**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторной работе №6

«Построение общей архитектуры проекта»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-32Б |  | преподаватель каф. ИУ5 |
| Старкин Александр |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2024 г.

### Задание:

Разработать общий план для работы приложения с использованием кода (по необходимости).

Разбить код на структуры, интерфейсы и функции.

**Выполнение:**

При запуске программы происходит инициализация уже существующих пользователей и их сессий.

package main

import (

    "context"

    "fmt"

    "log"

    "os"

    "sync"

    "time"

    "github.com/Robocotik/IT-s\_Friends/internal/database"

    "github.com/Robocotik/IT-s\_Friends/internal/database/postgres"

    "github.com/Robocotik/IT-s\_Friends/internal/models/structures"

    "github.com/Robocotik/IT-s\_Friends/internal/services/logic"

    "github.com/Robocotik/IT-s\_Friends/internal/services/notify"

    "github.com/joho/godotenv"

    "github.com/mymmrac/telego"

    th "github.com/mymmrac/telego/telegohandler"

)

func initEnv() {

    err := godotenv.Load("../configs/.env")

*if* err != nil {

        log.Fatal("Error loading .env file")

    }

}

func main() {

    initEnv()

    sessions := make(map[int64]\*structures.User)

    var sessionsMutex sync.Mutex

    botToken := os.Getenv("TOKEN")

    bot, err := telego.NewBot(botToken)

    botUser, err := bot.GetMe()

*if* err != nil {

        fmt.Println("Ошибка при получении бота " + err.Error())

    }

    updates, err := bot.UpdatesViaLongPolling(nil)

*if* err != nil {

        fmt.Println("Ошибка при update " + err.Error())

    }

    ctxMain, mainCancel := context.WithCancel(context.Background())

*defer* mainCancel()

    ctxTimer, cancel := context.WithTimeout(ctxMain, 10\*time.Second)

*defer* cancel()

    var psql postgres.Postgres

    psql.Conn, err = database.NewDatabase("POSTGRES\_DRIVER", "POSTGRES\_USER", "POSTGRES\_PASSWORD", "POSTGRES\_HOST", "POSTGRES\_PORT", "POSTGRES\_TABLE")

*defer* psql.Conn.Close(context.Background())

    psql.InitSessions(&sessions)

    bh, \_ := th.NewBotHandler(bot, updates)

*go* notify.CronMain(psql.Conn, bot, botUser.ID)

*if* err != nil {

        fmt.Println(err)

        os.Exit(1)

    }

*defer* bh.Stop()

*defer* bot.StopLongPolling()

    bh.HandleMessage(func(bot \*telego.Bot, msg telego.Message) {

        userID := msg.From.ID

        sessionsMutex.Lock()

        user, exists := sessions[userID]

*if* !exists {

            user = &structures.User{

                Id:     userID,

                State:  structures.StateStart,

                Exists: false,

            }

            sessions[userID] = user

            psql.AddUserId(bot, msg.Chat.ChatID().ID, msg.Chat.ChatID().ID, msg.From.Username)

        }

        sessionsMutex.Unlock()

*go* func() {

            logic.DoSwitch(ctxMain, psql, user, &user.Friend, bot, msg)

        }()

    })

    bh.Start()

*for* {

*select* {

*case* <-ctxTimer.Done():

            fmt.Println("Ошибка при установки соединения с базой данных")

*default*:

            fmt.Println("Соединение с базой данных установлено")

        }

    }

}

Каждый handleMessage каждого пользователя уходит в свою горутину, что позволяет обрабатывать независимо и асинхронно.

Функция DoSwitch представляет из себя state machine, что перешагивает на нужную тсадию в зависимости от ответа пользователя.

package logic

import (

    "context"

    "errors"

    "github.com/Robocotik/IT-s\_Friends/assets/consts"

    keyboard "github.com/Robocotik/IT-s\_Friends/assets/keyboards"

    "github.com/Robocotik/IT-s\_Friends/internal/database"

    errorsCustom "github.com/Robocotik/IT-s\_Friends/internal/models/errors"

    "github.com/Robocotik/IT-s\_Friends/internal/models/structures"

    "github.com/Robocotik/IT-s\_Friends/internal/services/input"

    "github.com/Robocotik/IT-s\_Friends/internal/services/output"

    output\_friends "github.com/Robocotik/IT-s\_Friends/internal/services/output/friends"

    output\_schedule "github.com/Robocotik/IT-s\_Friends/internal/services/output/schedule"

    "github.com/Robocotik/IT-s\_Friends/internal/services/utils"

    handle "github.com/Robocotik/IT-s\_Friends/internal/transport/handlers"

    "github.com/mymmrac/telego"

)

var ch\_zn\_selected = ""

func DoSwitch(ctx context.Context, bd database.IBd, user \*structures.User, friend \*structures.AskedFriend, bot \*telego.Bot, msg telego.Message) {

    var err error

*switch* user.State {

*case* structures.StateStart:

        handle.HandleStart(bot, msg)

        user.State = structures.StateDefault

*case* structures.StateDefault:

        \_, err = input.ParseString(bot, msg, errors.New("ответ"), []string{"Погнали"})

*if* err != nil {

            handle.HandleStart(bot, msg)

*break*

        }

*if* user.Exists {

            handle.HandleMenuStart(bot, msg)

            user.State = structures.StateStartMenu

        } *else* {

            handle.HandleAddMe(bot, msg)

            user.State = structures.UserNotExists

        }

*case* structures.UserNotExists:

        handle.HandleSelectFilial(bd, bot, msg)

        user.State = structures.StateAskForMe

*case* structures.StateAskForMe:

        FillObjectWithInfo(&user.StateFilling, bd, bot, msg, &user.Identity, true)

*if* user.StateFilling == structures.StateSearch {

            handle.HandleMenuStart(bot, msg)

            user.State = structures.StateStartMenu

            user.Exists = true

            bd.UpdateUser(bot, msg.Chat.ID, user.Identity, msg.Chat.ID)

        }

*case* structures.StateStartMenu:

        \_, err = input.ParseString(bot, msg, errors.New("ответ"), []string{consts.FIND\_NEW\_FRIENDS, consts.SHOW\_FRIENDS, consts.SET\_NOTIFICATIONS})

*if* err != nil {

            handle.HandleMenuStart(bot, msg)

*break*

        }

*switch* msg.Text {

*case* consts.FIND\_NEW\_FRIENDS:

            handle.HandleSelectFilial(bd, bot, msg)

            user.State = structures.StateAskForFriend

*case* consts.SET\_NOTIFICATIONS:

            handle.HandleSetNotifications(bot, msg)

            user.State = structures.StateSetNotifications

*default*:

            favs, err := bd.GetFriendsFromId(msg.Chat.ChatID().ID)

            output.RiseError(bot, msg.Chat.ID, err)

            utils.FuncWithKeyboard(bot, msg, func() (string, error) {

*return* output\_friends.ShowFavs(favs)

            }, keyboard.CreateKeyboardReturnToSearch())

            user.State = structures.StateRedirectToStartSearch

        }

*case* structures.StateSetNotifications:

        val, err2 := input.ParseString(bot, msg, errors.New("вариант"), []string{consts.CUSTOM\_TIME, consts.H1\_BEFORE, consts.H2\_BEFORE, consts.H3\_BEFORE})

        output.RiseError(bot, msg.Chat.ID, err2)

*if* err2 != nil {

            handle.HandleSetNotifications(bot, msg)

*break*

        }

*if* val == consts.CUSTOM\_TIME {

            user.State = structures.StateSetCustomNotification

            handle.HandleSetCustomNotification(bot, msg)

        } *else* {

            handle.HandleNotificationCreated(bot, msg)

            user.State = structures.StateRedirectToStartSearch

*// add server notification*

        }

*case* structures.StateSetCustomNotification:

        user.NotifyPeriod, err = input.CheckPeriod(msg.Text)

*if* err != nil {

            output.WriteMessage(bot, msg.Chat.ID, "Неверный формат данных")

            handle.HandleSetCustomNotification(bot, msg)

        }

*// add server notification*

*case* structures.StateAskForFriend:

        FillObjectWithInfo(&user.Friend.State, bd, bot, msg, &friend.Identity, false)

*if* user.Friend.State == structures.StateSearch {

            user.State = structures.StateSearch

            user.Friend.State = structures.StateAskFilial

        }

*case* structures.StateSearch:

        friend.Identity.Uuid = bd.GetGroupByUID(bot, msg.Chat.ID, &friend.Identity)

        friend.Request = DoRequest(bot, msg, friend.Identity.Uuid)

*if* len(friend.Request.Data.Schedule) != 0 { *// проверка на наличие расписания*

            handle.HandleGroupFound(bot, msg)

            user.State = structures.StateGroupFound

        } *else* {

            handle.HandleGroupNotFound(bot, msg)

            user.State = structures.StateRedirectToStartSearch

        }

*case* structures.StateGroupFound: *// Группа была найдена*

        \_, err = input.ParseString(bot, msg, errors.New("ответ"), []string{consts.ADD\_TO\_FAVOURITE, consts.SHOW\_SCHEDULE})

*if* err != nil {

            handle.HandleConfirm(bot, msg, &friend.Identity, false)

*break*

        }

*if* msg.Text == consts.ADD\_TO\_FAVOURITE {

            handle.HandleSelectNickname(bot, msg)

            user.State = structures.StateAskNickname

        } *else* {

            user.State = structures.StateShowTimetable

            handle.HandleShowTimetable(bot, msg)

        }

*case* structures.StateAskNickname:

        friend.NickName = msg.Text

        friend\_id, err := bd.AddFriend(bot, msg, friend)

*if* err != nil {

*if* err.Error() == errorsCustom.ErrTooLongMessage\_23514 {

                output.WriteMessage(bot, msg.Chat.ID, errorsCustom.ErrTooLongMessage\_23514)

                handle.HandleSelectNickname(bot, msg)

*break*

            }

        } *else* {

            bd.AddConnection(ctx, bot, msg.Chat.ID, msg.Chat.ChatID().ID, friend\_id)

            handle.HandleAddToHavourite(bot, msg)

        }

        user.State = structures.StateRedirectToStartSearch

*case* structures.StateShowTimetable: *// Вывод расписания*

        ch\_zn\_selected, err := input.ParseContainString(bot, msg, []string{consts.Ch, consts.Zn})

        output.RiseError(bot, msg.Chat.ID, err)

        keyboard := keyboard.CreateKeyboardShowTimetable()

        output\_schedule.ShowTimetable(bot, msg, keyboard, friend.Request, ch\_zn\_selected)

        user.State = structures.StateRedirectToStartSearch

*case* structures.StateRedirectToStartSearch:

        handle.HandleMenuStart(bot, msg)

        user.State = structures.StateStartMenu

*default*:

        output.WriteMessage(bot, msg.Chat.ID, "Неизвестная команда")

        panic("unknown state")

    }

}

В начале каждого state происходит считывание значения, в обработка логики, а в конце переход на следующий.

Полезная работа шага разнообразна, в основном это:

* WriteMessage – функция вывода сообщения, с целью последующего перехода на следующий шаг программы.
* Handle – пакет с хендлерами для вывода сообщения с клавиатурой впоследствии.
* Bd – пакет для работы с базой данных.
* Request – пакет для отправки http запросов в интернет.

http запросы оборачиваются в context с таймером для отмены запросы при истечении предельного времени.

Результат:







