Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ имени Н.Э.БАУМАНА
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

Факультет: Информатика и системы управления

Кафедра: Теоретическая информатика и компьютерные технологии

Лабораторная работа №5

Веб-сокет клиент и сервер

Вариант №0

Выполнил
студент группы ИУ9-31Б
Лисов Алексей

Москва, 2023

1 Условие

Целью данной работы является создание SSH-клиента на языке Go. Условие задачи, исходный код и пример работы программы необходимо предоставить в формате LaTeX.

2 Код решения

Файл main.go

```
package main

import (
        "encoding/json"
        "github.com/Baldislayer/iu9-networks/lab5/models"
        "github.com/gorilla/websocket"
        "iu9-networks/lab5/configs"
        "log"
)

const (
        host = "185.139.70.64"
)

const path = "ws://" + host + configs.Port + "/ws"

func main() {
        // поключаемся к серверу посредством вебсокета
        conn, _, err := websocket.DefaultDialer.Dial(path, nil)
        if err != nil {
                log.Fatal("Error connecting to WebSocket server_2:", err)
        }

        // отложенная операция, при завершении функции main мы закроем соединение
        // подробнее про defer в файле server_2.go
        defer conn.Close()

        for {
                // ты спросишь меня наверное, почему я не сделал присвоение по ссы
                // а сделал через return
                // дело в том, что так делают обычно в языке Golang
```

```
                // это показывает, что данные, которые мы передаем в качестве пара
                // и человеку,который будет использовать твой код - будет намного
                message := models.Message{}
                message = message.InputFromConsole()

                // переводим нашу структуру в jsong
                msg, err := json.Marshal(message)
                if err != nil {
                        log.Println("Error marshalling JSON:", err)
                        continue
                }

                // отправляем как текстовое сообщение наш запрос на сервер по вебс
                err = conn.WriteMessage(websocket.TextMessage, msg)
                if err != nil {
                        log.Println("Error writing message:", err)
                        continue
                }

                // принимаем ответ
                _, res, err := conn.ReadMessage()
                if err != nil {
                        log.Println("Error reading message:", err)
                        continue
                }

                // читаем ответ, которое нам прислал сервер и с помощью функции
                // json.Unmarshal переводим из формата json
                // в нашу гошную структуру
                var response models.Response
                err = json.Unmarshal(res, &response)
                if err != nil {
                        log.Println("Error unmarshalling JSON:", err)
                        continue
                }

                // выводим все в консоль
                response.OutToConsole()
        }
}
```

Файл server.go

```
package main

import (
        "encoding/json"
        "fmt"
        "github.com/Baldislayer/iu9-networks/lab5/models"
        "github.com/gorilla/websocket"
        "iu9-networks/lab5/configs"
        "log"
        "net/http"
)

var upgrader = websocket.Upgrader{
        CheckOrigin: func(r *http.Request) bool {
                return true
        },
}

func handleClient(conn *websocket.Conn) {
        // закрываем соединение при завершении нашей функции
        // все, что написано после defer, выполняется при завершении функции, в ко
        // находится этот defer, если их несколько, то они выполняются в порядке с
        // последняя в коде функции будет выполнена первой
        defer conn.Close()

        fmt.Println("Соединение открыто")
        // запускаем бесконенчый цикл обработки сообщений от клиента
        for {
                _, msg, err := conn.ReadMessage()
                if err != nil {
                        // если нам выбило такую ошибку, то это значит, что просто
                        if err.Error() == "websocket: close 1006 (abnormal closure)
                                fmt.Println("Соединение закрыто")
                                break
                        }
                        log.Println("Error reading message:", err)

                        break
```

```
                }

                // читаем сообщение, которое нам прислал пользователь и с помощью
                // json.Unmarshal переводим из формата json в котором нам прислал
                // в нашу гошную структуру
                var message models.Message
                err = json.Unmarshal(msg, &message)
                if err != nil {
                        log.Println("Error unmarshalling JSON:", err)
                        break
                }

                // выводим то, что получили, это чисто для дебага, можешь убрать
                fmt.Println(message)

                // вызываем нашу функцию "подсчета" результата
                response, err := models.ResultFunction(message)
                if err != nil {
                        log.Println("Error unmarshalling JSON:", err)
                        break
                }

                // кастуем обратно в json для дальнейшей отправки результата на кл
                res, err := json.Marshal(response)
                if err != nil {
                        log.Println("Error marshalling JSON:", err)
                        break
                }

                // передаем наше сообщение на клиент как текст
                err = conn.WriteMessage(websocket.TextMessage, res)
                if err != nil {
                        log.Println("Error writing message:", err)
                        break
                }
        }
}

func main() {
        // тут мы собственно создаем "ручку" для обработки запросов на присоединен
```

```
        // ручка - обработчик каких-то определенных событий
        http.HandleFunc("/ws", func(w http.ResponseWriter, r *http.Request) {
                conn, err := upgrader.Upgrade(w, r, nil)
                if err != nil {
                        log.Println("Error upgrading to WebSocket:", err)
                        return
                }
                go handleClient(conn)
        })

        // поднимаем наш сервер
        log.Fatal(http.ListenAndServe(configs.Port, nil))
}
```

3 Скриншоты

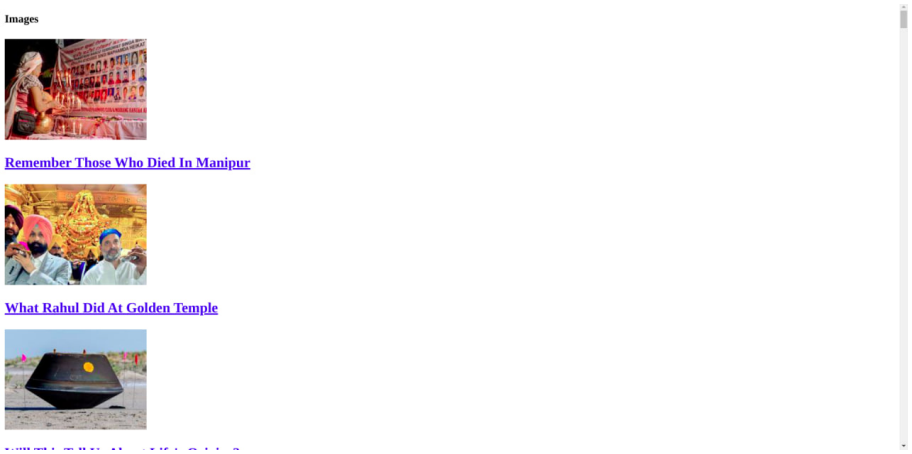
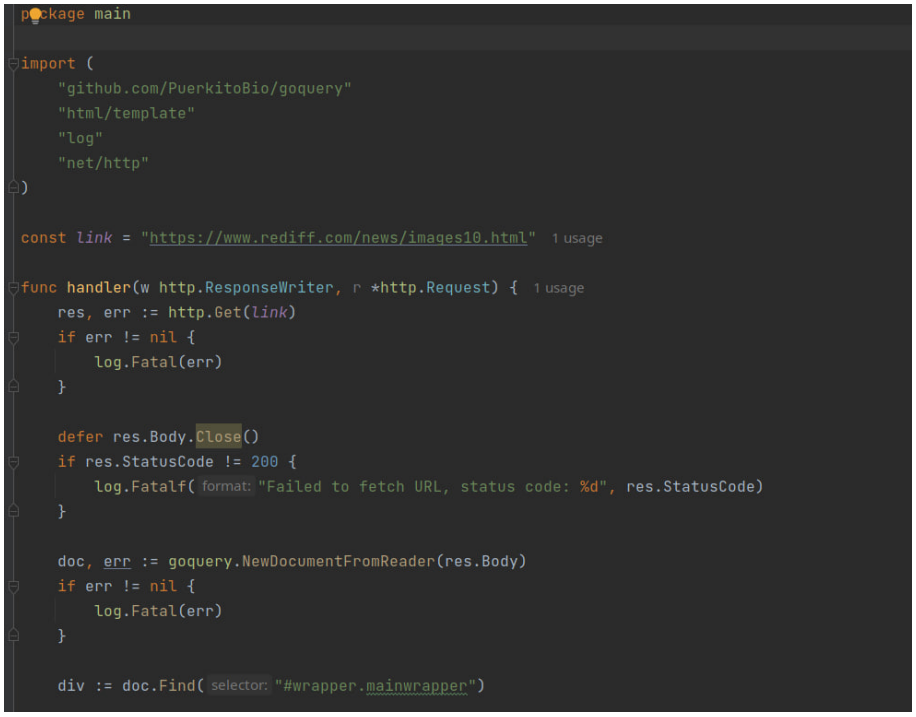


Рис. 1: Результат



```go
package main

import (
    "github.com/PuerkitoBio/goquery"
    "html/template"
    "log"
    "net/http"
)

const link = "https://www.rediff.com/news/images10.html"  1 usage

func handler(w http.ResponseWriter, r *http.Request) {  1 usage
    res, err := http.Get(link)
    if err != nil {
        log.Fatal(err)
    }

    defer res.Body.Close()
    if res.StatusCode != 200 {
        log.Fatalf( format: "Failed to fetch URL, status code: %d", res.StatusCode)
    }

    doc, err := goquery.NewDocumentFromReader(res.Body)
    if err != nil {
        log.Fatal(err)
    }

    div := doc.Find( selector: "#wrapper.mainwrapper")
```

Рис. 2: Код

```
    </head>
    <body>
        {{.}}
    </body>
    </html>
    `

    pageTmpl, err := template.New( name: "page").Parse(tmpl)
    if err != nil {
        log.Fatal(err)
    }

    w.Header().Set( key: "Content-Type",  value: "text/html; charset=utf-8")
    err = pageTmpl.Execute(w, template.HTML(html))
    if err != nil {
        log.Fatal(err)
    }
}

func main() {
    http.HandleFunc("/", handler)
    http.ListenAndServe( addr: ":8080",  handler: nil)
}
```

Рис. 3: Код