

# Prova-Backend

---

Backend feito em Golang e com a framework web Gin com padrão API REST. Servidor é executado na porta 8080 com a rota /verify. Autor do código Hícaro José Barbosa Soares [Josehpequeno](#).

## Dependências

- [Go](#)

## Executando Servidor

Dentro da pasta do projeto execute o comando:

```
go run main.go
```

## Explicando o Código

### Imports

```
import (  
    "net/http" // Utilizado para responder a requisição do cliente com status http  
  
    "github.com/gin-gonic/gin" // Web framework  
  
    "unicode" // Utilizado nas funções para verificar os caracteres de strings  
)
```

### Função Main

```
func main() {  
    port := "8080" // porta que irá ser executada a aplicação  
    r := gin.Default() //instanciando o router para criar rotas  
    r.POST("/verify", func(c *gin.Context) { // rota verify  
        var body Body  
        if err := c.ShouldBind(&body); err != nil {  
            // passando o json da requisição para a variavel body de tipo Body  
            c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})  
            // caso haja erro é retornado para o cliente  
            return  
        }  
        response := strongPassword(body)  
        // função que verifica a força da senha e retorna um objeto do tipo Respose  
        c.JSON(http.StatusOK, response)  
        // retorna a respota da função para o cliente  
    })  
}
```

```
    })
    r.Run(":" + port)
}
```

## Structs

```
/* tipo Body que é recebido na requisição, contendo:
Password (referenciado na json da requisição com a chave "password") e
Rules (referenciado na json da requisição com a chave "rules") que é um array de Rule.*/
type Body struct {
    Password string `json:"password" binding:"required"`
    Rules    []Rule  `json:"rules" binding:"required"`
}
/* tipo Rule contém o Content (referenciado na requisição com a chave "rule")
que armazena a string da regra e o Value (referenciado na requisição com a
chave "value") armazena o valor da regra.
*/
type Rule struct {
    Content string `json:"rule" binding:"required"`
    Value   int    `json:"value" binding:"required"`
}
/* Respose é o tipo da resposta ao cliente.
Contém Verify um boolean que recebe o valor true se a senha é forte de acordo com as
regras e false quando o contrário.
Contém também o NoMatch que é o array de regras que a senha não foi aprovada.
*/
type Response struct {
    Verify bool    `json:"verify"`
    NoMatch []string `json:"noMatch"`
}
```

## Função StrongPassword

```
func strongPassword(body Body) Response {
    noMatch := make([]string, 0) // array de regras em que password foi desaprovada
    lenght := len(body.Rules) // tamanho do array de regras
    flag := true
    // variável flag que indica se password foi desaprovada ou não nas regras.
    for i := 0; i < lenght; i++ { // percorrendo as regras
        rule := body.Rules[i].Content // regra a ser verificada
        x := body.Rules[i].Value // valor da regra
        switch rule {
            case "minSize": // caso a regra seja "minSize"
                if len(body.Password) < x {
                    // confere se tem o tamanho mínimo pedido
                    // caso não tenha altera a flag e adiciona regra ao array noMatch
                    flag = false
                    noMatch = append(noMatch, rule)
                }
            }
        }
    }
    return Response{Verify: flag, NoMatch: noMatch}
```

```

    }
    case "minUppercase":
        if !minUppercase(body.Password, x) {
            /*função que confere se tem o mínimo pedido de caracteres em
            uppercase */
            flag = false
            noMatch = append(noMatch, rule)
        }
    case "minLowercase":
        if !minLowercase(body.Password, x) {
            /*função que confere se tem o mínimo pedido de caracteres em
            lowercase*/
            flag = false
            noMatch = append(noMatch, rule)
        }
    case "minDigit":
        if !minDigit(body.Password, x) {
            //função que confere se tem o mínimo pedido de dígitos
            flag = false
            noMatch = append(noMatch, rule)
        }
    case "minSpecialChars":
        if !minSpecialChars(body.Password, x) {
            //função que confere se tem o mínimo pedido de caracteres especiais
            flag = false
            noMatch = append(noMatch, rule)
        }
    case "noRepeted":
        if !noRepeted(body.Password) {
            //função que confere se tem repetição em password
            flag = false
            noMatch = append(noMatch, rule)
        }
    }
}
return Response{flag, noMatch}
}

```

## Funções de verificação de regras

```

func minUppercase(s string, x int) bool {
    count := 0 // variável de contagem
    for _, r := range s {
        if unicode.IsLetter(r) && unicode.IsUpper(r) {
            // se for letra e estiver em uppercase
            count++
        }
    }
    if count >= x {
        // se a contagem for maior ou igual a pedida retorna true caso contrário retorna false
        return true
    }
}

```

```
    } else {
        return false
    }
}

func minLowercase(s string, x int) bool {
    count := 0 // variável de contagem
    for _, r := range s {
        if unicode.IsLetter(r) && unicode.IsLower(r) {
            // se for letra e estiver em lowercase
            count++
        }
    }
    if count >= x {
        // se a contagem for maior ou igual a pedida retorna true caso contrário retorna false
        return true
    } else {
        return false
    }
}

func minDigit(s string, x int) bool {
    count := 0 // variável de contagem
    for _, r := range s {
        if unicode.IsDigit(r) {
            // se for dígito
            count++
        }
    }
    if count >= x {
        // se a contagem for maior ou igual a pedida retorna true caso contrário retorna false
        return true
    } else {
        return false
    }
}

func minSpecialChars(s string, x int) bool {
    chars := "!@#$%^&*()-+/{ }[]\\\" // string de caracteres especiais
    count := 0 // variável de contagem
    for _, r := range s { // percorre a string
        for _, c := range chars { //percorre a string de caracteres especiais
            if r == c { // se for igual adiciona a contagem
                count++
                break
            }
            /* se for igual não vai ser igual a outro então faz o break no for e pula para
            próximo caracter da string*/
        }
    }
    if count >= x {
        // se a contagem for maior ou igual a pedida retorna true caso contrário retorna false
        return true
    } else {
```

```
        return false
    }
}
func noRepeted(s string) bool {
// 0 for só permitirá a entrada caso a string tenha tamanho 3 ou superior
for i := 1; i < len(s)-1; i++ {
/*percorre a string a partir do segundo caractere
comparando ele com o caracter anterior e com o próximo caracter */
    if s[i] == s[i-1] || s[i] == s[i+1] {
        return false
        /* se existir repetição ele já retorna á função anterior para que
        possa ser retornada ao cliente*/
    }
}
//caso não exista repetição
return true
}
```