

**Московский государственный технический
университет им. Н.Э. Баумана**

**Факультет ИУ
Кафедра ИУ5**

**Курс «Основы информатики»
Отчет по Домашнему заданию**

Выполнил студент группы ИУ5-33Б: Лачина Е.А.

Подпись и дата:

Проверил преподаватель каф.: Гапанюк Ю. Е.

Подпись и дата:

Москва, 2024 г

Calculator on http server

Инструментарий:

Golang - go version go1.23.3 darwin/arm64

Visual studio code

Для тестирования запросов использовался API testing platform Postman

Функционал:

Обработка реквестов сервером производится с использованием функций-хэндлеров:

- getRoot, getHello - стандартный запросы
- calculate - запрос с подсчетом введенного выражения

Директория **calculator**:

- IsDigit - функция проверки выражения - запроса
- Calc - калькулятор, возвращающий результат и сведения об ошибке

Middleware:

loggingMiddleware - лог пользователя, отправившего запрос.

Code Realisation:

Go.mod:

module github.com/server

go 1.23.3

Calculator/calculator.go

package calculator

```
import (  
    "errors"  
    "strconv"  
    "strings"  
)
```

```
func IsDigit(elem string) bool {  
    for _, letter := range "0123456789." {  
        if elem == string(letter) {  
            return true  
        }  
    }  
    return false  
}
```

```
func Calc(s string) (float64, error) {  
    s = strings.ReplaceAll(s, " ", "")
```

```
    // Проверка на некорректные символы в конце строки  
    if len(s) == 0 || strings.ContainsAny(s[len(s)-1:], "+-*/") {  
        return 0, errors.New("expression is not valid")  
    }
```

```
    var mix_s []interface{}  
    var current_num string
```

```
    // Разбиваем строку на числа и операторы  
    for i := 0; i < len(s); i++ {  
        if IsDigit(string(s[i])) {
```

```

    current_num += string(s[i])
} else {
    if current_num != "" {
        // Проверка на начало числа с нуля, исключая дробные числа
        if len(current_num) > 1 && current_num[0] == '0' && current_num[1]
!= '.' {
            return 0, errors.New("expression is not valid")
        }
        num, err := strconv.ParseFloat(current_num, 64)
        if err != nil {
            return 0, errors.New("ошибка преобразования числа: " +
err.Error())
        }
        mix_s = append(mix_s, num)
        current_num = ""
    }
    // Проверка на последовательные операторы
    if i > 0 && strings.ContainsAny(string(s[i]), "+-*/") &&
strings.ContainsAny(string(s[i-1]), "+-*/") {
        return 0, errors.New("expression is not valid")
    }
    mix_s = append(mix_s, string(s[i]))
}
}

// Если остались цифры в конце, добавляем их
if current_num != "" {
    if len(current_num) > 1 && current_num[0] == '0' && current_num[1] != '.'
{
        return 0, errors.New("expression is not valid")
    }
    num, err := strconv.ParseFloat(current_num, 64)
    if err != nil {
        return 0, errors.New("ошибка преобразования числа: " + err.Error())
    }
    mix_s = append(mix_s, num)
}
}

```

// Проходим по интерфейсу, заполняем стеки и выполняем мат.
действия

```
stek1 := make([]float64, 0)
```

```
stek2 := make([]string, 0)
```

```
for i := 0; i < len(mix_s); i++ {
```

```
    if num, ok := mix_s[i].(float64); ok {
```

```
        stek1 = append(stek1, num)
```

```
    } else if mix_s[i] == "(" {
```

```
        stek2 = append(stek2, mix_s[i].(string))
```

```
    } else if mix_s[i] == ")" {
```

```
        // Выполняем операции до открывающей скобки
```

```
        for len(stek2) > 0 && stek2[len(stek2)-1] != "(" {
```

```
            op := stek2[len(stek2)-1]
```

```
            stek2 = stek2[:len(stek2)-1]
```

```
            b := stek1[len(stek1)-1]
```

```
            stek1 = stek1[:len(stek1)-1]
```

```
            a := stek1[len(stek1)-1]
```

```
            stek1 = stek1[:len(stek1)-1]
```

```
            if op == "+" {
```

```
                stek1 = append(stek1, a+b)
```

```
            } else if op == "-" {
```

```
                stek1 = append(stek1, a-b)
```

```
            } else if op == "*" {
```

```
                stek1 = append(stek1, a*b)
```

```
            } else if op == "/" {
```

```
                if b == 0 {
```

```
                    return 0, errors.New("expression is not valid")
```

```
                }
```

```
                stek1 = append(stek1, a/b)
```

```
            }
```

```
        }
```

```
        stek2 = stek2[:len(stek2)-1] // Убираем (
```

```
    } else {
```

// Пока верхний оператор в стеке имеет такой же или более
высокий приоритет

```
        for len(stek2) > 0 && (stek2[len(stek2)-1] == "*" || stek2[len(stek2)-1]  
== "/" || (stek2[len(stek2)-1] == "+" || stek2[len(stek2)-1] == "-") && (mix_s[i]  
== "+" || mix_s[i] == "-")) {
```

```

    op := stek2[len(stek2)-1]
    stek2 = stek2[:len(stek2)-1]
    b := stek1[len(stek1)-1]
    stek1 = stek1[:len(stek1)-1]
    a := stek1[len(stek1)-1]
    stek1 = stek1[:len(stek1)-1]
    if op == "+" {
        stek1 = append(stek1, a+b)
    } else if op == "-" {
        stek1 = append(stek1, a-b)
    } else if op == "*" {
        stek1 = append(stek1, a*b)
    } else if op == "/" {
        if b == 0 {
            return 0, errors.New("expression is not valid")
        }
        stek1 = append(stek1, a/b)
    }
}
// Добавляем текущий оператор в стек операторов
stek2 = append(stek2, mix_s[i].(string))
}
}
// Выполняем оставшиеся операции
for len(stek2) > 0 {
    op := stek2[len(stek2)-1]
    stek2 = stek2[:len(stek2)-1]
    b := stek1[len(stek1)-1]
    stek1 = stek1[:len(stek1)-1]
    a := stek1[len(stek1)-1]
    stek1 = stek1[:len(stek1)-1]
    if op == "+" {
        stek1 = append(stek1, a+b)
    } else if op == "-" {
        stek1 = append(stek1, a-b)
    } else if op == "*" {
        stek1 = append(stek1, a*b)
    } else if op == "/" {
        if b == 0 {

```

```

        return 0, errors.New("expression is not valid")
    }
    stek1 = append(stek1, a/b)
}
}
return stek1[0], nil
}

```

Main.go

```
package main
```

```
import (
    "encoding/json"
    "errors"
    "fmt"
    "io"
    "log"
    "net/http"
    "os"

    "github.com/server/calculator"
)

```

```
type Request struct {
    Expression string `json:"expression"`
}

```

```
type Response struct {
    Result float64 `json:"result"`
    Error string `json:"error"`
}

```

```
// Handler
```

```
func getRoot(w http.ResponseWriter, r *http.Request) {
    fmt.Printf("got / request\n")
    io.WriteString(w, "This is my website!\n")
}

```

```

func getHello(w http.ResponseWriter, r *http.Request) {
    fmt.Printf("got /hello request\n")
    io.WriteString(w, "Hello, HTTP!\n")
}

func calculate(w http.ResponseWriter, r *http.Request) {
    var req Request
    var res Response
    err := json.NewDecoder(r.Body).Decode(&req)
    if err != nil {
        http.Error(w, err.Error(), http.StatusBadRequest)
        return
    }

    result, err := calculator.Calc(req.Expression)
    if err != nil {
        if errors.Is(err, errors.New("expression is not valid")) {
            res = Response{0, "expression is not valid"}
            w.WriteHeader(http.StatusUnprocessableEntity)
        } else {
            res = Response{0, "internal server error"}
            w.WriteHeader(http.StatusInternalServerError)
        }
    } else {
        res = Response{result, "nil"}
    }

    w.Header().Set("Content-Type", "application/json")
    jRes, err := json.Marshal(res)
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
    }
    w.WriteHeader(http.StatusOK)
    w.Write(jRes)
}

// Middleware

```



```
func loggingMiddleware(next http.HandlerFunc) http.HandlerFunc {  
    return func(w http.ResponseWriter, r *http.Request) {  
        log.Printf("User %s Hit Endpoint", r.FormValue("user"))  
        next(w, r)  
    }  
}
```

```
func main() {  
    mux := http.NewServeMux()  
    mux.HandleFunc("/", loggingMiddleware(getRoot))  
    mux.HandleFunc("/hello", loggingMiddleware(getHello))  
    mux.HandleFunc("POST /calculate", loggingMiddleware(calculate))  
  
    err := http.ListenAndServe(":3333", mux)  
  
    if errors.Is(err, http.ErrServerClosed) {  
        fmt.Printf("server closed\n")  
    } else if err != nil {  
        fmt.Printf("error starting server: %s\n", err)  
        os.Exit(1)  
    }  
}
```

HTTP New Collection / ServerTest Save

POST localhost:3333/calculate Send

Params Auth Headers (8) Body Scripts Settings Cookies

raw JSON Beautify

```
1 {  
2   "expression" : "2+2"  
3 }
```

Body 200 OK • 7 ms • 134 B • Save Response

Pretty Raw Preview Visualize JSON

```
1 {  
2   "result": 4,  
3   "error": "nil"  
4 }
```

Testing:

```
○ (base) jane_air@Janes-MacBook-Pro CalucatorHttp % go run main.go  
2024/12/22 17:28:37 User Hit Endpoint  
2024/12/22 17:28:45 User Hit Endpoint  
2024/12/22 17:28:45 http: superfluous response.WriteHeader call from main.calculate (main.go:62)  
2024/12/22 17:28:56 User Hit Endpoint
```



New Collection / ServerTest



Save



POST



localhost:3333/calculate

Send



Params

Auth

Headers (8)

Body

Scripts

Settings

Cookies

raw



JSON



Beautify

```
1  {
2  |    "expression" : "2+2/0"
3  }
```

Body



500 Internal Server Error

• 2 ms

• 180 B



Save Response



Pretty

Raw

Preview

Visualize

Text



```
1  {"result":0,"error":"internal server error"}
```