МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

ЗВІТ

Розробка чат-боту.

Виконав:
студент 2 курсу
групи КІ-32
Ільїн Сергій
Анатолійович
Перевірив:
Терентьєв О. М.

Київ – 2025

Загальний опис чат-бота

Назва: Linus Friends

Тег в телеграм: @Linus_friends_bot

Призначення: Телеграм чат бот для знайомств, де кожна людина, яка має досвід в IT та не тільки, може зареєструватись та знайти людину, яка володіє такими ж навичками, як і вона.

Предметна область: Допомога розробниками будувати нові соціальні зв’язки, підбирати людей на проєкт, допомога найму.

Перелік основних впроваджених правил

Представимо основні впроваджені правила у формі блок-схеми.

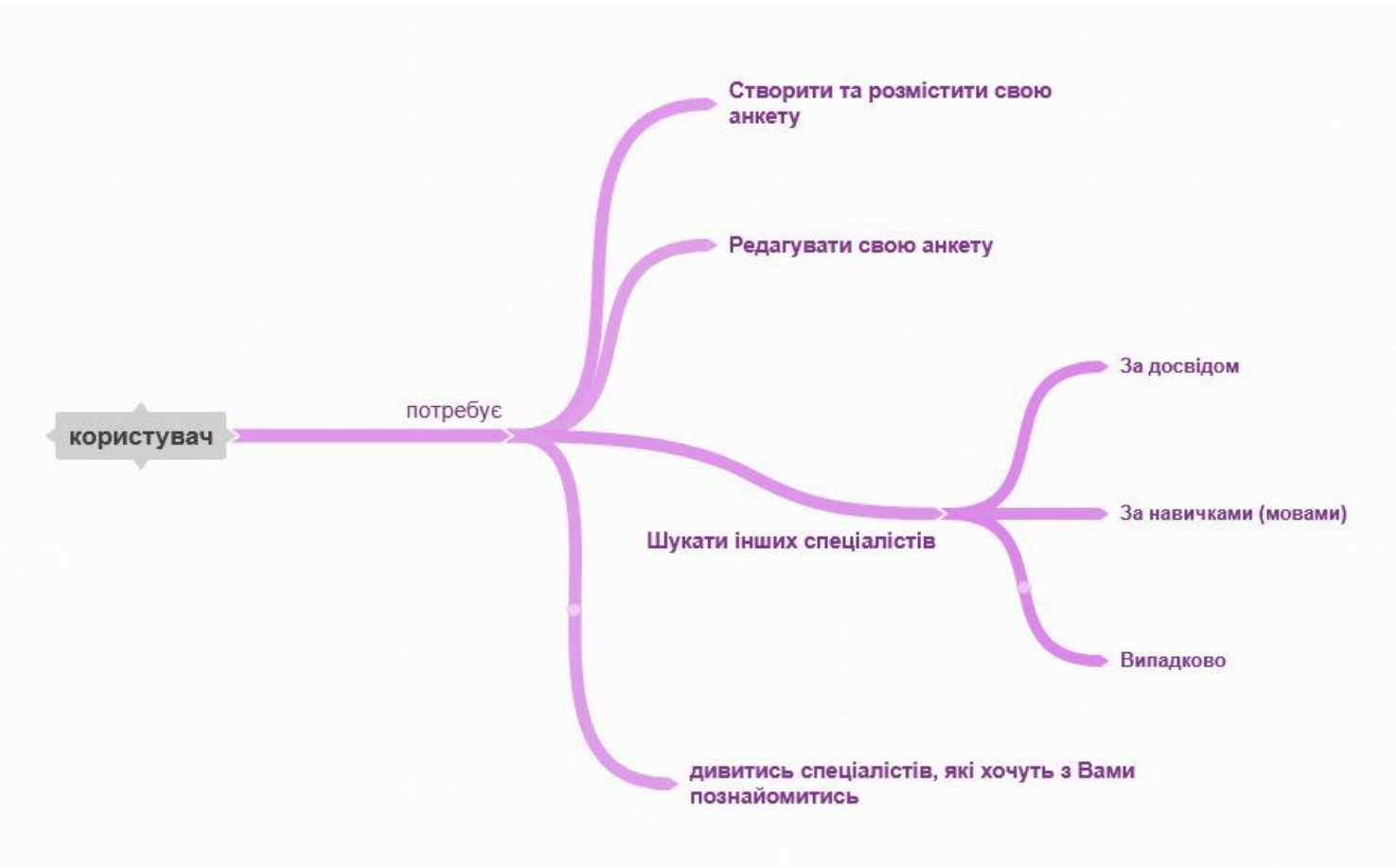


Рисунок 1 — Візуальне представлення можливостей бота

З рисунку 1 видно, що бот має всі основі функції бота для знайомств, представимо їх в вигляді впорядкованого списку, де номер пункту умовно відповідає за порядок виконання функцій користувачем впродовж використання бота:

1) Створення анкети, виконується лише один раз при першому запуску бота (див. рисунок 2).
2) Вибір відповідної функції, яку ми хочемо використати. З рисунка 3 видно, що ми можемо вибрати пункти «Шукати спеціалістів», «подивитись мій профіль» (в подальшому його змінити, див. рисунок), «показати спеціалістів, які зацікавлені в знайомстві з Вами».
3) При натисканні на 1, «Шукати спеціалістів», ми маємо вибір (див. рис. 4) шукати за досвідом, шукати за навичками, шукати навмання та вийти до головного меню. На рисунку 5 видно приклад пошуку спеціалістів, а на рисунку 6 видно як при цьому пошуку може прийти сповіщення про «лайк».
4) Натиснувши на 2, користувач опиняється в редагуванні свого профілю (див. рис. 7). Тут він може змінити своє фото, ім’я, роки досвіду, текст анкети, редагувати свої навички та додати нові. Приклад зміни навичок анкети представлений на рисунку 8.
5) Натиснувши на 3 (див. рис. 9), користувач може подивитись на бажаючих познайомитись, прийняти або ігнорувати запит, а також може зупинити перегляд та повернутись до головного меню.

Результат роботи бота представлені на рис. 2-9:

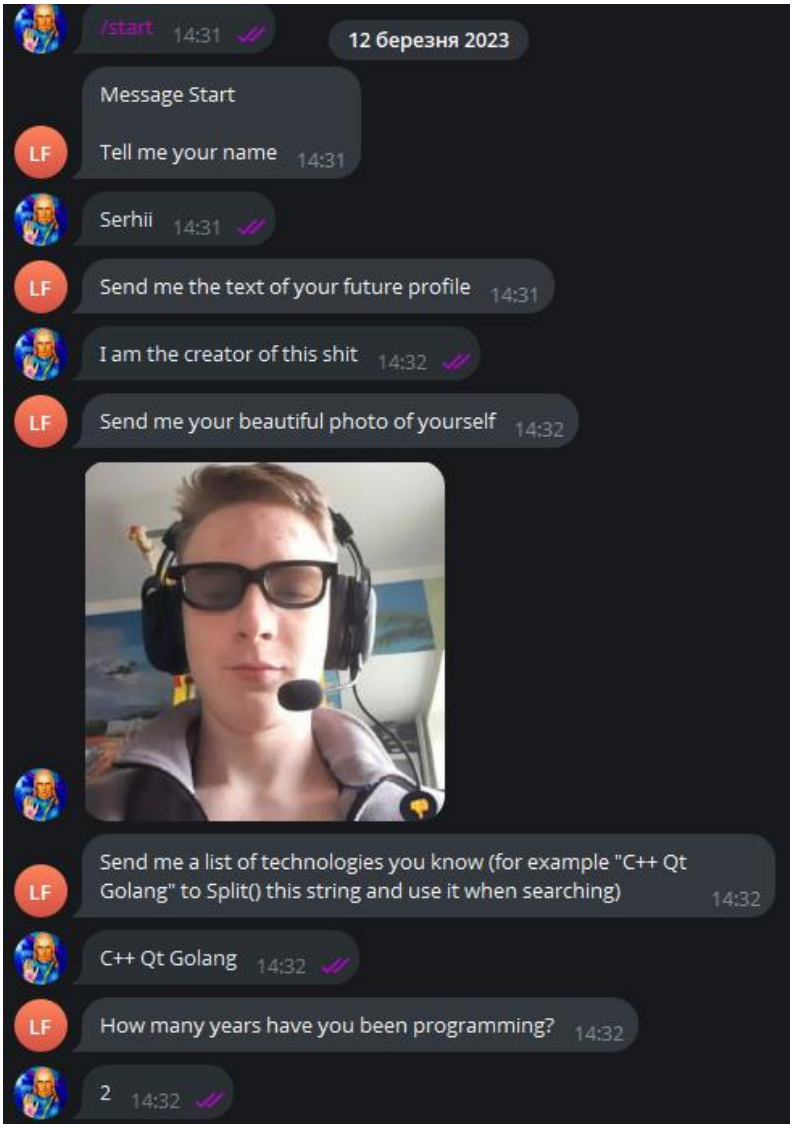


Рисунок 2 — Створення анкети на початку користування бота.

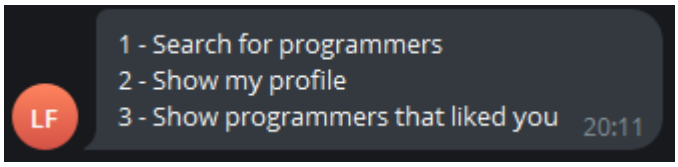


Рисунок 3 — головне меню.

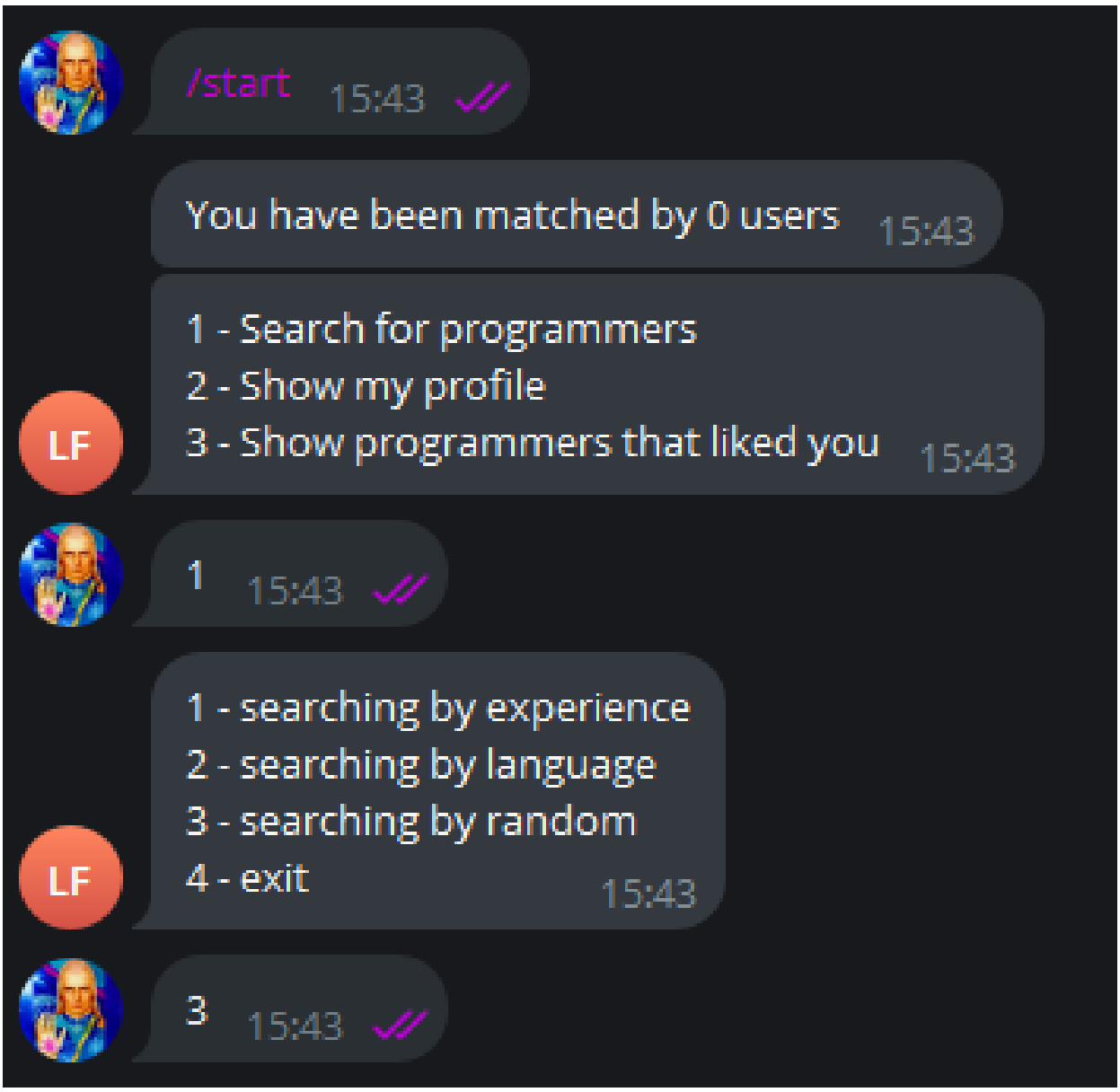
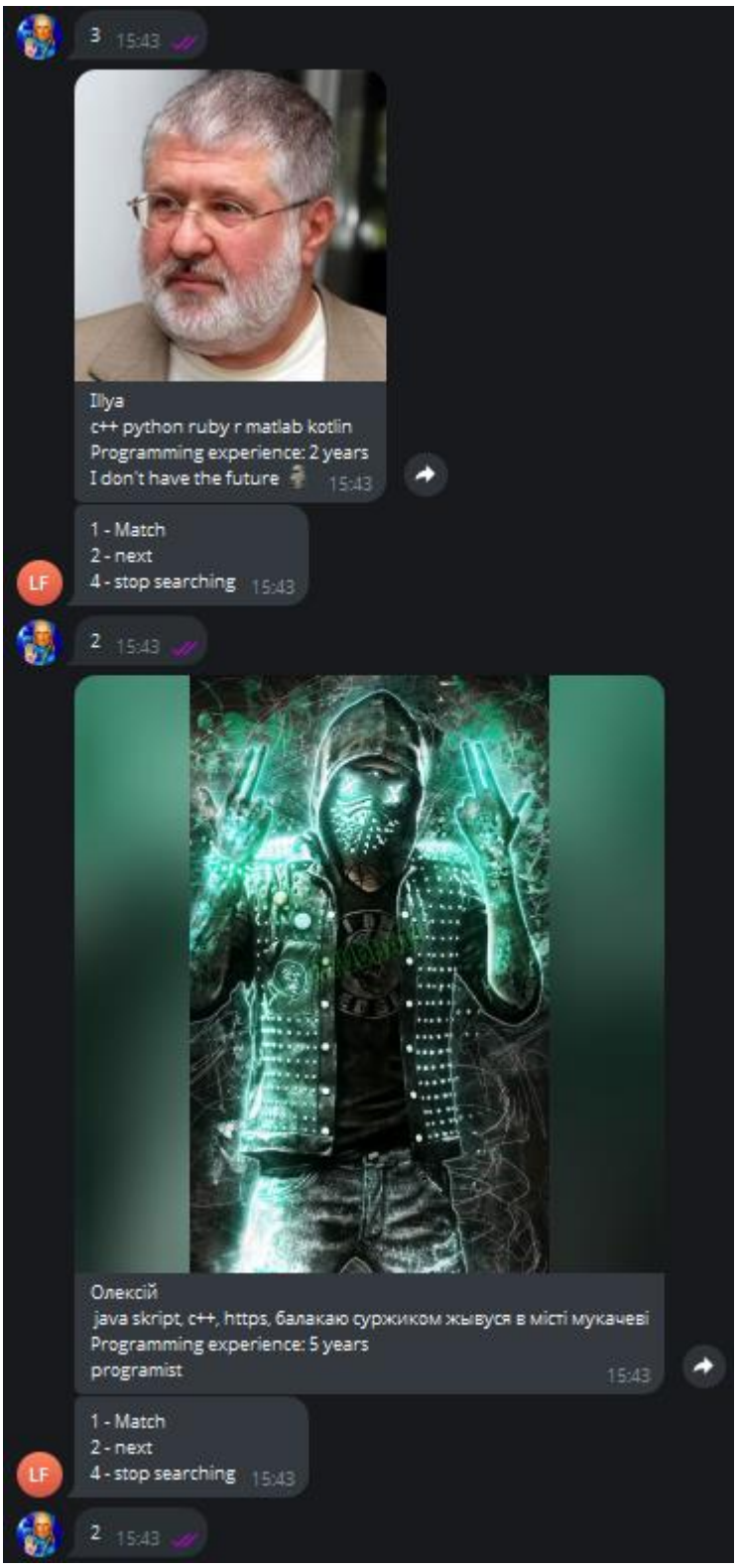


Рисунок 4 — меню «Шукати спеціалістів» та приклад його використання.



Рисунки 5 –приклад пошуку спеціалістів.

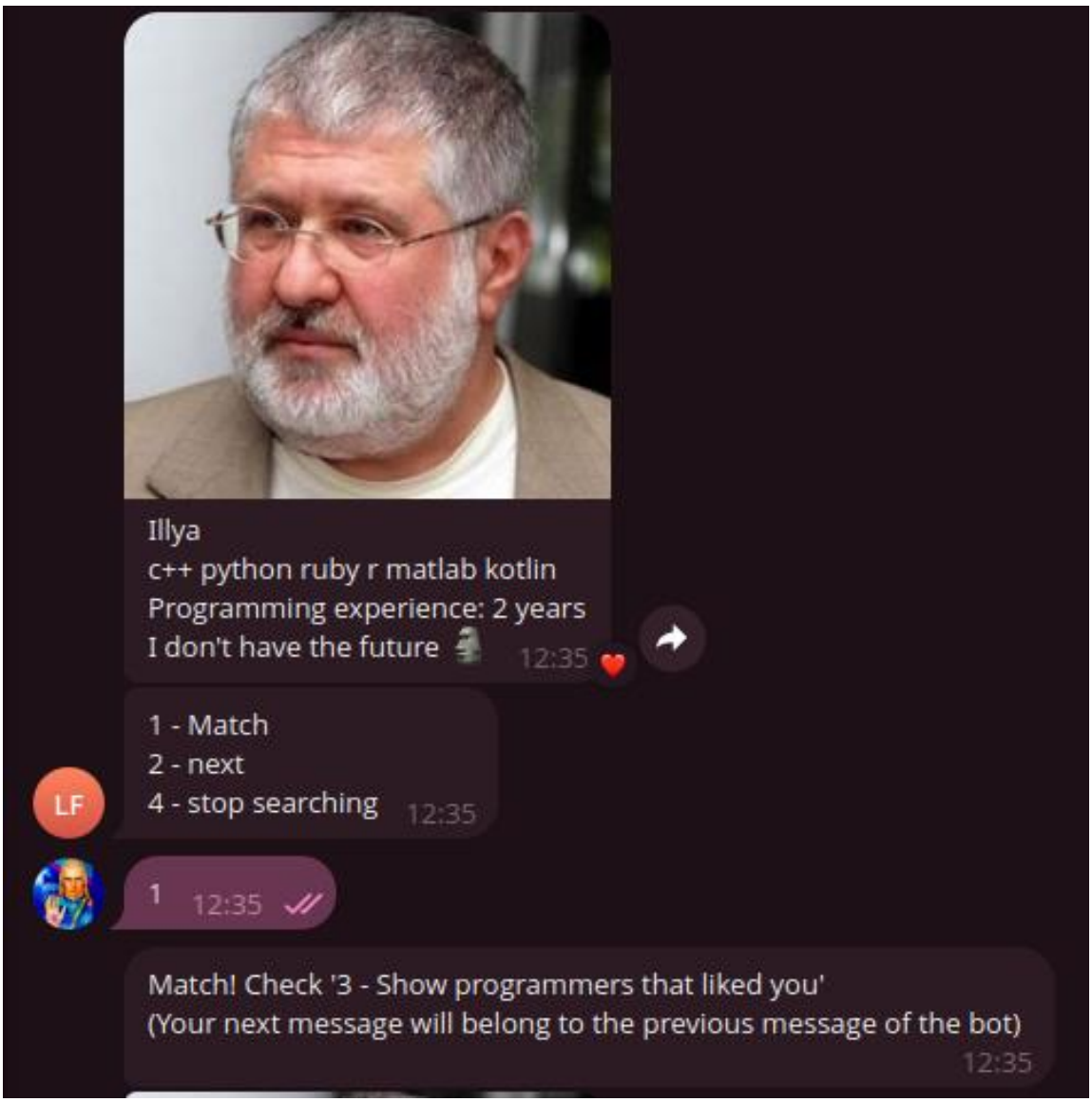


Рисунок 6 - приклад пошуку, де може прийти сповіщення про «лайк».

Рисунок 7 — меню редагування свого профілю.

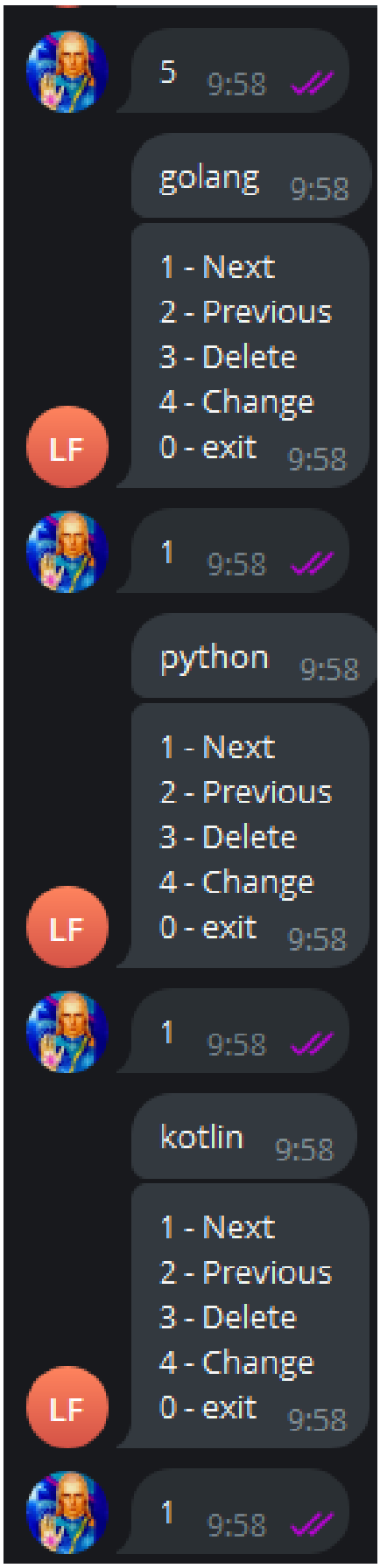


Рисунок 8 – приклад меню зміни навичок в анкеті.

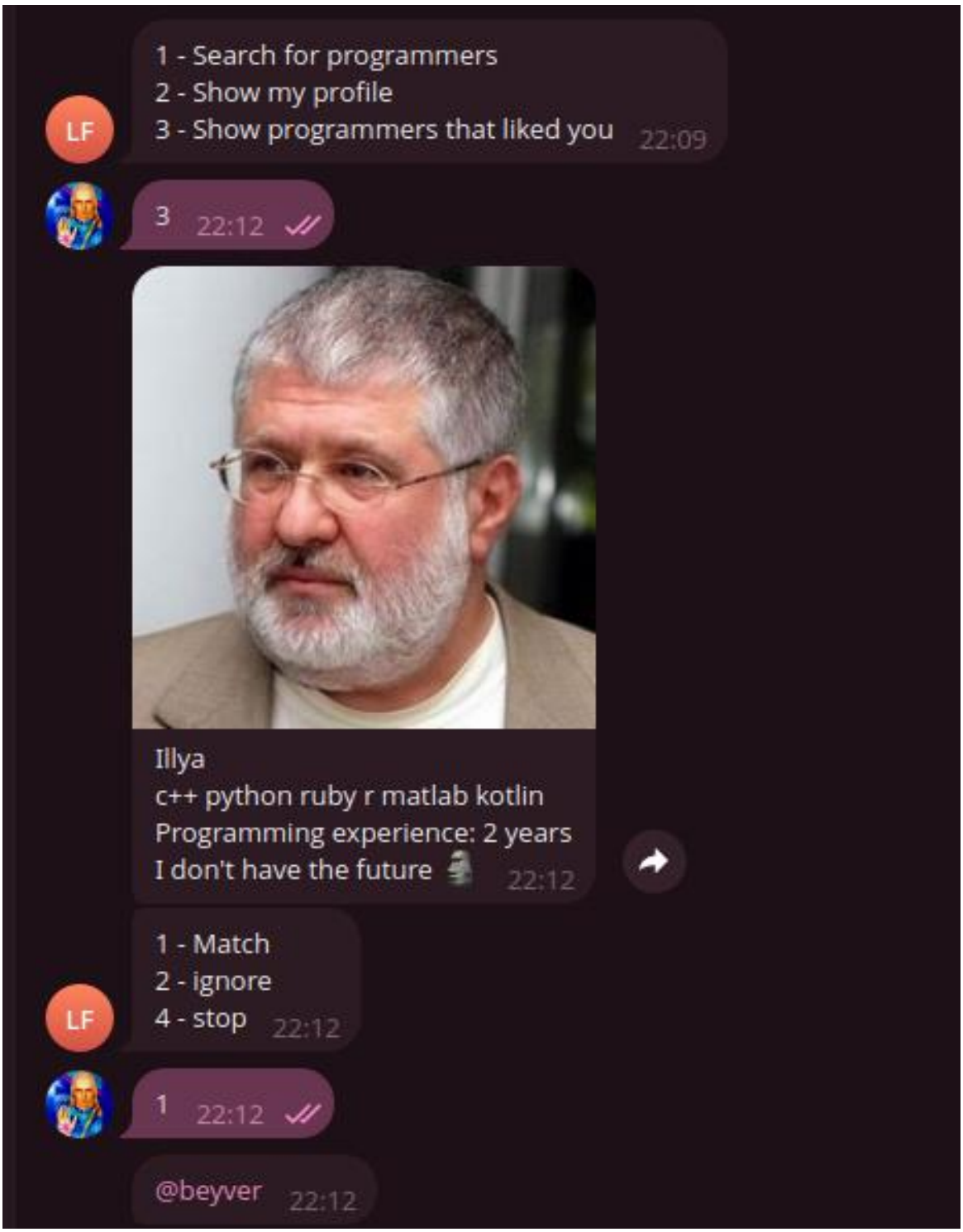


Рисунок 9 – приклад меню «подивитись на спеціалістів, які лайнули вашу анкету».

Тест Тюрінга

Через те, що цей бот саме для знайомств, йому не обов’язково мати функціонал живого спілкування з користувачем. Тому, при спробі користувача поговорити з ботом по людські, бот буде нагадувати, що він створений для пошуку спеціалістів, а не для балачок.

Технічні аспекти

Весь код проєкту написаний на мові програмування Go, використовуючи як основний засіб спілкування з API телеграма бібліотеку tgbotapi версії 5. Для зберігання користувачів використовувалась база даних SQLite.

Код з репозиторію [1] мого гітхаб акаунту [2]

Структура проєкту в вигляді файлового каталогу представлена на рисунку 10.

```
Sergey@DESKTOP-JOUIM9D MINGW64 ~/Desktop/Linus-friends-tg-bot (main)
$ ls -R
.:
LICENSE.md  README.md       db/     go.sum  main.go      storage/
LinusUser/  advertisement/  go.mod  libs/   processing/

./LinusUser:
user.go

./advertisement:
advertisement.go

./db:
storage.db   storage3.db       storageForTesting.db
storage1.db  storageForAds.db

./libs:
e/

./libs/e:
e.go

./processing:
messages.go              processCMDStart.go     processShowProfile.go
messagesForAdmin.go      processPostAdvert.go   processing.go
processAdmin.go          processSearch.go
processAdvertisement.go  processShowMatches.go

./storage:
sqlite/  storage.go

./storage/sqlite:
database.go  databaseAds.go
```

Рисунок 10 – представлення проєкту в вигляді файлового каталогу.

Розпишемо по пунктам яка папка за що відповідає:

- Корінна папка відповідає за збереження всього проєкту, тут можна знайти точку запуску проєкту файл main.go, конфігураційні golang файли go.sum та go.mod, ліцензію та readme.
- Папка LinusUser зберігає в собі файл user.go, який в собі зберігає тільки структуру з даними користувача.
- Папка advertisement зберігає в собі файл advertisement.go, в якому прописана структура даних реклами.

- Папка db зберігає службові файли бази даних SQLite.
- Папка Libs/e зберігає в собі файл e.go, який зберігає в собі функціонал зручного форматування помилок.
- Папка storage зберігає в собі папку sqlite та файл storage.go, який описує інтерфейс роботи з базою даних (не обов'язково SQLite) та константи вибору пошуку спеціалістів.
- Папка processing створена для опису головного функціоналу бота, зберігає в собі файли messages.go (константи з текстом можливих повідомлень бота), processCMDStart.go (логіка виконання команди "/start"), processShowProfile.go (логіка меню редагування профілю), messagesForAdmin.go (константи з текстом можливих повідомлень бота для адміністратора), processPostAdvert.go (логіка додавання реклами), processing.go (файл опису та логіки самого класу processing, з головними методами по типу реалізації головного меню), processAdmin.go (логіка спілкування бота з адміністратором), processSearch.go (логіка пошуку спеціалістів за певними критеріями), processAdvertisement.go (логіка маніпуляцій та показів реклами), processShowMatches.go (логіка показу та маніпуляції з бажаючими познайомитись).

Варто зазначити, що кожна папка – це пакет, вся структура проекту реалізує логіку створення структури пакетів за правилами мови програмування golang. Код проєкту представлений в додатку А.

Пропозиції щодо подальших покращень

Ось кілька можливих покращень для чат-бота Linus friends:

1. Розширення тематики та відповідного функціоналу, а саме створити області, в яких можуть бути спеціалісти (тобто не тільки програмісти, а ще й інженери, вчені, так далі); створити відповідні категорії для простих знайомств, проєктної роботи, найму; зробити характеризацію акаунту по типах спеціаліста або рекрутера, додати їм відповідний функціонал, тощо.

2. Вдосконалити реалізацію адміністрування бота (створити адекватну CRM-систему), зробити відповідну програму або веб-сторінку, створення функціоналу висвітлювання статистики користування ботом.
3. Рефакторинг коду.
4. Оптимізація бази даних та SQL запитів.
5. Використання більш професійної бази даних, як Postgress.
6. Додання більшої кількості декоративних деталей: аватарки, текстових прикрас, можливість додати до свого профілю не одну картинку, а принаймні 3, інше.
7. Реліз проєкту для загального користування та реклама бота в соціальних мережах.

Висновок:

Отже, в результаті виконання цієї лабораторії роботи було створено бота за допомогою мови програмування golang та бази даних SQLite, який виконує роль платформи для знайомств спеціалістів.

За допомогою бібліотеки tgbotapi версії 5 для мови програмування golang була успішна реалізація діалогу між сервером бота та API телеграму.

У процесі розробки було проведено кілька тренувальних сесій за допомогою чотирьох учасників, у результаті яких була продемонстрована коректна робота бота.

Отже, “Linus friends” - це не просто чат бот для знайомств, це рушій спільноти, який породжує нові знайомства, які в свою чергу роблять нові круті звершення в вигляді інноваційних стартапів, допомога між собою, навчання інших людей і так далі. Також це комерційно вдалий проєкт, так як він передбачає розміщення реклами та навіть пошук рекрутерами претендентів на робочі посади.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Посилання на репозиторій проєкту:
   https://github.com/njnjfnj/Linus-friends-tg-bot
2) Посилання на мій гітхаб акаунт: https://github.com/njnjfnj

Додаток А

Код відповідних файлів.

main.go:

```
package main

import (
	"context"
	"flag"
	"log"
	"strings"

	"LinusFriends/advertisement"
	"LinusFriends/processing"
	Database "LinusFriends/storage/sqlite"

	tgbotapi "github.com/go-telegram-bot-api/telegram-bot-api/v5"
)

func main() {
	//getting token
	token := flag.String(
		"token",
		"",
		"token for access to tg bot",
	)
	adminPassword := flag.String(
		"pswrd",
		"",
		"password for administrator",
	)
```

```
	adminChoice := flag.String(
		"adminChoice",
		"",
		"value that is going to be a choice value (like 1 - Search for programmers, 2 - show my profile, 3 - show matches)",
	)

	flag.Parse()

	if *token == "" {
		log.Fatal("token is not specified")
	}
	if *adminPassword == "" {
		log.Fatal("admin password is not specified")
	}
	if *adminChoice == "" {
		log.Fatal("admin choice is not specified")
	}

	// creating db
	db, err := Database.NewDatabase("db/storageForTesting.db")
	if err != nil {
		log.Fatal("Can not make a db: ", err)
	}
	if err := db.Init(context.TODO()); err != nil {
		log.Fatal("Can not init a db: ", err)
	}

	// bot init
	bot, err := tgbotapi.NewBotAPI(*token) // *token
```

```
	if err != nil {
		log.Panic(err)
	}

	bot.Debug = true
	log.Printf("Authorized on account %s", bot.Self.UserName)

	u := tgbotapi.NewUpdate(0)
	u.Timeout = 60

	UsersInProcess := make(map[int]chan tgbotapi.Update)

	// creating processing object
	// cheking updates
	updates := bot.GetUpdatesChan(u)
	process := processing.NewProcessing(bot, db, *adminPassword, *adminChoice,
make(chan advertisement.Ad))
	for upd := range updates {
		go func(update tgbotapi.Update) {
			if update.Message != nil {
				chat_id := update.FromChat().ChatConfig().ChatID
				if UsersInProcess[int(chat_id)] == nil {
					if len(update.Message.Text) != 0 &&
update.Message.Text[0] == '/' {
						UsersInProcess[int(chat_id)] = make(chan
tgbotapi.Update)
						if err :=
process.CMD(strings.ReplaceAll(update.Message.Text, " ", ""), chat_id,
UsersInProcess[int(chat_id)]); err != nil {
							log.Println("Can not process update: ", err)
						}
```

```
                                close(UsersInProcess[int(chat_id)])
                                UsersInProcess[int(chat_id)] = nil
                        } else {
                                bot.Send(tgbotapi.NewMessage(chat_id, "Unknown
message"))
                                return
                        }
                } else {
                        UsersInProcess[int(chat_id)] <- upd
                }
        }
    }(upd)
  }
}
```

LinusUser/user.go:

```
package LinusUser

type User struct {
        ChatID          int
        Name            string
        Description     string
        SkillsString    string
        YearsOfProgramming int
        Image           []byte
}
```

Advertisement/advertisement.go:

```
package advertisement

type Ad struct {
	Advert_id   int
	Content     []byte
	Description string
	Seen        int
	Rated       int
	Rate        int
}
```

Libs/e/e.go:

```
package e

import "fmt"

func Wrap(msg string, err error) error {
	return fmt.Errorf("%s: %w", msg, err)
}

func WrapIfErr(msg string, err error) error {
	if err != nil {
		return Wrap(msg, err)
	}
	return nil
}
```

Storage/ storage.go:

```
package storage

import (
	"LinusFriends/LinusUser"
	"LinusFriends/advertisement"
	"errors"
)

var ErrNoFriends = errors.New("no friends :(")
var ErrNoAds = errors.New("no adverts")

const (
	SearchingByExperience = iota
	SearchingByRandom
	SearchingByLanguage
)

type Storage interface {
	UserCount() (int, error)
	AddNewUser(u LinusUser.User) error
	DeleteUser(chat_id int) error
	IsUserExists(chat_id int) (bool, error)
	GetUser(chat_id int) (LinusUser.User, error)
	UpdateUser(u LinusUser.User) error
	GetRandomUserForUser(chat_id int64, SearchingByWhat int, user LinusUser.User)
(LinusUser.User, string, error)
	DeleteSkill(chat_id int, skill string) error
	AddSkill(chat_id int, skill string) error

	AddMatch(chat_id int64, user LinusUser.User) error
```

```
	GetMatches(chat_id int64) (string, error)
	SetMatches(chat_id int64, matchesLeft string) error

	AddNewAd(ad advertisement.Ad) error
	DeleteAd(advert_id int) error
	UpdateAdContent(content []byte, description string, advert_id int) error
	UpdateAdRatingAndViews(rating int, seen int, rated int, advert_id int) error
	GetAd(advert_id int) (advertisement.Ad, error)
	GetAds() ([]advertisement.Ad, error)
	GetAdsIds() ([]int, error)
	IsAdExists(advert_id int) (bool, error)
}
```

Storage/ sqlite/database.go:

```
package sqlite

import (
	"LinusFriends/LinusUser"
	"LinusFriends/libs/e"
	"LinusFriends/storage"
	"context"
	"database/sql"
	"strconv"
	"strings"

	_ "github.com/mattn/go-sqlite3"
)

type Database struct {
	db    *sql.DB
	cntxt context.Context
```

```
}

func NewDatabase(path string) (*Database, error) {
	DB, err := sql.Open("sqlite3", path)
	if err != nil {
		return nil, e.Wrap("can not open DB", err)
	}
	if err := DB.Ping(); err != nil {
		return nil, e.Wrap("can not connect to DB", err)
	}
	return &Database{db: DB}, nil
}

func (d *Database) Init(context context.Context) error {
	q := `CREATE TABLE IF NOT EXISTS db (chat_id INT NOT NULL, name TEXT, description TEXT, skillsString TEXT NOT NULL, years_of_programming INT NOT NULL, photo BLOB NOT NULL); CREATE TABLE IF NOT EXISTS skls (language TEXT NOT NULL, IDs TEXT); CREATE TABLE IF NOT EXISTS mtchs (chat_id INT NOT NULL, IDs TEXT NOT NULL); CREATE UNIQUE INDEX IF NOT EXISTS idx_id ON db (chat_id); CREATE UNIQUE INDEX IF NOT EXISTS idx_lng ON skls (language); CREATE UNIQUE INDEX IF NOT EXISTS idx_id ON mtchs (chat_id);
	CREATE TABLE IF NOT EXISTS ads(advert_id INT NOT NULL, content BLOB NOT NULL, rating INT NOT NULL, seen INT, rated INT, description TEXT); CREATE UNIQUE INDEX IF NOT EXISTS idx_id ON ads (advert_id);`
	if _, err := d.db.ExecContext(context, q); err != nil {
		return e.Wrap("Can not create DB", err)
	}
	d.cntxt = context

	return nil
}
```

```
func (d *Database) AddNewUser(u LinusUser.User) (err error) {
	defer func() { err = e.WrapIfErr("Can not add new user", err) }()

	q5 := `INSERT INTO mtchs (chat_id, IDs) VALUES(?, ?)`
	if _, err := d.db.ExecContext(d.cntxt, q5, u.ChatID, ""); err != nil {
		return e.Wrap("Can not add to matches db", err)
	}

	q2 := `SELECT COUNT(*) FROM skls WHERE language = ?`
	q3 := `UPDATE skls SET IDs = IDs || ' ' || ? WHERE language = ?`
	q4 := `INSERT INTO skls (language, IDs) VALUES(?, ?)`
	var res int
	for _, i := range strings.Split(u.SkillsString, " ") {
		if err := d.db.QueryRowContext(d.cntxt, q2, i).Scan(&res); err != nil {
			return e.Wrap("Can not check if language exists", err)
		} else if res > 0 {
			if _, err := d.db.ExecContext(d.cntxt, q3, strconv.Itoa(u.ChatID), i); err
!= nil {
				return e.Wrap("Can not update skills IDs", err)
			}
		} else if _, err := d.db.ExecContext(d.cntxt, q4, i, strconv.Itoa(u.ChatID)); err
!= nil {
			return e.Wrap("Can not add new language", err)
		}

	}

	q1 := `INSERT INTO db (chat_id, name, description, skillsString,
years_of_programming, photo) VALUES(?, ?, ?, ?, ?, ?)`
	if _, err := d.db.ExecContext(d.cntxt, q1,
```

```
            u.ChatID,
            u.Name,
            u.Description,
            u.SkillsString,
            u.YearsOfProgramming,
            u.Image); err != nil {
            return err
        }

        return nil
}

func (d *Database) DeleteUser(chat_id int) error {
        q := `DELETE FROM db WHERE chat_id = ?`
        if _, err := d.db.ExecContext(d.cntxt, q, chat_id); err != nil {
            return e.Wrap("Can not delete user", err)
        }
        return nil
}

func (d *Database) IsUserExists(chat_id int) (bool, error) {
        q := `SELECT COUNT(*) FROM db WHERE chat_id = ?`
        var res int

        if err := d.db.QueryRowContext(d.cntxt, q, chat_id).Scan(&res); err != nil {
            return false, e.Wrap("Can not check if user exists", err)
        }
        return res > 0, nil
}

func (d *Database) GetUser(chat_id int) (LinusUser.User, error) {
```

```
	var res LinusUser.User
	q := `SELECT * FROM db WHERE chat_id = ?`
	err := d.db.QueryRowContext(d.cntxt, q, chat_id).Scan(
		&res.ChatID,
		&res.Name,
		&res.Description,
		&res.SkillsString,
		&res.YearsOfProgramming,
		&res.Image)

	if err == sql.ErrNoRows {
		return LinusUser.User{}, storage.ErrNoFriends
	}
	if err != nil {
		return LinusUser.User{}, e.Wrap("Can not get user", err)
	}

	return res, nil
}

func (d *Database) UpdateUser(u LinusUser.User) error {
	q := `UPDATE db SET name = ?, description = ?, skillsString = ?,
years_of_programming = ?, photo = ? WHERE chat_id = ?`
	if _, err := d.db.ExecContext(d.cntxt, q,
		u.Name,
		u.Description,
		u.SkillsString,
		u.YearsOfProgramming,
		u.Image,
		u.ChatID); err != nil {
		return e.Wrap("Can not update user", err)
```

```
	}
	return nil
}

func (d *Database) GetRandomUserForUser(chat_id int64, SearchByWhat int, user
LinusUser.User) (LinusUser.User, string, error) {
	var res LinusUser.User
	var ids = ""
	var err error
	switch SearchByWhat {
	case storage.SearchingByExperience:
		q := `SELECT * FROM db WHERE years_of_programming = ? EXCEPT
SELECT * FROM db WHERE chat_id = ?`
		err = d.db.QueryRowContext(d.cntxt, q, user.YearsOfProgramming,
chat_id).Scan(
			&res.ChatID,
			&res.Name,
			&res.Description,
			&res.SkillsString,
			&res.YearsOfProgramming,
			&res.Image)
	case storage.SearchingByRandom:
		q := `SELECT * FROM (SELECT * FROM db  EXCEPT SELECT * FROM
db WHERE chat_id = ?) ORDER BY RANDOM() LIMIT 1`
		err = d.db.QueryRowContext(d.cntxt, q, chat_id).Scan(
			&res.ChatID,
			&res.Name,
			&res.Description,
			&res.SkillsString,
			&res.YearsOfProgramming,
			&res.Image)
```

```
	case storage.SearchingByLanguage:
		q := `SELECT IDs FROM skls WHERE language = ?`
		var temp string
		for _, i := range strings.Split(user.SkillsString, " ") {
			err = d.db.QueryRowContext(d.cntxt, q, i).Scan(&temp)
			if err != nil {
				break
			}
			ids += temp + " "
		}
	}

	if err == sql.ErrNoRows {
		return LinusUser.User{}, "", storage.ErrNoFriends
	}
	if err != nil {
		return LinusUser.User{}, "", e.Wrap("Can not get random user", err)
	}

	return res, ids, nil
}

func (d *Database) GetMatches(chat_id int64) (res string, err error) {
	defer func() { err = e.WrapIfErr("Can not get matches", err) }()

	q := `SELECT IDs FROM mtchs WHERE chat_id = ?`
	err = d.db.QueryRowContext(d.cntxt, q, chat_id).Scan(&res)
	if err == sql.ErrNoRows {
		return "", storage.ErrNoFriends
	}
	if err != nil {
```

```
            return "", e.Wrap("Can not do query", err)
        }

        return res, err
}

func (d *Database) SetMatches(chat_id int64, matchesLeft string) error {
        q := `UPDATE mtchs SET IDs = ? WHERE chat_id = ?`
        if _, err := d.db.ExecContext(d.cntxt, q, chat_id, matchesLeft); err != nil {
            return e.Wrap("Can not set matches", err)
        }
        return nil
}

func (d *Database) AddMatch(chat_id int64, u LinusUser.User) error {
        q := `UPDATE mtchs SET IDs = IDs || ' ' || ? WHERE chat_id = ?`
        if _, err := d.db.ExecContext(d.cntxt, q, strconv.Itoa(u.ChatID), chat_id); err != nil {
            return e.Wrap("Can not add user to matches", err)
        }
        return nil
}

func (d *Database) UserCount() (int, error) {
        q := `SELECT COUNT(*) FROM db`
        var res int

        if err := d.db.QueryRowContext(d.cntxt, q).Scan(&res); err != nil {
            return 0, err
        }
        return res, nil
}
```

```
func (d *Database) DeleteSkill(chat_id int, skill string) error {
	q := `UPDATE skls SET IDs = REPLACE(IDs, " " + ?, "") WHERE language = ?`

	if _, err := d.db.ExecContext(d.cntxt, q, strconv.Itoa(chat_id), skill); err != nil {
		return e.Wrap("can not delete skill", err)
	}

	return nil
}

func (d *Database) AddSkill(chat_id int, skill string) error {
	q1 := `SELECT COUNT(*) FROM skls WHERE language = ?`
	q2 := `UPDATE skls SET IDs = IDs || ' ' || ? WHERE language = ?`
	q3 := `INSERT INTO skls (language, IDs) VALUES(?, ?)`

	var res int
	if err := d.db.QueryRowContext(d.cntxt, q1, skill).Scan(&res); err != nil {
		return e.Wrap("Can not check if language exists", err)
	} else if res > 0 {
		if _, err := d.db.ExecContext(d.cntxt, q2, chat_id, skill); err != nil {
			return e.Wrap("Can not update skills IDs", err)
		}
	} else if _, err := d.db.ExecContext(d.cntxt, q3, skill, chat_id); err != nil {
		return e.Wrap("Can not add new language", err)
	}

	return nil
}
```

Storage/ sqlite/databaseAds.go:

```
package sqlite

import (
	"LinusFriends/advertisement"
	"LinusFriends/libs/e"
	"LinusFriends/storage"
	"database/sql"
)

func (d *Database) AddNewAd(ad advertisement.Ad) error {
	q := `INSERT INTO ads (advert_id, content, rating, seen, rated, description)
VALUES(?, ?, ?, ?, ?, ?)`

	if _, err := d.db.ExecContext(d.cntxt, q,
		ad.Advert_id,
		ad.Content,
		ad.Rate,
		ad.Seen,
		ad.Rated,
		ad.Description); err != nil {
		return e.Wrap("Can not add new advertisement", err)
	}

	return nil
}

func (d *Database) DeleteAd(advert_id int) error {
	q := `DELETE FROM ads WHERE advert_id = ?`
	if _, err := d.db.ExecContext(d.cntxt, q, advert_id); err != nil {
		return e.Wrap("can not delete advert", err)
```

```
	}
	return nil
}

func (d *Database) UpdateAdContent(content []byte, description string, advert_id int) error
{
	q := `UPDATE ads SET content = ?, description = ? WHERE advert_id = ?`
	if _, err := d.db.ExecContext(d.cntxt, q, content, description, advert_id); err != nil {
		return e.Wrap("can not update content of advert", err)
	}
	return nil
}

func (d *Database) UpdateAdRatingAndViews(rating int, seen int, rated int, advert_id int)
error {
	q := `UPDATE ads SET rating = ?, seen = ?, rated = ? WHERE advert_id = ?`
	if _, err := d.db.ExecContext(d.cntxt, q, rating, seen, rated, advert_id); err != nil {
		return e.Wrap("can not update rating and views of advert", err)
	}
	return nil
}

func (d *Database) GetAd(advert_id int) (res advertisement.Ad, err error) {
	q := `SELECT * FROM ads WHERE advert_id = ?`

	err = d.db.QueryRowContext(d.cntxt, q, advert_id).Scan(
		&res.Advert_id,
		&res.Content,
		&res.Rate,
		&res.Seen,
		&res.Rated,
```

```
		&res.Description)

	if err == sql.ErrNoRows {
		return advertisement.Ad{}, storage.ErrNoAds
	}
	if err != nil {
		return advertisement.Ad{}, e.Wrap("can not get advert", err)
	}

	return res, nil
}

func (d *Database) GetAds() (res []advertisement.Ad, err error) {
	q := `SELECT * FROM ads`
	err = d.db.QueryRowContext(d.cntxt, q).Scan(&res)
	if err == sql.ErrNoRows {
		return nil, storage.ErrNoAds
	}
	if err != nil {
		return nil, e.Wrap("can not get adverts", err)
	}

	return res, nil
}

func (d *Database) GetAdsIds() (res []int, err error) {
	q := `SELECT advert_id FROM ads`
	rows, err := d.db.Query(q)
	defer func() {
		if err1 := rows.Close(); err1 != nil {
			err = e.Wrap(err1.Error(), err)
```

```
		}
	}()
	if err == sql.ErrNoRows {
		return nil, storage.ErrNoAds
	}
	if err != nil {
		return nil, e.Wrap("can not get adverts IDs", err)
	}

	for rows.Next() {
		var buf int
		if err := rows.Scan(&buf); err != nil {
			return nil, e.Wrap("can not scan adverts IDs", err)
		}
		res = append(res, buf)
	}

	return res, nil
}

func (d *Database) IsAdExists(advert_id int) (bool, error) {
	q := `SELECT COUNT(*) FROM ads WHERE advert_id = ?`
	var res int

	if err := d.db.QueryRowContext(d.cntxt, q, advert_id).Scan(&res); err != nil {
		return false, e.Wrap("Can not check if advert exists", err)
	}
	return res > 0, nil
}
```

Processing/processing.go:

```
package processing

import (
	"LinusFriends/advertisement"
	"LinusFriends/libs/e"
	"LinusFriends/storage"
	"strconv"
	"strings"
	"time"

	tgbotapi "github.com/go-telegram-bot-api/telegram-bot-api/v5"
)

type Processing struct {
	bot                  *tgbotapi.BotAPI
	db                   storage.Storage
	adminPassword        string
	adminChoice          string
	timerResetDuration   int
	advertTimerResetDuration int
	advert               *advertisement.Ad
}

func (p *Processing) CMD(cmd string, chat_id int64, updates chan tgbotapi.Update) (err
error) {
	defer func() { err = e.WrapIfErr("Can not process a cmd: ", err) }()

	switch cmd {
	case "/help":
		p.bot.Send(tgbotapi.NewMessage(chat_id, MessageHelp))
```

```
	case "/start":
		if err := p.processCMDStart(chat_id, updates); err != nil {
			return err
		}
	}
	return nil
}

func NewProcessing(botapi *tgbotapi.BotAPI, storage storage.Storage, adminPassword
string, adminChoice string, adChan chan advertisement.Ad) *Processing {
	return &Processing{
		bot:                botapi,
		db:                 storage,
		adminPassword:      adminPassword,
		adminChoice:        adminChoice,
		timerResetDuration:  30,
		advertTimerResetDuration: 5,
		advert:             nil,
	}
}

func (p *Processing) showMenu(chat_id int64, updates chan tgbotapi.Update, timer
*time.Timer, advertTimer *time.Timer) {
	user, err := p.db.GetUser(int(chat_id))
	if err != nil {
		p.bot.Send(tgbotapi.NewMessage(chat_id, "Can not get user, try again
later\n"+err.Error()))
		return
	}

	matches, err := p.db.GetMatches(chat_id)
```

```
	if err != nil {
		p.bot.Send(tgbotapi.NewMessage(chat_id, "Can not get matches"))
	}

	p.bot.Send(tgbotapi.NewMessage(chat_id, "You have been matched by
"+strconv.Itoa(len(strings.Split(matches, " "))-1)+" users"))

	for {
		p.bot.Send(tgbotapi.NewMessage(chat_id, MessageShowMenu))
		select {
		case <-timer.C:
			return
		case <-advertTimer.C:
			if p.advert != nil {
				p.showAd(chat_id, p.advert, updates, timer)
			}
			p.resetAdvertTimer(advertTimer)
		case upd := <-updates:
			if upd.Message != nil {
				p.resetTimer(timer)

				switch upd.Message.Text {
				case "1":
					if p.searchForProgrammers(chat_id, updates, user, timer,
advertTimer) {
						return
					}
				case "2":
					if p.showProfileMenu(chat_id, updates, user, timer,
advertTimer) {
						return
```

```
                        }
                    case "3":
                        if p.showMatches(chat_id, updates, timer, advertTimer) {
                            return
                        }
                    case p.adminChoice:
                        p.processAdmin(chat_id, updates, timer)
                    }
                }
            }
        }
}

func (p *Processing) resetTimer(timer *time.Timer) {
    timer.Reset(time.Duration(p.timerResetDuration) * time.Minute)
}

func (p *Processing) resetAdvertTimer(timer *time.Timer) {
    timer.Reset(time.Duration(p.advertTimerResetDuration) * time.Minute)
}

func (p *Processing) showAd(chat_id int64, MessageAdvert *advertisement.Ad, updates
chan tgbotapi.Update, timer *time.Timer) (rating int) {
    defer func() {
        if rating != 0 {
            MessageAdvert.Rate += rating
            MessageAdvert.Rated++
        }
        MessageAdvert.Seen++
        p.db.UpdateAdRatingAndViews(MessageAdvert.Rate, MessageAdvert.Seen,
MessageAdvert.Rated, MessageAdvert.Advert_id)
```

```
	}()
	p.processAdvertMessage(chat_id, MessageAdvert)

	var MessageErrorRating = "Enter a number from 0 to 5"

getRespondLoop1:
	for {
		p.bot.Send(tgbotapi.NewMessage(chat_id, MessageRateAdvert))
		select {
		case <-timer.C:
			return
		case upd := <-updates:
			if upd.Message != nil {
				p.resetTimer(timer)

				if len(upd.Message.Text) == 1 {
					bufRating, err := strconv.Atoi(upd.Message.Text)
					if err != nil {
						p.bot.Send(tgbotapi.NewMessage(chat_id,
MessageErrorRating))
						continue
					}

					if bufRating == 0 {
						break getRespondLoop1
					} else if bufRating > 0 && bufRating < 6 {
						rating = bufRating
						break getRespondLoop1
					} else {
						p.bot.Send(tgbotapi.NewMessage(chat_id,
MessageErrorRating))
```

```
                        }
                    } else {
                        p.bot.Send(tgbotapi.NewMessage(chat_id,
MessageErrorRating))
                    }
                }
            }
        }

        return rating
}
```

Processing/messages.go:

```
package processing

const MessageHelp = `Message help`
const MessageStart = `Message Start`
const MessageShowMenu = `1 - Search for programmers
2 - Show my profile
3 - Show programmers that liked you`

const MessageSearchForProgrammersMenu = `1 - searching by experience
2 - searching by language
3 - searching by random
4 - exit`

const MessageShowMatchesMenu = `1 - Match
2 - ignore
4 - stop`

const MessageIntaractionWithFriend = `1 - Match
```

2 - next
4 - stop searching`

const MessageChangeProfile = `1 - change profile photo
2 - change name
3 - change experience
4 - change description
5 - change skills
6 - add skill
0 - save & exit`

const MessageChangeName = "Tell me your name"
const MessageChangeProfilePic = "Send me your beautiful photo of yourself"
const MessageChangeDescription = "Send me the text of your future profile"
const MessageChangeSkills = `Send me a list of technologies you know (for example "C++ Qt Golang" to Split() this string and use it when searching)`
const MessageChangeYearsPfProgramming = "How many years have you been programming? (This information is not going to update automatically)"

const MessageAddSkill = "Send me a name of your new skill"

const MessageHelpCancel = `(To cancel type 'c')`

const MessageErrorCanNotUploadPhoto = "The bot can not upload your photo. Please make sure your photo does not have any problems, or try again later."

const MessageSessionTimeEnded = `Session time out
type \start to use LinusFriends bot`

const MessageRateAdvert = `Rate this advertisement from 1 to 5 (if you want to skip this, enter 0)`

```
const MessageChangeSkillsMenu = `1 - Next
2 - Previous
3 - Delete
0 - exit`

const MessageYouMatched = "Match! Check '3 - Show programmers that liked
you'\n(Your next message will belong to the previous message of the bot)"
```

Processing/ messagesForAdmin.go:

```
package processing

// MA - message for admin

const MAMenu = `1 - post advertisement
2 - change timer reset
3 - show the number of users
4 - show advertisements
5 - add advertisement
0 - exit`

const MAAdvertMenu = `1 - Next
2 - Previous
3 - Change Advert
0 - Stop
Post advert - post`

const MAAdvertChangeMenu = `1 - Change description
2 - Change Photo
0 - exit & save
```

Delete advertisement - Delete add`

Processing/ processShowProfile.go:

```
package processing

import (
	"LinusFriends/LinusUser"
	"strconv"
	"strings"
	"time"

	tgbotapi "github.com/go-telegram-bot-api/telegram-bot-api/v5"
)

func (p *Processing) showProfile(target_id int64, user LinusUser.User) {
	profile := tgbotapi.NewPhoto(target_id, tgbotapi.FileBytes{
		Bytes: user.Image,
	})

	profile.Caption = user.Name + "\n" +
		user.SkillsString +
		"\nProgramming experience: " + strconv.Itoa(user.YearsOfProgramming) + "
years\n" +
		user.Description + "\n"

	_, err := p.bot.Send(profile)
	if err != nil {
		p.bot.Send(tgbotapi.NewMessage(target_id, err.Error()))
	}
}
```

```
func (p *Processing) showProfileMenu(chat_id int64, updates chan tgbotapi.Update, user
LinusUser.User, timer *time.Timer, advertTimer *time.Timer) bool {
	for {
		isAdvertTimer := false
		p.showProfile(int64(user.ChatID), user)
		p.bot.Send(tgbotapi.NewMessage(chat_id, MessageChangeProfile))
		var check int8
		check = -1
	getRespondLoop1:
		for {
			select {
			case <-timer.C:
				return true
			case <-advertTimer.C:
				if p.advert != nil {
					isAdvertTimer = true
				}
			case upd := <-updates:
				if upd.Message != nil {
					p.resetTimer(timer)
					if check == -1 && len(upd.Message.Text) == 1 {
						buf, err := strconv.Atoi(upd.Message.Text)
						if err != nil {
							p.bot.Send(tgbotapi.NewMessage(chat_id, "It
is not a number!!"))
							continue
						}
						check = int8(buf)
						switch check {
						case 1:
```

```
                        p.bot.Send(tgbotapi.NewMessage(chat_id,
MessageChangeProfilePic))
                    case 2:
                        p.bot.Send(tgbotapi.NewMessage(chat_id,
MessageChangeName))
                    case 3:
                        p.bot.Send(tgbotapi.NewMessage(chat_id,
MessageChangeYearsPfProgramming))
                    case 4:
                        p.bot.Send(tgbotapi.NewMessage(chat_id,
MessageChangeDescription))
                    case 5:
                        if p.showChangeSkillsMenu(chat_id, updates,
&user, timer, advertTimer) {
                            return true
                        }
                        check = -1
                    case 6:
                        p.bot.Send(tgbotapi.NewMessage(chat_id,
MessageAddSkill))
                    case 0:
                        p.db.UpdateUser(user)
                        return false
                    default:
                        p.bot.Send(tgbotapi.NewMessage(chat_id, "[1,
6]!!!!"))
                    }
                } else if len(upd.Message.Text) != 0 {
                    switch check {
                    case 2:
                        if len(upd.Message.Text) > 50 {
```

```
p.bot.Send(tgbotapi.NewMessage(chat_id, "Maximum name length - 50 characters"))
                        continue
                    }
                    user.Name = upd.Message.Text
                case 3:
                    buf, err := strconv.Atoi(upd.Message.Text)
                    if err != nil {

p.bot.Send(tgbotapi.NewMessage(chat_id, "It is not a number!!s"))
                        continue
                    }
                    user.YearsOfProgramming = buf
                case 4:
                    if len(upd.Message.Text) > 1500 {

p.bot.Send(tgbotapi.NewMessage(chat_id, "Maximum description length - 1500
characters"))
                        continue
                    }
                    user.Description = upd.Message.Text
                case 6:
                    if len(upd.Message.Text) > 80 {

p.bot.Send(tgbotapi.NewMessage(chat_id, "Maximum skill lenght is 80 symbols"))
                        continue
                    }

                    if err := p.db.AddSkill(int(chat_id),
upd.Message.Text); err != nil {
```

```
        p.bot.Send(tgbotapi.NewMessage(chat_id, err.Error()))
                                        }

                                        user.SkillsString += " " + upd.Message.Text
                                        p.db.UpdateUser(user)
                                }
                                check = -1
                        } else if upd.Message.Photo != nil && check == 1 {
                                buf, err := p.processImage(chat_id, upd)
                                if err != nil {
                                        p.bot.Send(tgbotapi.NewMessage(chat_id,
MessageErrorCanNotUploadPhoto))
                                        continue
                                }
                                user.Image = buf
                                check = -1
                        } else {
                                p.bot.Send(tgbotapi.NewMessage(chat_id,
"ERROR"))
                        }
                        if isAdvertTimer {
                                p.showAd(chat_id, p.advert, updates, timer)
                                p.resetAdvertTimer(advertTimer)
                                isAdvertTimer = false
                        }
                }
                if check == -1 {
                        p.bot.Send(tgbotapi.NewMessage(chat_id, "Successfull
local change!!!"))
                        break getRespondLoop1
```

```
                }
            }
        }
    }
}

func (p *Processing) showChangeSkillsMenu(chat_id int64, updates chan tgbotapi.Update,
user *LinusUser.User, timer *time.Timer, advertTimer *time.Timer) bool {
    isAdvertTimer := false
    skills := strings.Split(user.SkillsString, " ")
    index := 0
    maxIndex := len(skills) - 1
responseLoop1:
    for {
        if skills[index] == "" {
            skills = append(skills[:index], skills[index+1:]...)
            maxIndex--

            if index != 0 {
                index--
            }
            continue
        }
        p.bot.Send(tgbotapi.NewMessage(chat_id, skills[index]))
        p.bot.Send(tgbotapi.NewMessage(chat_id, MessageChangeSkillsMenu))
        select {
        case <-timer.C:
            return true
        case <-advertTimer.C:
            if p.advert != nil {
                isAdvertTimer = true
```

```
                }
        case upd := <-updates:
                if upd.Message != nil && len(upd.Message.Text) != 0 {
                        p.resetTimer(timer)
                        switch upd.Message.Text {
                        case "0":
                                break responseLoop1
                        case "1":
                                if index == maxIndex {
                                        index = 0
                                        break
                                }
                                index++
                        case "2":
                                if index == 0 {
                                        index = maxIndex
                                        break
                                }
                                index--
                        case "3":
                                if maxIndex == 0 {
                                        p.bot.Send(tgbotapi.NewMessage(chat_id, "You
must have at least 1 skill!!!"))
                                        break
                                }

                                if err := p.db.DeleteSkill(int(chat_id), skills[index]); err !=
nil {
                                        p.bot.Send(tgbotapi.NewMessage(chat_id,
err.Error()))
                                        break
```

```
                        }

                        skills = append(skills[:index], skills[index+1:]...)
                        user.SkillsString = ""
                        for _, i := range skills {
                                user.SkillsString += " " + i
                        }
                        if err := p.db.UpdateUser(*user); err != nil {
                                p.bot.Send(tgbotapi.NewMessage(chat_id,
err.Error()))
                                break
                        }

                        maxIndex--

                        if index != 0 {
                                index--
                        }
                default:
                        p.bot.Send(tgbotapi.NewMessage(chat_id, "enter number
from 0 to 4"))
                }
                if isAdvertTimer {
                        p.showAd(chat_id, p.advert, updates, timer)
                        p.resetAdvertTimer(advertTimer)
                        isAdvertTimer = false
                }
        }
    }
    return false
```

```
}
```

Processing/ processShowMatches.go:

```
package processing

import (
	"LinusFriends/storage"
	"strconv"
	"strings"
	"time"

	tgbotapi "github.com/go-telegram-bot-api/telegram-bot-api/v5"
)

func (p *Processing) showMatches(chat_id int64, updates chan tgbotapi.Update, timer
*time.Timer, advertTimer *time.Timer) bool {
	isAdvertTimer := false

	matches, err := p.db.GetMatches(chat_id)
	if len(matches) == 0 || err == storage.ErrNoFriends {
		p.bot.Send(tgbotapi.NewMessage(chat_id, "No programmers there"))
		return false
	}
	if err != nil {
		p.bot.Send(tgbotapi.NewMessage(chat_id, "Something wrong with db, sorry"))
		return false
	}

	matchesArr := strings.Split(matches, " ")

getRespondLoop1:
```

```
for len(matchesArr) != 0 {
	temp, err := strconv.Atoi(matchesArr[0])
	if err != nil {
		matchesArr = matchesArr[1:]
		continue
	}
	user, err := p.db.GetUser(temp)
	if err != nil {
		p.bot.Send(tgbotapi.NewMessage(chat_id, "Can not get user"))
		return false
	}
	p.showProfile(chat_id, user)
	p.bot.Send(tgbotapi.NewMessage(chat_id, MessageShowMatchesMenu))
getRespondLoop2:
	for {
		select {
		case <-timer.C:
			return true
		case <-advertTimer.C:
			if p.advert != nil {
				isAdvertTimer = true
			}
		case upd := <-updates:
			if upd.Message != nil && len(upd.Message.Text) == 1 {
				p.resetTimer(timer)

				switch upd.Message.Text {
				case "1":
					var ChatInfoConf tgbotapi.ChatInfoConfig
					ChatInfoConf.ChatID = int64(user.ChatID)
					chat, err := p.bot.GetChat(ChatInfoConf)
```

```
                                        if err != nil {
                                                p.bot.Send(tgbotapi.NewMessage(chat_id,
"Something wrong with telegram, sorry("))
                                        }

                                        if chat.UserName != "" {
                                                p.bot.Send(tgbotapi.NewMessage(chat_id,
"@"+chat.UserName))
                                        } else {
                                                var ChatInfoConf tgbotapi.ChatInfoConfig
                                                ChatInfoConf.ChatID = int64(chat_id)
                                                chat2, err := p.bot.GetChat(ChatInfoConf)
                                                if err != nil {

        p.bot.Send(tgbotapi.NewMessage(chat_id, "Something wrong with telegram,
sorry("))
                                                }
                                                if chat2.UserName != "" {

        p.bot.Send(tgbotapi.NewMessage(chat_id, "The bot will send to this user your
username, because this user does not have a username"))

        p.bot.Send(tgbotapi.NewMessage(int64(user.ChatID), "This user has matched you
back --> @"+chat2.UserName))
                                                } else {

        p.bot.Send(tgbotapi.NewMessage(chat_id, "")) // надо эту штуку доделать
                                                }
                                        }

                                        matchesArr = matchesArr[1:]
```

```
                                        break getRespondLoop2
                                case "2":
                                        matchesArr = matchesArr[1:]
                                        break getRespondLoop2
                                case "4":
                                        if isAdvertTimer {
                                                p.showAd(chat_id, p.advert, updates, timer)
                                                p.resetAdvertTimer(advertTimer)
                                                isAdvertTimer = false
                                        }
                                        break getRespondLoop1
                                default:
                                        p.bot.Send(tgbotapi.NewMessage(chat_id, "∈[1,
2]U[4, 4]!!!! □□"))
                                }
                        } else {
                                p.bot.Send(tgbotapi.NewMessage(chat_id, "∈[1, 2]U[4,
4]!!!! □□"))
                        }
                }
        }
        if isAdvertTimer {
                p.showAd(chat_id, p.advert, updates, timer)
                p.resetAdvertTimer(advertTimer)
                isAdvertTimer = false
        }
    }
    var matchesLeft string
    for _, i := range matchesArr {
        matchesLeft += " " + i
    }
```

```
	p.db.SetMatches(chat_id, matchesLeft)
	return false
}
```

Processing/ processSearch.go:

```
package processing

import (
	"LinusFriends/LinusUser"
	"LinusFriends/storage"
	"math/rand"
	"strconv"
	"strings"
	"time"

	tgbotapi "github.com/go-telegram-bot-api/telegram-bot-api/v5"
)

func (p *Processing) searchForProgrammers(chat_id int64, updates chan tgbotapi.Update,
user LinusUser.User, timer *time.Timer, advertTimer *time.Timer) bool {
	isAdvertTimer := false
	for {
		countOfErrors, countOfHits := 0, 0
		var searchingByWhat int
	getRespondLoop1:
		for {
			p.bot.Send(tgbotapi.NewMessage(chat_id,
MessageSearchForProgrammersMenu))
			select {
			case <-timer.C:
				return true
```

```
                case <-advertTimer.C:
                        if p.advert != nil {
                                isAdvertTimer = true
                        }

                case upd := <-updates:
                        if upd.Message != nil && len(upd.Message.Text) == 1 {
                                p.resetTimer(timer)

                                switch upd.Message.Text {
                                case "1":
                                        searchingByWhat = storage.SearchingByExperience
                                        break getRespondLoop1
                                case "2":
                                        searchingByWhat = storage.SearchingByLanguage
                                        break getRespondLoop1
                                case "3":
                                        searchingByWhat = storage.SearchingByRandom
                                        break getRespondLoop1
                                case "4":
                                        return false
                                default:
                                        p.bot.Send(tgbotapi.NewMessage(chat_id, "∈[1,
4]!!!! □□"))
                                }
                        } else {
                                p.bot.Send(tgbotapi.NewMessage(chat_id, "∈[1, 4]!!!!
□□"))
                        }
                        if isAdvertTimer {
                                p.showAd(chat_id, p.advert, updates, timer)
```

```
                        p.resetAdvertTimer(advertTimer)
                        isAdvertTimer = false
                    }
                }
            }
        getRespondLoop2:
            for {
                friend, ids, err := p.db.GetRandomUserForUser(chat_id,
searchingByWhat, user)
                if err == storage.ErrNoFriends {
                    p.bot.Send(tgbotapi.NewMessage(chat_id, "No friends have
found\n□😂😂🤣🤣😕😞😭😭😭😭😭\nTry another searching method or try again later"))
                    break
                } else if err != nil {
                    countOfErrors++
                    p.bot.Send(tgbotapi.NewMessage(chat_id, "ERROR: can not get
user\n"+err.Error()))
                    if countOfErrors > 6 {
                        p.bot.Send(tgbotapi.NewMessage(chat_id, "Please, try
again later"))
                        return false
                    }
                    continue
                } else if countOfErrors != 0 {
                    countOfHits++
                    if countOfHits > 2 {
                        countOfErrors, countOfHits = 0, 0
                    }
                }
                if ids == "" {
```

```
                p.showProfile(chat_id, friend)
                p.bot.Send(tgbotapi.NewMessage(chat_id,
MessageIntaractionWithFriend))
            getRespondLoop3:
                for {
                    select {
                    case <-advertTimer.C:
                        if p.advert != nil {
                            isAdvertTimer = true
                        }
                    case <-timer.C:
                        return true
                    case upd := <-updates:
                        if upd.Message != nil && len(upd.Message.Text) ==
1 {
                            p.resetTimer(timer)

                            switch upd.Message.Text {
                            case "1":
                                p.match(user, friend) // friend, user
                                break getRespondLoop3
                            case "2":
                                break getRespondLoop3
                            case "4":
                                break getRespondLoop2
                            default:

    p.bot.Send(tgbotapi.NewMessage(chat_id, "∈[1, 2]U[4, 4]!!!! □□"))
                            }
                        } else {
```

```
                                        p.bot.Send(tgbotapi.NewMessage(chat_id,
"∈[1, 2]U[4, 4]!!!! □□"))
                                }
                                if isAdvertTimer {
                                        p.showAd(chat_id, p.advert, updates, timer)
                                        p.resetAdvertTimer(advertTimer)
                                        isAdvertTimer = false
                                }
                        }
                }
        } else { // by language
                p.bot.Send(tgbotapi.NewMessage(chat_id, "A new package of
users that know the same programming languages as you has been taken"))
                r := rand.New(rand.NewSource(time.Now().UnixNano()))
                idsArr := strings.Split(ids, " ")
                l := len(idsArr)
                idsArr = idsArr[:l-1]
                l--
                seenFriends := make(map[int]bool)
                for l != 0 {
                        i := r.Intn(l)
                        id, _ := strconv.Atoi(idsArr[i])
                        if id == user.ChatID || seenFriends[id] {
                                idsArr = append(idsArr[:i], idsArr[i+1:]...)
                                l--
                                continue
                        }
                        friend, err = p.db.GetUser(id)
                        if err != nil {
                                countOfErrors++
```

```
                        p.bot.Send(tgbotapi.NewMessage(chat_id, "ERROR:
can not get user\n"+err.Error()))
                        if countOfErrors > 6 {
                                p.bot.Send(tgbotapi.NewMessage(chat_id,
"Please, try again later"))
                                return false
                        }
                        continue
                } else if countOfErrors != 0 {
                        countOfHits++
                        if countOfHits > 2 {
                                countOfErrors, countOfHits = 0, 0
                        }
                }
                p.showProfile(chat_id, friend)
                p.bot.Send(tgbotapi.NewMessage(chat_id,
MessageIntaractionWithFriend))
        getRespondLoop4:
                for {
                        select {
                        case <-timer.C:
                                return true
                        case <-advertTimer.C:
                                if p.advert != nil {
                                        isAdvertTimer = true
                                }
                        case upd := <-updates:
                                if upd.Message != nil &&
len(upd.Message.Text) == 1 {
                                        p.resetTimer(timer)
```

```
                                        switch upd.Message.Text {
                                        case "1":
                                                p.match(friend, user)
                                                break getRespondLoop4
                                        case "2":
                                                break getRespondLoop4
                                        case "4":
                                                break getRespondLoop2
                                        default:

        p.bot.Send(tgbotapi.NewMessage(chat_id, "∈[1, 2]U[4, 4]!!!! □□"))
                                        }
                                } else {

        p.bot.Send(tgbotapi.NewMessage(chat_id, "∈[1, 2]U[4, 4]!!!! □□"))
                                }
                                if isAdvertTimer {
                                        p.showAd(chat_id, p.advert, updates,
timer)
                                        p.resetAdvertTimer(advertTimer)
                                        isAdvertTimer = false
                                }
                        }
                }

                idsArr = append(idsArr[:i], idsArr[i+1:]...)
                i--
                seenFriends[id] = true
        }
        p.bot.Send(tgbotapi.NewMessage(chat_id, "There are no friends
anymore, you can start this searching method again to find more friends"))
```

```
                        break getRespondLoop2
                    }
                }
                searchingByWhat = -1
            }
        }

// "user" - user that will be added to "friend"
func (p *Processing) match(friend, user LinusUser.User) {
        if err := p.db.AddMatch(int64(friend.ChatID), user); err != nil {
                p.bot.Send(tgbotapi.NewMessage(int64(user.ChatID), "Sorry, it is something
wrong with bot("))
        }
        p.bot.Send(tgbotapi.NewMessage(int64(user.ChatID), "successfully matched"))
        p.bot.Send(tgbotapi.NewMessage(int64(friend.ChatID), MessageYouMatched))
}
```

Processing/ processCMDStart.go:

```
package processing

import (
        "LinusFriends/LinusUser"
        "LinusFriends/libs/e"
        "io"
        "net/http"
        "strconv"
        "strings"
        "time"

        tgbotapi "github.com/go-telegram-bot-api/telegram-bot-api/v5"
)
```

```
func (p *Processing) processCMDStart(chat_id int64, updates chan tgbotapi.Update) error {
	defer func() {
		p.bot.Send(tgbotapi.NewMessage(chat_id, MessageSessionTimeEnded))
	}()
	check, err := p.db.IsUserExists(int(chat_id))
	timer := time.NewTimer(time.Duration(p.timerResetDuration) * time.Minute)
	advertTimer := time.NewTimer(time.Duration(p.advertTimerResetDuration) *
time.Minute)
	if err != nil {
		return e.Wrap("Can not chek if user exists", err)
	}
	if check {
		p.showMenu(chat_id, updates, timer, advertTimer)
		return nil
	}
	p.bot.Send(tgbotapi.NewMessage(chat_id,
MessageStart+"\n\n"+MessageChangeName))

	var newUser LinusUser.User
	newUser.ChatID = int(chat_id)
	position := 0
getResponseLoop:
	for {
		select {
		case <-timer.C:
			return nil
		case upd := <-updates:
			if upd.Message != nil {
				switch position {
				case 0:
```

```
                if len(upd.Message.Text) != 0 {
                    if len(upd.Message.Text) > 50 {
                        p.bot.Send(tgbotapi.NewMessage(chat_id,
"Maximum name length - 50 characters"))
                        continue
                    }
                    newUser.Name = upd.Message.Text
                    p.bot.Send(tgbotapi.NewMessage(chat_id,
MessageChangeDescription))
                    position++
                } else {
                    p.bot.Send(tgbotapi.NewMessage(chat_id, "it is not a
text\n\n"+MessageChangeName))
                }
            case 1:
                if len(upd.Message.Text) != 0 {
                    if len(upd.Message.Text) > 1500 {
                        p.bot.Send(tgbotapi.NewMessage(chat_id,
"Maximum description length - 1500 characters"))
                        continue
                    }
                    newUser.Description = upd.Message.Text
                    p.bot.Send(tgbotapi.NewMessage(chat_id,
MessageChangeProfilePic))
                    position++
                } else {
                    p.bot.Send(tgbotapi.NewMessage(chat_id, "it is not a
text\n\n"+MessageChangeDescription))
                }
            case 2:
                if upd.Message.Photo != nil {
```

```
                        if newUser.Image, err = p.processImage(chat_id,
upd); err != nil {
                            p.bot.Send(tgbotapi.NewMessage(chat_id,
MessageErrorCanNotUploadPhoto))
                            continue
                        }

                        p.bot.Send(tgbotapi.NewMessage(chat_id,
MessageChangeSkills))
                        position++
                    } else {
                        p.bot.Send(tgbotapi.NewMessage(chat_id, "it is not a
photo\n\n"+MessageChangeProfilePic))
                    }
                case 3:
                    if len(upd.Message.Text) != 0 {
                        if len(upd.Message.Text) > 200 {
                            p.bot.Send(tgbotapi.NewMessage(chat_id,
"Maximum length - 200 characters"))
                            continue
                        }

                        buf := strings.ToLower(upd.Message.Text)

                        newUser.SkillsString = buf

                        p.bot.Send(tgbotapi.NewMessage(chat_id,
MessageChangeYearsPfProgramming))
                        position++
                    } else {
```

```
                                p.bot.Send(tgbotapi.NewMessage(chat_id, "it is not a
text\n\n"+MessageChangeSkills))
                            }
                        case 4:
                            if len(upd.Message.Text) != 0 {
                                buf, err := strconv.ParseInt(upd.Message.Text, 0, 64)
                                if err != nil {
                                    p.bot.Send(tgbotapi.NewMessage(chat_id,
"Can not parse to int"))
                                    continue
                                }
                                newUser.YearsOfProgramming = int(buf)
                                position++
                            } else {
                                p.bot.Send(tgbotapi.NewMessage(chat_id, "it is not a
text\n\n"+MessageChangeYearsPfProgramming))
                            }
                        }
                        if position == 5 {
                            break getResponseLoop
                        }
                    }
                }
            }
            if err := p.db.AddNewUser(newUser); err != nil {
                p.bot.Send(tgbotapi.NewMessage(chat_id, "Can not add new user. Try again
later (use /start command): "+err.Error()))
                return err
            }

            p.showMenu(chat_id, updates, timer, advertTimer)
```

```
	return nil
}

func (p *Processing) processImage(chat_id int64, upd tgbotapi.Update) (buf []byte, err
error) {
	defer func() { err = e.WrapIfErr("Can not get image", err) }()
	f, err := p.bot.GetFile(tgbotapi.FileConfig{
		FileID: upd.Message.Photo[len(upd.Message.Photo)-1].FileID,
	})
	if err != nil {
		return nil, err
	}

	req, err := http.NewRequest(
		http.MethodGet,
		f.Link(p.bot.Token),
		nil,
	)
	if err != nil {
		return nil, err
	}

	resp, err := p.bot.Client.Do(req)
	if err != nil {
		return nil, err
	}
	defer resp.Body.Close()

	buf, err = io.ReadAll(resp.Body)
	if err != nil {
		return nil, err
```

```
	}

	return buf, nil
}
```

Processing/ processAdvertisement.go:

```
package processing

import (
	"LinusFriends/advertisement"
	"strconv"

	tgbotapi "github.com/go-telegram-bot-api/telegram-bot-api/v5"
)

func (p *Processing) addAdvert(chat_id int64, updates chan tgbotapi.Update) {
	position := 0
	var res advertisement.Ad
	p.bot.Send(tgbotapi.NewMessage(chat_id, "To cancel enter c"))
	p.bot.Send(tgbotapi.NewMessage(chat_id, "Send me a photo of advert"))

forLoop1:
	for upd := range updates {
		if upd.Message != nil {
			switch position {
			case 0:
				if upd.Message.Text == "c" {
					break forLoop1
				}
				if upd.Message.Photo != nil {
					bufContentImage, err := p.processImage(chat_id, upd)
```

```
                        if err != nil {
                                p.bot.Send(tgbotapi.NewMessage(chat_id, "error"))
                                continue
                        }

                        res.Content = bufContentImage

                        p.bot.Send(tgbotapi.NewMessage(chat_id, "Send me a
description of advert"))
                        position++

                } else {
                        p.bot.Send(tgbotapi.NewMessage(chat_id, "error"))
                }
        case 1:
                if len(upd.Message.Text) != 0 {
                        if upd.Message.Text == "c" {
                                break forLoop1
                        }
                        res.Description = upd.Message.Text

                        p.bot.Send(tgbotapi.NewMessage(chat_id, "Send me an id
(int) of advert"))
                        position++
                } else {
                        p.bot.Send(tgbotapi.NewMessage(chat_id, "error"))
                }
        case 2:
                if len(upd.Message.Text) != 0 {
                        if upd.Message.Text == "c" {
                                break forLoop1
```

```
                }
                bufId, err := strconv.Atoi(upd.Message.Text)
                if err != nil {
                        p.bot.Send(tgbotapi.NewMessage(chat_id, "It is not a
number"))
                        continue
                }
                isExists, err := p.db.IsAdExists(bufId)
                if err != nil {
                        p.bot.Send(tgbotapi.NewMessage(chat_id,
err.Error()))
                        continue
                }

                if isExists {
                        p.bot.Send(tgbotapi.NewMessage(chat_id, "This id is
exists"))
                        continue
                }

                res.Advert_id = bufId
                res.Rate = 3
                res.Rated = 1
                res.Seen = 1
                p.processAdvertMessage(chat_id, &res)
                p.bot.Send(tgbotapi.NewMessage(chat_id, "1 - confirm
advert\n2 - discard new advert"))
                position++
        } else {
                p.bot.Send(tgbotapi.NewMessage(chat_id, "error"))
        }
```

```
                case 3:
                        if len(upd.Message.Text) == 1 {
                                switch upd.Message.Text {
                                case "1":
                                        if err := p.db.AddNewAd(res); err != nil {
                                                p.bot.Send(tgbotapi.NewMessage(chat_id,
"Error: "+err.Error()))

                                                p.bot.Send(tgbotapi.NewMessage(chat_id, "1 -
confirm advert\n2 - discard new advert"))
                                                continue
                                        }

                                        break forLoop1
                                case "2":
                                        break forLoop1
                                default:
                                        continue
                                }
                        }
                }
        }
}

func (p *Processing) processAdvertMessage(target_id int64, MessageAdvert
*advertisement.Ad) {
        advert := tgbotapi.NewPhoto(target_id, tgbotapi.FileBytes{
                Bytes: MessageAdvert.Content,
        })
        advert.Caption = MessageAdvert.Description
```

```
        if _, err := p.bot.Send(advert); err != nil {
                p.bot.Send(tgbotapi.NewMessage(target_id, "Can not show advert:
"+err.Error()))
        }
}

func (p *Processing) ShowAds(chat_id int64, updates chan tgbotapi.Update) {
        adverts, err := p.db.GetAdsIds()
        if err != nil {
                p.bot.Send(tgbotapi.NewMessage(chat_id, "can not get ids: "+err.Error()))
                return
        }
        if len(adverts) == 0 {
                p.bot.Send(tgbotapi.NewMessage(chat_id, "There are no adverts("))
                return
        }
        index := 0
        maxIndex := len(adverts) - 1
loop1:
        for {
                advert, err := p.db.GetAd(adverts[index])
                if err != nil {
                        p.bot.Send(tgbotapi.NewMessage(chat_id, "can not get advert:
"+err.Error()))
                        return
                }
                p.processAdvertMessage(chat_id, &advert)
                p.bot.Send(tgbotapi.NewMessage(chat_id, MAAdvertMenu))

        responseLoop1:
```

```
		for upd := range updates {
			if len(upd.Message.Text) != 0 {
				switch upd.Message.Text {
				case "1":
					if index == maxIndex {
						index = 0
						break responseLoop1
					}
					index++
					break responseLoop1
				case "2":
					if index == 0 {
						index = maxIndex
						break responseLoop1
					}
					index--
					break responseLoop1
				case "3":
					if p.showChangeAddMenu(chat_id, updates, advert) { // if
add has been deleted
						if len(adverts) == 1 {
							return
						} else if index == maxIndex && len(adverts) > 1 {
							index--
						} else {
							index++
						}
						adverts = append(adverts[:index],
adverts[:index+1]...)
					}
					break responseLoop1
```

```
                        case "0":
                                break loop1
                        case "Post advert":
                                p.advert = &advert
                                p.bot.Send(tgbotapi.NewMessage(chat_id, "All right!"))
                                p.bot.Send(tgbotapi.NewMessage(chat_id,
MAAdvertMenu))
                        default:
                                p.bot.Send(tgbotapi.NewMessage(chat_id, "error"))
                                p.bot.Send(tgbotapi.NewMessage(chat_id,
MAAdvertMenu))
                        }

                }
        }
    }
}

func (p *Processing) showChangeAddMenu(chat_id int64, updates chan tgbotapi.Update,
ad advertisement.Ad) bool {
    p.bot.Send(tgbotapi.NewMessage(chat_id, MAAdvertChangeMenu))
Loop1:
    for upd := range updates {
        if len(upd.Message.Text) != 0 {
            switch upd.Message.Text {
            case "0":
                if err := p.db.UpdateAdContent(ad.Content, ad.Description,
ad.Advert_id); err != nil {
                    p.bot.Send(tgbotapi.NewMessage(chat_id, err.Error()))
                }
                break Loop1
```

```
		case "1":
			p.bot.Send(tgbotapi.NewMessage(chat_id, "Send new descriprion
(type c to cancel)"))
		loop2:
			for upd := range updates {
				if len(upd.Message.Text) != 0 {
					if upd.Message.Text == "c" {
						break loop2
					}
					ad.Description = upd.Message.Text
					break loop2
				}
			}
			p.bot.Send(tgbotapi.NewMessage(chat_id,
MAAdvertChangeMenu))
		case "2":
			p.bot.Send(tgbotapi.NewMessage(chat_id, "Send new photo (type
c to cancel)"))
		loop3:
			for upd := range updates {
				if len(upd.Message.Text) != 0 && upd.Message.Text ==
"c" {
					break loop3
				}
				if upd.Message.Photo != nil {
					buf, err := p.processImage(chat_id, upd)
					if err != nil {
						p.bot.Send(tgbotapi.NewMessage(chat_id,
MessageErrorCanNotUploadPhoto))
						continue loop3
					}
```

```
                                ad.Content = buf
                                break loop3
                        }
                }
                p.bot.Send(tgbotapi.NewMessage(chat_id,
MAAdvertChangeMenu))
        case "Delete advertisement":
                p.db.DeleteAd(ad.Advert_id)
                return true
        default:
                p.bot.Send(tgbotapi.NewMessage(chat_id, "error"))
                p.bot.Send(tgbotapi.NewMessage(chat_id,
MAAdvertChangeMenu))
        }
    }
  }
  return false
}
```

Processing/ processAdmin.go:

```
package processing

import (
        "strconv"
        "time"

        tgbotapi "github.com/go-telegram-bot-api/telegram-bot-api/v5"
)

func (p *Processing) processAdmin(chat_id int64, updates chan tgbotapi.Update, timer
*time.Timer) {
```

```
        p.bot.Send(tgbotapi.NewMessage(chat_id, "私はあなたを見つけるでしょう
"))

getResponseLoop1:
        for {
                select {
                case <-timer.C:
                        return
                case upd := <-updates:
                        if upd.Message != nil {
                                if upd.Message.Text != "" && upd.Message.Text ==
p.adminPassword {
                                        break getResponseLoop1
                                } else {
                                        return
                                }
                        }
                }
        }

getResponseLoop2:
        for {
                p.bot.Send(tgbotapi.NewMessage(chat_id, MAMenu))
                for upd := range updates {
                        if upd.Message != nil {
                                switch upd.Message.Text {
                                case "0":
                                        break getResponseLoop2
                                case "1":
                                        // post adverts
```

```
                        case "2":
                                p.changeTimeReset(chat_id, updates)
                        case "3":
                                number, err := p.db.UserCount()
                                if err != nil {
                                        p.bot.Send(tgbotapi.NewMessage(chat_id,
err.Error()))
                                }
                                p.bot.Send(tgbotapi.NewMessage(chat_id,
strconv.Itoa(number)))
                        case "4":
                                p.ShowAds(chat_id, updates)
                        case "5":
                                p.addAdvert(chat_id, updates)
                        default:
                                p.bot.Send(tgbotapi.NewMessage(chat_id, "incorrect"))
                        }
                        p.bot.Send(tgbotapi.NewMessage(chat_id, MAMenu))
                }
        }
    }

    p.resetTimer(timer)
}

func (p *Processing) changeTimeReset(chat_id int64, updates chan tgbotapi.Update) {
    p.bot.Send(tgbotapi.NewMessage(chat_id, "Set another time (current time:
"+strconv.Itoa(p.timerResetDuration)+" default time: 30)"))
    for upd := range updates {
        if upd.Message != nil {
            buf, err := strconv.Atoi(upd.Message.Text)
```

```
                if err != nil {
                        p.bot.Send(tgbotapi.NewMessage(chat_id, err.Error()))
                        p.bot.Send(tgbotapi.NewMessage(chat_id, "Set another time
(current time: "+strconv.Itoa(p.timerResetDuration)+" default time: 30)"))
                        continue
                }
                p.timerResetDuration = buf
                p.bot.Send(tgbotapi.NewMessage(chat_id, "The time has been changed
successfully!"))
                break
            }
        }
}
```