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 **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.01 **Информатика и вычислительная техника**

О Т Ч Е Т

по лабораторной работе № 6

Название: **Основы Back-End разработки на Golang**

Дисциплина: Языки Интернет Программирования

Студент	ИУ6-32Б		А.С.Авдеева
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель			В.Д.Шульман
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2024

Цель работы — изучение основ сетевого взаимодествия и серверной разработки с использованием языка Golang.

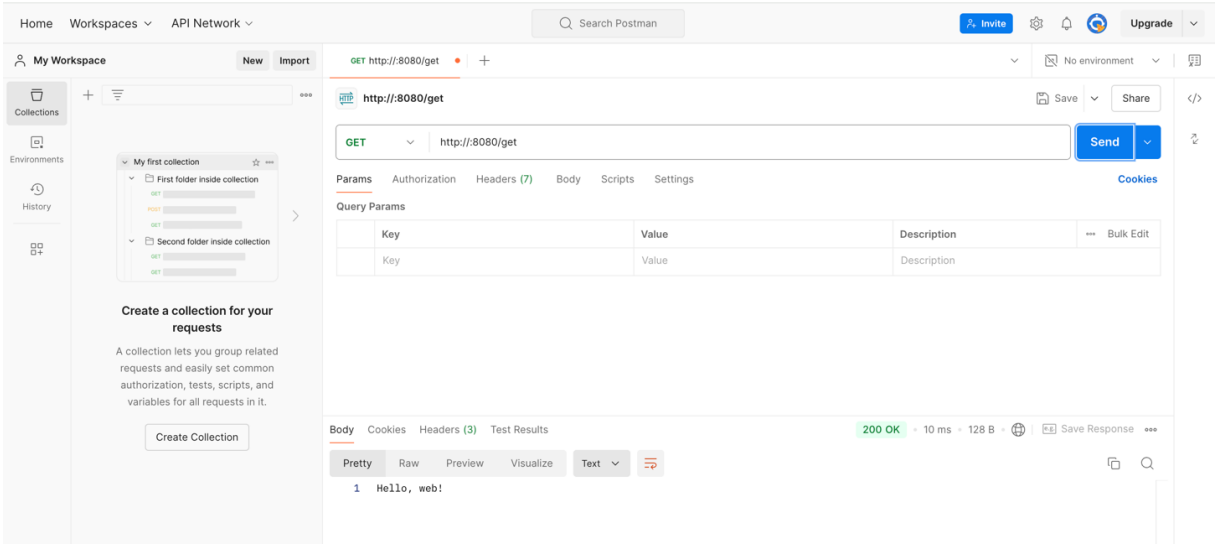
Задание 1_hello– Необходимо написать веб-сервер, который по пути /get отдает текст "Hello, web!". Порт должен быть :8080.
После решения задания полученный код main.go необходимо перенести в данный репозиторий.
Код должен компилироваться, а сервер запускаться и корректно обрабатывать запросы.
Для локальной отладки можно использовать Postman или Insomnia.

Ход работы и результаты:

1)Написание кода на Golang:

```go
package main

import (
    "fmt"
    "net/http"
)

func handler(w http.ResponseWriter, r *http.Request) {
    w.Write([]byte("Hello, web!"))
}

func main() {
    http.HandleFunc("/get", handler)
    err := http.ListenAndServe(":8080", nil)
    if err != nil {
       fmt.Println(err)
    }
}
```

2)Результат с использованием Postman(рис.№1)



(рис.№1)

Задание 2_query – Необходимо написать веб-сервер, который по пути /api/user приветствует пользователя. Сервер по этому пути должен принимать и парсить параметр name, после этого отвечая в формате: "Hello,<name>!". Пример url: /api/user?name=Golang
После решения задания полученный код main.go необходимо перенести в данный репозиторий в директорию с данным файлом README.md
Код должен компилироваться, а сервер запускаться и корректно обрабатывать запросы.
Для локальной отладки можно использовать Postman или Insomnia.

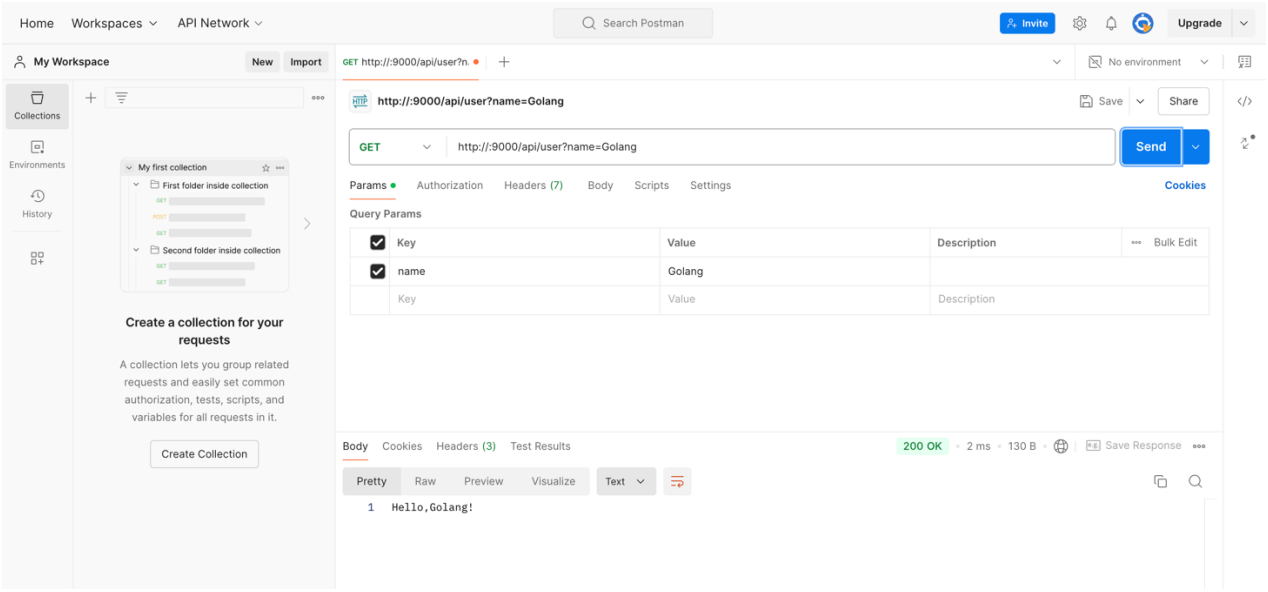
Ход работы и результаты:

1) Написание кода на Golang:

```
package main

import (
    "fmt"
    "net/http"
)

func handler(w http.ResponseWriter, r *http.Request){
    w.Write([]byte("Hello," + r.URL.Query().Get("name") + "!"))
}

func main() {
    http.HandleFunc("/api/user", handler)
    err := http.ListenAndServe(":9000", nil)
    if err != nil{
```

```
        fmt.Println(err)
    }
}
```

2)Результат с использованием Postman(рис.№2)



(рис.№2)

Задание 3_count – Напиши веб сервер (**порт :3333**) - счетчик который будет обрабатывать GET (/count) и POST (/count) запросы:

GET: возвращает счетчик

POST: увеличивает ваш счетчик на значение (с ключом "count") которое вы получаете из формы, но если пришло НЕ число то нужно ответить клиенту: "это не число" со статусом http.StatusBadRequest (400).

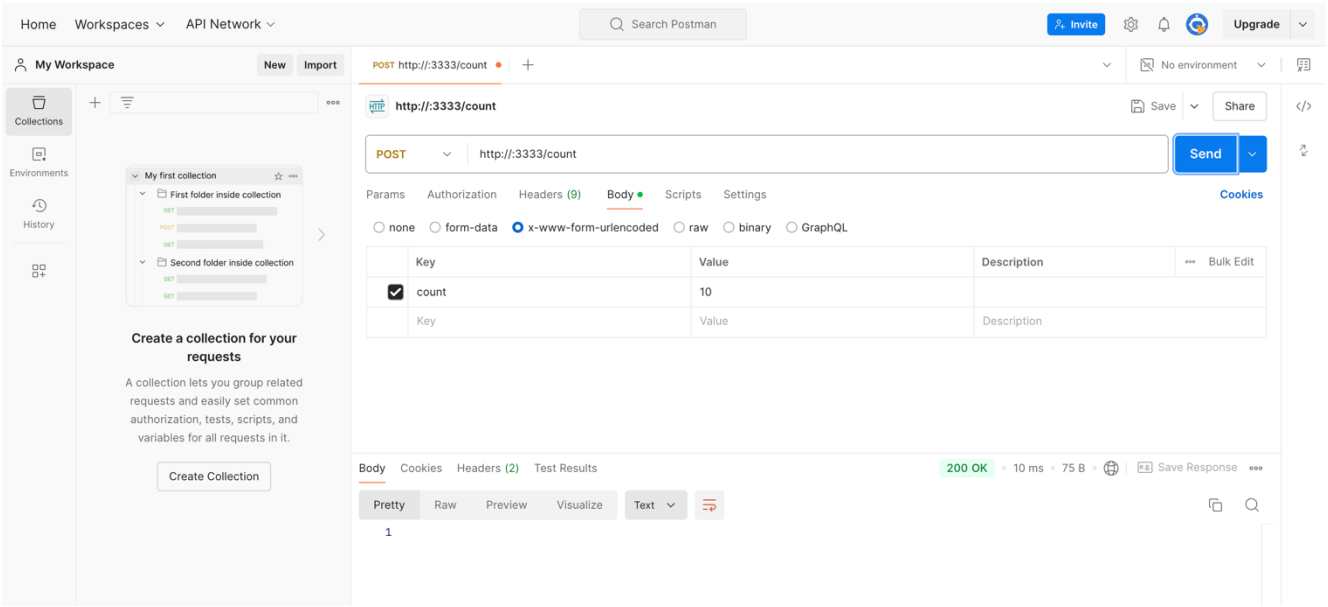
Ход работы и результаты:

1) Написание кода на Golang:

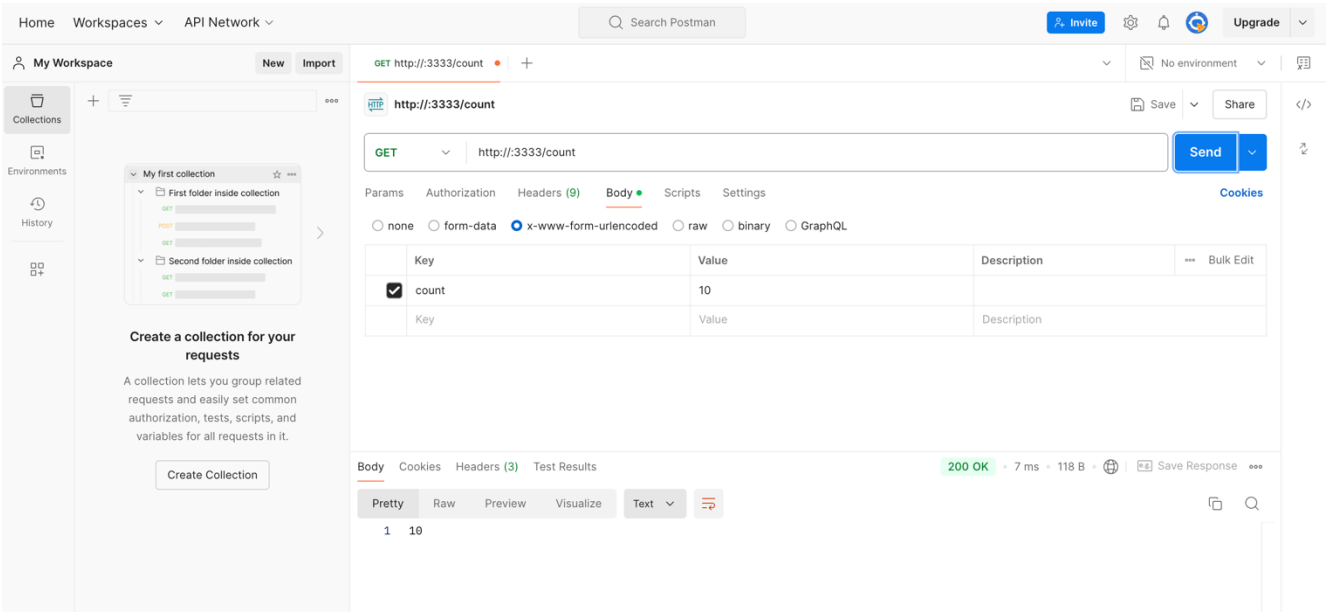
```
package main

import (
    "fmt"
    "net/http"
    "strconv"
)

var count int
```

```go
func handler(w http.ResponseWriter, r *http.Request){
    if r.Method == "GET"{
       w.Write([]byte(strconv.Itoa(count)))
       return
    }else if r.Method == "POST"{
       r.ParseForm()
       numS := r.Form.Get("count")
       num, err := strconv.Atoi(numS)
       if err != nil{
          w.WriteHeader(http.StatusBadRequest)
          w.Write([]byte("это не число"))
          return
       }
       count += num
    }
}

func main(){
    http.HandleFunc("/count", handler)
    err := http.ListenAndServe(":3333", nil)
    if err != nil{
       fmt.Println(err)
    }
}
```

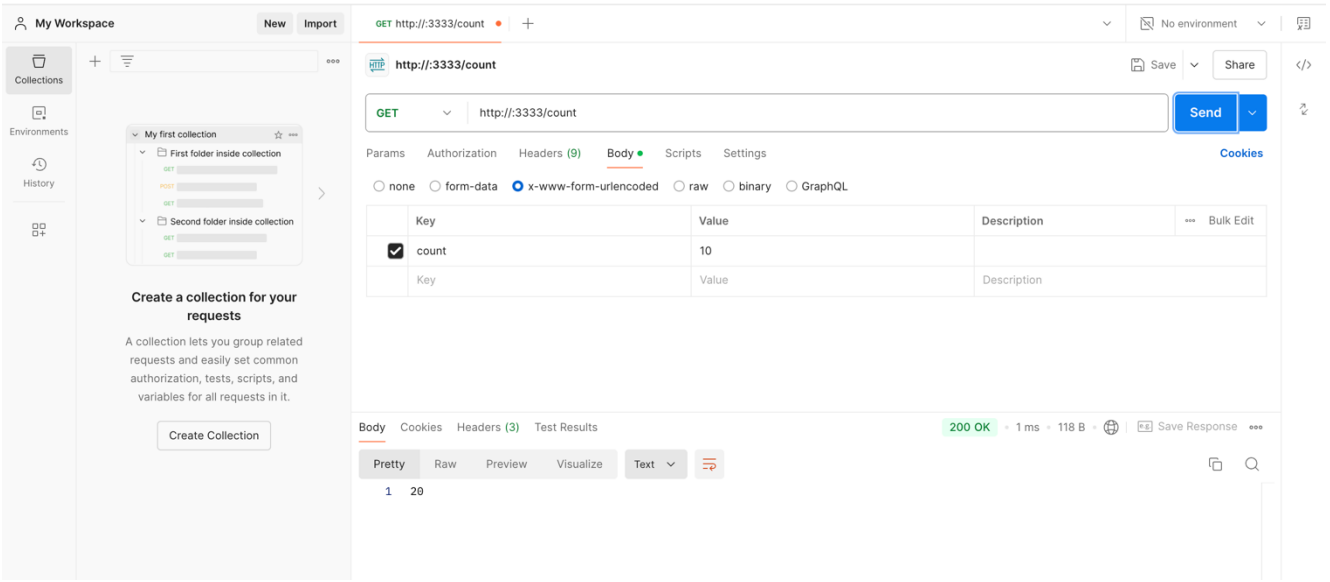
2)Результат с использованием Postman, если count - число(рис.№3, №4, №5)



(рис.№3)-count = 10(POST)

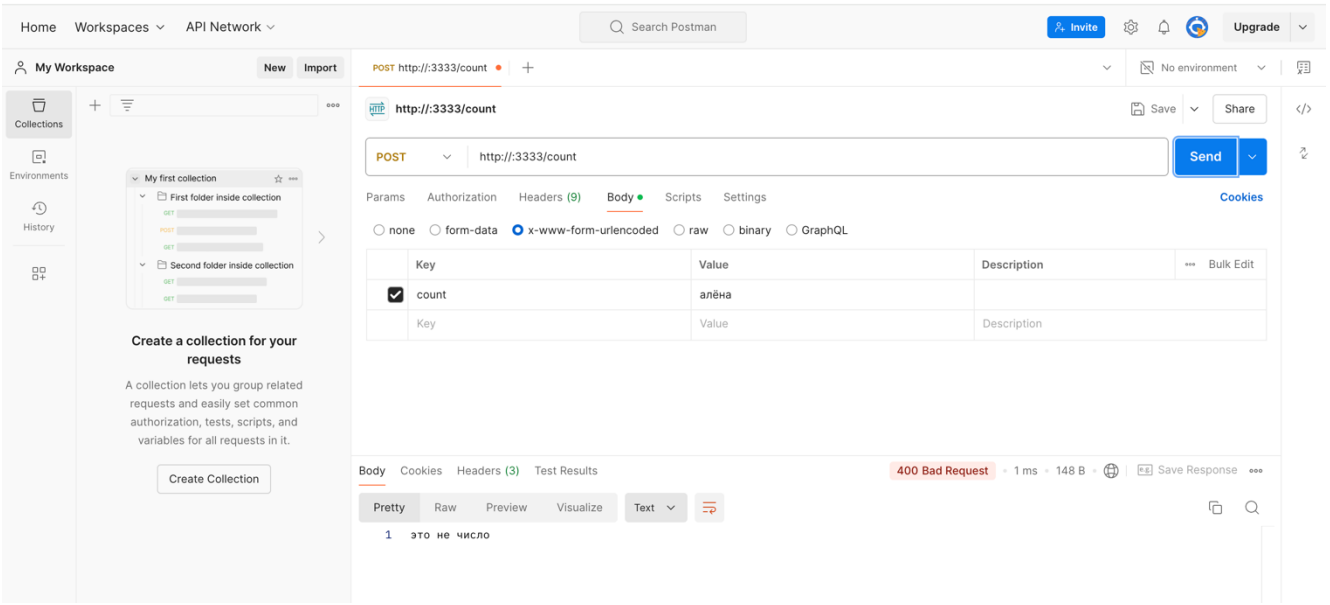


(рис.№4)-GET(count = 10)

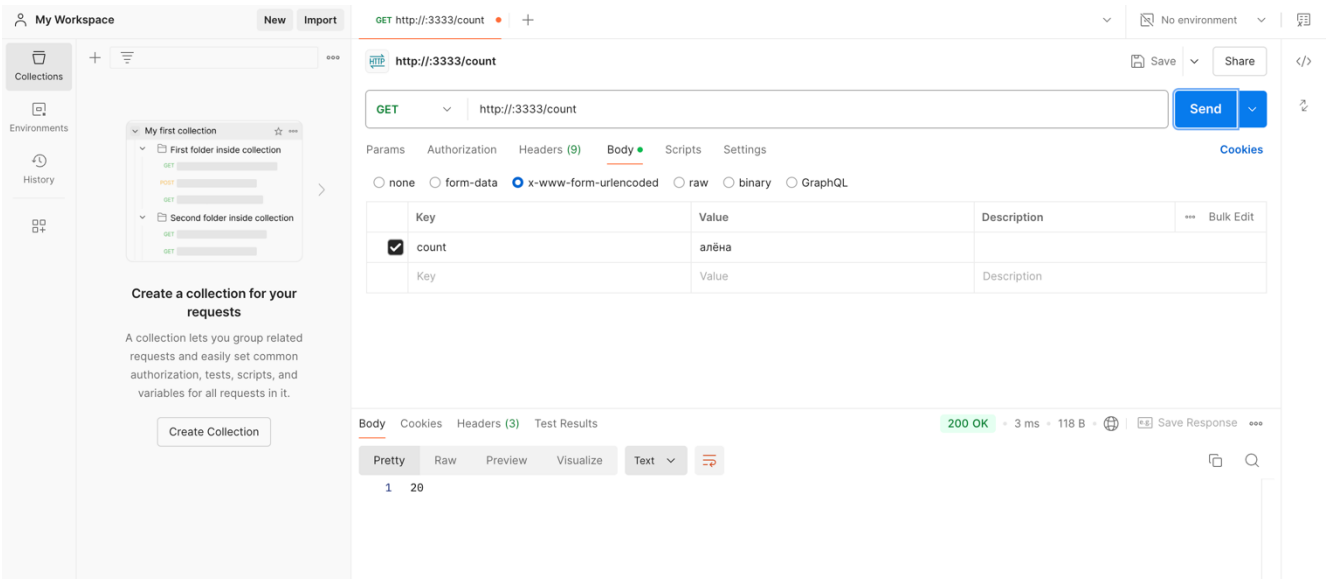


(рис.№5)-увеличили счётчик вновь на count = 10

3)если count – не число(рис.№6, рис.№7)



(рис.№6)-POST



(рис.№7)-GET, счётчик больше не увеличивается

Контрольные вопросы:

1)В чём разница между протоколами TCP и UDP ?

UDP протокол – это когда нам не особо важно и критично, если произойдёт небольшая потеря передачи данных, при этом важна скорость передачи данных, например, если используется стриминг, то есть если проводится прямая трансляция, то пользователю передаётся звук и видео, нам же не будет критично, если звук и видео будут немного лагать, при этом скорость предачи этих данных будет высокой, а TCP протокол – это более надёжная передача данных, где нам важно, чтобы все данные передались, то есть, например, веб-браузинг, когда при

открытии страницы пользователь видит все данные в правильном порядке и без потерь.

2)Для чего нужны IP Address и Port Number у веб-сервера и в чём разница?

IP Address и Port Number позволяют взаимодействовать друг с другом веб-серверу и клиентам. IP Address – это адрес устройства в сети, он показывает, где находится веб-сервер, а Port Number – это адрес приложения на устройстве, который указывает на конкретную программу, с которой хочет связаться клиент.

3)Какой набор методов в HTTP-request в полной мере релализует семантику CRUD ?

CRUD – это Create Read Update Delete.

Набор методов:

POST(create)

GET(read)

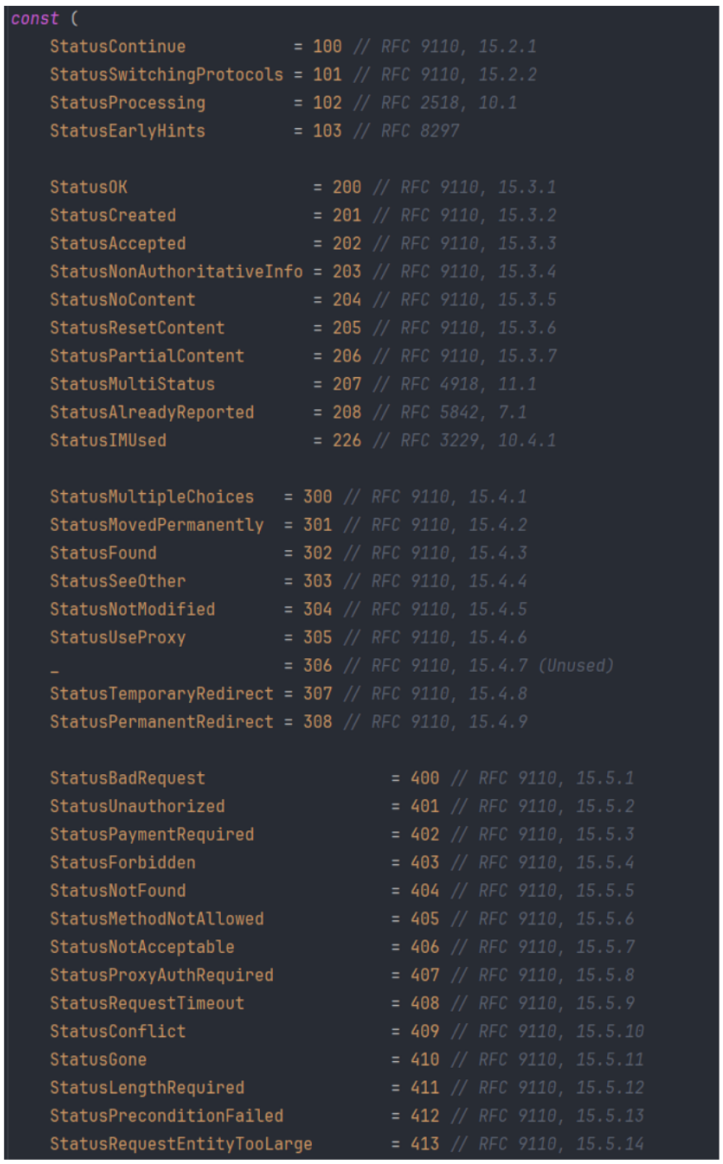
PUT(update)

PATCH (update)

DELETE(delete)

4)Какие группы status code существуют у HTTP-response (желательно, с примерами) ?

Status code – это код состояния, который информирует клиента о результате его запроса.



```
const (
    StatusContinue           = 100 // RFC 9110, 15.2.1
    StatusSwitchingProtocols = 101 // RFC 9110, 15.2.2
    StatusProcessing         = 102 // RFC 2518, 10.1
    StatusEarlyHints         = 103 // RFC 8297

    StatusOK                   = 200 // RFC 9110, 15.3.1
    StatusCreated              = 201 // RFC 9110, 15.3.2
    StatusAccepted             = 202 // RFC 9110, 15.3.3
    StatusNonAuthoritativeInfo = 203 // RFC 9110, 15.3.4
    StatusNoContent            = 204 // RFC 9110, 15.3.5
    StatusResetContent         = 205 // RFC 9110, 15.3.6
    StatusPartialContent       = 206 // RFC 9110, 15.3.7
    StatusMultiStatus          = 207 // RFC 4918, 11.1
    StatusAlreadyReported      = 208 // RFC 5842, 7.1
    StatusIMUsed               = 226 // RFC 3229, 10.4.1

    StatusMultipleChoices   = 300 // RFC 9110, 15.4.1
    StatusMovedPermanently  = 301 // RFC 9110, 15.4.2
    StatusFound             = 302 // RFC 9110, 15.4.3
    StatusSeeOther          = 303 // RFC 9110, 15.4.4
    StatusNotModified       = 304 // RFC 9110, 15.4.5
    StatusUseProxy          = 305 // RFC 9110, 15.4.6
    _                       = 306 // RFC 9110, 15.4.7 (Unused)
    StatusTemporaryRedirect = 307 // RFC 9110, 15.4.8
    StatusPermanentRedirect = 308 // RFC 9110, 15.4.9

    StatusBadRequest                   = 400 // RFC 9110, 15.5.1
    StatusUnauthorized                 = 401 // RFC 9110, 15.5.2
    StatusPaymentRequired              = 402 // RFC 9110, 15.5.3
    StatusForbidden                    = 403 // RFC 9110, 15.5.4
    StatusNotFound                     = 404 // RFC 9110, 15.5.5
    StatusMethodNotAllowed             = 405 // RFC 9110, 15.5.6
    StatusNotAcceptable                = 406 // RFC 9110, 15.5.7
    StatusProxyAuthRequired            = 407 // RFC 9110, 15.5.8
    StatusRequestTimeout               = 408 // RFC 9110, 15.5.9
    StatusConflict                     = 409 // RFC 9110, 15.5.10
    StatusGone                         = 410 // RFC 9110, 15.5.11
    StatusLengthRequired               = 411 // RFC 9110, 15.5.12
    StatusPreconditionFailed           = 412 // RFC 9110, 15.5.13
    StatusRequestEntityTooLarge        = 413 // RFC 9110, 15.5.14
```

(примеры)

5)Из каких составных элементов состоит HTTP-request и HTTP-response ?

HTTP-request:

1)Методы(POST, GET и т.д.)

2)URL-адрес ресурса, к которому указан запрос

3)заголовки(Headers)-дополнительная информация о запросе

4)тело(body)-содержит данные, которые отправляются серверу

HTTP-response

1)код состояния(status code)-указывают успешно ли был обработан запрос

2)заголовки(Headers)-содержат информацию об ответе

3)тело(body)-содержат данные, отправленные сервером в ответ на запрос

Заключение – проделана успешная работа в понимании работы с http-запросами на языке Golang!