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 ИУ5 «Системы обработки информации и управления»

Проект

«VK-bot выдачи расписания МГТУ им. Баумана»

по дисциплине

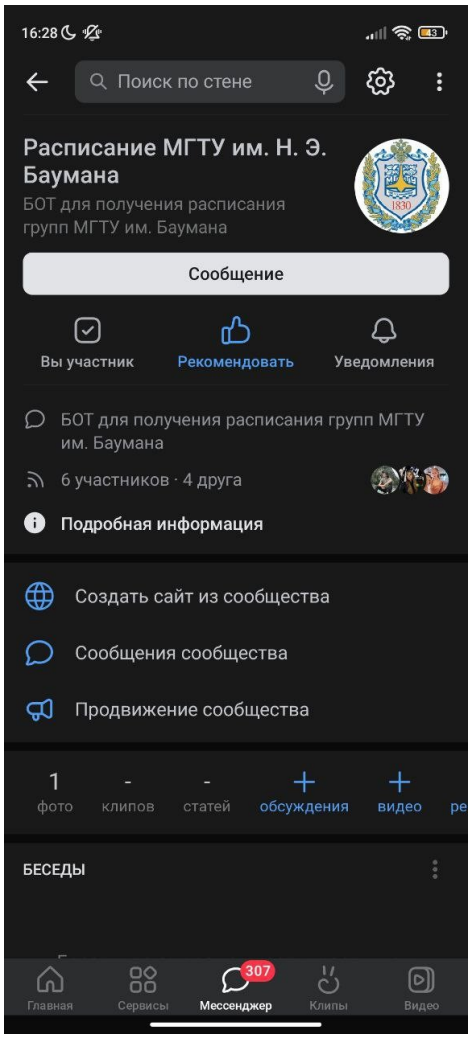
«Программирование сетевых приложений»

Выполнил:
студент группы ИУ5-43Б
Зорькин А.В.
Абрамов В.Г.

Проверила:
Аксенова М.В.

2022 г.

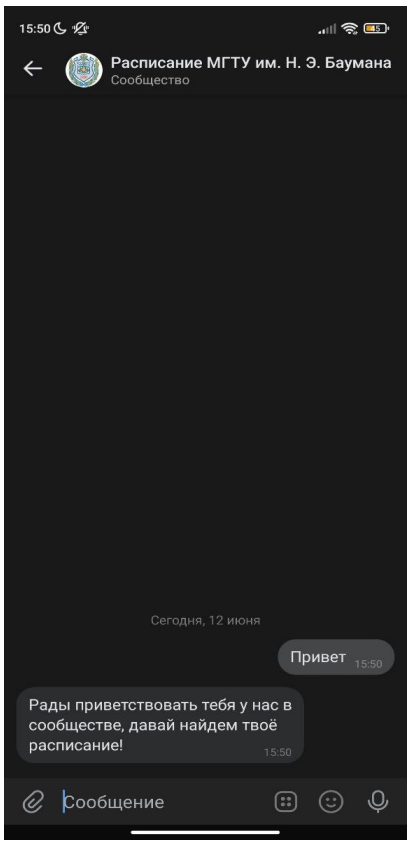
Бот позволяет получать расписание групп МГТУ им. Баумана. Для реализации бота было необходимо создать сообщество:

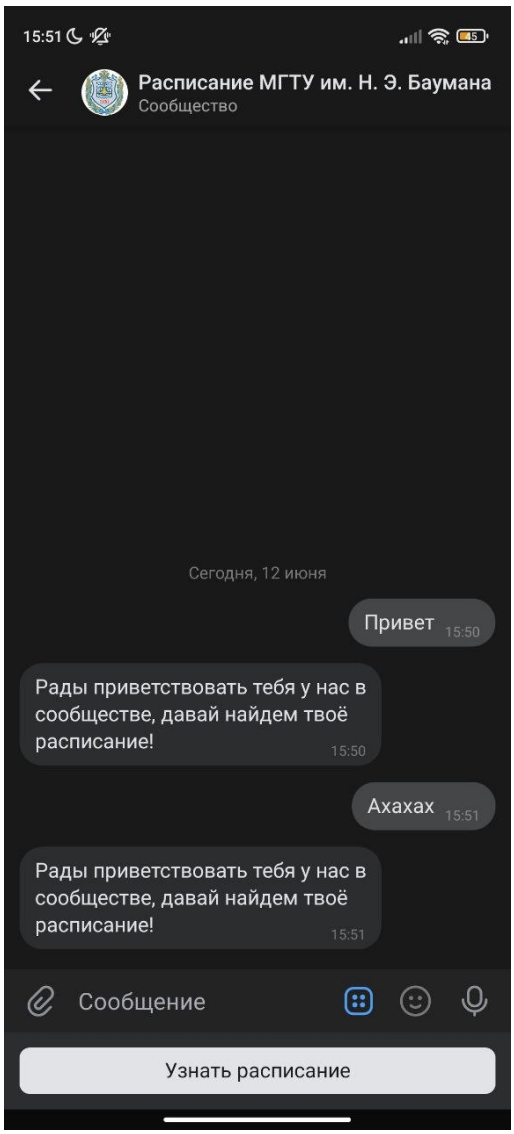


https://vk.com/club213613932

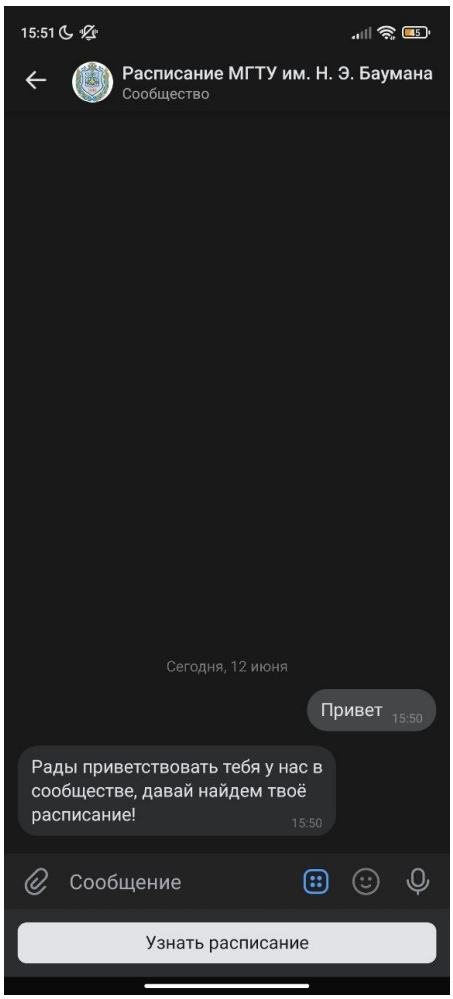
Для получения расписания требуется написать сообщение:

Можно написать абсолютно любое приветственное сообщение, после чего бот нам ответит:

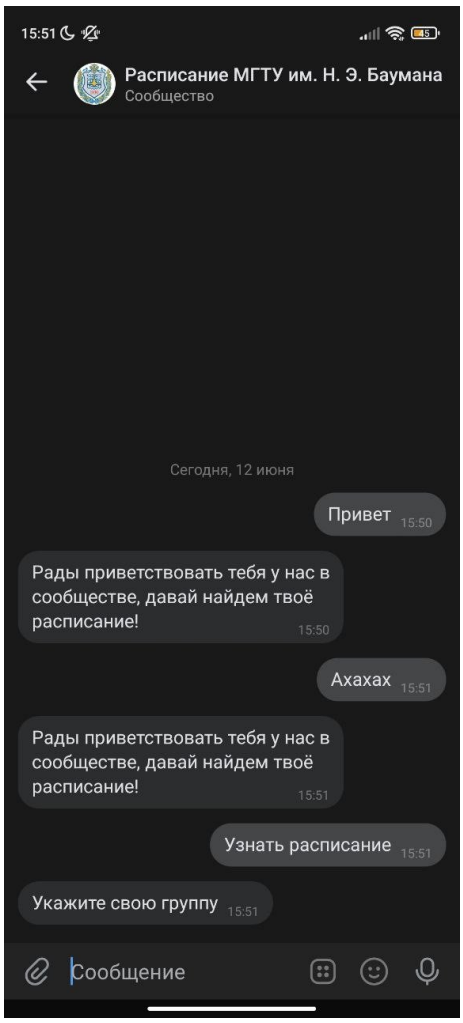

15:50
Расписание МГТУ им. Н. Э. Баумана
Сообщество
Сегодня, 12 июня
Привет
15:50
Рады приветствовать тебя у нас в сообществе, давай найдем твоё расписание!
15:50
Сообщение



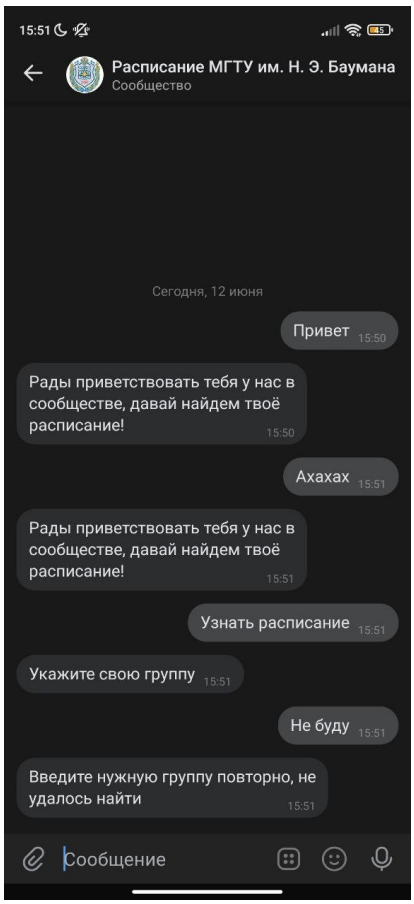
Имеется кнопка, на которую требуется нажать:



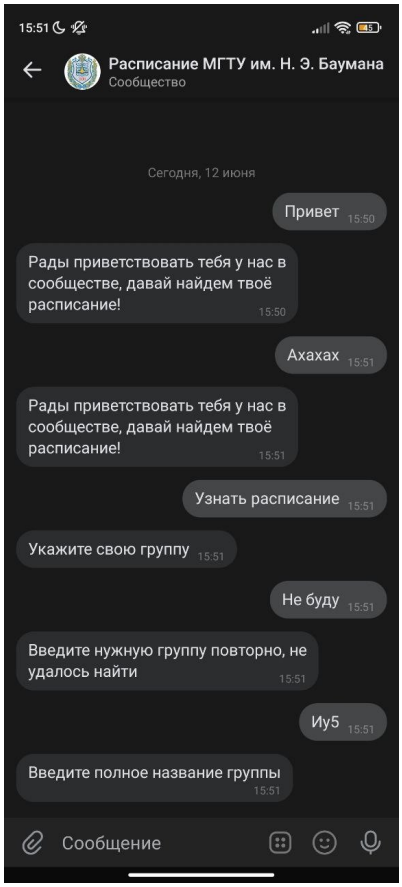
После нажатия на которую бот предложит выбрать группу:



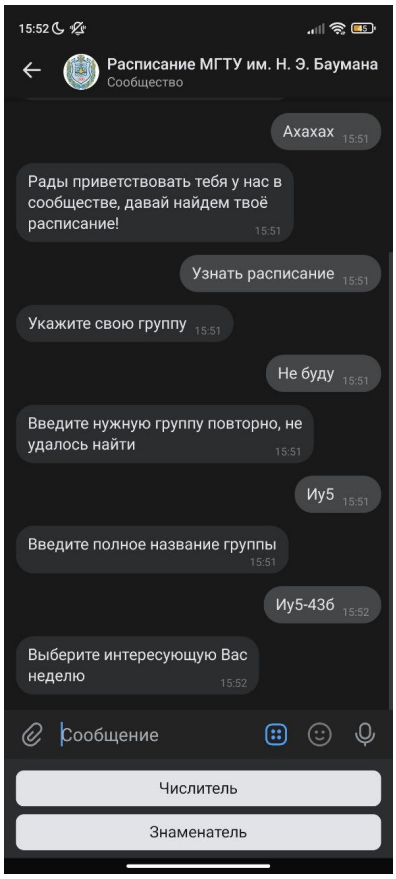
Если вводить несуществующую группу, то будет выдано сообщение:



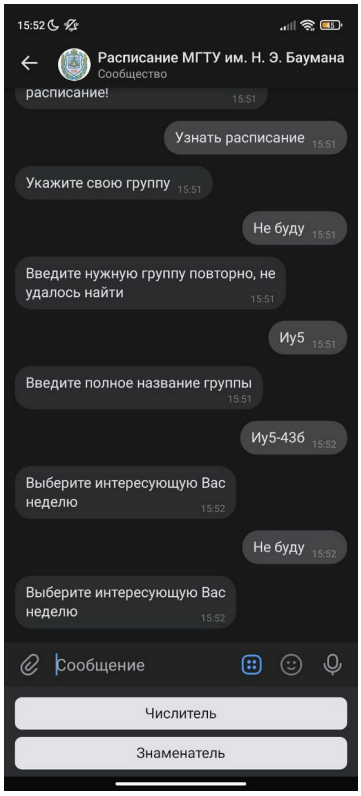
Если ввести неполное название группы, то бот выдаст:



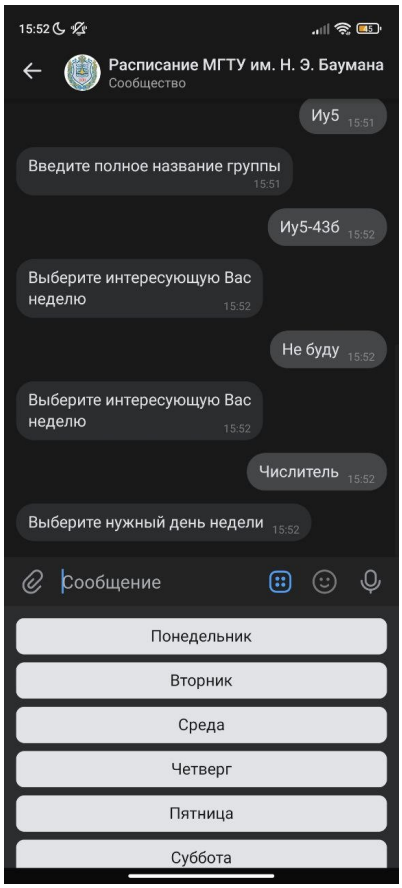
Если вбить название группы, то бот предложит выбрать нужную неделю (числитель, знаменатель):



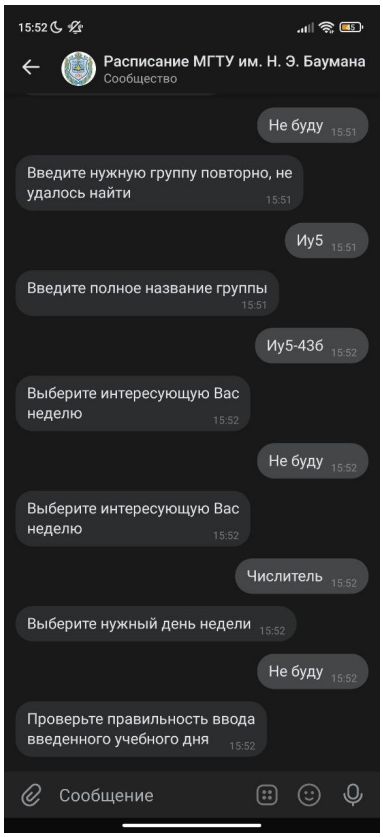
Если не написать нужную неделю, то бот выдаст:

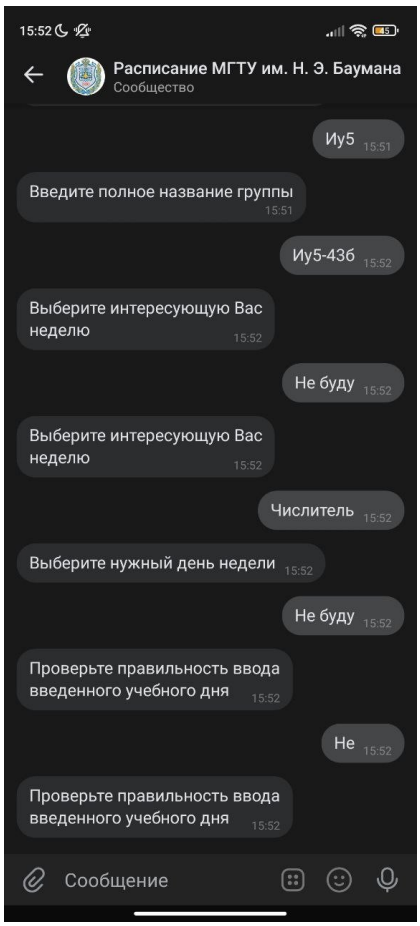


Если выбрать одну из возможных недель, то бот предложит выбрать день недели:

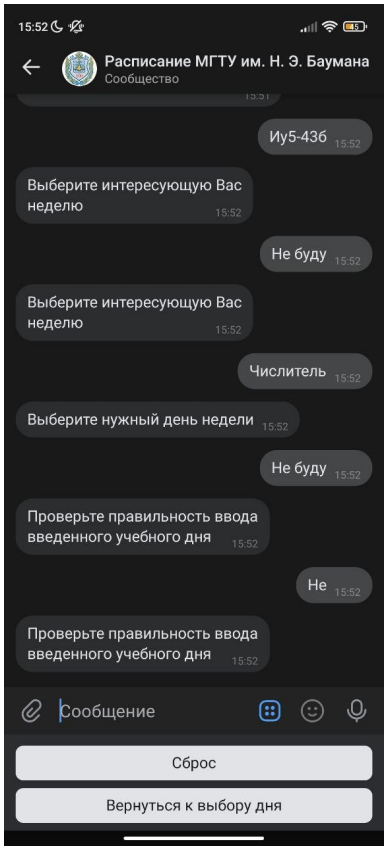


Если ввести несуществующий день недели:

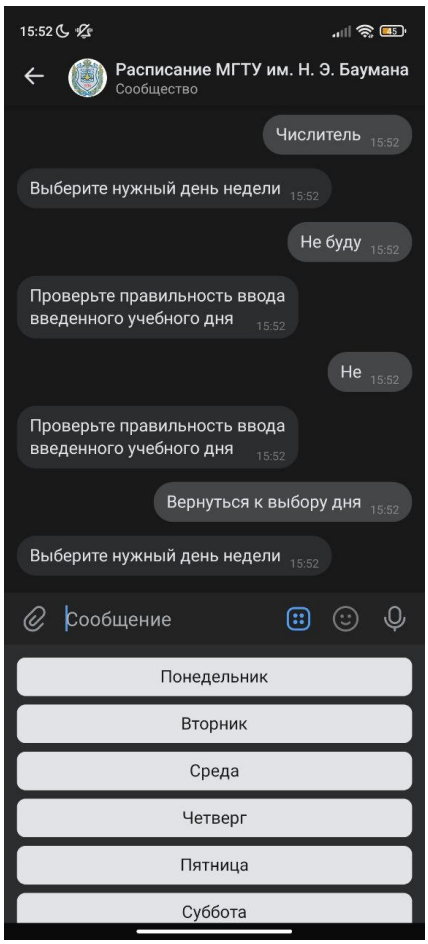




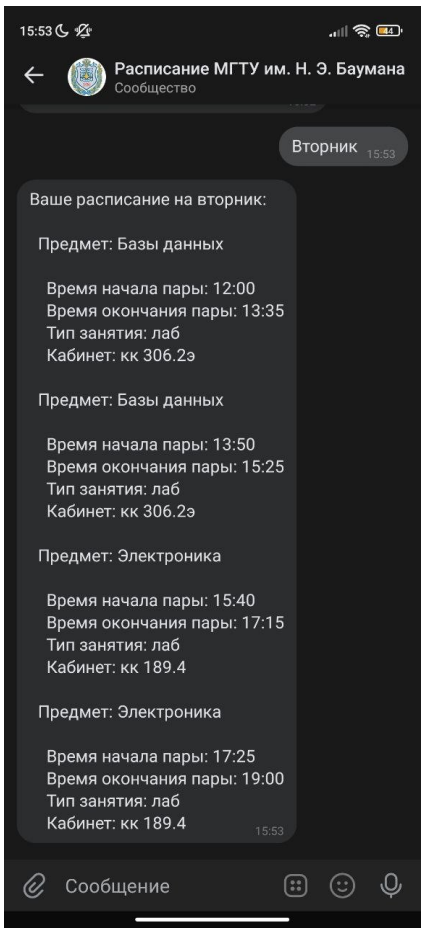
В таком случае будут кнопки:



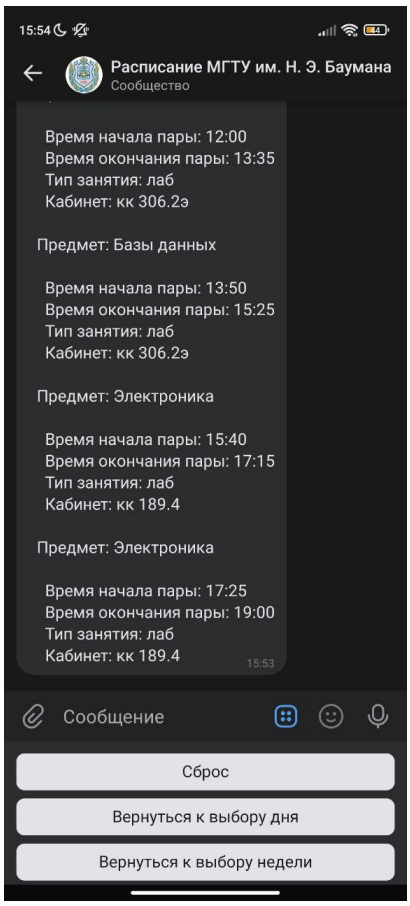
При нажатии на кнопку «Вернуться к выбору дня»:



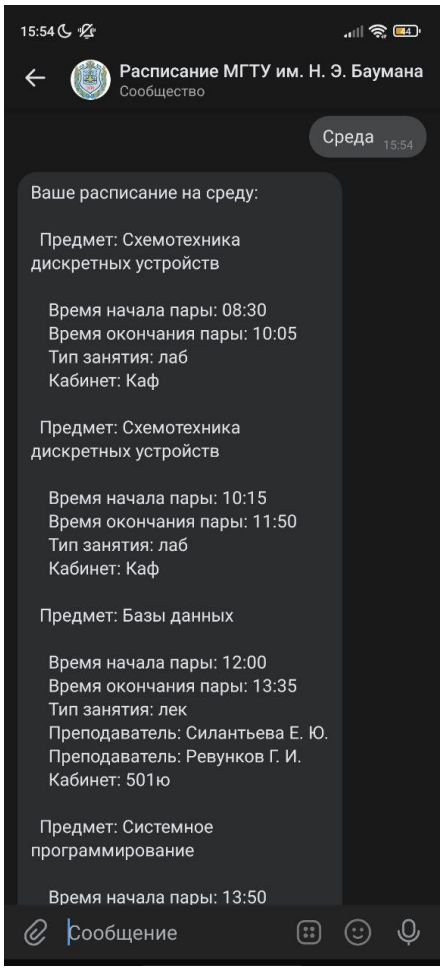
При выборе дня имеем:

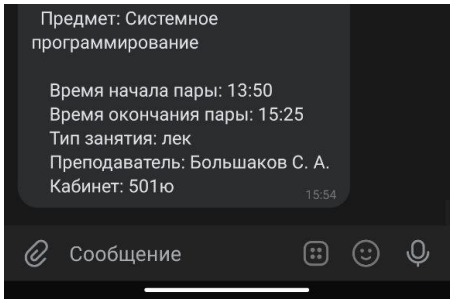


После получения расписания имеем кнопки:

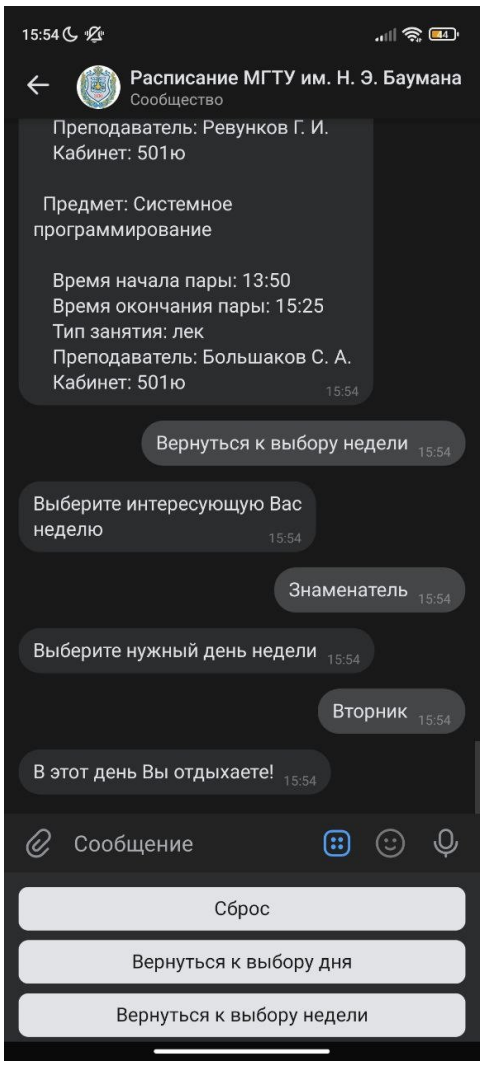


Выберем другой день:

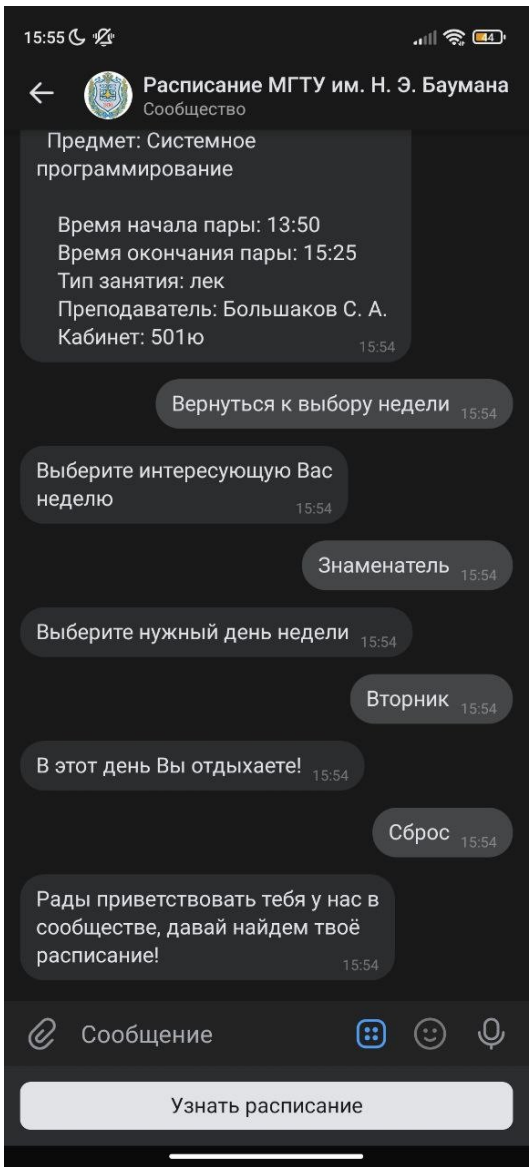




Выберем другую неделю:



После нажатия на кнопку «Сброс», имеем возвращение в начальное состояние:



Код проекта можно найти на GitHub:

https://github.com/Alekseizor/vk-bot-bmstu

Основные файлы:

main.go

```
package main

import (
        "context"
        log "github.com/sirupsen/logrus"
        "main/internal/app/config"
        "main/internal/pkg/app"
        "os"
)

func init() {
        // Log as JSON instead of the default ASCII formatter.
        log.SetFormatter(&log.JSONFormatter{})
```

```
        // Output to stdout instead of the default stderr
        // Can be any io.Writer, see below for File example
        log.SetOutput(os.Stdout)

        // Only log the warning severity or above.
        log.SetLevel(log.WarnLevel)
}

func main() {
        ctx := context.Background()

        cfg, err := config.NewConfig(ctx)
        if err != nil {
                log.WithContext(ctx).WithError(err).Error("cant init config")

                os.Exit(2)
        }

        ctx = config.WrapContext(ctx, cfg)

        application, err := app.New(ctx)
        if err != nil {
                log.WithContext(ctx).WithError(err).Error("cant create
application")

                os.Exit(2)
        }

        err = application.Run(ctx)
        if err != nil {
                log.WithContext(ctx).WithError(err).Error("cant run
application")

                os.Exit(2)
        }

}
```

app.go

```
package app

import (
        "context"
        "fmt"
        "github.com/SevereCloud/vksdk/v2/api"
        "github.com/SevereCloud/vksdk/v2/events"
        "github.com/SevereCloud/vksdk/v2/longpoll-bot"
        log "github.com/sirupsen/logrus"
        vk_client "main/internal/app/button"
        "main/internal/app/config"
        "main/internal/app/ds"
        "main/internal/app/redis"
        "main/internal/app/state"
        "main/internal/pkg/clients/bitop"
        "strings"
)

var start_message string = "Рады приветствовать тебя у нас в сообществе,
выбирай пункт меню и полетели!"
var chatcontext state.ChatContext
```

```
type App struct {
	// корневой контекст
	ctx context.Context
	vk  *api.VK
	lp  *longpoll.LongPoll

	vkClient    *vk_client.VkClient
	redisClient *redis.RedClient
	bitopClient *bitop.Client
}

func New(ctx context.Context) (*App, error) {
	cfg := config.FromContext(ctx)
	vk := api.NewVK(cfg.VKToken)
	group, err := vk.GroupsGetByID(nil)
	if err != nil {
		log.WithError(err).Error("cant get groups by id")

		return nil, err
	}

	log.WithField("group_id", group[0].ID).Info("init such group")

	c, err := redis.New(ctx)
	if err != nil {
		return nil, err
	}

	vkClient, err := vk_client.New(ctx)
	if err != nil {
		return nil, err
	}

	//starting long poll
	lp, err := longpoll.NewLongPoll(vk, group[0].ID)
	if err != nil {
		log.Fatal(err)
	}

	app := &App{
		ctx:         ctx,
		vk:          vk,
		lp:          lp,
		vkClient:    vkClient,
		redisClient: c,
	}

	return app, nil
}

func (a *App) Run(ctx context.Context) error {
	// New message event
	var ScheduleUser *ds.User
	var err error
	a.lp.MessageNew(func(_ context.Context, obj events.MessageNewObject) {

		messageText := obj.Message.Text
		fmt.Println(messageText)
		fmt.Println(obj.Message.PeerID)
		ScheduleUser, err = a.redisClient.GetUser(ctx,
obj.Message.PeerID)
		if err != nil {
			log.WithError(err).Error("cant set user")
```

```
                return
        }
        //if the user writes for the first time, add to the database
        if ScheduleUser == nil {
                ScheduleUser = &ds.User{}
                ScheduleUser.VkID = obj.Message.PeerID
                ScheduleUser.State = "StartState"
                err := a.redisClient.SetUser(ctx, *ScheduleUser)
                if err != nil {
                        log.WithError(err).Error("cant set user")
                        return
                }
        } else if ScheduleUser.State == "" {
                ScheduleUser.State = "StartState"
                err := a.redisClient.SetUser(ctx, *ScheduleUser)
                if err != nil {
                        log.WithError(err).Error("cant set user")
                        return
                }
        }
        fmt.Println(ScheduleUser.State) //норм

        if strings.EqualFold(messageText, "Сброс") {
                ScheduleUser.State = "StartState"
                err := a.redisClient.SetUser(ctx, *ScheduleUser)
                if err != nil {
                        log.WithError(err).Error("cant set user")
                        return
                }
        }

        strInState := map[string]state.State{
                state.RefStartState.Name():      state.RefStartState,
                state.RefBranchState.Name():     state.RefBranchState,
                state.RefFacultyState.Name():    state.RefFacultyState,
                state.RefDepartmentState.Name():
state.RefDepartmentState,
                state.RefGroupState.Name():      state.RefGroupState,
                state.RefWeekState.Name():       state.RefWeekState,
                state.RefNextWeekState.Name():
state.RefNextWeekState,
                state.RefDayState.Name():        state.RefDayState,
                state.RefErrorState.Name():      state.RefErrorState,
        }
        ctc := state.ChatContext{
                ScheduleUser,
                a.vk,
                a.redisClient,
                &ctx,
                a.bitopClient,
        }

        step := strInState[ScheduleUser.State]
        fmt.Println(step.Name())
        nextStep := step.Process(ctc, messageText)
        fmt.Println(nextStep.Name())
        ScheduleUser.State = nextStep.Name()
        err = a.redisClient.SetUser(ctx, *ScheduleUser)
        if err != nil {
                log.WithError(err).Error("cant set user")
                return
        }

        //ScheduleUser.State = nextStep.Name()
```

```
		//err = a.redisClient.SetUser(ctx, *ScheduleUser)
		//if err != nil {
		//	log.WithError(err).Error("cant set user")
		//	return
		//}
		/*messageText := obj.Message.Text
		ScheduleUser := &ds.User{}
		//check if we have such a user
		ScheduleUser, err := a.redisClient.GetUser(ctx,
obj.Message.PeerID)
		if err != nil {
			log.WithError(err).Error("cant set user")

			return
		}
		//if the user writes for the first time, add to the database
		if ScheduleUser == nil {
			ScheduleUser.VkID = obj.Message.PeerID
			ScheduleUser.State = "StartState"
			err := a.redisClient.SetUser(ctx, *ScheduleUser)
			if err != nil {
				log.WithError(err).Error("cant set user")
				return
			}
		}
		//to get states
		strInState := map[string]state.State{
			state.RefStartState.Name():      state.RefStartState,
			state.RefBranchState.Name():     state.RefBranchState,
			state.RefFacultyState.Name():    state.RefFacultyState,
			state.RefDepartmentState.Name():
state.RefDepartmentState,
			state.RefGroupState.Name():      state.RefGroupState,
			state.RefWeekState.Name():       state.RefWeekState,
			state.RefNextWeekState.Name():
state.RefNextWeekState,
			state.RefDayState.Name():        state.RefDayState,
			state.RefErrorState.Name():      state.RefErrorState,
		}
		if strings.EqualFold(messageText, "Сброс") {
			ScheduleUser.State = "StartState"
			err := a.redisClient.SetUser(ctx, *ScheduleUser)
			if err != nil {
				log.WithError(err).Error("cant set user")
				return
			}
		}
		ctc := state.ChatContext{
			ScheduleUser,
			a.vk,
			a.redisClient,
			a.ctx,
			a.bitopClient,
		}

		step := strInState[ScheduleUser.State]
		fmt.Println(step.Name())
		nextStep := step.Process(ctc, messageText)
		ScheduleUser.State = nextStep.Name()
		err = a.redisClient.SetUser(ctx, *ScheduleUser)
		if err != nil {
			log.WithError(err).Error("cant set user")
			return
		}
```

```
                */
        })
        log.Println("Start Long Poll")
        if err := a.lp.Run(); err != nil {
                log.Fatal(err)
                return nil
        }
        return nil
}
```

client.go

```
package bitop

import (
        "bytes"
        "context"
        "encoding/json"
        "errors"
        log "github.com/sirupsen/logrus"
        "io"
        "main/internal/app/config"
        "main/internal/app/model"
        "net/http"
        "net/url"
        "strconv"
)

type Client struct {
        ctx    context.Context
        body   model.BitopBody
        client *http.Client
}

func New(ctx context.Context) *Client {
        cfg := config.FromContext(ctx)
        return &Client{
                ctx: ctx,
                body: model.BitopBody{
                        Token: cfg.BITOPToken,
                },
                client: &http.Client{},
        }
}

// GetBranch get info about branch from request
func (c *Client) GetBranch(ctx context.Context, branch string)
(*model.ResponseBody, error) {
        cfg := config.FromContext(ctx).BITOP

        //creating url
        url := url.URL{
                Scheme: cfg.Protocol,
                Host:   cfg.SiteAdress,
                Path:   cfg.PathSearch,
        }

        log.Info("url created", url.String())

        //create request body
```

```
        reqBody, _ := json.Marshal(model.RequestBody{
                "",
                branch,
                "branch",
        })

        //create request
        reqToApi, err := http.NewRequest("POST", url.String(),
bytes.NewBuffer(reqBody))
        if err != nil {
                log.WithError(err).Error("cant create request")
                return nil, err
        }

        //create request headers
        reqToApi.Header = http.Header{
                "x-bb-token": {c.body.Token},
        }

        //do request
        rawResp, err := c.client.Do(reqToApi)
        if err != nil {
                log.WithError(err).Error("cant do request")
                return nil, err
        }

        var resp model.ResponseBody

        //status code check
        if rawResp.StatusCode != 200 {
                errLog := "status code is" + strconv.Itoa(rawResp.StatusCode)
                log.Error(errLog)
                return nil, errors.New(errLog)
        }

        //read response body
        body, err := io.ReadAll(rawResp.Body)
        if err != nil {
                log.WithError(err).Error("cant read response")
                return nil, err
        }

        //unmarshall response body
        err = json.Unmarshal(body, &resp)
        if err != nil {
                log.WithError(err).Error("cant unmarshal response")
                return nil, err
        }

        return &resp, err
}

// GetFaculty get info about faculty from parent uuid
func (c *Client) GetFaculty(ctx context.Context, parentUUID string)
(*model.ResponseBody, error) {
        cfg := config.FromContext(ctx).BITOP

        url := url.URL{
                Scheme: cfg.Protocol,
                Host:   cfg.SiteAdress,
                Path:   cfg.PathSearch,
        }

        reqBody, _ := json.Marshal(model.RequestBody{
```

```
                parentUUID,
                "",
                "faculty",
        })

        reqToApi, err := http.NewRequest("POST", url.String(),
bytes.NewBuffer(reqBody))
        if err != nil {
                log.WithError(err).Error("cant create request")
        }

        reqToApi.Header = http.Header{
                "x-bb-token": {c.body.Token},
        }

        rawResp, err := c.client.Do(reqToApi)
        if err != nil {
                log.WithError(err).Error("cant do request")
                return nil, err
        }

        if rawResp.StatusCode != 200 {
                errLog := "status code is" + strconv.Itoa(rawResp.StatusCode)
                log.Error(errLog)
                return nil, errors.New(errLog)
        }

        body, err := io.ReadAll(rawResp.Body)
        if err != nil {
                log.WithError(err).Error("cant read response")
                return nil, err
        }
        var resp model.ResponseBody

        err = json.Unmarshal(body, &resp)
        if err != nil {
                log.WithError(err).Error("cant unmarshal response")
                return nil, err
        }

        return &resp, err
}

// GetDepartment get info about department from from parent uuid
func (c *Client) GetDepartment(ctx context.Context, parentUUID string)
(*model.ResponseBody, error) {
        cfg := config.FromContext(ctx).BITOP

        url := url.URL{
                Scheme: cfg.Protocol,
                Host:   cfg.SiteAdress,
                Path:   cfg.PathSearch,
        }

        reqBody, _ := json.Marshal(model.RequestBody{
                parentUUID,
                "",
                "department",
        })

        reqToApi, err := http.NewRequest("POST", url.String(),
bytes.NewBuffer(reqBody))
        if err != nil {
                log.WithError(err).Error("cant create request")
```

```
	}

	reqToApi.Header = http.Header{
		"x-bb-token": {c.body.Token},
	}

	rawResp, err := c.client.Do(reqToApi)
	if err != nil {
		log.WithError(err).Error("cant do request")
		return nil, err
	}

	if rawResp.StatusCode != 200 {
		errLog := "status code is" + strconv.Itoa(rawResp.StatusCode)
		log.Error(errLog)
		return nil, errors.New(errLog)
	}

	body, err := io.ReadAll(rawResp.Body)
	if err != nil {
		log.WithError(err).Error("cant read response")
		return nil, err
	}
	var resp model.ResponseBody

	err = json.Unmarshal(body, &resp)
	if err != nil {
		log.WithError(err).Error("cant unmarshal response")
		return nil, err
	}

	return &resp, err
}

// GetGroup get info about group, from parent uuid
func (c *Client) GetGroup(ctx context.Context, groupName string)
(*model.ResponseBody, error) {
	cfg := config.FromContext(ctx).BITOP

	url := url.URL{
		Scheme: cfg.Protocol,
		Host:   cfg.SiteAdress,
		Path:   cfg.PathSearch,
	}

	reqBody, _ := json.Marshal(model.RequestBody{
		"",
		groupName,
		"group",
	})

	reqToApi, err := http.NewRequest("POST", url.String(),
bytes.NewBuffer(reqBody))
	if err != nil {
		log.WithError(err).Error("cant create request")
	}

	reqToApi.Header = http.Header{
		"x-bb-token": {c.body.Token},
	}

	rawResp, err := c.client.Do(reqToApi)
	if err != nil {
		log.WithError(err).Error("cant do request")
```

```
		return nil, err
	}

	if rawResp.StatusCode != 200 {
		errLog := "status code is" + strconv.Itoa(rawResp.StatusCode)
		log.Error(errLog)
		return nil, errors.New(errLog)
	}

	body, err := io.ReadAll(rawResp.Body)
	if err != nil {
		log.WithError(err).Error("cant read response")
		return nil, err
	}
	var resp model.ResponseBody

	err = json.Unmarshal(body, &resp)
	if err != nil {
		log.WithError(err).Error("cant unmarshal response")
		return nil, err
	}

	return &resp, err
}

func (c *Client) GetSchedule(ctx context.Context, parentUUID string,
IsNumerator bool, message string) (*model.ResponseBodySchedule, error) {
	weekdays := map[string]int{
		"Понедельник": 1,
		"Вторник":     2,
		"Среда":       3,
		"Четверг":     4,
		"Пятница":     5,
		"Суббота":     6,
	}

	cfg := config.FromContext(ctx).BITOP

	url := url.URL{
		Scheme: cfg.Protocol,
		Host:   cfg.SiteAdress,
		Path:   cfg.PathPath,
	}

	urlS := url.String() + parentUUID

	reqToApi, err := http.NewRequest("GET", urlS, nil)
	if err != nil {
		log.WithError(err).Error("cant create request")
	}

	reqToApi.Header = http.Header{
		"x-bb-token": {c.body.Token},
	}

	rawResp, err := c.client.Do(reqToApi)
	if err != nil {
		log.WithError(err).Error("cant do request")
		return nil, err
	}

	if rawResp.StatusCode != 200 {
		errLog := "status code is" + strconv.Itoa(rawResp.StatusCode)
		log.Error(errLog)
```

```
                return nil, errors.New(errLog)
        }

        body, err := io.ReadAll(rawResp.Body)
        if err != nil {
                log.WithError(err).Error("cant read response")
                return nil, err
        }
        var resp model.ResponseBodySchedule

        err = json.Unmarshal(body, &resp)
        if err != nil {
                log.WithError(err).Error("cant unmarshal response")
                return nil, err
        }

        var result model.ResponseBodySchedule
        k := 0
        for _, item := range resp.Lessons {
                if item.Day == weekdays[message] && item.IsNumerator ==
IsNumerator {
                        k++
                        result.Lessons = append(result.Lessons, item)
                }
        }
        if k == 0 {
                return nil, nil
        }
        return &result, err
}
```

state.go

```
package state

import (
        "context"
        "fmt"
        "github.com/SevereCloud/vksdk/v2/api"
        "github.com/SevereCloud/vksdk/v2/api/params"
        "github.com/SevereCloud/vksdk/v2/object"
        log "github.com/sirupsen/logrus"
        "main/internal/app/ds"
        "main/internal/app/model"
        "main/internal/app/redis"
        "main/internal/pkg/clients/bitop"
        "strconv"
        "strings"
        "time"
)

//////////////////////////////////////////////////////
type ChatContext struct {
        User        *ds.User
        Vk          *api.VK
        RedisClient *redis.RedClient
        Ctx         *context.Context
        BitopClient *bitop.Client
        //получаем информацию о пользователе
```

```
	//используем для записи информации о выборе пользователя, на каком
состоянии он находится
}

func (chc ChatContext) ChatID() int {
	return chc.User.VkID
}
func (chc ChatContext) Get(VkID int, Field string) string { //получаем
информацию о пользователе(либо состояние, либо uuid)
	//в стрингу(входной параметр) будем писать нужный нам атрибут из БД,
возвращаем
	var err error
	chc.User, err = chc.RedisClient.GetUser(*chc.Ctx, VkID)
	if err != nil {
		log.Println("Failed to get record")
		log.Fatal(err)
	}
	if Field == "State" {
		return chc.User.State
	}
	if Field == "BranchUUID" {
		return chc.User.BranchUUID
	}
	if Field == "FacultyUUID" {
		return chc.User.FacultyUUID
	}
	if Field == "DepartmentUUID" {
		return chc.User.DepartmentUUID
	}
	if Field == "GroupUUID" {
		return chc.User.GroupUUID
	}
	if Field == "IsNumerator" {
		return strconv.FormatBool(chc.User.IsNumerator)
	}

	return "not found"
}
func (chc ChatContext) Set() { //записываем информацию в бд
	err := chc.RedisClient.SetUser(*chc.Ctx, *chc.User)
	if err != nil {
		log.WithError(err).Error("cant set user")
		return
	}
}

type State interface {
	Name() string                       //получаем название состояния в
виде строки, чтобы в дальнейшем куда-то записать(БД)
	Process(ChatContext, string) State //нужно взять контекст, посмотреть
на каком состоянии сейчас пользователь, метод должен вернуть состояние
}

//////////////////////////////////////////////////////
type StartState struct {
}

var RefStartState = &StartState{}

func (state StartState) Process(ctc ChatContext, messageText string) State {
	if messageText == "Узнать расписание" {
		b := params.NewMessagesSendBuilder()
		b.RandomID(0)
		b.Message("Укажи свою группу")
```

```
                b.PeerID(ctc.User.VkID)
                _, err := ctc.Vk.MessagesSend(b.Params)
                if err != nil {
                        log.Fatal(err)
                }
                return RefGroupState
        } else {
                b := params.NewMessagesSendBuilder()
                b.RandomID(0)
                b.PeerID(ctc.User.VkID)
                b.Message("Рады приветствовать тебя у нас в сообществе, давай
найдем твоё расписание!")
                k := &object.MessagesKeyboard{}
                k.AddRow()
                k.AddTextButton("Узнать расписание", "", "primary")
                b.Keyboard(k)
                _, err := ctc.Vk.MessagesSend(b.Params)
                if err != nil {
                        log.Fatal(err)
                }
                return RefStartState
        }
}

func (state StartState) Name() string {
        return "StartState"
}

//////////////////////////////////////////////////////
type BranchState struct {
}

var RefBranchState = &BranchState{}

func (state BranchState) Process(ctc ChatContext, messageText string) State {

        resp, _ := ctc.BitopClient.GetBranch(*ctc.Ctx, "messageText")
        if resp == nil {
                b := params.NewMessagesSendBuilder()
                b.RandomID(0)
                b.Message("Рады приветствовать тебя у нас в сообществе, давай
найдем твоё расписание!")
                k := &object.MessagesKeyboard{}
                k.AddRow()
                k.AddTextButton("Узнать расписание", "", "primary")
                b.Keyboard(k)
                b.RandomID(0)
                _, err := ctc.Vk.MessagesSend(b.Params)
                if err != nil {
                        log.Fatal(err)
                }
                return RefBranchState
        } else {

        }
        return RefFacultyState
}

func (state BranchState) Name() string {
        return "BranchState"
}

//////////////////////////////////////////////////////
type FacultyState struct {
```

```
}

var RefFacultyState = &FacultyState{}

func (state FacultyState) Process(ctx ChatContext, messageText string) State
{
	return RefDepartmentState
}

func (state FacultyState) Name() string {
	return "FacultyState"
}

//////////////////////////////////////////////////////////
type DepartmentState struct {
}

var RefDepartmentState = &DepartmentState{}

func (state DepartmentState) Process(ctx ChatContext, messageText string)
State {
	return RefGroupState
}
func (department DepartmentState) Name() string {
	return "DepartmentState"
}

//////////////////////////////////////////////////////////
type GroupState struct {
}

var RefGroupState = &GroupState{}

func (state GroupState) Process(ctc ChatContext, messageText string) State {
	ctc.BitopClient = bitop.New(*ctc.Ctx)
	resp, _ := ctc.BitopClient.GetGroup(*ctc.Ctx, messageText)
	if resp.Total > 1 {
		for _, group := range resp.Items {
			if group.Caption == strings.ToUpper(messageText) {
				ctc.User.GroupUUID = resp.Items[0].Uuid
				b := params.NewMessagesSendBuilder()
				b.PeerID(ctc.User.VkID)
				b.RandomID(0)
				b.Message("Выберите интересующую Вас неделю")
				k := &object.MessagesKeyboard{}
				k.AddRow()
				k.AddTextButton("Числитель", "", "primary")
				k.AddRow()
				k.AddTextButton("Знаменатель", "", "primary")
				b.Keyboard(k)
				_, err := ctc.Vk.MessagesSend(b.Params)
				if err != nil {
					log.Fatal(err)
				}
				return RefWeekState
			}
		}
		b := params.NewMessagesSendBuilder()
		b.PeerID(ctc.User.VkID)
		b.RandomID(0)
		b.Message("Введите полное название группы")
		_, err := ctc.Vk.MessagesSend(b.Params)
		if err != nil {
			log.Fatal(err)
```

```
                }
                return RefGroupState
        }
        if resp.Total == 1 {
                ctc.User.GroupUUID = resp.Items[0].Uuid
                b := params.NewMessagesSendBuilder()
                b.PeerID(ctc.User.VkID)
                b.RandomID(0)
                b.Message("Выберите интересующую Вас неделю")
                k := &object.MessagesKeyboard{}
                k.AddRow()
                k.AddTextButton("Числитель", "", "primary")
                k.AddRow()
                k.AddTextButton("Знаменатель", "", "primary")
                b.Keyboard(k)
                _, err := ctc.Vk.MessagesSend(b.Params)
                if err != nil {
                        log.Fatal(err)
                }
                return RefWeekState
        }
        b := params.NewMessagesSendBuilder()
        b.RandomID(0)
        b.PeerID(ctc.User.VkID)
        b.Message("Введите нужную группу повторно, не удалось найти")
        _, err := ctc.Vk.MessagesSend(b.Params)
        if err != nil {
                log.Fatal(err)
        }
        return RefGroupState
}

func (state GroupState) Name() string {
        return "GroupState"
}

//////////////////////////////////////////////////////
type WeekState struct {
}

var RefWeekState = &WeekState{}

func (state WeekState) Process(ctc ChatContext, messageText string) State {
        if messageText == "Числитель" {
                ctc.User.IsNumerator = true
                b := params.NewMessagesSendBuilder()
                b.PeerID(ctc.User.VkID)
                b.RandomID(0)
                b.Message("Выберите нужный день недели")
                k := &object.MessagesKeyboard{}
                k.AddRow()
                k.AddTextButton("Понедельник", "", "primary")
                k.AddRow()
                k.AddTextButton("Вторник", "", "primary")
                k.AddRow()
                k.AddTextButton("Среда", "", "primary")
                k.AddRow()
                k.AddTextButton("Четверг", "", "primary")
                k.AddRow()
                k.AddTextButton("Пятница", "", "primary")
                k.AddRow()
                k.AddTextButton("Суббота", "", "primary")
                b.Keyboard(k)
                _, err := ctc.Vk.MessagesSend(b.Params)
```

```
                if err != nil {
                        log.Fatal(err)
                }
                return RefDayState
        } else if messageText == "Знаменатель" {
                ctc.User.IsNumerator = false
                b := params.NewMessagesSendBuilder()
                b.PeerID(ctc.User.VkID)
                b.RandomID(0)
                b.Message("Выберите нужный день недели")
                k := &object.MessagesKeyboard{}
                k.AddRow()
                k.AddTextButton("Понедельник", "", "primary")
                k.AddRow()
                k.AddTextButton("Вторник", "", "primary")
                k.AddRow()
                k.AddTextButton("Среда", "", "primary")
                k.AddRow()
                k.AddTextButton("Четверг", "", "primary")
                k.AddRow()
                k.AddTextButton("Пятница", "", "primary")
                k.AddRow()
                k.AddTextButton("Суббота", "", "primary")
                b.Keyboard(k)
                _, err := ctc.Vk.MessagesSend(b.Params)
                if err != nil {
                        log.Fatal(err)
                }
                return RefDayState
        } else {
                b := params.NewMessagesSendBuilder()
                b.PeerID(ctc.User.VkID)
                b.RandomID(0)
                b.Message("Выберите интересующую Вас неделю")
                k := &object.MessagesKeyboard{}
                k.AddRow()
                k.AddTextButton("Числитель", "", "primary")
                k.AddRow()
                k.AddTextButton("Знаменатель", "", "primary")
                b.Keyboard(k)
                _, err := ctc.Vk.MessagesSend(b.Params)
                if err != nil {
                        log.Fatal(err)
                }
                return RefWeekState
        }
}

func (state WeekState) Name() string {
        return "WeekState"
}

//////////////////////////////////////////////////////

type NextWeekState struct {
}

var RefNextWeekState = &WeekState{}

func (state NextWeekState) Process(ctc ChatContext, messageText string) State
{
        return NextWeekState{}
}
```

```
func (state NextWeekState) Name() string {
	return "NextWeekState"
}

//////////////////////////////////////////////////////

type DayState struct {
}

var RefDayState = &DayState{}

func (state DayState) Process(ctc ChatContext, messageText string) State {
	var v string
	if (messageText == "Понедельник") || (messageText == "Вторник") ||
(messageText == "Среда") || (messageText == "Четверг") || (messageText ==
"Пятница") || (messageText == "Суббота") {
		fmt.Print("Я здесь")
		ctc.BitopClient = bitop.New(*ctc.Ctx)
		Schedule, err := ctc.BitopClient.GetSchedule(*ctc.Ctx,
ctc.User.GroupUUID, ctc.User.IsNumerator, messageText)
		if err != nil {
			log.WithError(err).Error("failed to get schedule")
		}
		if Schedule == nil {
			v := "В этот день Вы отдыхаете!"
			b := params.NewMessagesSendBuilder()
			b.PeerID(ctc.User.VkID)
			b.RandomID(0)
			k := &object.MessagesKeyboard{}
			k.AddRow()
			k.AddTextButton("Сброс", "", "primary")
			k.AddRow()
			k.AddTextButton("Вернуться к выбору дня", "",
"primary")
			k.AddRow()
			k.AddTextButton("Вернуться к выбору недели", "",
"primary")
			b.Message(v)
			b.Keyboard(k)
			_, err := ctc.Vk.MessagesSend(b.Params)
			if err != nil {
				log.Fatal(err)
			}
			return RefDayState
		}
		var lessons []model.Lesson
		var less model.Lesson
		var teach model.Teacher
		var teachs model.Teachers
		for _, lesson := range Schedule.Lessons {
			less.Name = lesson.Name
			less.Cabinet = lesson.Cabinet
			less.Type = lesson.Type
			for _, teacher := range lesson.Teachers {
				teach.Name = teacher.Name
				teachs = append(teachs, teach)
			}
			less.Teachers = teachs
			teachs = nil
			less.StartAt = lesson.StartAt
			less.EndAt = lesson.EndAt
			less.Day = lesson.Day
			less.IsNumerator = lesson.IsNumerator
			lessons = append(lessons, less)
```

```
		}
		lessons = quickSort(&lessons)
		switch messageText {
		case "Понедельник":
			{
				v = "Ваше расписание на понедельник:\n\n"
			}
		case "Вторник":
			{
				v = "Ваше расписание на вторник:\n\n"
			}
		case "Среда":
			{
				v = "Ваше расписание на среду:\n\n"
			}
		case "Четверг":
			{
				v = "Ваше расписание на четверг:\n\n"
			}
		case "Пятница":
			{
				v = "Ваше расписание на пятницу:\n\n"
			}
		case "Суббота":
			{
				v = "Ваше расписание на субботу:\n\n"
			}
		}
		for _, lesson := range lessons {
			v += ("\t\tПредмет: " + lesson.Name + "\n\n")
			v += ("\t\t\t\tВремя начала пары: " +
lesson.StartAt[0:5] + "\n")
			v += ("\t\t\t\tВремя окончания пары: " +
lesson.EndAt[0:5] + "\n")
			if (lesson.Type) != "" {
				v += ("\t\t\t\tТип занятия: " + lesson.Type +
"\n")
			}
			for _, teacher := range lesson.Teachers {
				v += ("\t\t\t\tПреподаватель: " + teacher.Name
+ "\n")
			}
			if (lesson.Cabinet) != "" {
				v += ("\t\t\t\tКабинет: " + lesson.Cabinet + "\
n")
			}
			v += "\n\n"
		}
		b := params.NewMessagesSendBuilder()
		b.PeerID(ctc.User.VkID)
		b.RandomID(0)
		b.Message(v)
		k := &object.MessagesKeyboard{}
		k.AddRow()
		k.AddTextButton("Сброс", "", "primary")
		k.AddRow()
		k.AddTextButton("Вернуться к выбору дня", "", "primary")
		k.AddRow()
		k.AddTextButton("Вернуться к выбору недели", "", "primary")
		b.Keyboard(k)
		_, err = ctc.Vk.MessagesSend(b.Params)
		if err != nil {
			log.Fatal(err)
		}
```

```
		return RefDayState
	} else if messageText == "Сброс" {
		b := params.NewMessagesSendBuilder()
		b.PeerID(ctc.User.VkID)
		b.RandomID(0)
		k := &object.MessagesKeyboard{}
		k.AddRow()
		k.AddTextButton("Узнать расписание", "", "primary")
		b.Keyboard(k)
		_, err := ctc.Vk.MessagesSend(b.Params)
		if err != nil {
			log.Fatal(err)
		}
		return RefStartState
	} else if messageText == "Вернуться к выбору дня" {
		b := params.NewMessagesSendBuilder()
		b.PeerID(ctc.User.VkID)
		b.RandomID(0)
		b.Message("Выберите нужный день недели")
		k := &object.MessagesKeyboard{}
		k.AddRow()
		k.AddTextButton("Понедельник", "", "primary")
		k.AddRow()
		k.AddTextButton("Вторник", "", "primary")
		k.AddRow()
		k.AddTextButton("Среда", "", "primary")
		k.AddRow()
		k.AddTextButton("Четверг", "", "primary")
		k.AddRow()
		k.AddTextButton("Пятница", "", "primary")
		k.AddRow()
		k.AddTextButton("Суббота", "", "primary")
		b.Keyboard(k)
		_, err := ctc.Vk.MessagesSend(b.Params)
		if err != nil {
			log.Fatal(err)
		}
		return RefDayState
	} else if messageText == "Вернуться к выбору недели" {
		b := params.NewMessagesSendBuilder()
		b.PeerID(ctc.User.VkID)
		b.RandomID(0)
		b.Message("Выберите интересующую Вас неделю")
		k := &object.MessagesKeyboard{}
		k.AddRow()
		k.AddTextButton("Числитель", "", "primary")
		k.AddRow()
		k.AddTextButton("Знаменатель", "", "primary")
		b.Keyboard(k)
		_, err := ctc.Vk.MessagesSend(b.Params)
		if err != nil {
			log.Fatal(err)
		}
		return RefWeekState
	} else {
		b := params.NewMessagesSendBuilder()
		v := "Проверьте правильность ввода введенного учебного дня"
		b.PeerID(ctc.User.VkID)
		b.RandomID(0)
		b.Message(v)
		k := &object.MessagesKeyboard{}
		k.AddRow()
		k.AddTextButton("Сброс", "", "primary")
		k.AddRow()
```

```
                k.AddTextButton("Вернуться к выбору дня", "", "primary")
                b.Keyboard(k)
                _, err := ctc.Vk.MessagesSend(b.Params)
                if err != nil {
                        log.Fatal(err)
                }
                return RefDayState
        }
}

func (state DayState) Name() string {
        return "DayState"
}

//////////////////////////////////////////////////////////

type ErrorState struct {
}

var RefErrorState = &ErrorState{}

func (state ErrorState) Process(ctx ChatContext, messageText string) State {
        return RefStartState
}

func (state ErrorState) Name() string {
        return "ErrorState"
}

//////////////////////////////////////////////////////////

func quickSort(lessons *[]model.Lesson) []model.Lesson {
        var lessonl, lessone, lessonm []model.Lesson
        if (len(*lessons) == 1) || (len(*lessons) == 0) {
                return *lessons
        }
        randomTime := (*lessons)[0].StartAt
        randomTimetime, _ := time.Parse("15:04:05", randomTime)
        fmt.Println(randomTimetime)
        for _, lesson := range *lessons {
                Timetime, _ := time.Parse("15:04:05", lesson.StartAt)
                fmt.Println(Timetime)
                if Timetime.Before(randomTimetime) { //если ли Timetime раньше
randomTimetime
                        lessonl = append(lessonl, lesson)
                } else if Timetime.After(randomTimetime) {
                        lessonm = append(lessonm, lesson)
                } else {
                        lessone = append(lessone, lesson)
                }
        }
        finalLessonsl := quickSort(&lessonl)
        for _, lesson := range lessone {
                finalLessonsl = append(finalLessonsl, lesson)
        }
        finalLessonsm := quickSort(&lessonm)
        for _, lesson := range finalLessonsm {
                finalLessonsl = append(finalLessonsl, lesson)
        }
        return finalLessonsl
}
```