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 № 9

Название: **Back-End разработка с использованием фреймворка Echo**

Дисциплина: Языки интернет программирования

Студент ИУ6-33Б 11.12.24 Пономаренко В.М.
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель 11.12.24 Шульман В.Д.
(Подпись, дата) (И.О. Фамилия)

Москва, 2024

Цель работы: получение первичных навыков использования веб-фрейворков в BackEnd-разработке на Golang

Задание:

Доработать сервисы таким образом, чтобы роутинг, обработка запросов, парсинг json, обработка ошибок и логирование осуществлялись на базе фреймворка Echo

Ход работы:

Задание 1. Hello

1.Описание реализуемого функционала

1) Get – выводит случайное сообщение из базы данных
2) Post – добавление в базу данных нового сообщения

Прописываем команды
go mod init hello
go get github.com/labstack/echo/v4
go get github.com/lib/pq
для установки Echo и драйвера PostgreSQL

2.Листинг файла hello.go

Реализуем дополнительную проверку, что сообщение не пустая строка (которая состоит только из пробелов)

```go
package main

import (
    "database/sql"
    "flag"
    "fmt"
    "log"
    "net/http"
    "strings"

    "github.com/labstack/echo/v4"
    _ "github.com/lib/pq"
)

const (
    host     = "localhost"
    port     = 5432
    user     = "postgres"
    password = "catjkm8800"
    dbname   = "hello"
)

type Handlers struct {
    dbProvider DatabaseProvider
}

type DatabaseProvider struct {
    db *sql.DB
}

// Обработчики HTTP-запросов
func (h *Handlers) GetHello(c echo.Context) error {
    msg, err := h.dbProvider.SelectHello()
    if err != nil {
        return c.String(http.StatusInternalServerError, err.Error())
    }
    return c.String(http.StatusOK, msg)
}
```

```go
func (h *Handlers) PostHello(c echo.Context) error {
    input := struct {
        Msg string `json:"msg"`
    }{}
    if err := c.Bind(&input); err != nil {
        return c.String(http.StatusBadRequest, "Incorrect format of  JSON")
    }
    //проверка, что в сообщении хоть что-то есть
    //TrimSpace удаляет все пробельные символы (включая пробелы, табуляции и переводы строк)
    if strings.TrimSpace(input.Msg) == "" {
        return c.String(http.StatusBadRequest, "No message")
    }
    err := h.dbProvider.InsertHello(input.Msg)
    if err != nil {
        return c.String(http.StatusInternalServerError, err.Error())
    }
    return c.String(http.StatusCreated, "Added")
}

// Методы для работы с базой данных

func (dp *DatabaseProvider) SelectHello() (string, error) {
    var msg string
    row := dp.db.QueryRow("SELECT message FROM hello ORDER BY RANDOM() LIMIT 1")
    err := row.Scan(&msg)
    if err != nil {
        return "", err
    }
    return msg, nil
}

func (dp *DatabaseProvider) InsertHello(msg string) error {
    _, err := dp.db.Exec("INSERT INTO hello (message) VALUES ($1)", msg)
    return err
}

func main() {
    address := flag.String("address", "127.0.0.1:8081", "адрес для запуска сервера")
    flag.Parse()

    psqlInfo := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s sslmode=disable", host,
port, user, password, dbname)

    db, err := sql.Open("postgres", psqlInfo)
    if err != nil {
        log.Fatal(err)
    }
    defer db.Close()

    err = db.Ping()
    if err != nil {
        log.Fatal("Failed to connect to the database:", err)
    }
    fmt.Println("Connected!")

    dp := DatabaseProvider{db: db}
    h := Handlers{dbProvider: dp}

    e := echo.New()

    e.GET("/get", h.GetHello)
    e.POST("/post", h.PostHello)
```

```
    err = e.Start(*address)
    if err != nil {
        log.Fatal(err)
    }
}
```

3.Приведем результаты работы

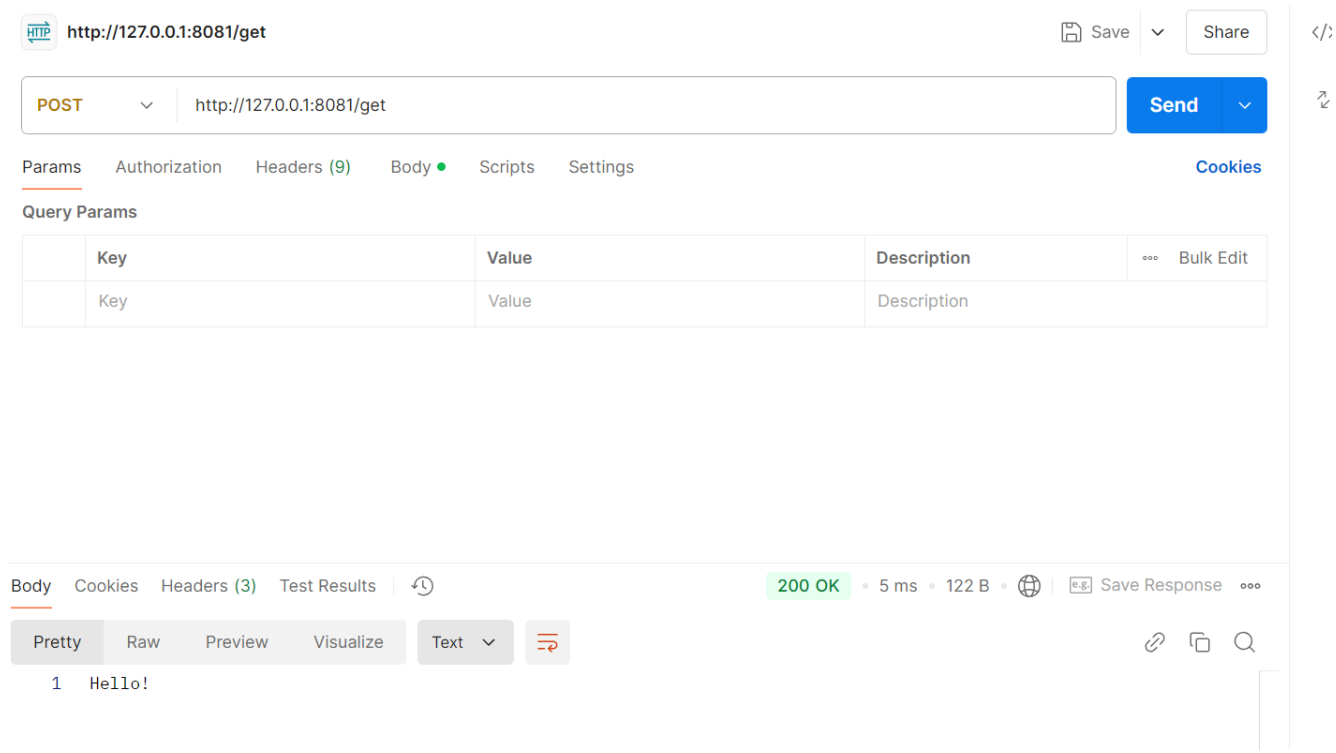

Рисунок 1 - тест 1 (GET)

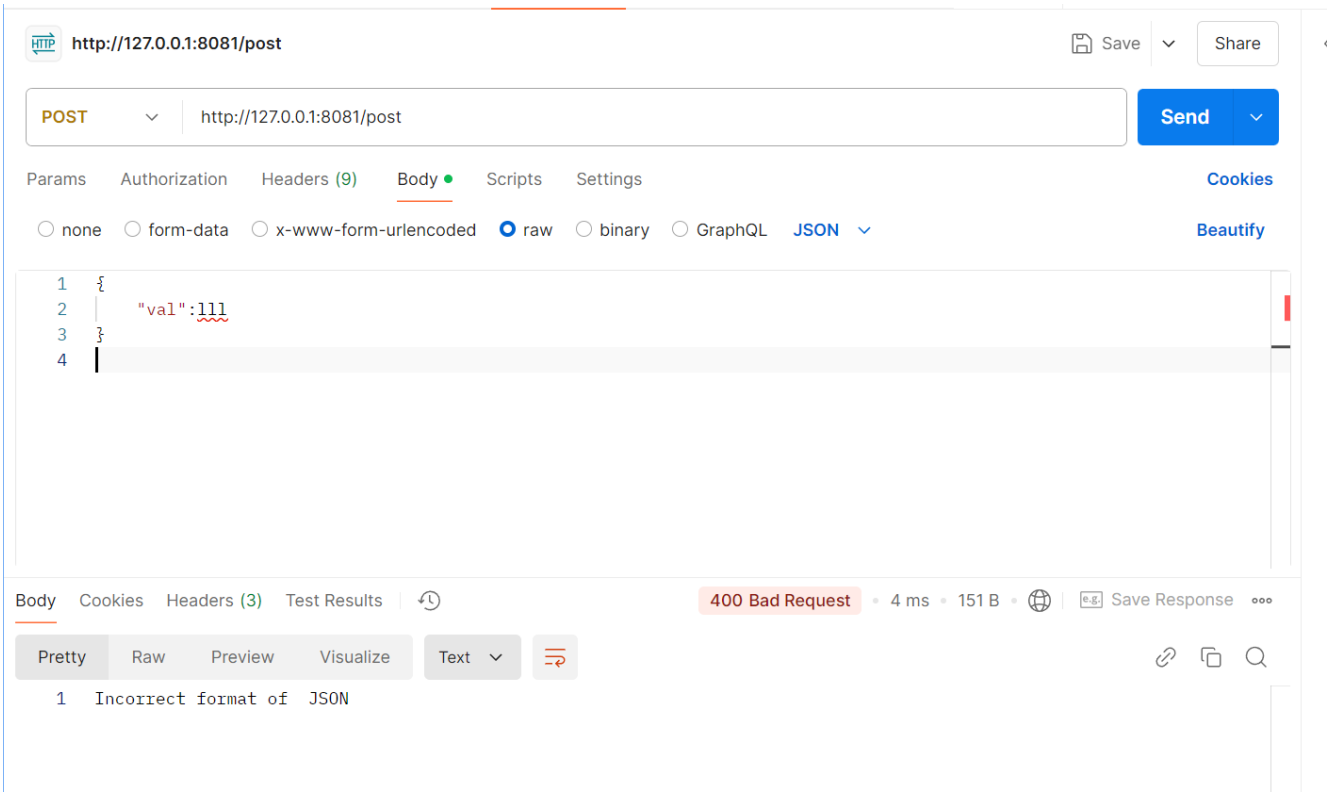

Рисунок 2 - тест 2 отсутсвие нужного поля в JSON

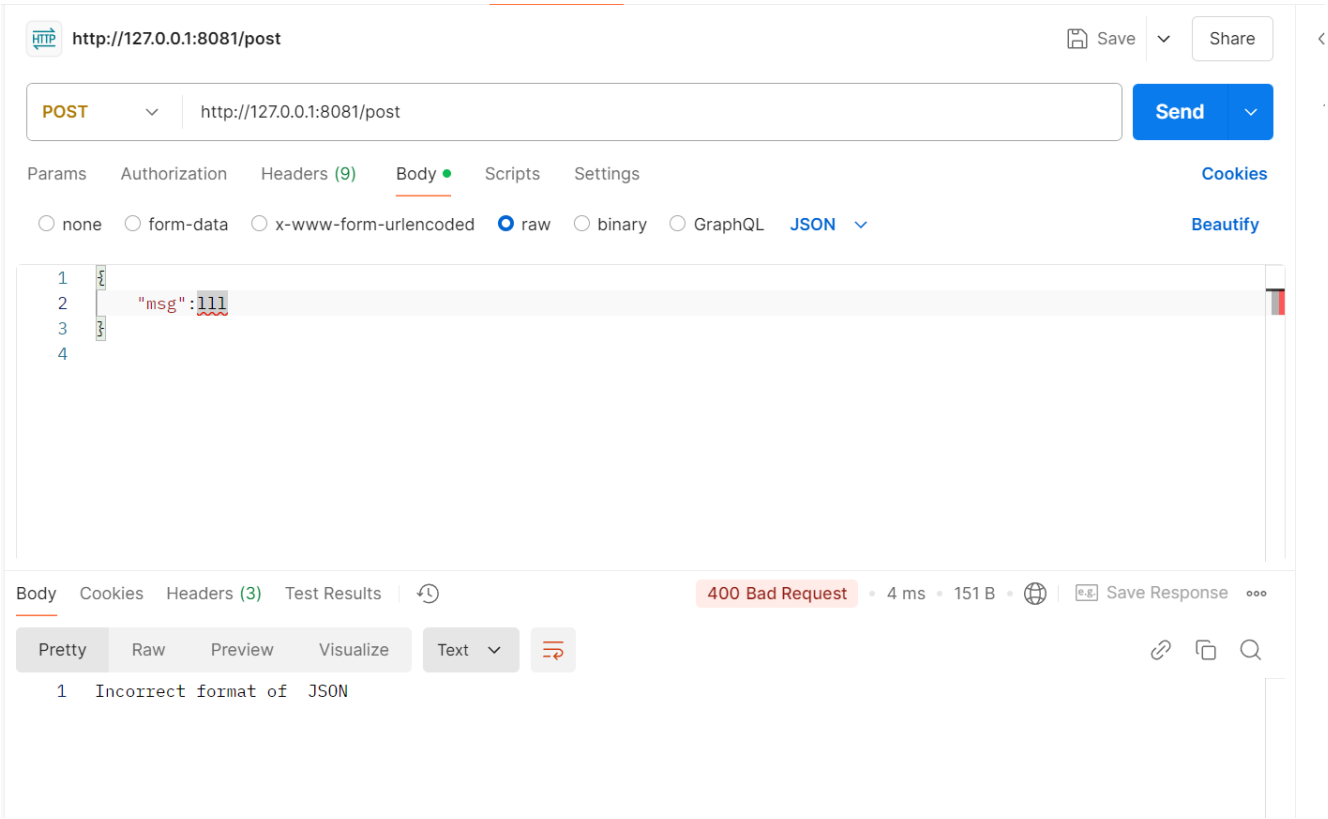

Рисунок 3 -тест 3 неккоректное значение

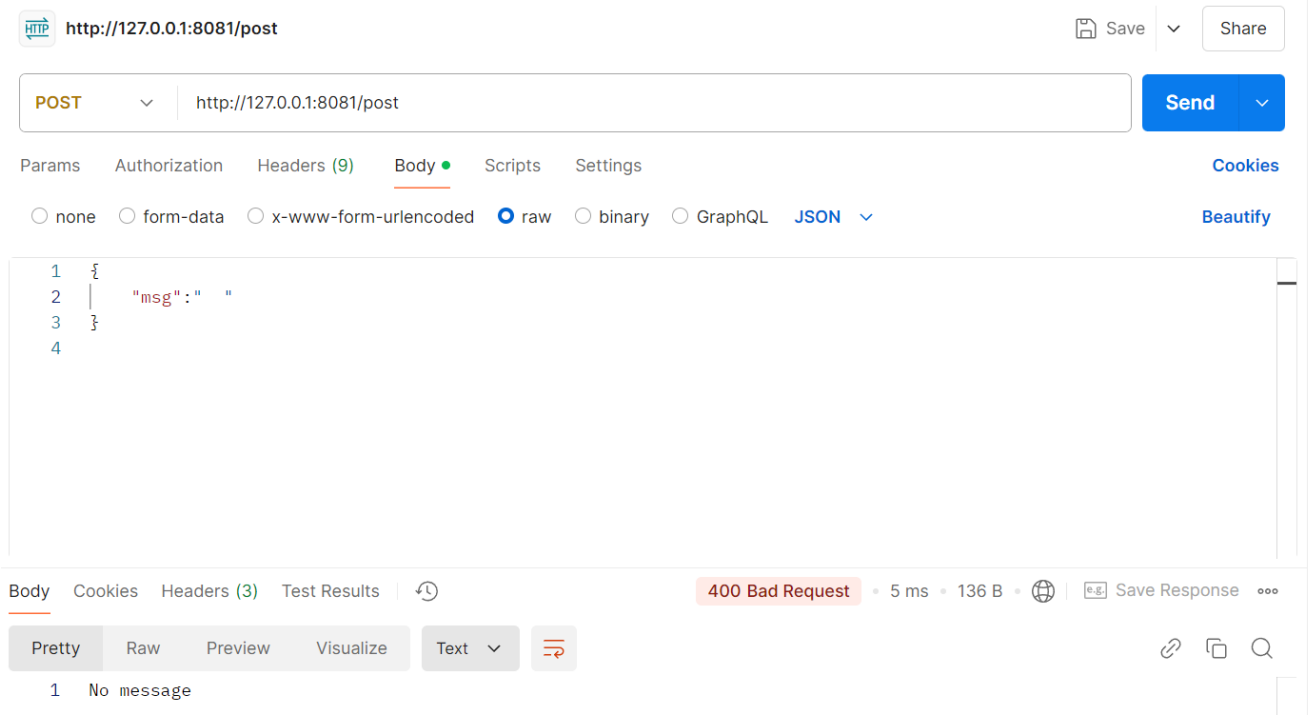

Рисунок 4 - тест 4 пустая строка

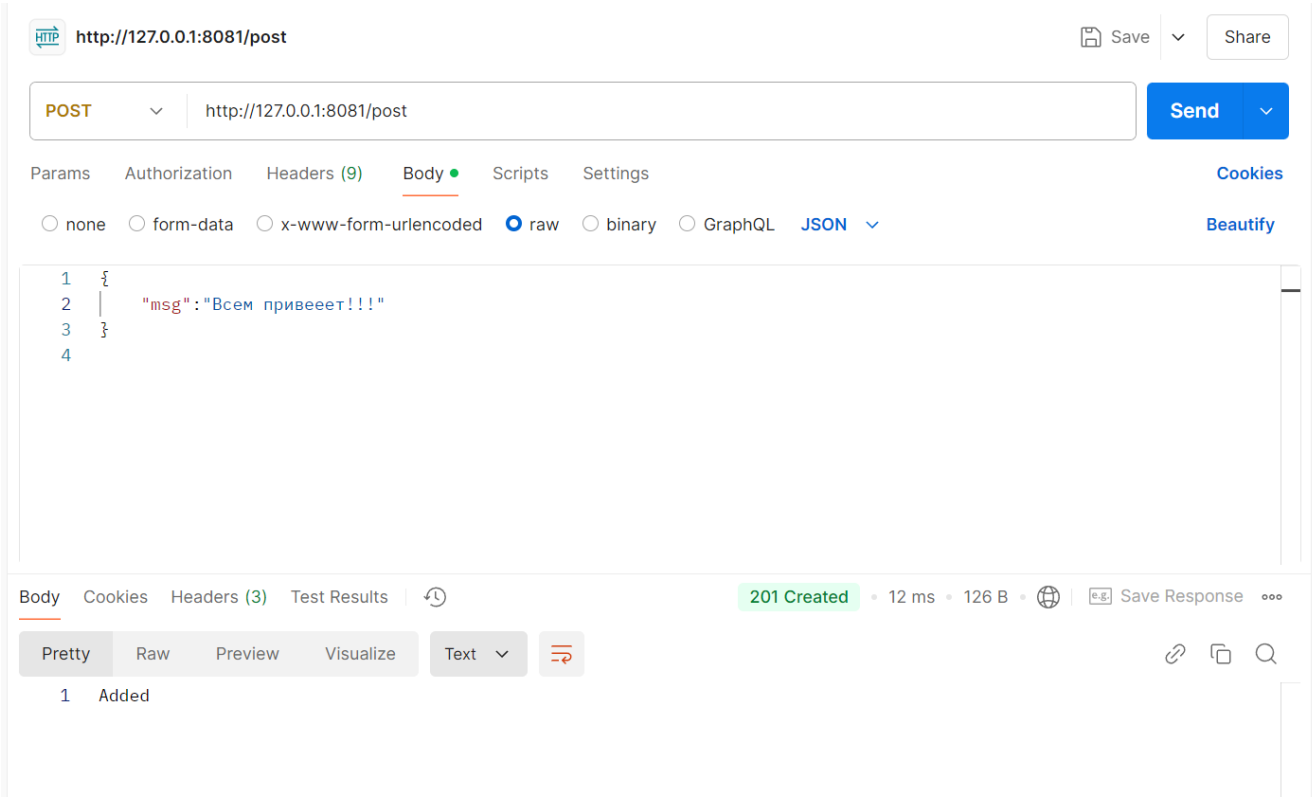

Рисунок 5 - тест 5 кореектный POST

Задание 2. Query

Прописываем команды
go mod init query
go get github.com/labstack/echo/v4
go get github.com/lib/pq
для установки Echo и драйвера PostgreSQL

```
PS D:\Go\lab9> go mod init query
go: creating new go.mod: module query
go: to add module requirements and sums:
        go mod tidy
PS D:\Go\lab9> go get github.com/labstack/echo/v4
go: downloading github.com/labstack/echo/v4 v4.13.1
go: downloading github.com/labstack/echo v3.3.10+incompatible
go: downloading golang.org/x/crypto v0.22.0
go: downloading golang.org/x/net v0.24.0
go: downloading github.com/labstack/gommon v0.4.2
go: downloading github.com/valyala/fasttemplate v1.2.2
go: downloading golang.org/x/sys v0.19.0
go: downloading github.com/valyala/bytebufferpool v1.0.0
go: downloading golang.org/x/text v0.14.0
go: added github.com/labstack/echo/v4 v4.13.1
go: added github.com/labstack/gommon v0.4.2
go: added github.com/mattn/go-colorable v0.1.13
go: added github.com/mattn/go-isatty v0.0.20
go: added github.com/valyala/bytebufferpool v1.0.0
go: added github.com/valyala/fasttemplate v1.2.2
go: added golang.org/x/crypto v0.22.0
go: added golang.org/x/net v0.24.0
go: added golang.org/x/sys v0.19.0
go: added golang.org/x/text v0.14.0
PS D:\Go\lab9> go get github.com/lib/pq
go: added github.com/lib/pq v1.10.9
```

1. Описание реализуемого функционала
   3) Get – выводит приветствие для последнего пользователя, внесенного в БД
   4) Post – создает в БД новую запись для пользователя, чье имя отправлено в Query-параметре name

5) Put – изменяет имя последнего пользователя в БД на имя в Query-параметре name

2. Ниже приведен листинг файла query.go

```go
package main

import (
    "database/sql"
    "flag"
    "fmt"
    "log"
    "net/http"
    "regexp"

    "github.com/labstack/echo/v4"
    _ "github.com/lib/pq"
)

const (
    host     = "localhost"
    port     = 5432
    user     = "postgres"
    password = "catjkm8800"
    dbname   = "query"
)

type Handlers struct {
    dbProvider DatabaseProvider
}

type DatabaseProvider struct {
    db *sql.DB
}

// Обработчики HTTP-запросов
func (h *Handlers) GetQuery(c echo.Context) error {
    msg, err := h.dbProvider.SelectQuery()
    if err != nil {
        return c.String(http.StatusInternalServerError, err.Error())
    }
    return c.String(http.StatusOK, "Hello "+msg+"!")
}

func (h *Handlers) PostQuery(c echo.Context) error {
    nameInput := c.QueryParam("name") // Получаем Query-параметр
    if nameInput == "" {
        return c.String(http.StatusBadRequest, "Missing 'name' query parameter")
    }
    re := regexp.MustCompile(`[a-zA-Za-яА-Я]`)
    if !re.MatchString(nameInput) {
        return c.String(http.StatusBadRequest, "empty string")
    }
    err := h.dbProvider.InsertQuery(nameInput)
    if err != nil {
        return c.String(http.StatusInternalServerError, err.Error())
    }
    return c.String(http.StatusCreated, "Created")
}

func (h *Handlers) PutQuery(c echo.Context) error {
    nameInput := c.QueryParam("name") // Получаем Query-параметр
    if nameInput == "" {
        return c.String(http.StatusBadRequest, "Missing 'name' query parameter")
    }
    re := regexp.MustCompile(`[a-zA-Za-яА-Я]`)
    if !re.MatchString(nameInput) {
        return c.String(http.StatusBadRequest, "empty string")
    }
    err := h.dbProvider.UpdateQuery(nameInput)
    if err != nil {
        return c.String(http.StatusInternalServerError, err.Error())
    }
    return c.String(http.StatusOK, "Updated")
}

// Методы для работы с базой данных
func (dbp *DatabaseProvider) SelectQuery() (string, error) {
    var msg string
    row := dbp.db.QueryRow("SELECT name FROM query ORDER BY id DESC LIMIT 1")
    err := row.Scan(&msg)
    if err != nil {
        return "", err
    }
    return msg, nil
}

func (dbp *DatabaseProvider) UpdateQuery(n string) error {
```

```
    _, err := dbp.db.Exec("UPDATE query SET name = $1 WHERE id = (SELECT MAX(id) FROM query)", n)
    if err != nil {
        return err
    }
    return nil
}

func (dbp *DatabaseProvider) InsertQuery(n string) error {
    _, err := dbp.db.Exec("INSERT INTO query (name) VALUES ($1)", n)
    if err != nil {
        return err
    }
    return nil
}

func main() {
    address := flag.String("address", "127.0.0.1:8081", "адрес для запуска сервера")
    flag.Parse()

    psqlInfo := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s sslmode=disable", host, port, user, password, dbname)

    db, err := sql.Open("postgres", psqlInfo)
    if err != nil {
        log.Fatal(err)
    }
    defer db.Close()

    err = db.Ping()
    if err != nil {
        log.Fatal("Failed to connect to the database:", err)
    }
    fmt.Println("Connected!")

    dp := DatabaseProvider{db: db}
    h := Handlers{dbProvider: dp}

    e := echo.New()

    e.GET("/get", h.GetQuery)
    e.POST("/post", h.PostQuery)
    e.PUT("/put", h.PutQuery)

    err = e.Start(*address)
    if err != nil {
        log.Fatal(err)
    }
}
```

3. Дополнительная доработка: проверка, что строка содержит хотя бы 1 букву русского или латинского алфавита (в условиях задачи будем считать это корректным именем). Реализовано, чтобы не допустить появления в базе данных строк с пустым значением name.
4. Приведем примеры работы программы

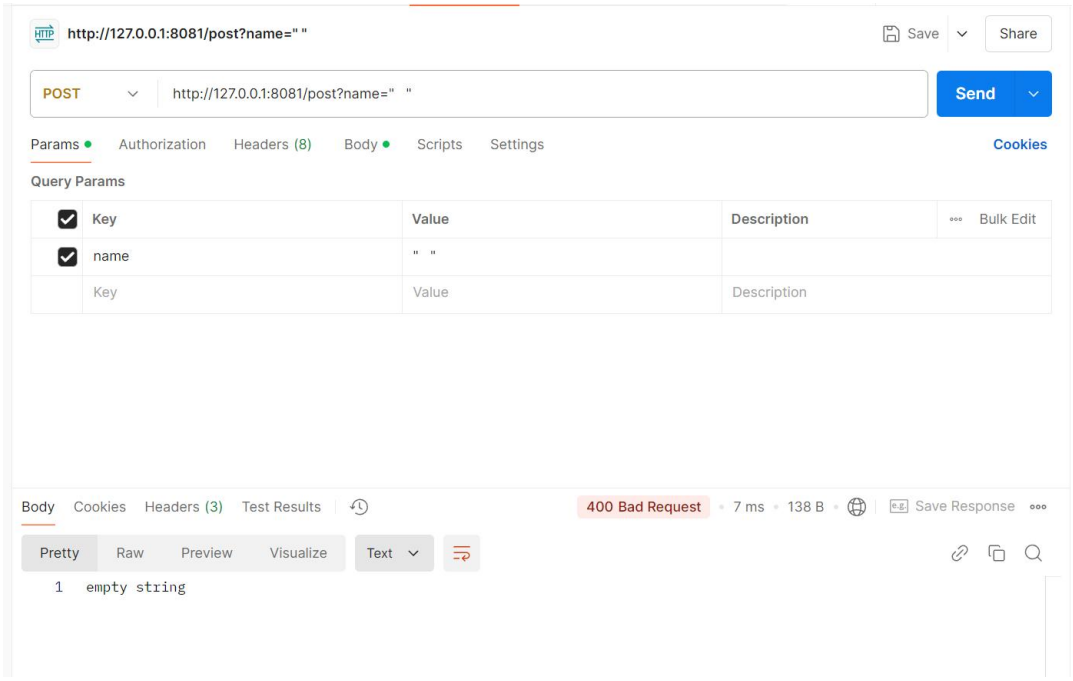

Рисунок 6 – тест 1 - с пустой строкой

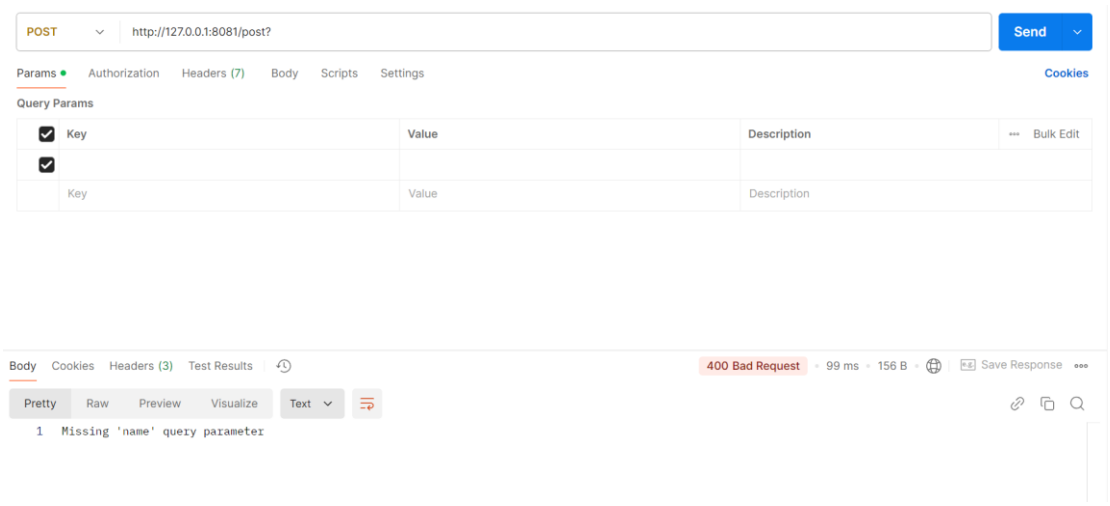

Рисунок 7 - тест 2 - отсутствует параметр

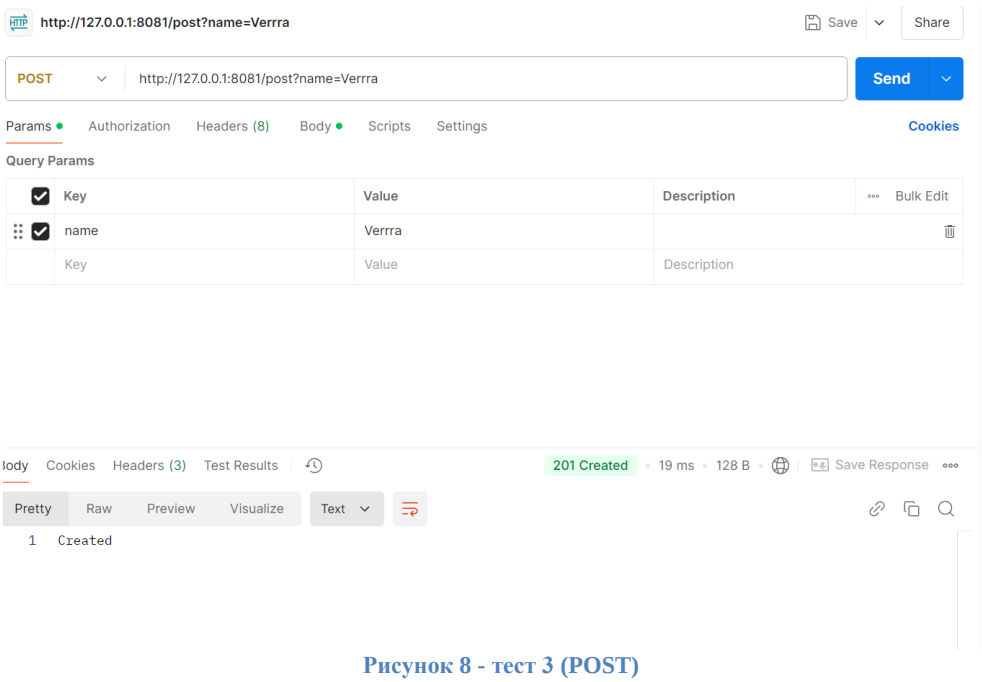

Рисунок 8 - тест 3 (POST)

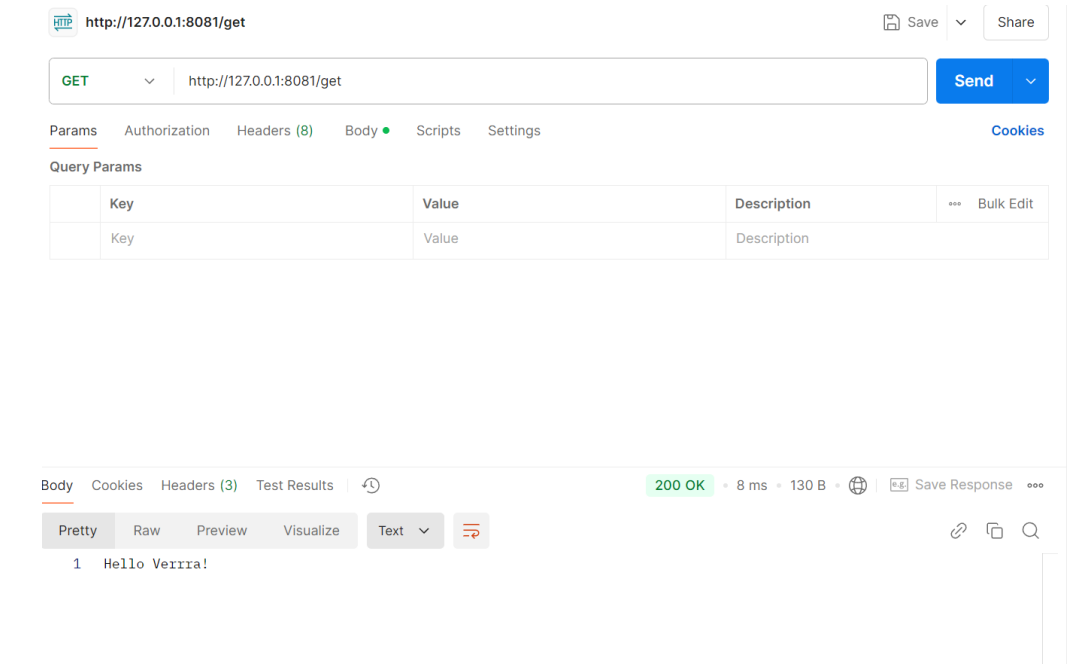

Рисунок 9 - тест 3 (GET)

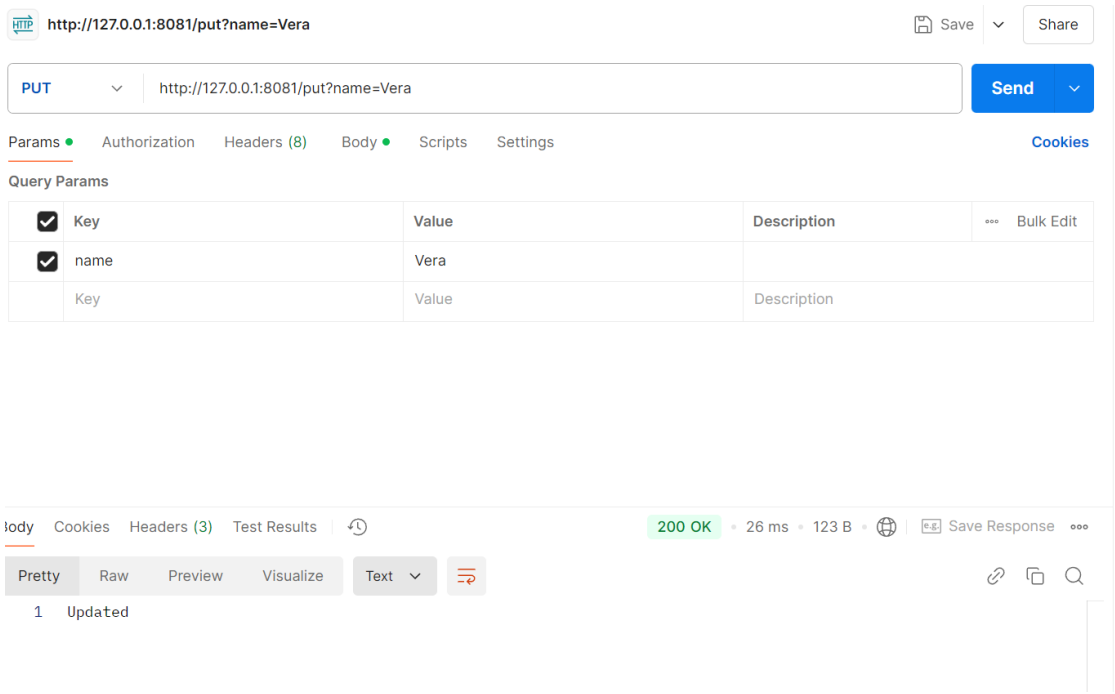

Рисунок 10 - тест 4 (PUT)

Для демонстрации функционала изменения последнего пользователя приведем саму БД.

21	21	qwe1
22	22	qwe2
23	23	qwe3
24	24	Verra

21	21	qwe1
22	22	qwe2
23	23	qwe3
24	24	Vera

Рисунок 11 – демонстрация изменения БД

5. Так как работа производится под ОС Windows, для которой команда make lint не является традиционной, то проверка осуществлялась альтернативным способом.

Приведем листинг специального файла build.ps1

```
# Set the output encoding to UTF-8
[Console]::OutputEncoding = [System.Text.Encoding]::UTF8
```

```powershell
# Start golangci-lint
Write-Host "Running golangci-lint..." -ForegroundColor Cyan
golangci-lint run ./...

# Check the status of the linter execution
if ($LASTEXITCODE -ne 0) {
    Write-Host "Error during golangci-lint execution." -ForegroundColor Red
    exit $LASTEXITCODE
} else {
    Write-Host "golangci-lint completed successfully." -ForegroundColor Green
}

# Compile the project
Write-Host "Compiling the project..." -ForegroundColor Cyan
go build -o myapp.exe .

# Check the status of the compilation
if ($LASTEXITCODE -ne 0) {
    Write-Host "Error during project compilation." -ForegroundColor Red
    exit $LASTEXITCODE
} else {
    Write-Host "Project successfully compiled to myapp.exe." -ForegroundColor Green
}

Write-Host "Script completed successfully." -ForegroundColor Green
```

go install github.com/golangci/golangci-lint/cmd/golangci-lint@latest
.\build.ps1

```
PS D:\Go\lab9\query> go install github.com/golangci/golangci-lint/cmd/golangci-lint@latest
go: downloading github.com/golangci/golangci-lint v1.62.2
go: downloading github.com/stretchr/testify v1.10.0
go: downloading github.com/polyfloyd/go-errorlint v1.7.0
go: downloading github.com/Crocmagnon/fatcontext v0.5.3
go: downloading github.com/nunnatsa/ginkgolinter v0.18.3
go: downloading github.com/uudashr/iface v1.2.1
go: downloading github.com/mgechev/revive v1.5.1
go: downloading github.com/Antonboom/testifylint v1.5.2
go: downloading golang.org/x/exp/typeparams v0.0.0-20241108190413-2d47ceb2692f
PS D:\Go\lab9\query> .\build.ps1
Running golangci-lint...
golangci-lint completed successfully.
Compiling the project...
Project successfully compiled to myapp.exe.
Script completed successfully.
```

Рисунок 12 - результат проверки через PowerShell

Результаты проверки удовлетворительные.

Задание 3. Counter

1. Описание реализуемого функционала
    1) Get – выводит текущее состояние счетчика
    2) Post – прибавление к счетчику значения, передаваемого через json
    3) Put – изменяет значение последнего отправленного значения и как следствие итоговую сумму

Ниже приведен листинг программы

```go
package main
```

```go
import (
    "database/sql"
    "flag"
    "fmt"
    "log"
    "net/http"

    "github.com/labstack/echo/v4"
    _ "github.com/lib/pq"
)

const (
    host     = "localhost"
    port     = 5432
    user     = "postgres"
    password = "catjkm8800"
    dbname   = "count"
)

type Handlers struct {
    dbProvider DatabaseProvider
}

type DatabaseProvider struct {
    db *sql.DB
}

// Структура для валидации входящих данных
type CountInput struct {
    Val float32 `json:"val"` // Используем float32 для автоматической проверки числового значения
}

// Обработчик GET запроса
func (h *Handlers) GetCount(c echo.Context) error {
    msg, err := h.dbProvider.SelectCount()
    if err != nil {
        return c.String(http.StatusInternalServerError, err.Error())
    }
    return c.String(http.StatusOK, msg)
}

// Обработчик POST запроса
func (h *Handlers) PostCount(c echo.Context) error {
    input := CountInput{}

    // Привязка входных данных и проверка на ошибки
    if err := c.Bind(&input); err != nil {
        return c.String(http.StatusBadRequest, "Неправильный формат JSON")
    }

    // Проверка на нечисловое значение

    if err := h.dbProvider.InsertCount(input.Val); err != nil {
        return c.String(http.StatusInternalServerError, err.Error())
    }

    return c.String(http.StatusCreated, "Значение успешно вставлено")
}

// Обработчик PUT запроса
func (h *Handlers) PutCount(c echo.Context) error {
    input := CountInput{}
```

```
    if err := c.Bind(&input); err != nil {
        return c.String(http.StatusBadRequest, "Неправильный формат JSON")
    }


    if err := h.dbProvider.UpdateCount(input.Val); err != nil {
        return c.String(http.StatusInternalServerError, err.Error())
    }

    return c.String(http.StatusOK, "Значение успешно обновлено")
}

// Методы для работы с базой данных
func (dbp *DatabaseProvider) SelectCount() (string, error) {
    var msg string
    row := dbp.db.QueryRow("SELECT summa FROM count ORDER BY id DESC LIMIT 1")

    err := row.Scan(&msg)
    if err != nil {
        return "", err
    }
    return msg, nil
}

func (dbp *DatabaseProvider) InsertCount(v float32) error {
    _, err := dbp.db.Exec("INSERT INTO count (val, summa) VALUES ($1, $1 + (SELECT COALESCE(summa,
0) FROM count ORDER BY id DESC LIMIT 1))", v)


    return err
}

func (dbp *DatabaseProvider) UpdateCount(v float32) error {
    _, err := dbp.db.Exec("UPDATE count SET val = $1, summa = (val + (SELECT summa FROM count WHERE
id = ((SELECT MAX(id) FROM count) - 1))) WHERE id = (SELECT MAX(id) FROM count)", v)
    return err
}

func main() {
    address := flag.String("address", "127.0.0.1:8081", "адрес для запуска сервера")
    flag.Parse()

    psqlInfo := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s sslmode=disable", host,
port, user, password, dbname)

    db, err := sql.Open("postgres", psqlInfo)
    if err != nil {
        log.Fatal(err)
    }
    defer db.Close()

    if err = db.Ping(); err != nil {
        log.Fatal("Не удалось подключиться к базе данных:", err)
    }

    fmt.Println("Подключено к базе данных!")

    dp := DatabaseProvider{db: db}
    h := Handlers{dbProvider: dp}

    e := echo.New()

    e.GET("/get", h.GetCount)
```

```
    e.POST("/post", h.PostCount)
    e.PUT("/put", h.PutCount)

    if err = e.Start(*address); err != nil {
        log.Fatal(err)
    }
}
```

Приведем результаты работы программы

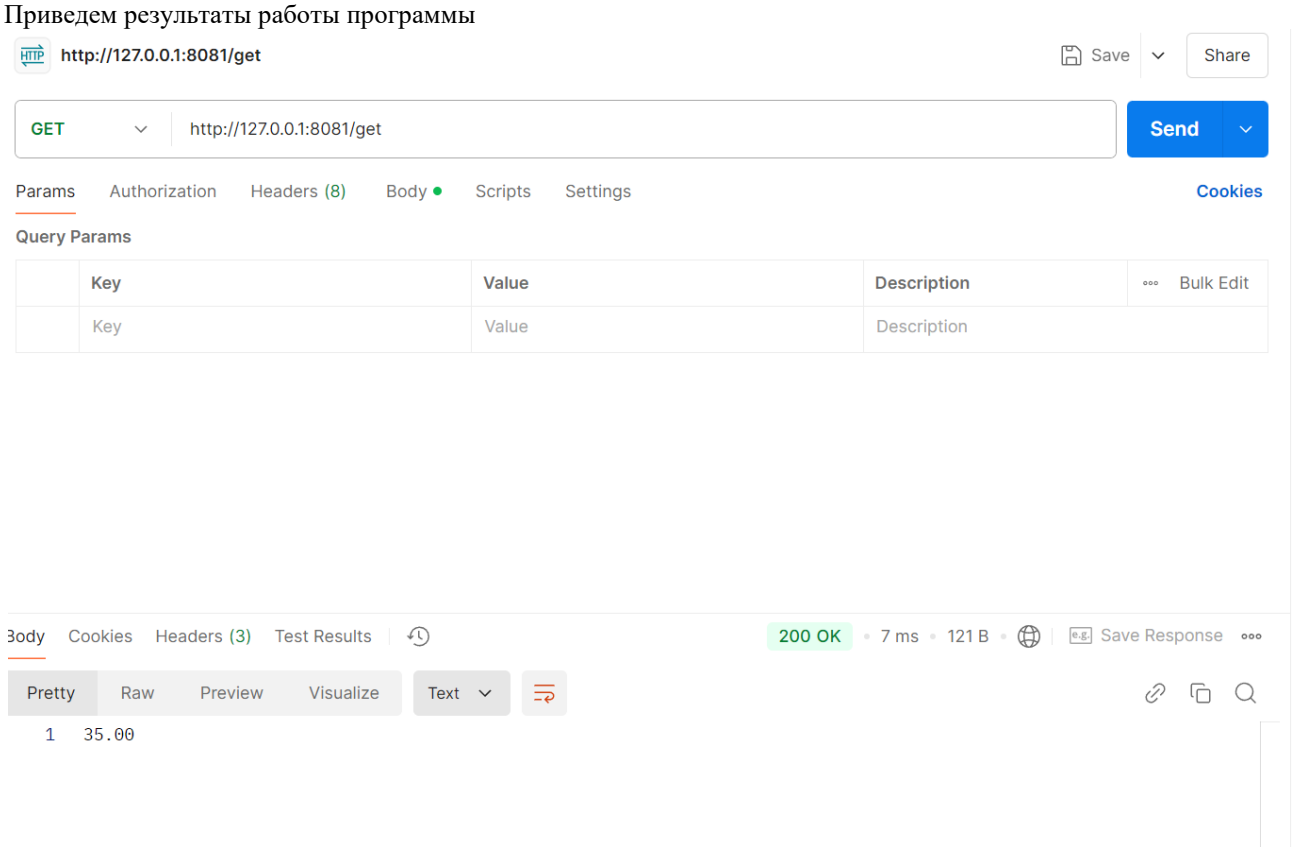

Рисунок 8 - тест 1 (GET)

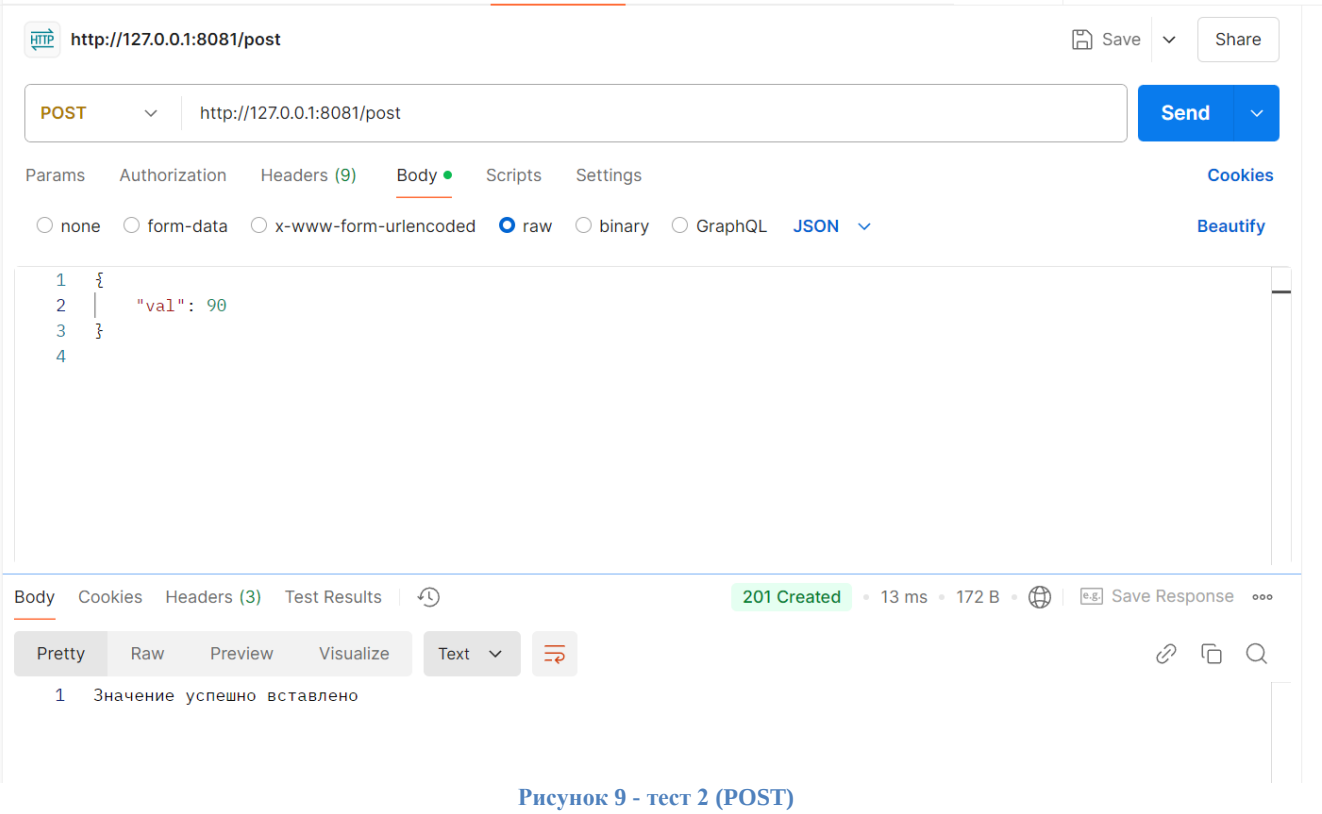

Рисунок 9 - тест 2 (POST)

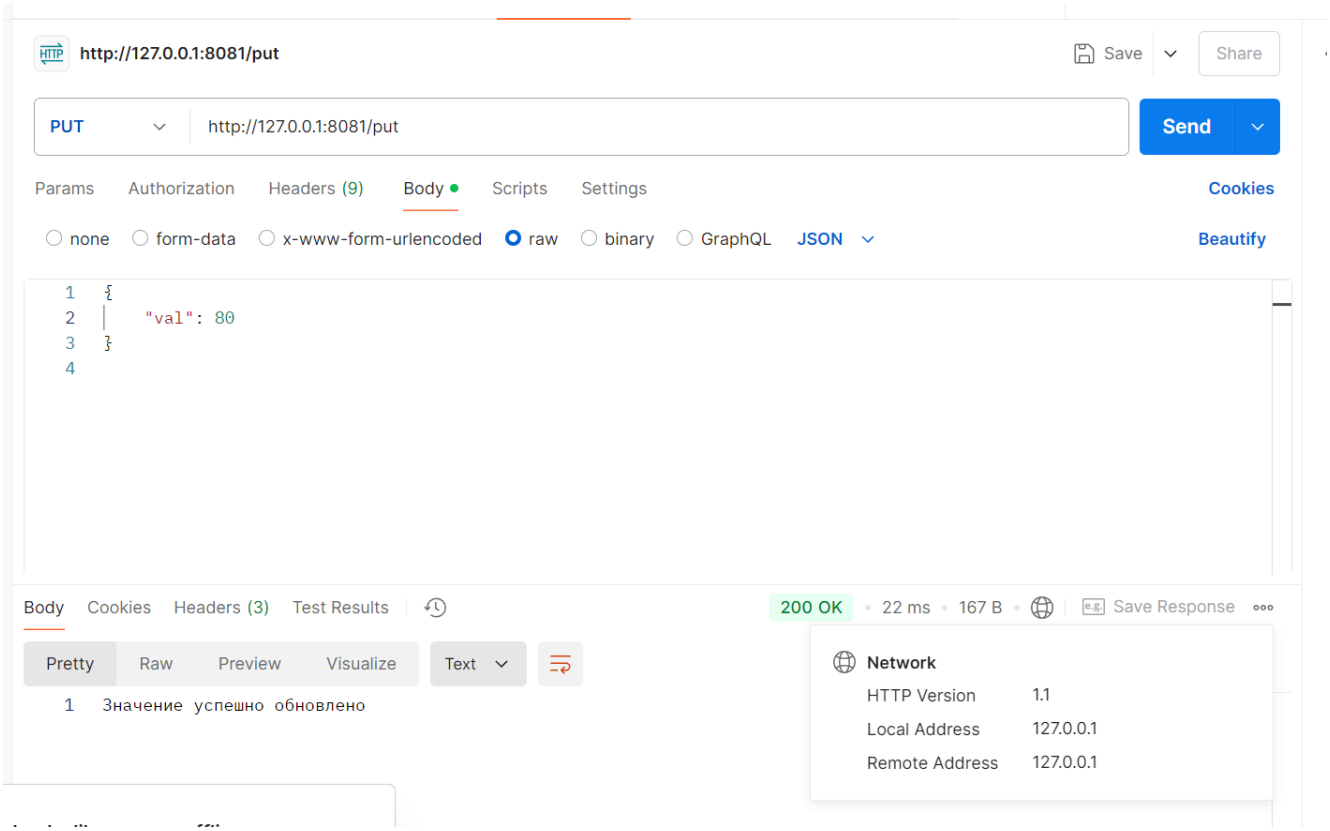

Рисунок 10 - тест 3 (PUT)

	id [PK] integer	val numeric (10,2)	summa numeric (10,2)
1	1	0.00	0.00
2	2	1.00	1.00
3	3	3.00	4.00
4	4	5.00	9.00
5	5	24.00	33.00
6	6	2.00	35.00
7	7	10.00	45.00
8	8	90.00	135.00

	id [PK] integer	val numeric (10,2)	summa numeric (10,2)
1	1	0.00	0.00
2	2	1.00	1.00
3	3	3.00	4.00
4	4	5.00	9.00
5	5	24.00	33.00
6	6	2.00	35.00
7	7	10.00	45.00
8	8	80.00	125.00

Рисунок 11 - БД

Для демонстрации работы обработчиков PUT и POST приведены скриншоты с состоянием БД. Из них виден пересчет значений при изменении последнего отправленного числа.

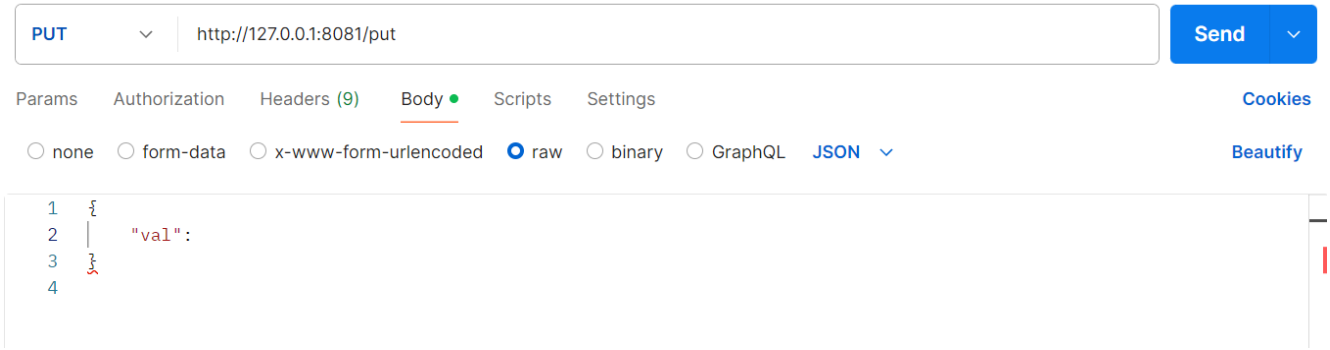

Рисунок 12 - тест 4 - пропущенное значение

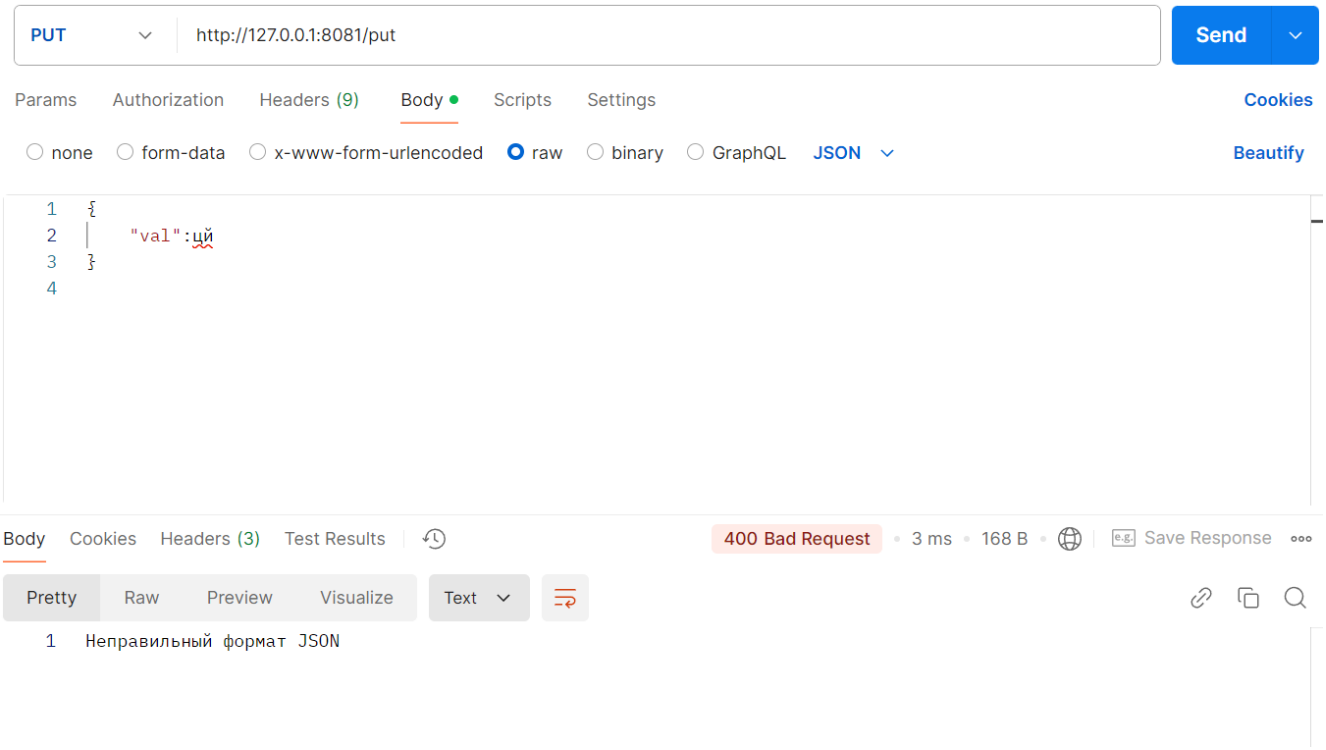

Рисунок 13 - тест 5 - некорректные данные

Проверка линтерами аналогична задаче 2.

Выводы: В ходе выполнения лабораторной работы были изучены базовые инструменты фреймворка Echo для организации клиент-серверного взаимодействия между Golang и Postgresql, где в роли клиента выступает сервис Golang, а в роли сервера СУБД Postgresql.

Список использованных источников:

1) https://tproger.ru/articles/osnovy-postgresql-dlya-nachinayushhih--ot-ustanovki-do-pervyh-zaprosov-250851
2) https://golangdocs.com/golang-postgresql-example
3) https://stepik.org/course/63054/syllabus
4) https://echo.labstack.com/docs/quick-start