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 ИУ6-31Б _____ Д.Ю. Воронин
(Группа) (Подпись, дата) (И.О. Фамилия)

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

Москва, 2024

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

Задание:

1. Ознакомиться с теоретическими сведениями
2. Сделать форк данного репозитория в GitHub, склонировать получившуюся копию локально, создать от мастера ветку dev и переключиться на неё
3. Перекопировать код сервисов, полученный в ходе выполнения 8-й лабораторной работы, в соответствующие поддиректории в директории cmd
4. Доработать сервисы таким образом, чтобы роутинг, обработка запросов, парсинг json, обработка ошибок и логирование осуществлялись на базе фреймворка Echo
5. Проверить свой код линтерами с помощью команды make lint
6. Сделать отчёт и поместить его в директорию docs
7. Зафиксировать изменения, сделать коммит и отправить получившееся состояние ветки дев в личный форк данного репозитория в GitHub
8. Через интерфейс GitHub создать Pull Request dev --> master
9. На защите лабораторной продемонстрировать корректность работы обновленных сервисов на Golang

Ход работы

1. **Задача Hello**

Доработаем код согласно условию:

```
package main

import (
	"database/sql"
	"fmt"
	"net/http"

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

const (
	host     = "localhost"
	port     = 5432
	user     = "postgres"
	password = "postgre"
	dbname   = "sandbox"
)

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)
}
func (h *Handlers) PostHello(c echo.Context) error {
	input := struct {
		Msg *string `json:"msg"`
	}{}

	err := c.Bind(&input)
	if input.Msg == nil {
		return c.String(http.StatusBadRequest, "Отсутствует поле 'msg'!")
	} else if *input.Msg == "" {
		return c.String(http.StatusBadRequest, "Строка пустая!")
	} else if err != nil {
		return c.String(http.StatusBadRequest, err.Error())
	}

	err = h.dbProvider.InsertHello(*input.Msg)
	if err != nil {
		return c.String(http.StatusInternalServerError, err.Error())
	}

	return c.String(http.StatusCreated, "")
}

// Методы для работы с базой данных
func (dp *DatabaseProvider) SelectHello() (string, error) {
```

```
	var msg string

	// Получаем одно сообщение из таблицы hello, отсортированной в случайном порядке
	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)
	if err != nil {
		return err
	}

	return nil
}

func main() {
	e := echo.New()

	// Формирование строки подключения для postgres
	psqlInfo := fmt.Sprintf("host=%s port=%d user=%s "+
		"password=%s dbname=%s sslmode=disable",
		host, port, user, password, dbname)

	// Создание соединения с сервером postgres
```

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

        // Создаем провайдер для БД с набором методов
        dp := DatabaseProvider{db: db}

        // Создаем экземпляр структуры с набором обработчиков
        h := Handlers{dbProvider: dp}

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

        fmt.Println("Сервер запущен")
        // Запускаем веб-сервер на указанном адресе
        e.Logger.Fatal(e.Start(":8081"))
}
```

Проверим работоспособность программы – рисунки 1-4.

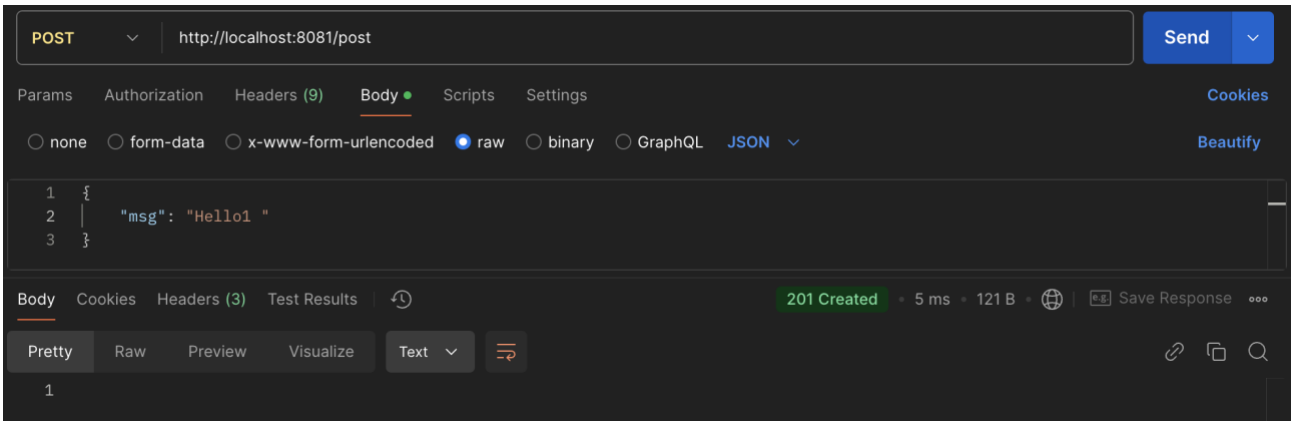

Рисунок 1 – Успешный POST запрос

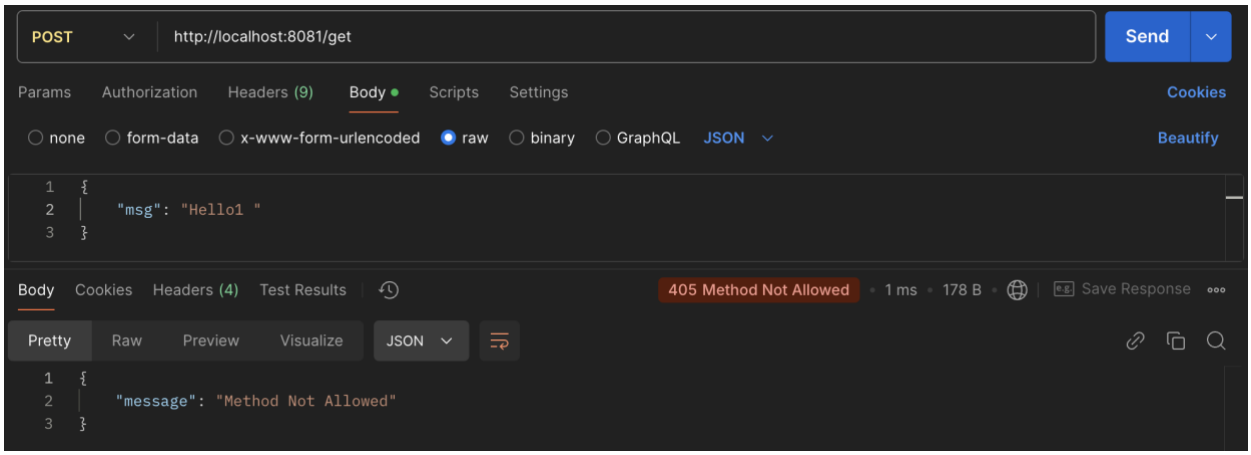

Рисунок 2 – Неудачный POST запрос

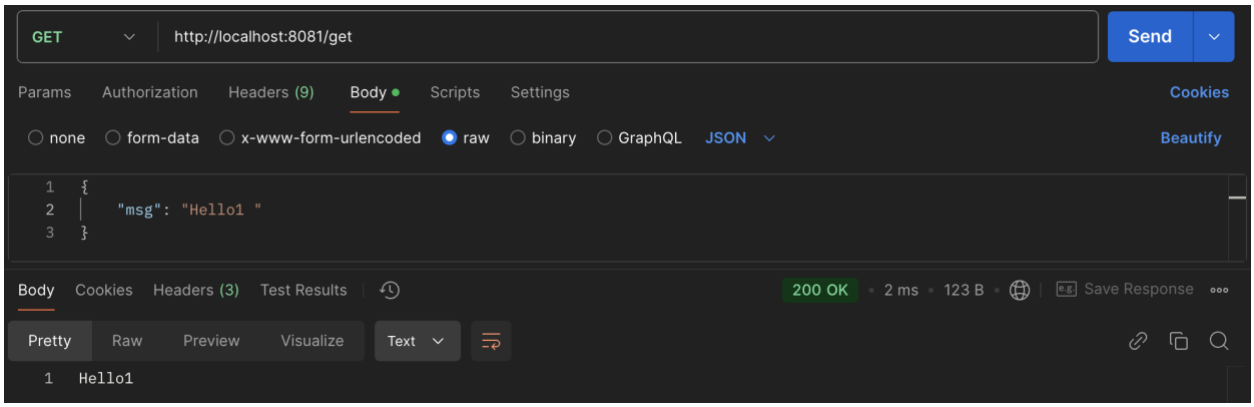

Рисунок 3 – Успешный GET запрос

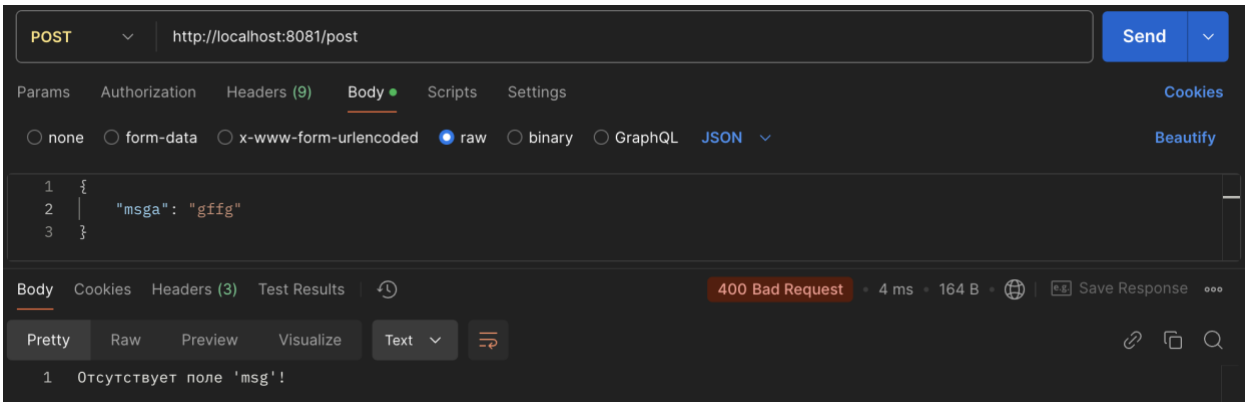

Рисунок 4 – Неудачный POST запрос

2. **Задача query**

Доработаем код согласно условию:

```
package main


import (
	"database/sql"
	"fmt"
	"net/http"
```

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

const (
	host     = "localhost"
	port     = 5432
	user     = "postgres"
	password = "postgre"
	dbname   = "sandbox"
)

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())
	} else if msg == "" {
		return c.String(http.StatusBadRequest, "Неинициализированное
значение в базе данных!")
	} else {
```

```
            return c.String(http.StatusOK, "Hello, "+msg+"!")
        }
}
func (h *Handlers) PostQuery(c echo.Context) error {

        input := struct {
            Msg *string `json:"name"`
        }{}

        err := c.Bind(&input)
        if err != nil {
            return c.String(http.StatusBadRequest, err.Error())
        } else if input.Msg == nil {
            return c.String(http.StatusBadRequest, "Отсутствует поле 'name'!")
        } else if *input.Msg == "" {
            return c.String(http.StatusBadRequest, "Пустая строка!")
        } else {
            err = h.dbProvider.UpdateQuery(*input.Msg)
            if err != nil {
                return c.String(http.StatusInternalServerError, err.Error())
            } else {
                return c.String(http.StatusAccepted, "")
            }
        }
}

// Методы для работы с базой данных
func (dp *DatabaseProvider) SelectQuery() (string, error) {
        var msg string

        row := dp.db.QueryRow("SELECT name FROM query")
```

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

	return msg, nil
}
func (dp *DatabaseProvider) UpdateQuery(msg string) error {
	_, err := dp.db.Exec("UPDATE query SET name = $1", msg)
	if err != nil {
		return err
	}

	return nil
}

func main() {
	e := echo.New()

	// Формирование строки подключения для postgres
	psqlInfo := fmt.Sprintf("host=%s port=%d user=%s "+
		"password=%s dbname=%s sslmode=disable",
		host, port, user, password, dbname)

	// Создание соединения с сервером postgres
	db, err := sql.Open("postgres", psqlInfo)
	if err != nil {
		e.Logger.Fatal(err)
	}
	defer db.Close()
```

```
    // Создаем провайдер для БД с набором методов
    dp := DatabaseProvider{db: db}
    // Создаем экземпляр структуры с набором обработчиков
    h := Handlers{dbProvider: dp}

    // Регистрируем обработчики
    e.GET("/get", h.GetQuery)
    e.POST("/post", h.PostQuery)

    fmt.Println("Сервер запущен")
    // Запускаем веб-сервер на указанном адресе
    e.Logger.Fatal(e.Start(":8082"))
}
```

Проверим работоспособность программы – рисунки 5-9.

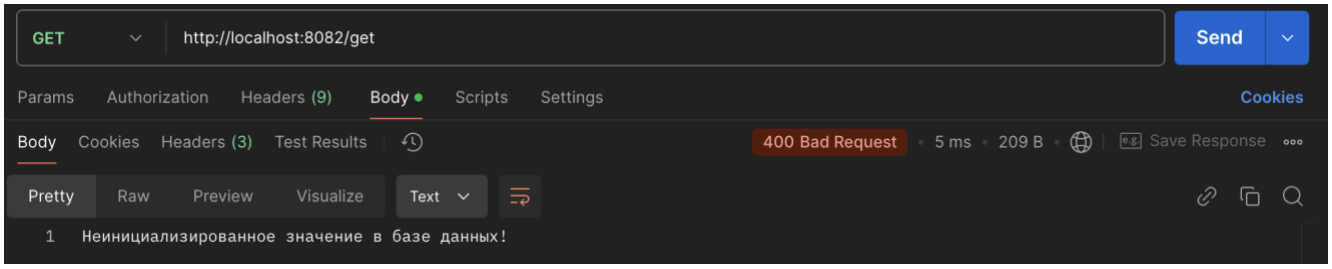

Рисунок 5 – Неудачный GET запрос

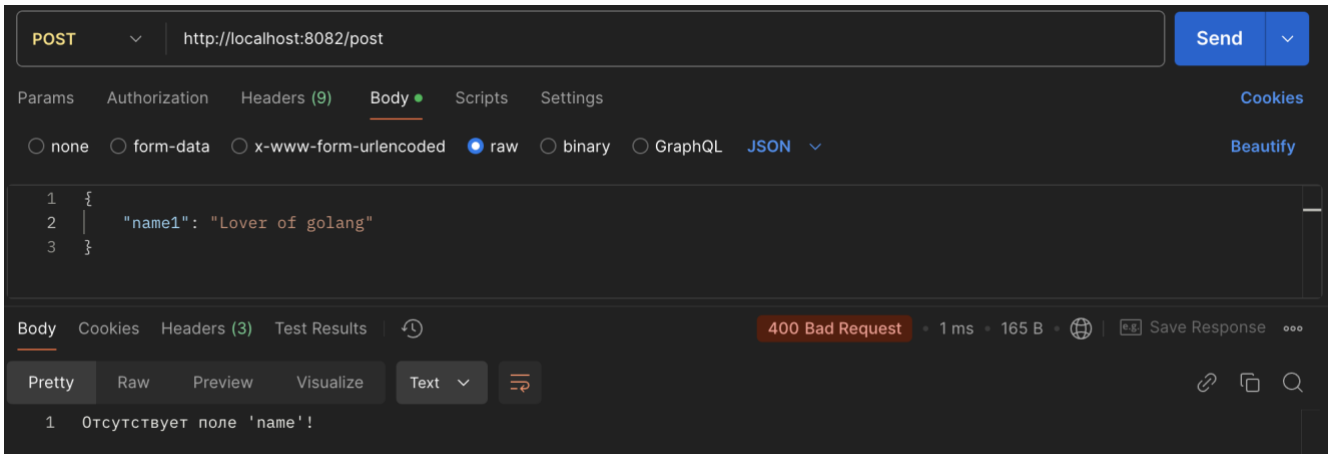

Рисунок 6 – Неудачный POST запрос

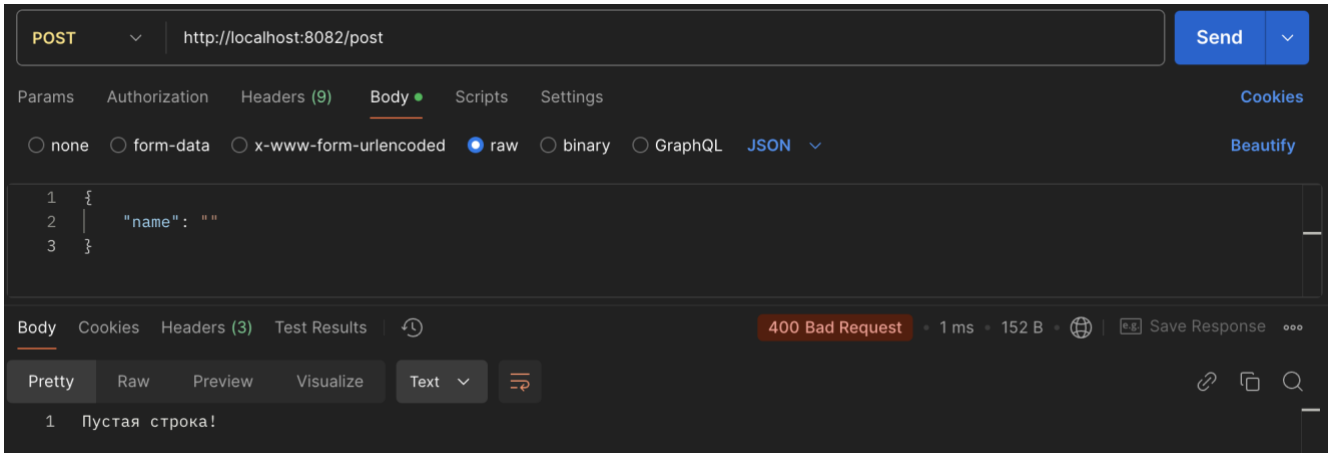

Рисунок 7 – Неудачный POST запрос

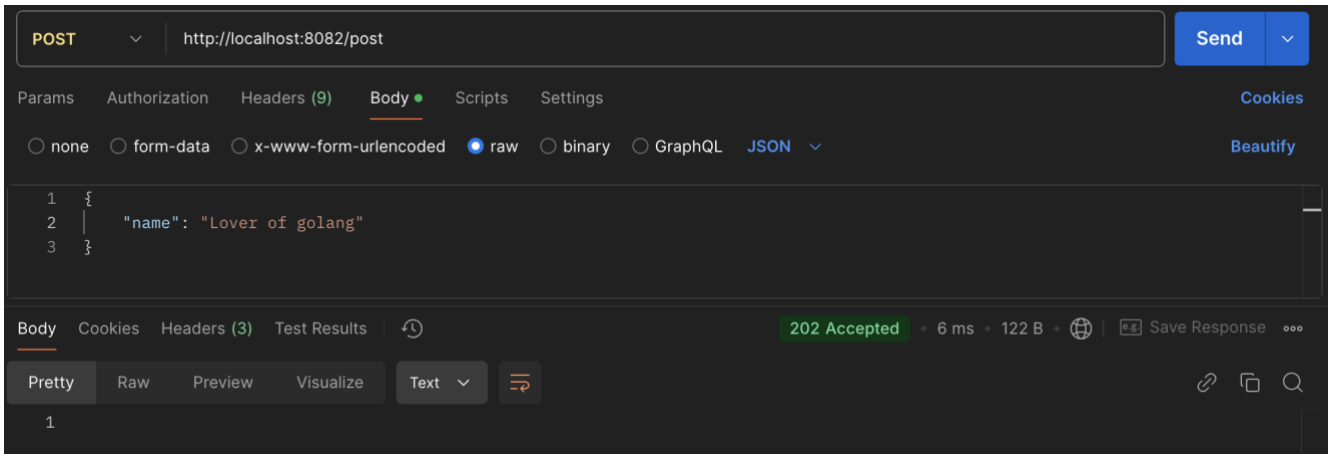

Рисунок 8 – Успешный POST запрос

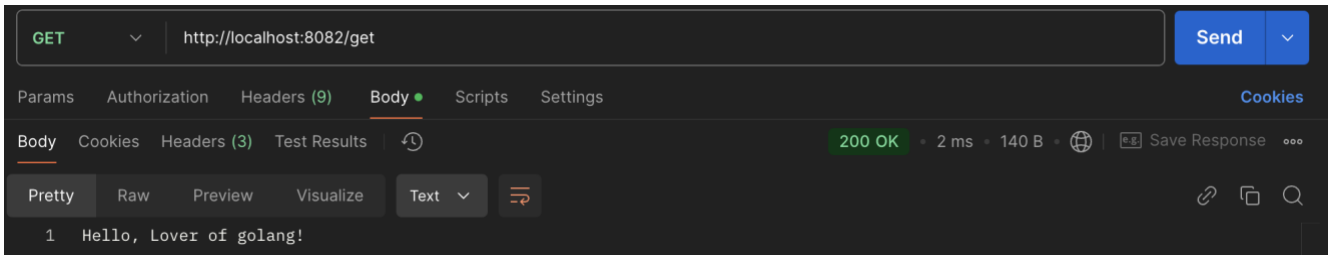

Рисунок 9 – Успешный GET запрос

3. Задача count

Доработаем код согласно условию:

```
package main

import (
	"database/sql"
	"fmt"
	"net/http"
```

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

const (
	host     = "localhost"
	port     = 5432
	user     = "postgres"
	password = "postgre"
	dbname   = "sandbox"
)

type Handlers struct {
	dbProvider DatabaseProvider
}

type DatabaseProvider struct {
	db *sql.DB
}

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

```
	input := struct {
		Msg *int `json:"count"`
	}{}
	err := c.Bind(&input)
	if input.Msg == nil {
		return c.String(http.StatusBadRequest, "Отсутствует поле 'count'!")
	} else if err != nil {
		return c.String(http.StatusBadRequest, err.Error())
	} else {
		err = h.dbProvider.UpdateCount(*input.Msg)
		if err != nil {
			return c.String(http.StatusInternalServerError, err.Error())
		} else {
			return c.String(http.StatusAccepted, "")
		}
	}
}

// Методы для работы с базой данных
func (dp *DatabaseProvider) SelectCount() (string, error) {
	var msg string

	row := dp.db.QueryRow("SELECT num FROM counter")
	err := row.Scan(&msg)
	if err != nil {
		return "", err
	}

	return msg, nil
}
func (dp *DatabaseProvider) UpdateCount(msg int) error {
```

```
	_, err := dp.db.Exec("UPDATE counter SET num = num + $1", msg)
	if err != nil {
		return err
	}

	return nil
}

func main() {
	e := echo.New()

	// Формирование строки подключения для postgres
	psqlInfo := fmt.Sprintf("host=%s port=%d user=%s "+
		"password=%s dbname=%s sslmode=disable",
		host, port, user, password, dbname)

	// Создание соединения с сервером postgres
	db, err := sql.Open("postgres", psqlInfo)
	if err != nil {
		e.Logger.Fatal(err)
	}
	defer db.Close()

	// Создаем провайдер для БД с набором методов
	dp := DatabaseProvider{db: db}
	// Создаем экземпляр структуры с набором обработчиков
	h := Handlers{dbProvider: dp}

	// Регистрируем обработчики
	e.GET("/get", h.GetCount)
	e.POST("/post", h.PostCount)
```

```
    fmt.Println("Сервер запущен")
    // Запускаем веб-сервер на указанном адресе
    e.Logger.Fatal(e.Start(":8083"))
}
```

Проверим работоспособность программы – рисунки 10-17.

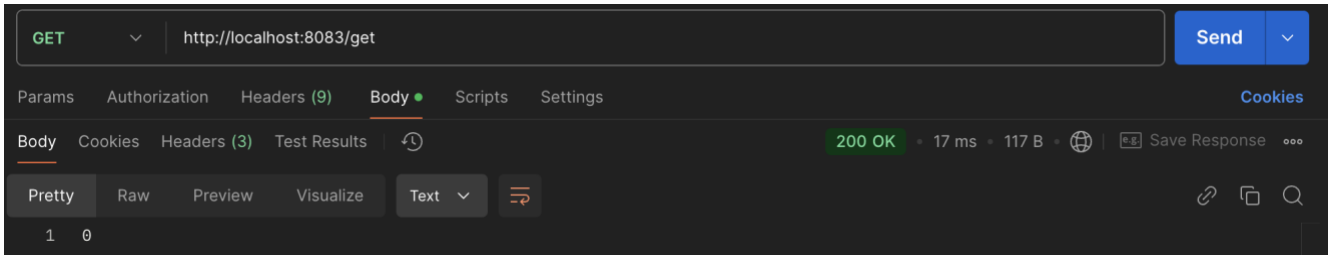

Рисунок 10 – Успешный GET запрос

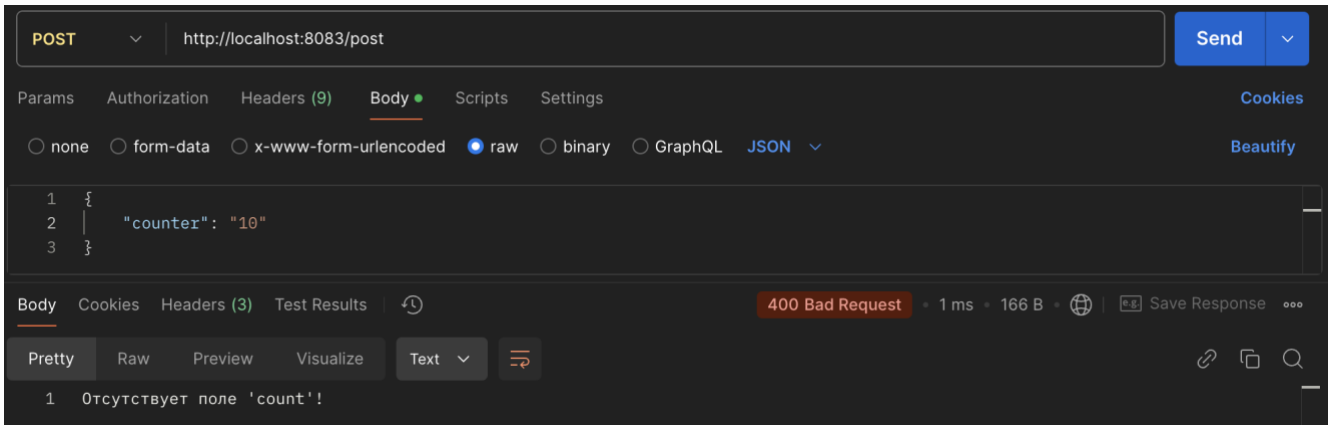

Рисунок 11 – Неудачный POST запрос

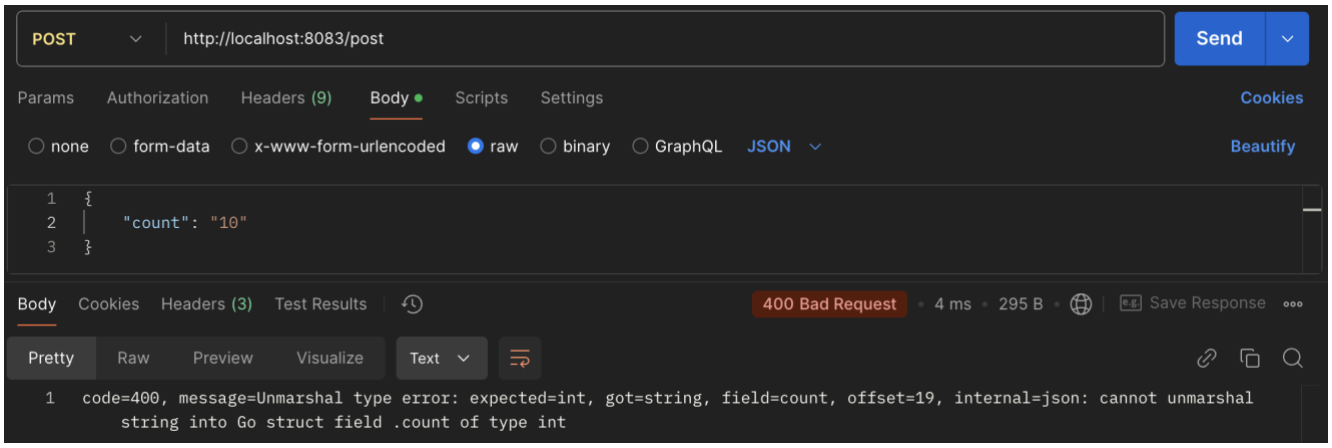

Рисунок 12 – Неудачный POST запрос из-за неверного типа данных

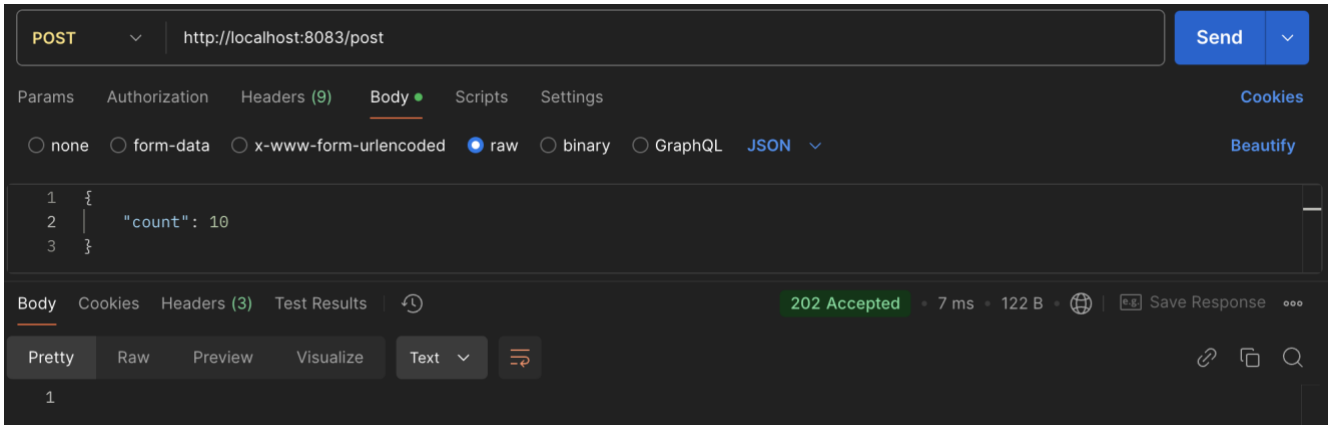

Рисунок 13 – Успешный POST запрос с положительным числом

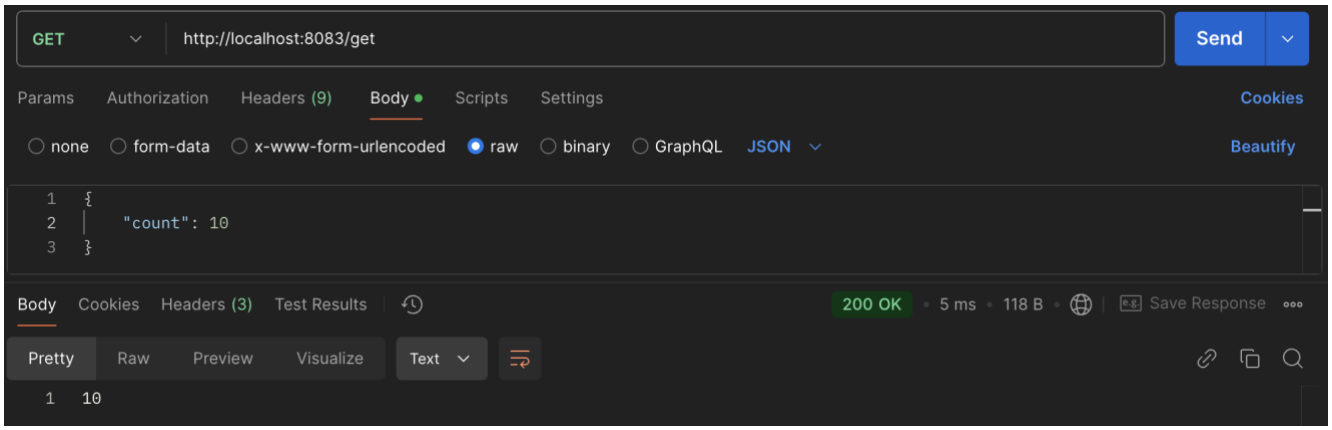

Рисунок 14 – Успешный GET запрос с результатом, внесённым предыдущим POST запросом

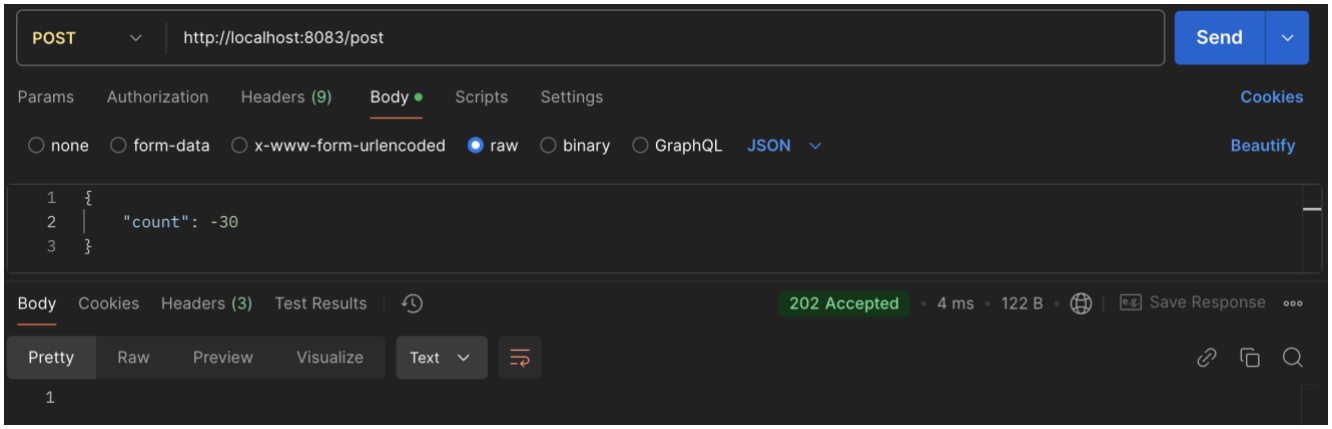

Рисунок 15 – Успешный POST запрос с отрицательным числом

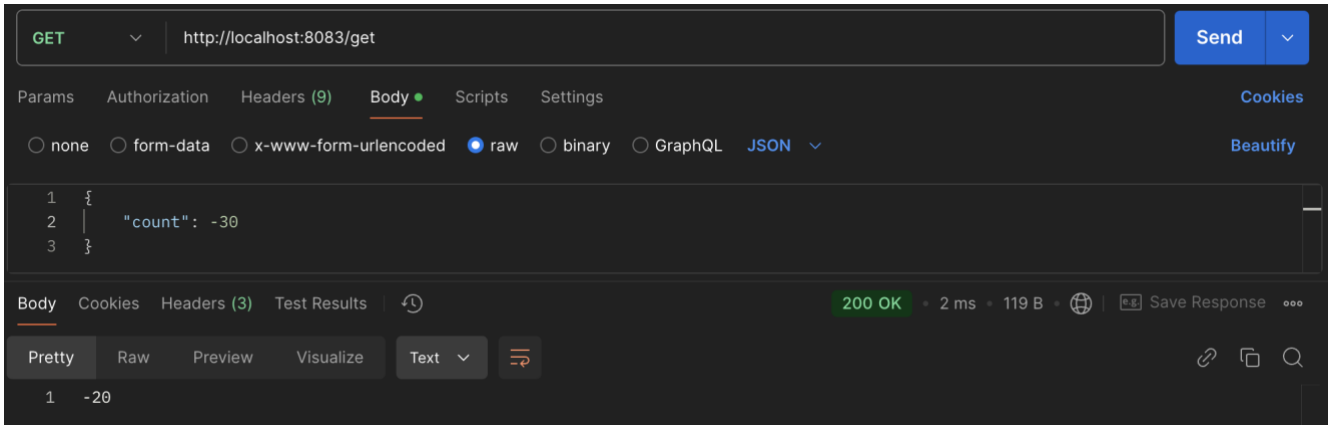

Рисунок 16 – Успешный GET запрос для проверки корректности данных после второго POST запроса

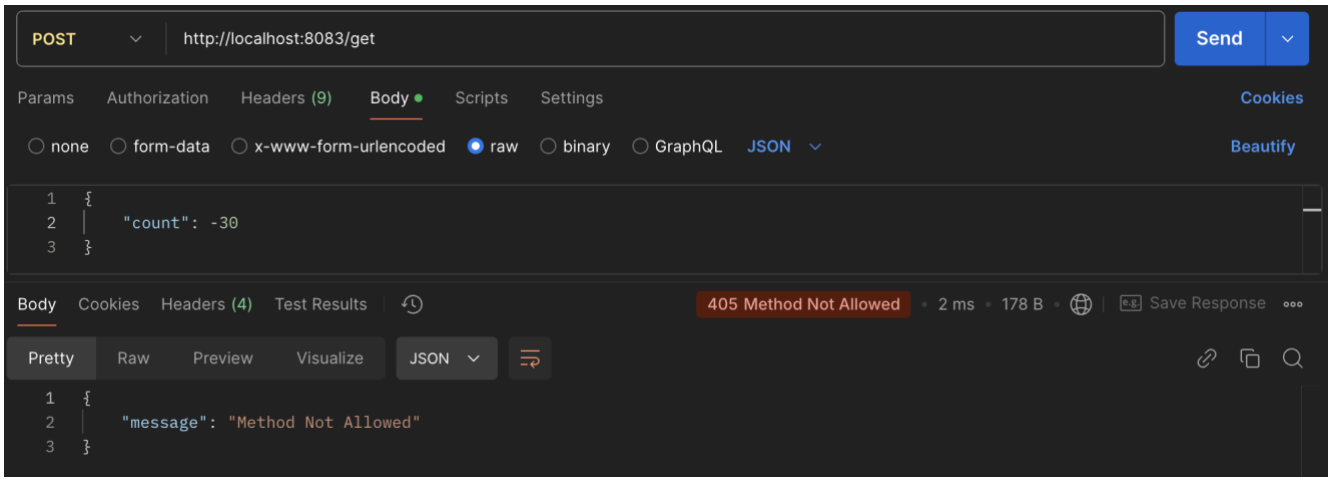

Рисунок 17 – Неудачный POST запрос, так как данный путь не поддерживает POST запрос

Заключение:

В ходе работы были получены базовые навыки разработки серверной части веб-приложений на языке Go с использованием веб-фреймворков. В частности, был использован веб-фреймворк Echo.

Список использованных источников:

https://github.com/ValeryBMSTU/web-9