Московский государственный технический университет им. Н.Э. Баумана

Факультет информатика и системы управления
Кафедра системы обработки информации и управления

Курс «Парадигмы и конструкции языков программирования»
Отчет по домашнему заданию

Выполнил:
студент группы ИУ5-32Б:
Шпакова Д. Л.
Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю. Е.
Подпись и дата:

Описание задания

Цель работы: Изучение языка программирования Go (Golang) и взаимодействия с внешними API путем реализации чат-бота для управления задачами.

Выбранный язык: Go (Golang).

Инструментарий: Visual Studio Code (с использованием WSL), Git, библиотека

Задача: Разработать Telegram-бота «TaskManager» для управления списком задач через текстовый интерфейс, поддерживающего создание, назначение исполнителей и завершение задач.

Описание проекта и изученных аспектов языка

Архитектура решения

Проект представляет собой серверное приложение, работающее по принципу Long Polling для получения обновлений от Telegram:

- **Bot Logic:** Основной модуль, обрабатывающий входящие текстовые команды (/tasks, /new, /assign и др.) и формирующий ответы пользователю.
- **Data Model:** Слой хранения данных, реализованный In-Memory (в оперативной памяти) без использования внешних СУБД, что упрощает развертывание и тестирование.

Основные возможности и изученные конструкции

В процессе разработки были изучены и применены следующие особенности Go:

- **Струтуры и Слайсы (Structs & Slices):** Использованы для моделирования сущности «Задача» (Task) и хранения списка задач в памяти вместо базы данных. Реализован поиск и фильтрация данных внутри слайса (например, для команд /my и /owner).
- **Работа со строками и Парсинг:** Реализована логика разбора входящих сообщений (парсинг аргументов команд), например, извлечение заголовка задачи из команды /new или ID задачи из /assign_$ID.
- **Взаимодействие с API (Telegram Bot API):** Освоена работа со сторонней библиотекой telegram-bot-api для отправки и получения сообщений, а также принципы работы с обновлениями (Updates) от сервера Telegram.

Текст программы

b

```
package main

import (
    "context"
    "fmt"
    "log"
    "net/http"
    "os"
    "sort"
    "strconv"
    "strings"
    "sync"

    "github.com/joho/godotenv"
    tgbotapi "github.com/skinass/telegram-bot-api/v5"
)

var (
    BotToken   string
    WebhookURL string
)

// Task представляет собой одну задачу
type Task struct {
    ID        int
    Title     string
    OwnerID   int64
    OwnerUser *tgbotapi.User
    Assignee  *tgbotapi.User // nil если не назначена
}

// Глобальные переменные для хранения состояния
var (
    bot        *tgbotapi.BotAPI
    tasks      map[int]*Task          // Хранилище задач
    nextID     int                    // Автоинкрементный ID для новых задач
    tasksMu    sync.RWMutex           // Мьютекс для защиты доступа к tasks и
nextID
    userStates = make(map[int64]bool) // true, если ждем название задачи от
пользователя
)

// Создаем клавиатуру для меню
var numericKeyboard = tgbotapi.NewReplyKeyboard(
    tgbotapi.NewKeyboardButtonRow(
        tgbotapi.NewKeyboardButton("➕ Новая задача"),
    ),
```

```
        tgbotapi.NewKeyboardButtonRow(
            tgbotapi.NewKeyboardButton("📋 Все задачи"),
            tgbotapi.NewKeyboardButton("👤 Мои задачи"),
        ),
        tgbotapi.NewKeyboardButtonRow(
            tgbotapi.NewKeyboardButton("👑 Я создал"),
            tgbotapi.NewKeyboardButton("❓ Помощь"),
        ),
    )

    // Инициализация конфигурации
    func init() {
        if err := godotenv.Load(); err != nil {
            log.Println("Не найден файл .env, используются переменные окружения")
        }

        BotToken = os.Getenv("BOT_TOKEN")
        WebhookURL = os.Getenv("WEBHOOK_URL")

        if BotToken == "" || WebhookURL == "" {
            log.Fatal("ERROR: BOT_TOKEN или WEBHOOK_URL не заданы")
        }

        tasks = make(map[int]*Task)
        nextID = 1

        userStates = make(map[int64]bool)
    }

    // startTaskBot запускает бота и HTTP-сервер для вебхуков
    func startTaskBot(ctx context.Context) error {
        var err error
        bot, err = tgbotapi.NewBotAPI(BotToken)
        if err != nil {
            return fmt.Errorf("NewBotAPI failed: %w", err)
        }

        wh, err := tgbotapi.NewWebhook(WebhookURL)
        if err != nil {
            return fmt.Errorf("NewWebhook failed: %w", err)
        }

        _, err = bot.Request(wh)
        if err != nil {
            return fmt.Errorf("SetWebhook failed: %w", err)
        }

        updates := bot.ListenForWebhook("/")

        // Настройка адреса сервера.
```

```
    // В локальной среде или при специфической конфигурации сети адрес может
отличаться.
    addr := "127.0.0.1:8081"

    // Если мы развернуты в облаке (например, Heroku/Render), порт часто
передается через env PORT.
    if port := os.Getenv("PORT"); port != "" {
        addr = ":" + port
    } else if !strings.Contains(WebhookURL, "127.0.0.1") {
        // Fallback для стандартного запуска, если не указан локальный IP
        addr = ":8080"
    }

    srv := &http.Server{Addr: addr}
    log.Printf("Starting server on %s", addr)

    // Запускаем сервер в отдельной горутине
    go func() {
        if err := srv.ListenAndServe(); err != nil && err != http.ErrServerClosed
{
            log.Fatalf("ListenAndServe failed: %v", err)
        }
    }()

    // Обработчик обновлений
    go func() {
        for update := range updates {
            handleUpdate(update)
        }
    }()

    // Ждем сигнала отмены (ctx.Done()) для graceful shutdown
    <-ctx.Done()
    log.Println("Shutting down server...")
    return srv.Shutdown(context.Background())
}

// handleUpdate - главный обработчик входящих сообщений
func handleUpdate(update tgbotapi.Update) {
    if update.Message == nil {
        return
    }

    msg := update.Message
    user := msg.From
    if user == nil {
        return
    }

    text := msg.Text
```

```
    // 1. Если нажать кнопку "Новая задача"
    if text == "➕ Новая задача" {
        tasksMu.Lock()
        userStates[user.ID] = true // Включаем режим ожидания
        tasksMu.Unlock()

        sendMessage(msg.Chat.ID, "✍️ Напишите задачу в чат (одним сообщением):")
        return
    }

    // 2. Если мы ждем название задачи от этого пользователя
    tasksMu.RLock()
    isWaiting := userStates[user.ID]
    tasksMu.RUnlock()

    if isWaiting {
        // Если пользователь передумал и нажал другую кнопку - отменяем ожидание
        if strings.HasPrefix(text, "/") || text == "📋 Все задачи" || text == "👤
Мои задачи" || text == "👑 Я создал" || text == "❓ Помощь" {
            tasksMu.Lock()
            delete(userStates, user.ID)
            tasksMu.Unlock()
            // И код пойдет дальше обрабатывать эту команду как обычно
        } else {
            handleNew(user, msg.Chat.ID, text)

            // Выключаем режим ожидания
            tasksMu.Lock()
            delete(userStates, user.ID)
            tasksMu.Unlock()
            return
        }
    }

    var cmd, args string

    switch text {
    case "📋 Все задачи":
        cmd = "/tasks"
    case "👤 Мои задачи":
        cmd = "/my"
    case "👑 Я создал":
        cmd = "/owner"
    case "❓ Помощь":
        cmd = "/help"
    default:
        cmd, args = parseCommand(text)
    }

    switch cmd {
    case "/start":
```

```
        resp := tgbotapi.NewMessage(msg.Chat.ID, "Привет! Я твой задачник.
Используй кнопки внизу 👇")
        resp.ReplyMarkup = numericKeyboard
        bot.Send(resp)

    case "/help":
        sendMessage(msg.Chat.ID, "Нажми '➕ Новая задача' и напиши её.\nИли
используй кнопки меню.")

    case "/new":
        handleNew(user, msg.Chat.ID, args)
    case "/tasks":
        handleTasks(user.ID, msg.Chat.ID)
    case "/assign":
        handleAssign(user, msg.Chat.ID, args)
    case "/unassign":
        handleUnassign(user, msg.Chat.ID, args)
    case "/resolve":
        handleResolve(user, msg.Chat.ID, args)
    case "/my":
        handleMy(user.ID, msg.Chat.ID)
    case "/owner":
        handleOwner(user.ID, msg.Chat.ID)

    default:
        resp := tgbotapi.NewMessage(msg.Chat.ID, "Я не понимаю 😕\nЧтобы создать
задачу, нажмите кнопку '➕ Новая задача'.")
        resp.ReplyMarkup = numericKeyboard
        bot.Send(resp)
    }
}

// parseCommand разбирает текст сообщения на команду и аргументы
func parseCommand(text string) (cmd, args string) {
    if !strings.HasPrefix(text, "/") {
        return "", ""
    }

    parts := strings.SplitN(text, " ", 2)
    cmd = parts[0]
    if len(parts) == 2 {
        args = strings.TrimSpace(parts[1])
    }

    // Обработка команд с ID ( /assign_1, /unassign_1, /resolve_1 )
    if strings.HasPrefix(cmd, "/assign_") {
        args = strings.TrimPrefix(cmd, "/assign_")
        cmd = "/assign"
    } else if strings.HasPrefix(cmd, "/unassign_") {
        args = strings.TrimPrefix(cmd, "/unassign_")
        cmd = "/unassign"
```

```
    } else if strings.HasPrefix(cmd, "/resolve_") {
        args = strings.TrimPrefix(cmd, "/resolve_")
        cmd = "/resolve"
    }
    return cmd, args
}

// sendMessage - утилита для отправки сообщения
func sendMessage(chatID int64, text string) {
    if bot == nil {
        log.Println("ERROR: Bot is not initialized")
        return
    }
    msg := tgbotapi.NewMessage(chatID, text)
    bot.Send(msg)
}

// --- Обработчики команд ---

// handleNew создает новую задачу
func handleNew(user *tgbotapi.User, chatID int64, title string) {
    if title == "" {
        sendMessage(chatID, "Задача не может быть пустой. Пример: /new Купить
продукты")
        return
    }

    tasksMu.Lock()
    defer tasksMu.Unlock()

    id := nextID
    nextID++

    task := &Task{
        ID:        id,
        Title:     title,
        OwnerID:   user.ID,
        OwnerUser: user,
        Assignee:  nil,
    }
    tasks[id] = task

    sendMessage(chatID, fmt.Sprintf(`Задача "%s" создана`, title))
}

// getSortedTasks возвращает отсортированный по ID срез задач (для RLock)
func getSortedTasks() []*Task {
    ids := make([]int, 0, len(tasks))
    for id := range tasks {
        ids = append(ids, id)
    }
```

```
    sort.Ints(ids)

    sorted := make([]*Task, 0, len(ids))
    for _, id := range ids {
        sorted = append(sorted, tasks[id])
    }
    return sorted
}

// handleTasks показывает все активные задачи
func handleTasks(viewerID int64, chatID int64) {
    tasksMu.RLock()
    defer tasksMu.RUnlock()

    if len(tasks) == 0 {
        sendMessage(chatID, "Нет задач")
        return
    }

    sortedTasks := getSortedTasks()
    msgs := make([]string, 0, len(sortedTasks))
    for _, task := range sortedTasks {
        msgs = append(msgs, formatTask(task, viewerID, "tasks"))
    }

    sendMessage(chatID, strings.Join(msgs, "\n\n"))
}

// handleMy показывает задачи, назначенные на пользователя
func handleMy(viewerID int64, chatID int64) {
    tasksMu.RLock()
    defer tasksMu.RUnlock()

    sortedTasks := getSortedTasks()
    msgs := make([]string, 0, len(sortedTasks))
    for _, task := range sortedTasks {
        if task.Assignee != nil && task.Assignee.ID == viewerID {
            msgs = append(msgs, formatTask(task, viewerID, "my"))
        }
    }

    if len(msgs) == 0 {
        sendMessage(chatID, "Нет задач")
        return
    }
    sendMessage(chatID, strings.Join(msgs, "\n\n"))
}

// handleOwner показывает задачи, созданные пользователем
func handleOwner(viewerID int64, chatID int64) {
    tasksMu.RLock()
```

```
    defer tasksMu.RUnlock()

    sortedTasks := getSortedTasks()
    msgs := make([]string, 0, len(sortedTasks))
    for _, task := range sortedTasks {
        if task.OwnerID == viewerID {
            msgs = append(msgs, formatTask(task, viewerID, "owner"))
        }
    }

    if len(msgs) == 0 {
        sendMessage(chatID, "Нет задач")
        return
    }
    sendMessage(chatID, strings.Join(msgs, "\n\n"))
}

// formatTask форматирует задачу в строку для вывода
// context: "tasks", "my", "owner" (влияет на отображение assignee)
func formatTask(task *Task, viewerID int64, context string) string {
    var b strings.Builder

    // 1. Шапка карточки (ID и Название)
    icon := "🆕"
    if task.Assignee != nil {
        icon = "👤"
    }

    b.WriteString(fmt.Sprintf("%s Задача №%d\n", icon, task.ID))
    b.WriteString(fmt.Sprintf("📝 %s\n", task.Title))
    b.WriteString(fmt.Sprintf("👤 Автор: @%s\n", task.OwnerUser.UserName))

    // 2. Статус и Кнопки действий
    b.WriteString("--------------------\n")

    showAssignee := (context == "tasks")

    if task.Assignee == nil {
        // --- ЗАДАЧА СВОБОДНА ---
        b.WriteString("👤 СВОБОДНА\n")
        b.WriteString(fmt.Sprintf("👉 Взять в работу: /assign_%d", task.ID))

    } else if task.Assignee.ID == viewerID {
        // --- ЗАДАЧА НА МНЕ ---
        b.WriteString("👤 В РАБОТЕ (у Вас)\n\n")
        b.WriteString("Что делаем?\n")
        b.WriteString(fmt.Sprintf("✅ Готово:      /resolve_%d\n", task.ID))
        b.WriteString(fmt.Sprintf("❌ Отказаться:  /unassign_%d", task.ID))

    } else {
        // --- ЗАДАЧА НА ДРУГОМ ---
```

```
        if showAssignee {
            b.WriteString(fmt.Sprintf("🔒 Исполнитель: @%s",
task.Assignee.UserName))
        }
    }

    b.WriteString("\n") // Пустая строка в конце для отступа между карточками
    return b.String()
}

// handleAssign назначает задачу на пользователя
func handleAssign(user *tgbotapi.User, chatID int64, args string) {
    taskID, err := strconv.Atoi(args)
    if err != nil {
        sendMessage(chatID, "Неверный ID задачи")
        return
    }

    tasksMu.Lock()
    defer tasksMu.Unlock()

    task, ok := tasks[taskID]
    if !ok {
        sendMessage(chatID, "Задача не найдена")
        return
    }

    oldAssignee := task.Assignee
    task.Assignee = user

    // 1. Отвечаем тому, кто назначил
    sendMessage(chatID, fmt.Sprintf(`Задача "%s" назначена на вас`, task.Title))

    // 2. Логика уведомлений
    notificationSent := false

    // 2a. Уведомляем старого исполнителя
    if oldAssignee != nil && oldAssignee.ID != user.ID {
        sendMessage(oldAssignee.ID, fmt.Sprintf(`Задача "%s" назначена на @%s`,
task.Title, user.UserName))
        notificationSent = true
    }

    // 2b. Если старого исполнителя не было, уведомляем владельца
    if !notificationSent && task.OwnerID != user.ID {
        sendMessage(task.OwnerID, fmt.Sprintf(`Задача "%s" назначена на @%s`,
task.Title, user.UserName))
    }
}

// handleUnassign снимает задачу с исполнителя
```

```
func handleUnassign(user *tgbotapi.User, chatID int64, args string) {
    taskID, err := strconv.Atoi(args)
    if err != nil {
        sendMessage(chatID, "Неверный ID задачи")
        return
    }

    tasksMu.Lock()
    defer tasksMu.Unlock()

    task, ok := tasks[taskID]
    if !ok {
        sendMessage(chatID, "Задача не найдена")
        return
    }

    // Проверяем, что задачу снимает текущий исполнитель
    if task.Assignee == nil || task.Assignee.ID != user.ID {
        sendMessage(chatID, "Задача не на вас")
        return
    }

    task.Assignee = nil
    sendMessage(chatID, "Принято")

    // Уведомляем владельца
    if task.OwnerID != user.ID {
        sendMessage(task.OwnerID, fmt.Sprintf(`Задача "%s" осталась без
исполнителя`, task.Title))
    }
}

// handleResolve выполняет (удаляет) задачу
func handleResolve(user *tgbotapi.User, chatID int64, args string) {
    taskID, err := strconv.Atoi(args)
    if err != nil {
        sendMessage(chatID, "Неверный ID задачи")
        return
    }

    tasksMu.Lock()
    defer tasksMu.Unlock()

    task, ok := tasks[taskID]
    if !ok {
        return
    }

    if task.Assignee == nil || task.Assignee.ID != user.ID {
        sendMessage(chatID, "Задача не на вас")
        return
```

```
    }

    delete(tasks, taskID)

    sendMessage(chatID, fmt.Sprintf(`Задача "%s" выполнена`, task.Title))

    if task.OwnerID != user.ID {
        sendMessage(task.OwnerID, fmt.Sprintf(`Задача "%s" выполнена @%s`,
task.Title, user.UserName))
    }
}

func main() {
    if err := startTaskBot(context.Background()); err != nil {
        log.Fatal(err)
    }
}
```

Скриншот работы бота

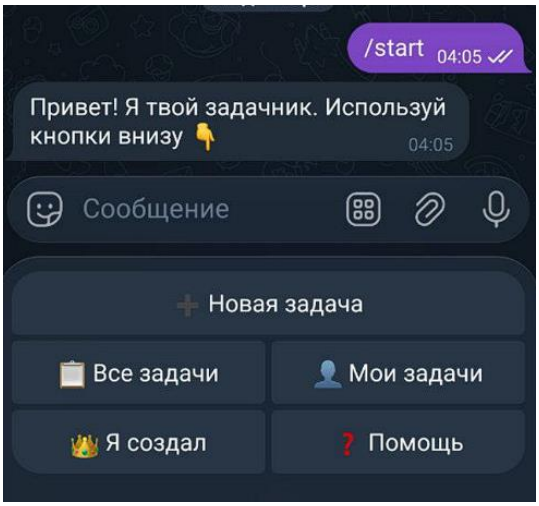

Рис. 1 Ответ на /start

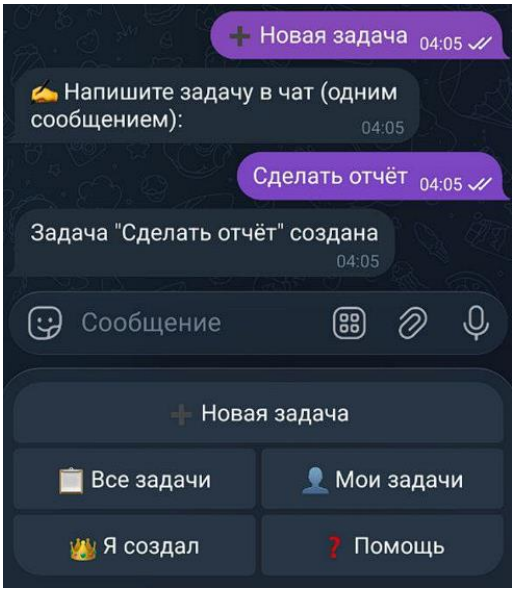

Рис. 2 Новая задача

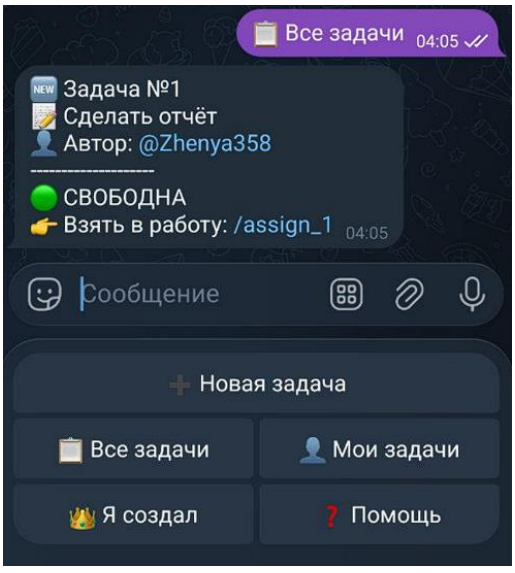

Рис. 3 Просмотр всех задач

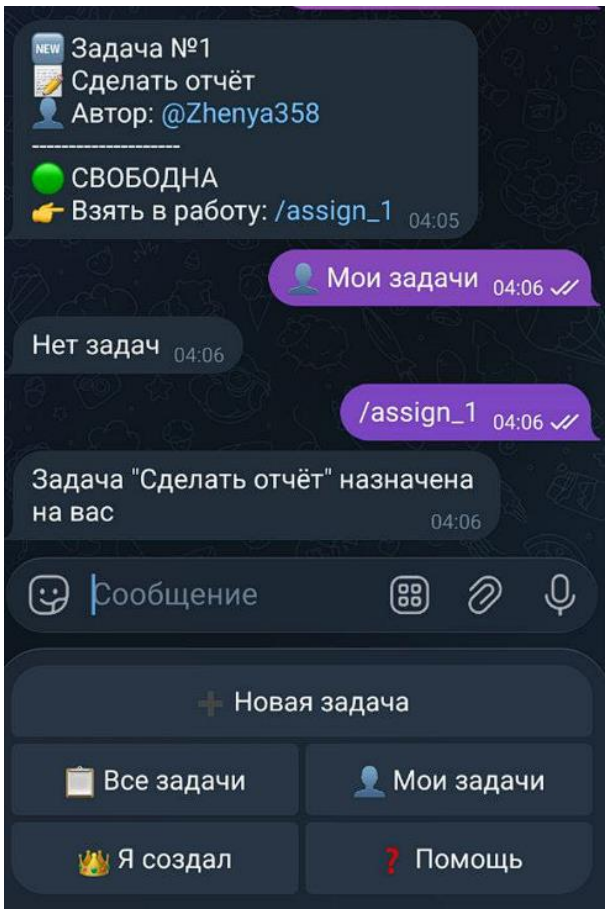

Рис. 4 Назначение задачи на себя

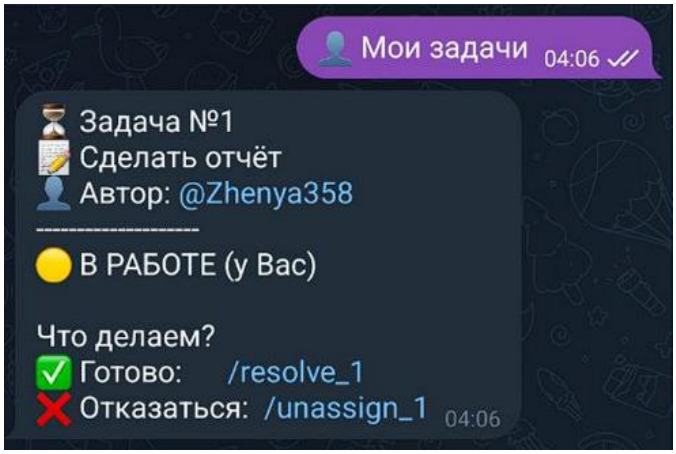

Рис. 5 Моя задача в работе

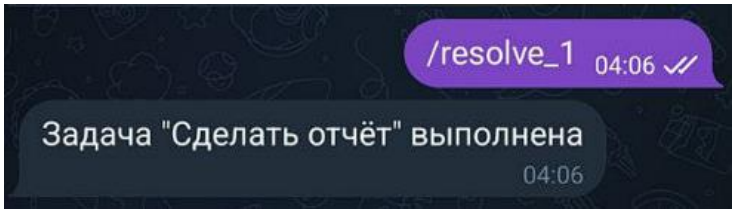

Рис. 6 Отметка о выполнении задачи

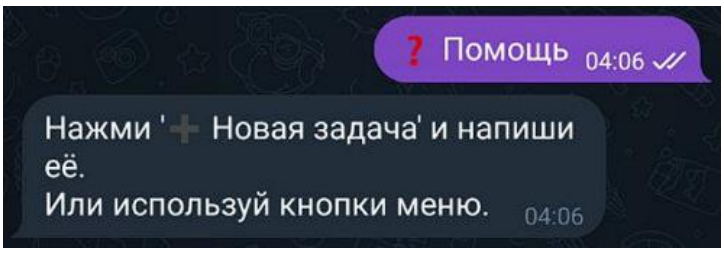

Рис. 7 Ответ на кнопку “Помощь”

Заключение

В ходе выполнения проекта были освоены принципы разработки серверных приложений на языке Go и взаимодействие с Telegram Bot API. Изучены механизмы конкурентного доступа (использование sync.RWMutex) и обработки HTTP-запросов (Webhooks) для обеспечения стабильной работы сервиса. Реализованный проект «TaskBot» успешно решает задачу управления жизненным циклом задач (создание, назначение, выполнение) и координации действий пользователей внутри мессенджера.