c" é porque o usuario deseja consultar um livro,
    caso contrário ele deseja deletar um livro */
    if op == "c" {
        /* Com as informações retornadas pelo livrodal.go ele irá inserir na struct Erro se houve erro ou não
        e a sua mensagem */
        livro, valorErro, valorMens := livrodal.ConsultaUmLivro(*l)
        e.SetErro(valorErro)
        e.SetMens(valorMens)

        /* Se a consulta ao livro foi um sucesso ele setará os valores no livro que faz referencia
        ao livro do arquivo livroihm.go */
        if valorErro == false {
            l.SetCodigo(livro.GetCodigo())
            l.SetTitulo(livro.GetTitulo())
            l.SetAutor(livro.GetAutor())
            l.SetEditora(livro.GetEditora())
            l.SetAno(livro.GetAno())
        }
    } else {
        /* Com as informações retornadas pelo livrodal.go ele irá inserir na struct Erro se houve erro ou não
        e a sua mensagem */
        valorErro, valorMens := livrodal.ExcluiUmLivro(*l)
        e.SetErro(valorErro)
        e.SetMens(valorMens)
    }
}
```

Note que na verificação a saber se o usuário quer uma consulta, a função ConsultaUmLivro() do pacote livrodal.go além de retornar o valor do erro e a sua mensagem, é retornado o livro consultado também. Logo depois é feita a verificação se houve algum erro ao consultar, caso não tenha ocorrido nenhum erro, será setado as informações na instância da struct Livro que será usada para exibir os dados do livro na tela.

livrodal.go

Este arquivo, como já dito, será responsável por agir diretamente com o banco de dados. Quando as funções forem chamadas ele abrirá uma conexão com banco de dados, irá inserir um livro com os dados digitados pelo usuário na tabela, alterar um livro com base no código digitado pelo o usuário, deletar um livro do banco de dados com base no código digitado pelo o usuário e poder consultar um livro da tabela do banco de dados.
Ademais, as funções de inserir, alterar, consultar e deletar, retornarão um valor booleano e uma mensagem de erro ao livrobll.go, com o objetivo de informar se a operação feita pelo usuário deu certo ou não. A função consultar também retornará o livro consultado.

Veja o código super comentado a seguir:



```
livrodal.go
livrodal > livrodal.go > ConsultaUmLivro
package livrodal

import (
    "POO/livro"
    "database/sql"
    "fmt"

    _ "github.com/go-sql-driver/mysql"
)

// Variaveis que armazenará o estado do banco de dados que será aberto na função Conecta()
var db *sql.DB
var erroBD error

/* Variaveis criadas com intuito de retornar valores para o package livro.bll
que utilizará essas informações para a validação */
var valorMens string
var valorErro bool

/* Como já visto abrirá uma conexão com o banco de dados "cadastrolivro" que criamos no workbench
e foi ativado no wampserver */
func Conecta() {
    db, erroBD = sql.Open("mysql", "root:@tcp(localhost:3306)/cadastrolivro")
    if erroBD != nil {
        panic(erroBD.Error())
    }

}
```

```go
func Desconecta() {
    defer db.Close()
}

/* Função responsável por inserir um livro na nossa tabela do banco de dados.
Ela receberá um livro instanciado e setará as informações da struct na tabela do nosso banco de dados */
func InserirLivro(livro livro.Livro) (bool, string) {
    // Armazenando a pesquisa responsavel por inserir um livro na tabela do mysql
    var pesquisa = "insert into livros (codigo,titulo,autor,editora,ano) values (?,?,?,?,?)"

    // Preparando a pesquisa
    stmt, erroPesquisa := db.Prepare(pesquisa)

    // Caso ocorra um erro na pesquisa será retornado um erro ao livrobll.go
    if erroPesquisa != nil {
        valorErro = true
        valorMens = "erro ao preparar a consulta ao banco de dados!"
        return valorErro, valorMens
    }

    // Substituindo os "?" da variavel "pesquisa" pelos valores do livro instanciado e inserindo os dados na tabela
    respostaInserir, erroInserir := stmt.Exec(livro.GetCodigo(), livro.GetTitulo(), livro.GetAutor(), livro.GetEditora(), livro.GetAno())

    // Caso ocorra um erro ao inserir os dados na tabela será retornado um erro ao livrobll.go
    if erroInserir != nil {
        valorErro = true
        valorMens = "erro ao inserir o livro ao banco de dados!"
        return valorErro, valorMens
    }

    fmt.Print(respostaInserir)
```

```go
    // Relatando ao livrobll.go que não houve erro ao inserir o livro
    valorErro = false
    valorMens = "livro cadastrado com sucesso!"
    return valorErro, valorMens
}

/* Essa será responsável por excluir um livro da nossa tabela do nosso banco de dados. Receberá um livro instanciado
com o código digitado pelo usuário e excluirá o livro com base nele */
func ExcluiUmLivro(livro livro.Livro) (bool, string) {
    // Armazenando a pesquisa responsavel por deletar um livro na tabela do mysql
    var pesquisa = "delete from livros where codigo = (?)"

    // Preparando a pesquisa ao banco de dados
    stmt, erroPesquisa := db.Prepare(pesquisa)

    // Caso ocorra um erro na pesquisa será retornado um erro ao livrobll.go
    if erroPesquisa != nil {
        valorErro = true
        valorMens = "erro ao preparar a consulta ao banco de dados!"
        return valorErro, valorMens
    }

    // Substituindo o "?" da variavel "pesquisa" pelo código do livro instanciado e deletando os dados da tabela
    respostaExclui, erroExclui := stmt.Exec(livro.GetCodigo())

    // Caso ocorra um erro ao deletar os dados da tabela do banco de dados será retornado um erro ao livrobll.go
    if erroExclui != nil {
        valorErro = true
        valorMens = "Houve um erro ao excluir o livro!"
        return valorErro, valorMens
    }
```

```go
    fmt.Print(respostaExclui)
    // Relatando ao livrobll.go que não houve erro ao deletar o livro
    valorErro = false
    valorMens = "o livro foi excluido com sucesso"
    return valorErro, valorMens
}

/* Essa função será responsável por atualizar um livro existente na nossa tabela do nosso banco de dados.
Receberá um livro instanciado e com base no código digitado pelo usuário atualizará o livro com as informações digitadas */
func AtualizaUmLivro(livro livro.Livro) (bool, string) {
    // Armazenando a pesquisa responsavel por atualizar um livro na tabela do mysql
    var pesquisa = "update livros set titulo=(?), autor=(?), editora=(?), ano=(?) where codigo = (?)"

    // Preparando a pesquisa ao banco de dados
    stmt, erroPesquisa := db.Prepare(pesquisa)

    // Caso ocorra um erro na pesquisa será retornado um erro ao livrobll.go
    if erroPesquisa != nil {
        valorErro = true
        valorMens = "erro ao preparar a consulta ao banco de dados!"
        return valorErro, valorMens
    }

    // Substituindo os "?" da variavel "pesquisa" pelas informações do livro instanciado e atualizando os dados da tabela
    respostaAtualiza, erroAtualiza := stmt.Exec(livro.GetTitulo(), livro.GetAutor(), livro.GetEditora(), livro.GetAno(), livro.GetCodigo())

    // Caso ocorra um erro ao atualizar o livro será retornado um erro ao livrobll.go
    if erroAtualiza != nil {
        valorErro = true
        valorMens = "não foi possível alterar o livro!"
        return valorErro, valorMens
    }
```

```
    // Relatando ao livrobll.go que não houve erro ao deletar o livro
    fmt.Print(respostaAtualiza)
    valorErro = false
    valorMens = "o livro foi alterado com sucesso!"
    return valorErro, valorMens
}

/* Essa função será responsável por consultar um livro existente na nossa tabela do nosso banco de dados.
Receberá um livro instanciado e com base no código digitado pelo usuário retornará o livro pesquisado
na tabela do banco de dados. Note que nessa função além de retornar o valor do erro e a mensagem de erro,
estamos retornando também o livro para o livrobll.go que será utilizado no livroihm.go */
func ConsultaUmLivro(livro livro.Livro) (livro.Livro, bool, string) {
    // Armazenando a pesquisa responsavel por consultar um livro na tabela do mysql
    var pesquisa = "select * from livros where codigo=(?)"
    var entrou = false

    // Preparando a pesquisa ao banco de dados
    smtr, erroPesquisa := db.Prepare(pesquisa)

    // Caso ocorra um erro na pesquisa será retornado um erro ao livrobll.go
    if erroPesquisa != nil {
        valorErro = true
        valorMens = "erro ao preparar a consulta ao banco de dados!"
        return livro, valorErro, valorMens
    }

    /* Dessa vez não utilizaremos o Exec(), pois queremos fazer uma consulta e não alterar diretamente os valores da tabela
    do banco de dados, para isso utilizamos o a função Query() */
    respostaConsulta, erroConsulta := smtr.Query(livro.GetCodigo())
```

```
    /* Nessa parte do código definimos os valores padrão da struct e dos valores da mensagem e de erro,
    pois se a função Query() não achar o livro com o código consultado a struct livro retornará o livro
    que foi consultado anteriormente. Note que não damos "return", pois alteraremos este valor mais para frente
    caso a função Query() ache um livro na tabela do banco de dados */
    livro.SetCodigo("")
    livro.SetTitulo("")
    livro.SetAutor("")
    livro.SetEditora("")
    livro.SetAno(0)
    valorErro = true
    valorMens = "erro ao consultar o livro!"

    // Caso ocorra um erro brusco ao consultar o livro será retornado um erro ao livrobll.go
    if erroConsulta != nil {
        valorErro = true
        valorMens = "erro ao consultar o livro!"
        return livro, valorErro, valorMens
    }

    // Esse for irá percorrer o livro somente se existir na tabela um livro com o código digitado pelo usuário
    for respostaConsulta.Next() {
        entrou = true

        var codigo string
        var titulo string
        var autor string
        var editora string
        var ano int

        // A função Scan irá pegar os valores pesquisados na tabela e jogará nas váriaveis criadas
        erroLeituraDados := respostaConsulta.Scan(&codigo, &titulo, &autor, &editora, &ano)
```

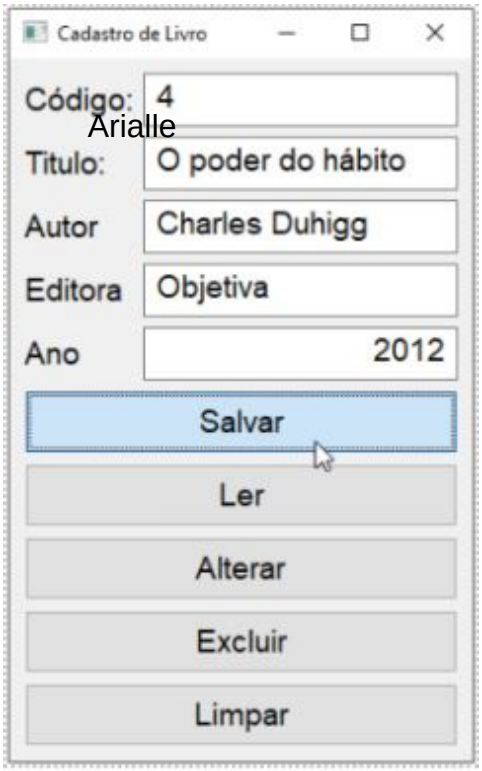
```

        // Caso haja um erro ao setar os valores da tabela nas variaveis criadas será retornado um erro ao livrobll.go
        if erroLeituraDados != nil {
            valorErro = true
            valorMens = "houve um erro ao ler os dados da tabela do banco de dados!"
            return livro, valorErro, valorMens
        }

        /* E finalmente caso não haja nenhum erro setaremos os valores pesquisados na struct livro que será
        retornada ao livro bll.go */
        livro.SetCodigo(codigo)
        livro.SetTitulo(titulo)
        livro.SetAutor(autor)
        livro.SetEditora(editora)
        livro.SetAno(ano)
    }

    /* Caso ele tenha entrado no for e chegou até essa parte do código é porque deu tudo certo.
    Portanto relatamos ao livrobll.go que a consulta foi um sucesso */
    if entrou {
        valorErro = false
        valorMens = "livro consultado com sucesso"
    }

    // Retornando o livro consultado, o valor do erro e a sua mensagem ao livrobll.go
    return livro, valorErro, valorMens
}
```

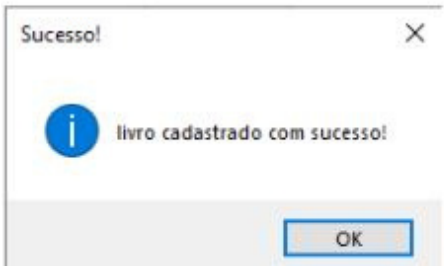
Note que em todas as funções relacionadas ao banco de dados, como por exemplo o Prepare(), Query(), Open(), retornam dois valores. O primeiro valor sempre vai ser o resultado referente a operação e o segundo um possível erro, que sempre é verificado após a chamada da função.
Após a codificação, vamos mostrar o funcionamento do nosso projeto com as telas de execução. Mostraremos cada operação e o resultado delas na nossa tabela do banco de dados.

Importante!! Lembre-se de importar para o seu projeto o arquivo erro.go e o livro.go do projeto passado. São de extrema importância para o funcionamento da nossa aplicação atual utilizando o banco de dados.

Executando e testando as funcionalidades do nosso projeto

Inserindo um livro





Veja que no banco de dados o livro “O poder do hábito” foi inserido corretamente.

```
24 •   select * from livros;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

codigo	titulo	autor	editora	ano
1	Dom Quixote	Miguel de Cervantes	Pinguim Companhia	1605
2	Guerra e Paz	Liev Tolstói	Companhia das Letras	1869
3	A montanha mágica	Thomas Mann	Companhia das Letras	1924
4	O poder do hábito	Charles Duhigg	Objetiva	2012

Alterando um livro

Houve um erro ao digitar o ano do livro. Vamos corrigir ele através do botão alterar, alterando o ano do livro para “2013” ao invés de “2012”.

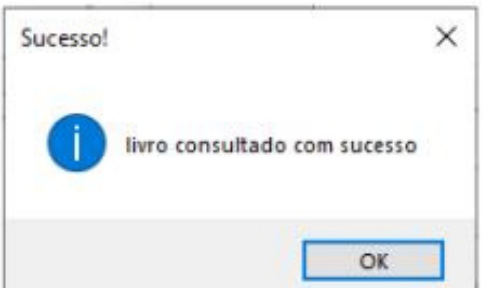
Note que o livro foi alterado com sucesso na tabela do banco de dados.

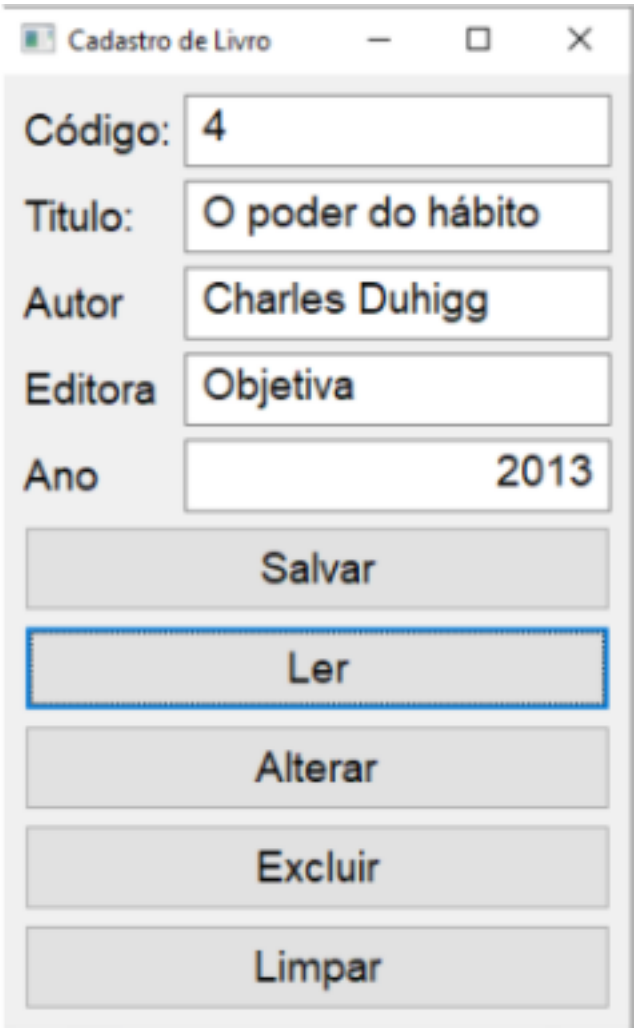
```
24 •  select * from livros;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

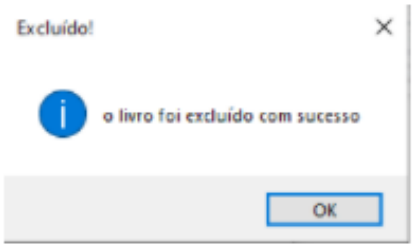
codigo	titulo	autor	editora	ano
1	Dom Quixote	Miguel de Cervantes	Pinguim Companhia	1605
2	Guerra e Paz	Liev Tolstói	Companhia das Letras	1869
3	A montanha mágica	Thomas Mann	Companhia das Letras	1924
4	O poder do hábito	Charles Duhigg	Objetiva	2013

Consultando um livro

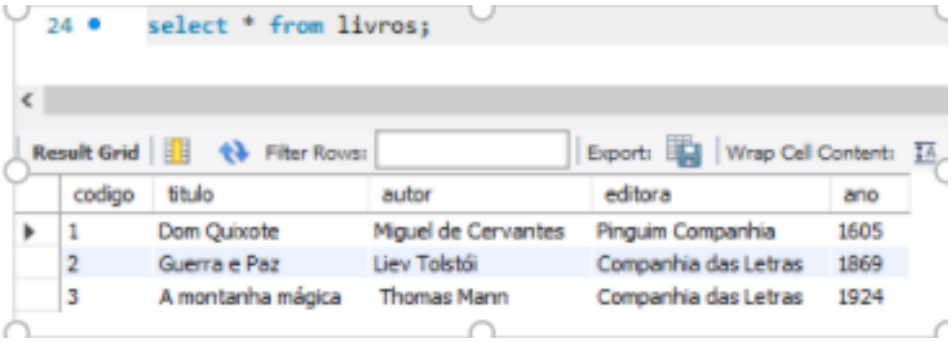




Deletando um livro



Agora na tabela sobrou apenas os livros que havíamos inserido pelo workbench. Veja:



Consumindo uma API em golang

Nesse último capítulo ensinaremos a consumir uma API já desenvolvida.
Vamos lá? Faremos juntos uma aplicação que nos retorna a UF, cidade, bairro e rua através de um CEP. A API utilizada será o ViaCEP.

Começaremos nosso programa importando algumas bibliotecas que serão utilizadas posteriormente!

```go
package main

// importação de bibliotecas
import (
    "encoding/json"
    "io/ioutil"
    "net/http"

    "github.com/lxn/walk"
    . "github.com/lxn/walk/declarative"
)
```

É necessário entender que através da API ViaCEP (e uma boa parte das demais API's existentes no mundo) nos retorna dados em JSON.

O que é JSON?

O JSON é, basicamente, um formato leve de troca de informações/dados entre sistemas. É um formato de leitura extremamente simples, veloz na execução e transporte de dados e leve!

```json
{
  "cep": "01001-000",
  "logradouro": "Praça da Sé",
  "complemento": "lado ímpar",
  "bairro": "Sé",
  "localidade": "São Paulo",
  "uf": "SP",
  "ibge": "3550308",
  "gia": "1004",
  "ddd": "11",
  "siafi": "7107"
}
```

Como podemos notar, é extremamente simples sua leitura. Está bem claro que recebemos um cep, logradouro, complemento, bairro, localidade, uf, ibge, gia, ddd e siafi e respectivamente seus valores.

Retornando ao nosso programa, iremos criar uma struct do tipo ConsultaCEP, que terá os campos Cep, Logradouro, Complemento, Bairro, Localidade, Uf, Ibge, Gia, Ddd e Siafi.

```
13  // estrutura JSON que recebemos através da API
14  type ConsultaCEP struct {
15      Cep         string `json:"cep"`
16      Logradouro  string `json:"logradouro"`
17      Complemento string `json:"complemento"`
18      Bairro      string `json:"bairro"`
19      Localidade  string `json:"localidade"`
20      Uf          string `json:"uf"`
21      Ibge        string `json:"ibge"`
22      Gia         string `json:"gia"`
23      Ddd         string `json:"ddd"`
24      Siafi       string `json:"siafi"`
25  }
```

Percebe-se uma diferença do que vimos até agora: após definirmos o tipo de cada campo, há a palavra json seguida do nome dos campos que a API nos retorna em JSON. Para transformarmos os campos que o JSON nos retorna, existe um site que pode facilitar nossa vida: https://mholt.github.io/json-to-go/

Copiamos o modelo de JSON retornado e automaticamente teremos uma struct da forma que precisamos.

Seguindo, iremos iniciar nossas variáveis. Uma variável para termos controle da MainWindow, variáveis para receber os dados cep, uf, cidade, bairro e rua e um botão para a consulta dos dados.

```
27  func main() {
28      // setando o valor das variáveis
29      var form *walk.MainWindow
30      var cep, uf, cidade, bairro, rua *walk.TextEdit
31      var consultar *walk.PushButton
```

Seguimos com ajustes na forma do nosso programa, criando uma label para informar que é necessário inserir o CEP e o campo para a inserção dele.

```
32      MainWindow{
33          AssignTo: &form,
34          Title:    "Consultar endereço",
35          Size:     Size{500, 200},
36          Layout:   Grid{Columns: 2},
37          Font:     Font{Family: "Arial", PointSize: 14},
38          Children: []Widget{
39
40              Label{
41                  Text: "Digite seu CEP:",
42              },
43
44              // Campo CEP
45              TextEdit{
46                  AssignTo: &cep,
47                  Text:     "",
48              },
49
```

Dentro do nosso botão para a consulta começaremos o consumo da API.

O CEP digitado pelo usuário será necessário para termos os dados do local e ele funcionará como um elemento da nossa URL, pois ela funcionará da seguinte maneira: "https://viacep.com.br/ws/cepDigitado/json/".

```
50          // Botão de consulta
51          PushButton{
52              AssignTo:   &consultar,
53              ColumnSpan: 2,
54              Text:       "Consultar Endereço",
55              OnClicked: func() {
56                  cepDigitado := cep.Text()
57                  url := "https://viacep.com.br/ws/" + cepDigitado + "/json/"
58
59                  // O método http.Get nos traz os valores que a API nos manda, caso dê erro, será retornado na segunda variável.
60                  response, _ := http.Get(url)
61
62                  // O método ioutil.ReadAll serve para convertermos o response.Body (corpo do response), que vem como formato de bytes, para string.
63                  responseData, _ := ioutil.ReadAll(response.Body)
64
65                  // Iniciamos uma variável do tipo ConsultaCEP
66                  var consulta ConsultaCEP
67
68                  // A função json.Unmarshal transforma nossa string para os valores recebidos em cada campo, indo diretamente para cada campo setado
                    pelo `json:"nome do campo"`
69                  json.Unmarshal(responseData, &consulta)
```

Através do método Get da biblioteca net/http traremos os valores que a API nos envia. Porém, ela não vem em um formato convencional: é trazida em formato de bytes. Para isso, é necessário utilizar a função ReadAll da biblioteca io/ioutil para convertermos o corpo desses dados em string.

Então iniciamos uma variável do tipo ConsultaCEP, para receber os dados de cada campo que a API nos retorna através de seus campos previamente criados. Para isso, é necessário utilizarmos a função Unmarshal da biblioteca encoding/json, que enviará os dados organizados em string para os campos da variável de tipo ConsultaCEP.

Caso os valores retornados sejam vazios, ocorrerá um erro na consulta dos dados, pois o CEP é inválido. Após isso, os valores de cada campo serão setados!

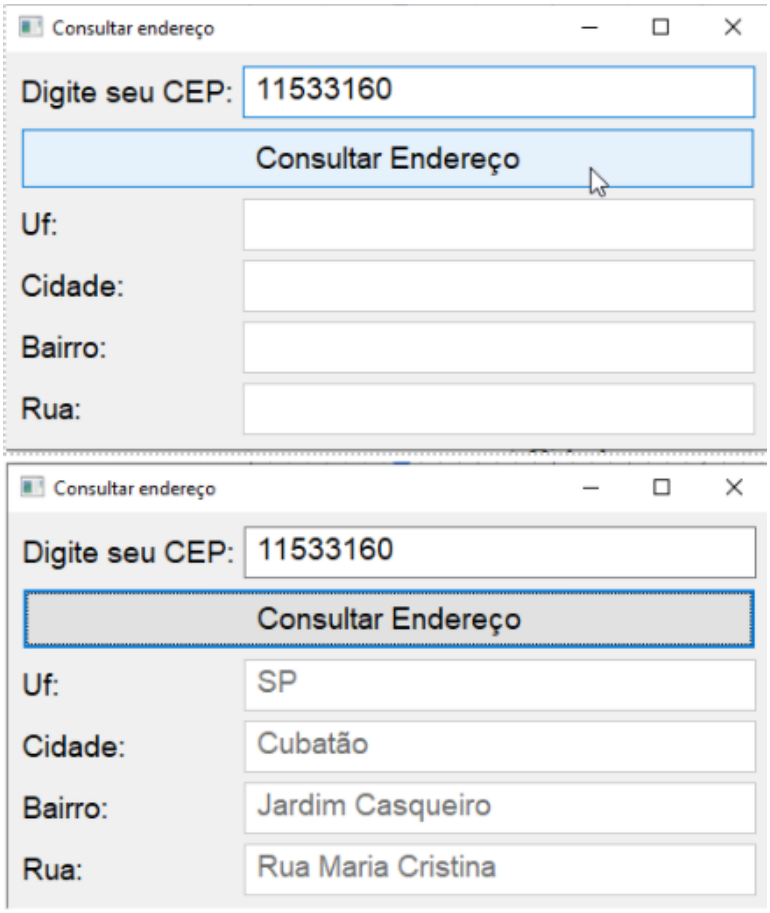
```
71              // Se os valores retornados forem vazios significa que o CEP inserido é inválido.
72              if consulta.Uf == "" && consulta.Localidade == "" && consulta.Bairro == "" && consulta.Logradouro == "" {
73                  walk.MsgBox(form, "Erro!", "Erro ao consultar os dados!", walk.MsgBoxStyle(walk.MsgBoxIconError))
74              }
75
76              // Setando os valores recebidos para os campos visuais de texto.
77              uf.SetText(consulta.Uf)
78              cidade.SetText(consulta.Localidade)
79              bairro.SetText(consulta.Bairro)
80              rua.SetText(consulta.Logradouro)
81          },
82      },
```

Validação de cada campo e resto do código:

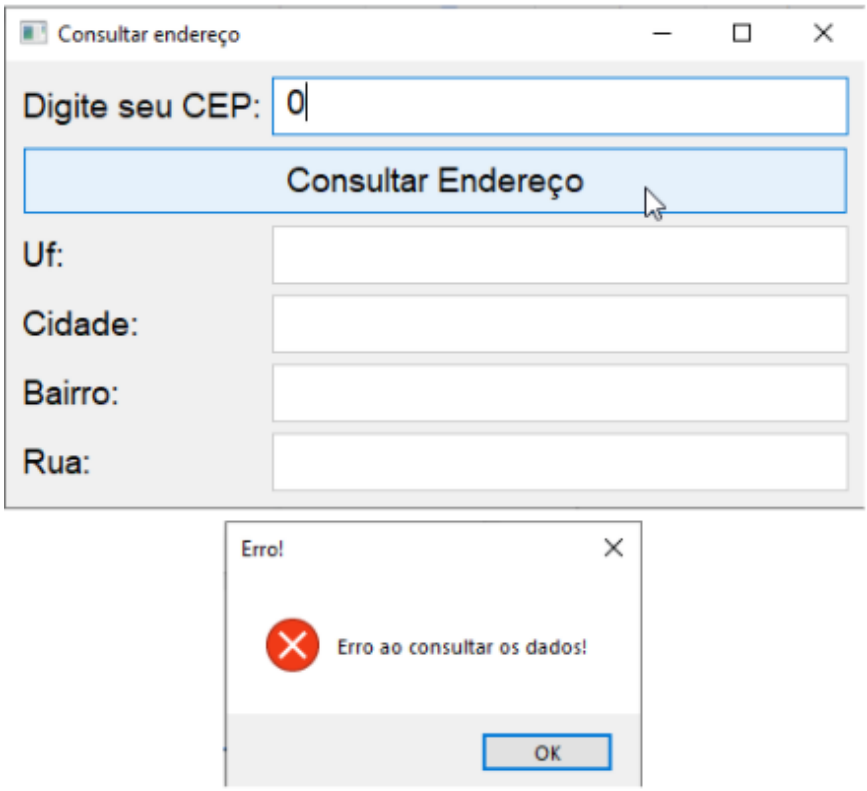
```
84              Label{
85                  Text: "Uf:",
86              },
87
88              TextEdit{
89                  AssignTo: &uf,
90                  Text:     "",
91                  Enabled:  false,
92              },
93
94              Label{
95                  Text: "Cidade:",
96              },
97
98              TextEdit{
99                  AssignTo: &cidade,
100                 Text:     "",
101                 Enabled:  false,
102             },
103
104             Label{
105                 Text: "Bairro:",
106             },
```

```
107
108             TextEdit{
109                 AssignTo: &bairro,
110                 Text:     "",
111                 Enabled:  false,
112             },
113
114             Label{
115                 Text: "Rua:",
116             },
117
118             TextEdit{
119                 AssignTo: &rua,
120                 Text:     "",
121                 Enabled:  false,
122             },
123         },
124     }.Run()
125 }
126
```

Executando o programa



Inserindo um CEP inválido



Agradecimento

Agradecemos primeiramente a Deus, por ter nos mantido na trilha certa durante este projeto de pesquisa, com saúde e forças para chegar até o final.
Somos grato à família pelo apoio que sempre nos deram durante toda a nossa vida. Deixamos um agradecimento especial para o nosso orientador, pelo incentivo e pela dedicação do seu escasso tempo em nosso projeto de pesquisa. Também agradecemos ao IFSP e a todos os professores do nosso curso pela elevada qualidade do ensino oferecido. Obrigado!

Para acompanhar os projetos desenvolvidos na apostila, acesse:

Clique aqui - Alisson de Sousa Vieira
Clique aqui - Everson Pereira da Silva