

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, назва факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
бакалавр
(ступінь вищої освіти)

на тему ПРОГРАМНА РЕАЛІЗАЦІЯ БОТА-КОНСУЛЬТАНТА ДЛЯ ПОТРЕБ
ПРИЙМАЛЬНОЇ КОМІСІЇ НАВЧАЛЬНОГО ЗАКЛАДУ
SOFTWARE IMPLEMENTATION OF A BOT CONSULTANT FOR THE
ADMISSIONS COMMITTEE OF AN EDUCATIONAL INSTITUTION

Виконав: студент(ка) 4 курсу, групи КНТ-137
Спеціальності 121 Інженерія програмного
забезпечення
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

Яремчак Б.М.
(прізвище та ініціали)

Керівник Степаненко О.О.
(прізвище та ініціали)

Рецензент Голуб Т.В.
(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Інститут, факультет ІРЕ, ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
 (код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
 “ ” 20 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТА(КИ)

Яремчака Богдана Михайловича
 (прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Програмна реалізація бота-консультанта для потреб приймальної комісії навчального закладу. Software Implementation of a Bot Consultant for the Admissions Committee of an Educational Institution
 керівник проекту (роботи) доцент Степаненко Олександр Олексійович
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
- затверджені наказом закладу вищої освіти від “30” березня 2020 року № 103
2. Строк подання студентом проекту (роботи) 28 травня 2020 року
3. Вихідні дані до проекту (роботи) рекомендована література, технічне завдання
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми. 3. Основні рішення щодо реалізації компонентів системи. 4. Експлуатація, тестування та експериментальне дослідження програми. 5. Технічне завдання. 6. Текст програми.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	Степаненко О.О., доцент		
Нормоконтроль	Камінська Ж.К., асистент		

7. Дата видачі завдання 5 квітня 2021р.**КАЛЕНДАРНИЙ ПЛАН**

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2–3 тижні	Розділ 1
3	Розробка архітектури програми.	4 тиждень	Розділ 2
4	Розробка програми.	5–6 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми.	7 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	8 тиждень	Додатки
7	Захист роботи.	9 тиждень	

Студент (ка) _____ **Яремчак Б.М.**
(підпис) (прізвище та ініціали)Керівник проекту (роботи) _____ **Степаненко О.О.**
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
63с., 47рис., 2 табл., 2 додатки, 17 джерел.

BOT-КОНСУЛЬТАНТ, TELEGRAM, PYTHON, SQL, JSON, PYCHARM
COMMUNITY EDITION

Об'єкт дослідження - програмні засоби для створення бота-консультанта у Telegram.

Предмет дослідження – Бот-консультант для потреб приймальної комісії у Telegram.

Мета роботи – створення бота-консультанта, що надає користувачам можливість залишати повідомлення та данні підтримці приймальної комісії навчальних закладів, через месенджер Telegram.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування на мові Python для створення бота у месенджері Telegram.

Результати. Створено бота-консультанта у Telegram за допомогою мови програмування Python, який допомагає користувачам та адміністратору приймальної комісії: залишати та відправляти повідомлення, збирати данні користувача.

Висновки. Розроблений бот-консультант допомагає приймальній комісії у спілкуванні з користувачами, та збирає їх данні для подальших дій приймальної комісії.

Галузь використання – середа месенджеру Telegram, як один із популярніших месенджерів нашого часу.

ABSTRACT

Explanatory note to the final qualifying work of the bachelor: 63 pages, 47 figures, 2 tables, 2 appendixes, 17 sources.

BOT CONSULTANT, TELEGRAM, PYTHON, SQL, JSON, PYCHARM
COMMUNITY EDITION

The object of study is a software for creating a bot-consultant in Telegram.

The subject of the study is a bot-consultant for the needs of the selection committee in Telegram.

The purpose of the work is to create a bot-consultant that allows users to leave messages and data to the support of the admissions committee of educational institutions, via the Telegram messenger.

Materials, methods and technical means: object-oriented programming in Python, to create a bot in Telegram messenger.

Results. Created a bot-consultant in Telegram using the programming language Telegram, which helps users and the administrator of the selection committee: to leave and send messages, and collect user data.

Conclusions. The developed bot-constant helps the selection committee to communicate with users, and collects their data for further actions of the selection committee.

The field of use is the Telegram messenger environment, as one of the most popular messengers of our time.

ЗМІСТ

Перелік скорочень та умовних познач	8
Вступ	9
1 Аналіз предметної області	10
1.1 Знайомство з предметною областю	10
1.2.1 Бот-консультант «Comfy.ua»	11
1.2.2 Бот-Консультант «monobank»	12
1.2.3 Бот-консультант «Foxtrot»	13
1.2.4 Порівняльна таблиця програмних продуктів	14
2 Вибір та обґрунтування структури системи, яка проектується	15
2.1 Вибір інструментарію розробки бота	15
2.1.1 Мессенджер Telegram	15
2.1.2 Мова програмування	15
2.1.3 Огляд особливостей середовища розробки	16
2.2 Структура компонентів бота-консультанта	18
2.3 Опис модулів та компонентів розробленого застосунку	20
2.3.1 Класи	20
2.3.2 Модулі та бібліотеки	20
3 Основні рішення під час реалізації основних компонентів застосунку, що розробляється	20
3.1 Сервіс BotFather	20
3.2 Функціонал бібліотеки telebot	22
3.3 Опис допоміжних класів	23
3.4 Використання бази даних	25
3.4.1 Основа бази	25
3.4.2 SQLiteStudio	26
3.4.3 Робота з базою	28
4 Керівництво програміста	30
4.1 Алгоритм послідовностей	30

3.4.1 Алгоритм основи 30

3.4.1 Алгоритм сценаріїв.....31

5 Експлуатація, тестування та ескпериментальне дослідження програми.....39

5.1 Тестування бота-консультанта 39

Висновки.....48

Перелік джерел посилань.....49

Додаток А Текст програми 51

Додаток Б Слайди презентації.....56

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface

JSON – JavaScript Object Notation

ORM - Object-Relational Mapping

SDK – Software Development Kit

SQL – Structured Query Language

ВСТУП

Мессенджери стали невід'ємною частиною нашого життя. Люди проводять у них велику долю свого часу спілкуючись, а також меседжери, є непоганим способом здобування та розповсюдження інформації. Людині просто необхідно спілкуватися та отримувати інформацію, адже це одна з її потреб.

Боти у месенджерах виконують дуже різні завдання. Починаючи від звичайного менеджменту чат-конференцій, і закінчуючи цілими чат-іграми розрахованими на багато користувачів. Також є багатофункціональні боти, які увібрали в себе одразу декілька цілей для виконання. Насамперед боти потрібні для автоматизування процесів та для полегшення їх виконання людьми.

Для бота-консультанта був обраний месенджер Telegram, бо його використовує велика кількість людей адже він є одним з найпопулярніших месенджерів на теперішній день.

Бот-консультант для приймальної комісії вирішує проблему у зборі та зберіганні даних користувачей, та допомагає у анонімному спілкуванні зі сторони адміністрації приймальної комісії. Адже він вміє не лише приймати, але й розсилати повідомлення користувачам.

Також основною перевагою вважається те, що бот може обробляти одночасно запити від декількох користувачей цілодобово, на відміну від звичайної людини, якій потрібен відпочинок. Саме цим він автоматизує процес збору даних, та запитів користувачей.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Знайомство з предметною областю

Як правило боти-консультанти, або боти, які створені для підтримки якихось проєктів, це індивідуальні боти. Вони створююця вузьконаправленно, саме під потреби тієї чи іншої сфери.

Основна функція бота-консультанта для потреб приймальної комісії навчального закладу – можливість надати користувачу консультацію що до виниклих в нього питань або проблем, за допомогою обробки його запитів та доставкою їх.

1.2 Огляд існуючих програмних засобів, що вирішують аналогічні завдання

У дуже багатьох компаній є своя підтримка у соцмережах, або у месенджерах. Дуже часто це зроблено як бот, через який можливо залишати та отримувати повідомлення.

Розроблений бот-консультант для потреб приймальної комісії стане індивідуальним ботом, через який підтримка зможе зручно надавати допомогу користувачам.

1.2.1 Бот-консультант «Comfy.ua»

Як приклад бота у месенджері Telegram, через який надаються консультації користувачам можна взяти бота торговельної мережі магазинів «Comfy» [1]. Функції консультації зображено на рисунку 1.1.

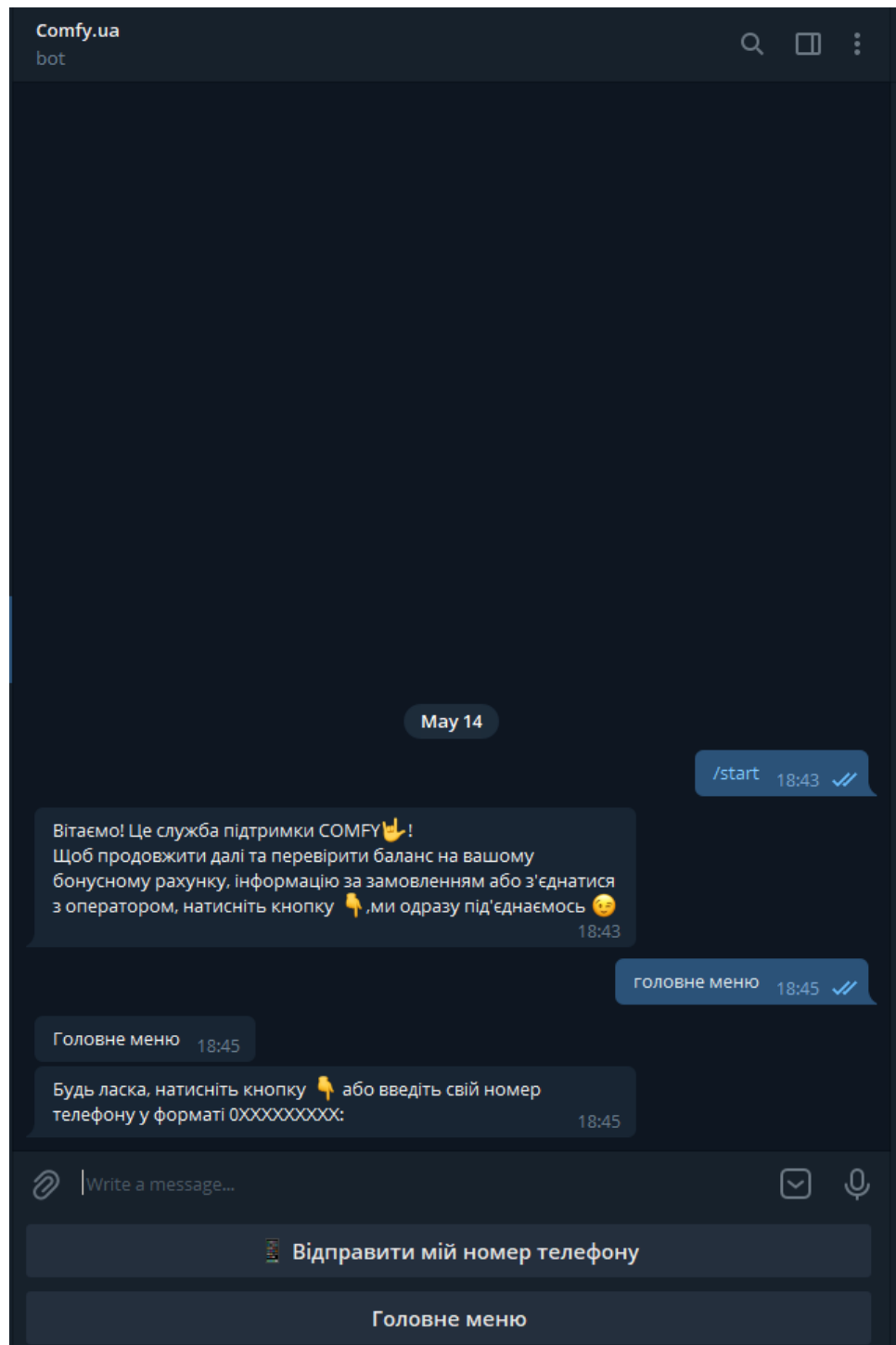


Рисунок 1.1 – Функція консультації бота «Comfy.ua»

1.2.2 Бот-Консультант «monobank»

Наступний аналог, що належить онлайн банку “monobank” переслідує таку саме ціль та створений так само для консультації користувачів[2]. Приклад інтерфейсу наведено на рисунку 1.2.

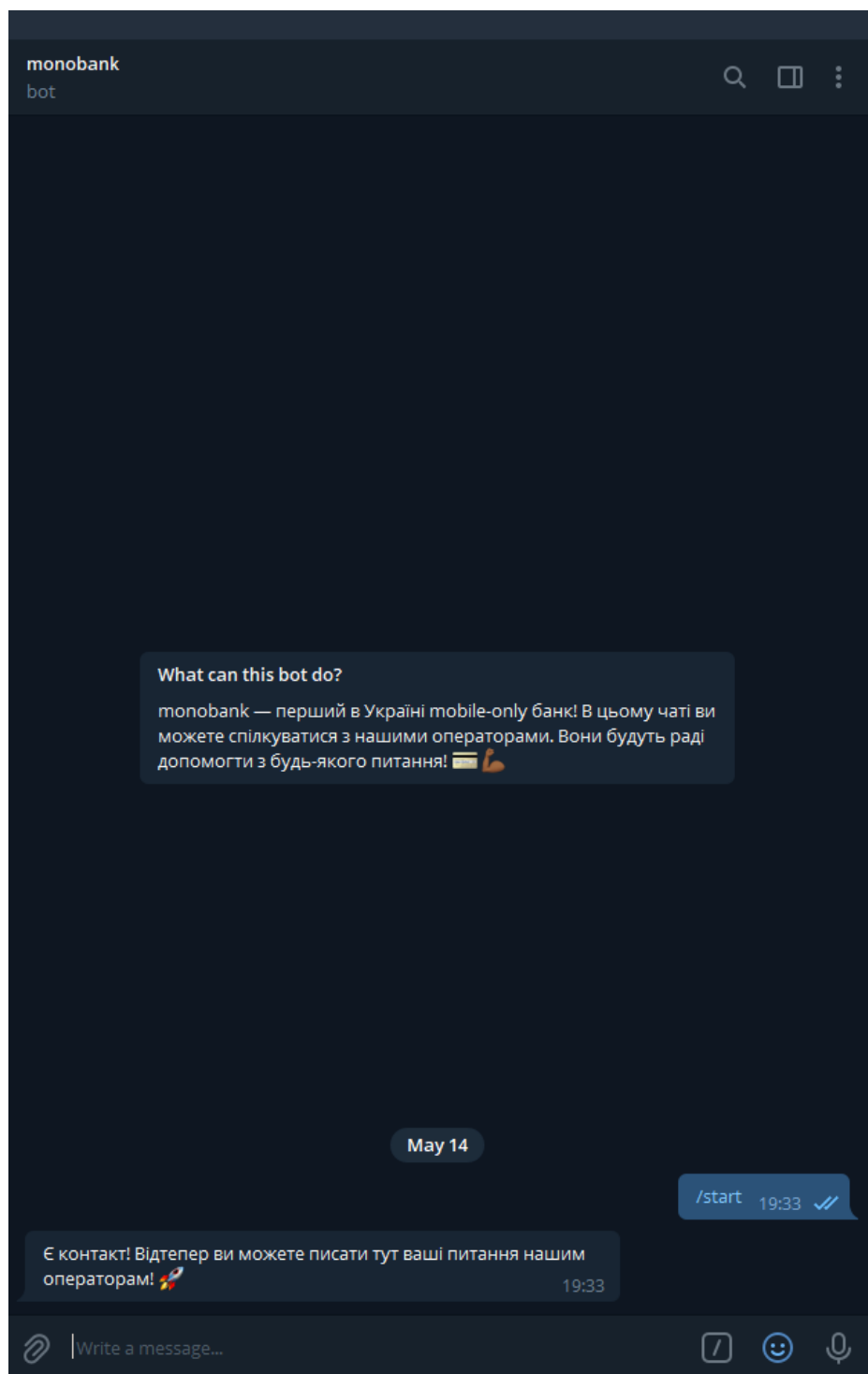


Рисунок 1.2 - Інтерфейс бота “monobank”

1.2.3 Бот-консультант «Foxtrot»

Бот для мережі торгівельних магазинів «Foxtrot» працює за схожим принципом як і бот «Comfi.ua» [3]. Приклад наведено на рисунку 1.3.

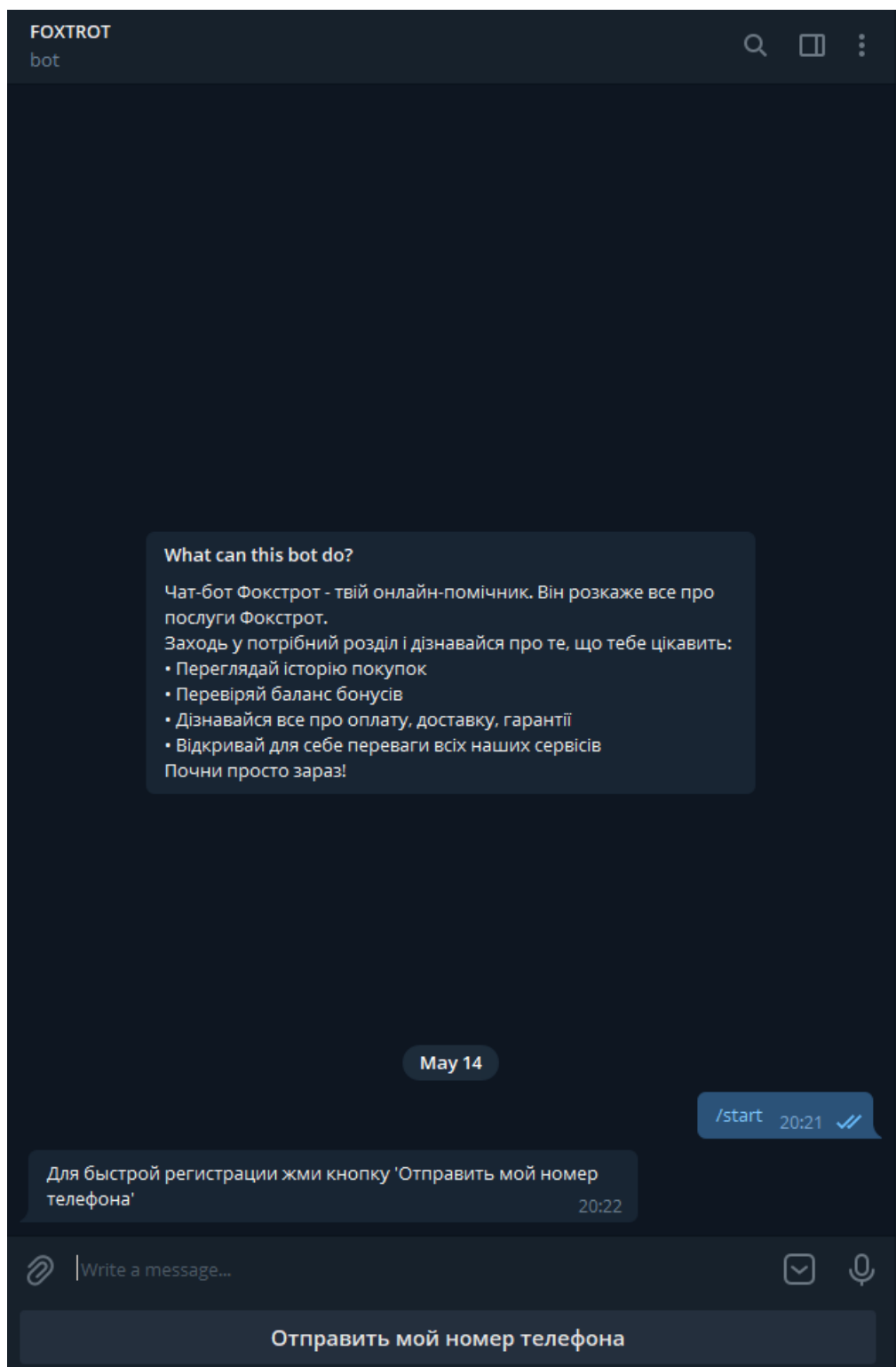


Рисунок 1.3 -- Приклад інтерфейсу бота“Foxtrot ”

1.2.4 Порівняльна таблиця програмних продуктів

Функціонал наведених ботів-консультантів дуже схожий. Кожен бот розроблено індивідуально під проєкт. Усі боти-консультанти достатньо прості та зручні у використанні користувачами. Те що відрізняє бота-консультанта приймальної комісії від інших, це можливість заповнити анкету, яка у свою чергу зберігає данні у зручну таблицю. Порівняння можна побачити у таблиці 1.1.

Таблиця 1.1 - Порівняльна характеристика застосунків

Назва	Зручність у використанні	індивідуальність	Збір даних у таблицю
Comfy.ua	+	+	-
Monobank	+	+	-
Foxtrot	+	+	-
Приймальна Комісія	+	+	+

2 ВИБІР ТА ОБҐРУНТУВАННЯ СТРУКТУРИ СИСТЕМИ, ЯКА ПРОЕКТУЄТЬСЯ

2.1 Вибір інструментарію розробки бота

2.1.1 Месенджер Telegram

Telegram - багатоплатформовий месенджер з функціями VoIP, що дозволяє обмінюватися текстовими, голосовими та відеоповідомленнями, стікерами та фотографіями, файлами багатьох форматів. Також можна здійснювати відео- і аудіозвонки, організовувати конференції, розраховані на багато користувачів групи і канали. Клієнтські програми Telegram доступні для Android, iOS, Windows Phone, Windows, macOS і GNU / Linux.

Для розробки бота-консультанта був обраний Telegram, так як він дуже популярний, а ще має зручні інструменти для цього[4].

2.1.2 Мова програмування

Перед початком розробки було проаналізовано мови програмування, що використовуються для розробки ботів для Telegram.

Таким чином, на вибір було взято: Node.js, PHP, Python.

Було обрано мову програмування Python, як класичну мову для програмування ботів у Telegram.

Node.js було б непоганим рішенням, як би використовувалось Microsoft Visual Studio. Зрозуміло що такий варіант підходить лише при використанні Windows[5].

PHP надає не чимало можливостей. Насамперед ця мова достатньо популярна. Якщо щось піде не так, завжди знайдеться з ким можливо проконсультуватись. Також мова має вже готові бібліотеки для роботи з Telegram[6].

Нарешті, серед мов програмування, на яких можливо без зайвих проблем розробити бота для Telegram, Python – одне з найпопулярніших рішень. В основному це зв'язано з широкими можливостями при використанні стандартних бібліотек, як наприклад telebot[7].

2.1.2 Огляд особливостей середи розробки

Під мову Python було обрано середу розробки PyCharm Community Edition, що є зараз максимально зручною під задачі, що для неї ставляться.

Порівняльні характеристики існуючих середовищ розробки наведено у таблиці 2.1

Таблиця 2.1 – Порівняльна характеристика

Назва	Мова(и) програмування	Операційні системи	Платна
PyCharm Community Edition	Python	Windows, Mac OS X и Linux	Hi
Microsoft Visual Studio	C#, Node.js	Windows	Hi
Eclipse IDE	Java, PHP, Ruby, Python	Linux, Mac OS X, Solaris, Windows	Hi

PyCharm – дуже знайоме всім розробникам на мові Python середовище, яке не просто так завоювало свою репутацію, бо надає дійсно зручні можливості для реалізації проекту[11]:

- статичність аналіз коду, підсвічування синтаксису и помилок;
- навігація среди проектів и сирцевої коду: відображення файлової структури проекту, швидко перехід между файлами, класами и методами;
- рефакторинг: перейменування, витяг методу, введення змінної, введення констант, що Підняття и опускання методу тощо;

- інструменти для веб-розробки з Використання фреймворку Django;
- вбудований зневаджувач для Python;
- вбудовані інструменти для юніт-тестування;
- розробка з Використання Google App Engine;
- підтримка систем контролю версій: Загальний користувачський інтерфейс для Mercurial, Git, Subversion, Perforce и CVS з підтримкою списків змін та злиттів.

PyCharm Community Edition було обрано виходячи з того, що програмування буде відбуватися саме на Python. PyCharm має підтримку всього що потрібно для цілей розробки. Інтерфейс PyCharm Community Edition зображено на рисунку 2.1

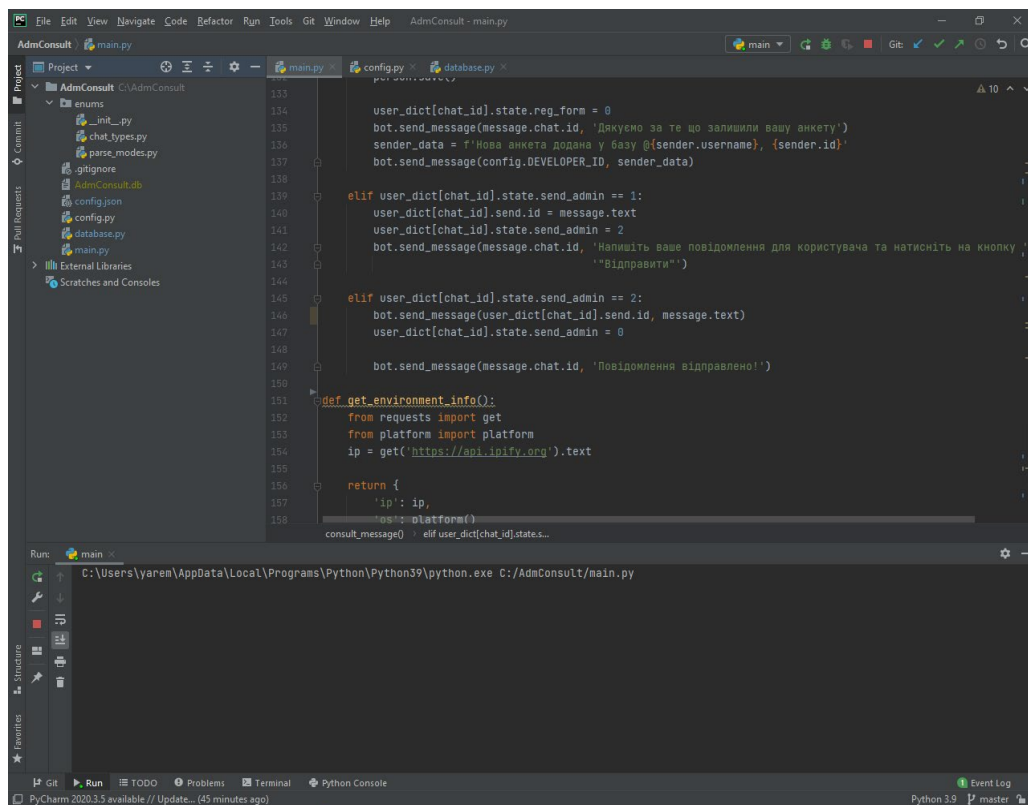


Рисунок 2.1 – Інтерфейс PyCharm Community Edition

2.2 Структура компонентів бота-консультанта

На рисунку 2.2 наведено структуру компонентів. Головними слід зазначити:

- main.py – тут описана основна логіка нашого бота, усі message_handler;
- config.py бере на себе роль центру нагалодження;
- У database.py описується логіка створення та взаємодії з базою даних, яку у подальшому ми будемо використати для зберігання даних користувачів.

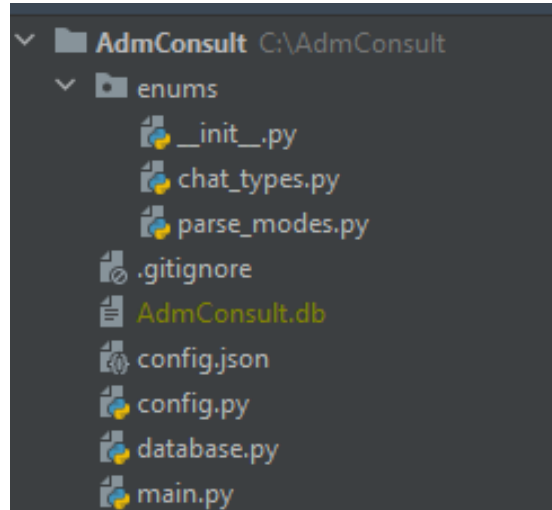


Рисунок 2.2 - Структура компонентів проекту

2.3 Опис модулів та компонентів розробленого бота

2.3.1 Класи

Можна виділити декілька класів, що розроблені для функціонування бота:

- User – Найголовніший клас, який тримає у собі основні данні користувачів. Для того, щоб бот бачив кожного користувача індивідуально від інших;
- State – клас, який служить як «state machine», для позначення стану у якому перебуває користувач у окремий момент часу користування ботом;
- Send – останній клас, який зберігає у собі id адресата, котрому адміністратор збирається відправити повідомлення.

Як було зазначено вище, найголовніший клас це User, саме через нього використовується інші класи.

2.3.2 Модулі та бібліотеки

Також можна виділити основні бібліотеки та модулі, які було використано у розробці бота:

- telebot – одна з найпопулярніших бібліотек для розробки бота у Telegram. Достатньо проста у використанні, та надає необхідні можливості для написання бота на Python[8];

- peewee – це простий і невеликий ORM. У ньому мало (але демонстративні) концепції, що полегшують навчання та інтуїтивне використання. Використовується для роботи з базою даних;[14]

- JSON – простий формат обміну даними, заснований на підмножині синтаксису JavaScript. Модуль json дозволяє кодувати і декодувати дані в зручному форматі[13];

- sys – модуль, що забезпечує доступ до деяких змінних і функцій, які взаємодіють з інтерпретатором Python[15].

3 ОСНОВНІ РІШЕННЯ ПІД ЧАС РЕАЛІЗАЦІЇ ОСНОВНИХ КОМПОНЕНТІВ ЗАСТОСУНКУ, ЩО РОЗРОБЛЯЄТЬСЯ

3.1 Сервіс BotFather

Перше що треба зробити для створення бота – отримати API Token. У месенджері Telegram це можна це зробити, скористувавшись сервісом BotFather у якому вказується назва та username бота [10]. Цей процес відображений на рисунку 3.1.

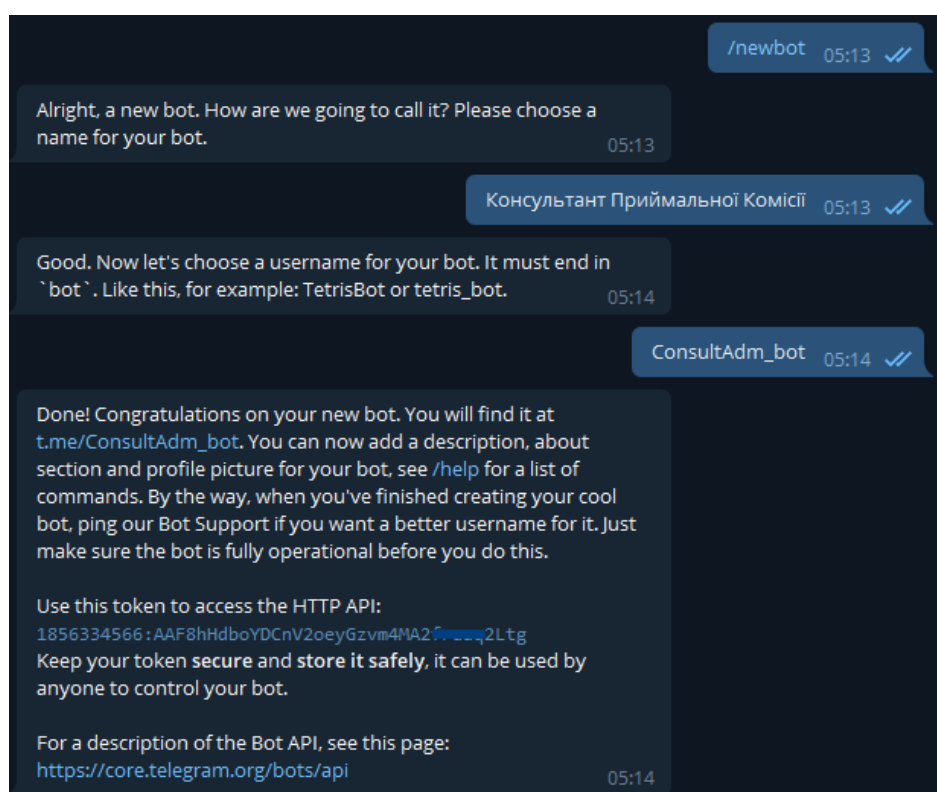


Рисунок 3.1 – Процес створення бота у BotFather

Після цього стає доступною можливість кастомізації бота, а саме: зробити боту опис, наділити бота зображенням, а також зробити повідомлення для користувачів, що перший раз відкривають бота. Для цього треба відправити команду /mybots, та обрати нашого бота.

Зображення для аватара бота було взяте з сервісу безкоштовних векторних зображень freepik.com. Зображення представлено на рисунку 3.2.

Також через сервіс BotFather був додан список команд бота, який відображаєся у впливаючому вікні бота після вводу “/”. Деталі зображені на рисунку 3.3.



Рисунок 3.2 – Аватар бота-консультанта

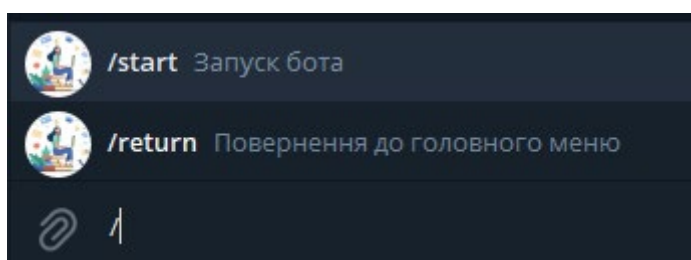


Рисунок 3.3 – Впливаюче вікно команд бота

3.2 Функціонал бібліотека telebot

Основною бібліотекою, яка використовується для написання логіки бота, буде telebot. Об'єкти, які використовуються для реалізації проекту: user, chat, message, ReplyKeyboardMarkup, KeyboardMarkup.

User - об'єкт що представляє Telegram користувача або бота. Завдяки йому ми можемо отримати id користувачів та бота, що дозволить зробити індивідуальні сесії запитів для користувачів.

Реалізація захвату команд та тексту повідомлень від користувачів за допомогою методу bot.message_handler():

@bot.message_handler(content_type=['command']) – захватує команди;

@bot.message_handler(content_type=['text']) – захватує текст повідомлень.

Відправка повідомлень ботом відбувається завдяки двом методам:

bot.send_message() – для відправки самостійного повідомлення;

bot.reply_to() – для відправлення повідомлення, яке посилається на інше.

Ще один з основних функціоналів, для якого використовується бібліотека, це створення інтерактивних кнопок, ознайомитись з якими можна на рисунку 3.4 та 3.5. А саме створення клавіш виглядає так:

markup = types.ReplyKeyboardMarkup() – створює об'єкт markup що є самою клавіатурою;

types.KeyboardButton() – створює кнопку, яка відправляє текст написаний на ній;

<Об'єкт клавіатура>.add() – використовуюця для наповнення створеної раніше клавіатури об'єктами(у данному випадку кнопки) [9].

(...)

```
markup = types.ReplyKeyboardMarkup (resize_keyboard = True)
```

```
if chat_id == config.DEVELOPER_ID:
```

```
    markup.add (
```

```
        types.KeyboardButton ('Написати в підтримку ☒'),
```

```
        types.KeyboardButton ('Залишити Анкету ☐'),
```

```
        types.KeyboardButton ('Написати листа 📧')
```

```
    )
```

```

else:
    markup.add (
        types.KeyboardButton ('Написати в підтримку 📧'),
        types.KeyboardButton ('Залишити Анкету 📄')
    )
bot.send_message (message.chat.id, text, reply_markup = markup)
(...)

```

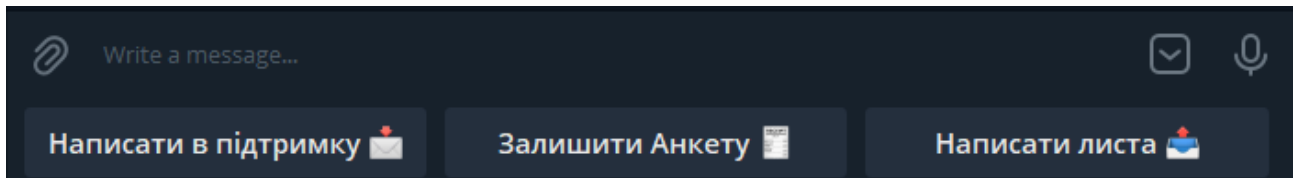


Рисунок 3.4 – Інтерфейс інтерактивних клавiш для адміністратора

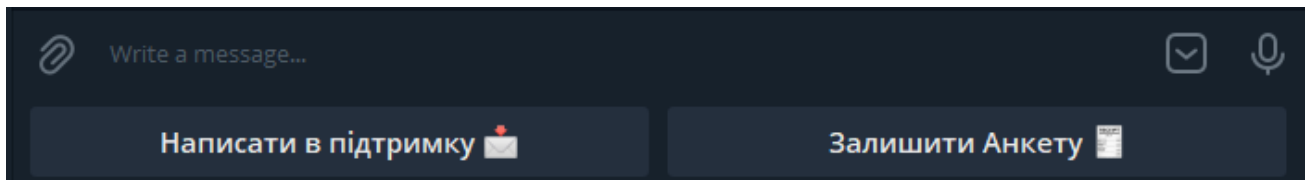


Рисунок 3.5– Інтерфейс інтерактивних клавiш для звичайних користувачів

3.3 Опис допоміжних класів

Для реалізації індивідуального використання бота-консультанта знадобилось створити 5 класів: User, State, Send, BaseModel, Preson.

Як було сказано раніше, головний допоміжний клас це User. Інформація що зберігаєця у ньому це id користувача ботом-консультантом, та його username у месенджері Telegram(рисунок 3.6).

```

class User:
    def __init__(self, tg_id, username):
        self.tg_id = tg_id
        self.username = username
        self.state = State()
        self.send = Send()

```

Рисунок 3.6 – Клас User

Другий клас це State. Як було сказано раніше, він бере на себе функцію state machine. Зберігає у собі стан кожного з сценаріїв які запускаюця після натискання на інтерактивні кнопок(рисунок 3.7). Усього параметрів у класі 3, для кожного з сценаріїв інтерактивних кнопок.

```
class State:
    def __init__(self):
        self.send = False
        self.reg_form = 0
        self.send_admin = 0
```

Рисунок 3.7 – Клас State

Третій клас Send найменший. Він створений для того, щоб зберігати у собі id користувача, якому адміністратор бажає відправити повідомлення за допомогою інтерактивної кнопки «Написати листа » (рисунок 3.8).

```
class Send:
    def __init__(self):
        self.id = {}
```

Рисунок 3.8 – Клас Send

Четвертий та п'ятий класи створені для таблиці та бази даних. Клас Preson (рисунок 3.9) спадкує клас BaseModel. А також має атрибути telegram_id, first_name, last_name, phone_number. Діаграма класів зображена на рисунку 3.10.

```
class Person(BaseModel):
    telegram_id = IntegerField(primary_key=True)
    first_name = CharField(null=True)
    last_name = CharField(null=True)
    phone_number = BlobField(null=True)
```

Рисунок 3.9– Клас Person

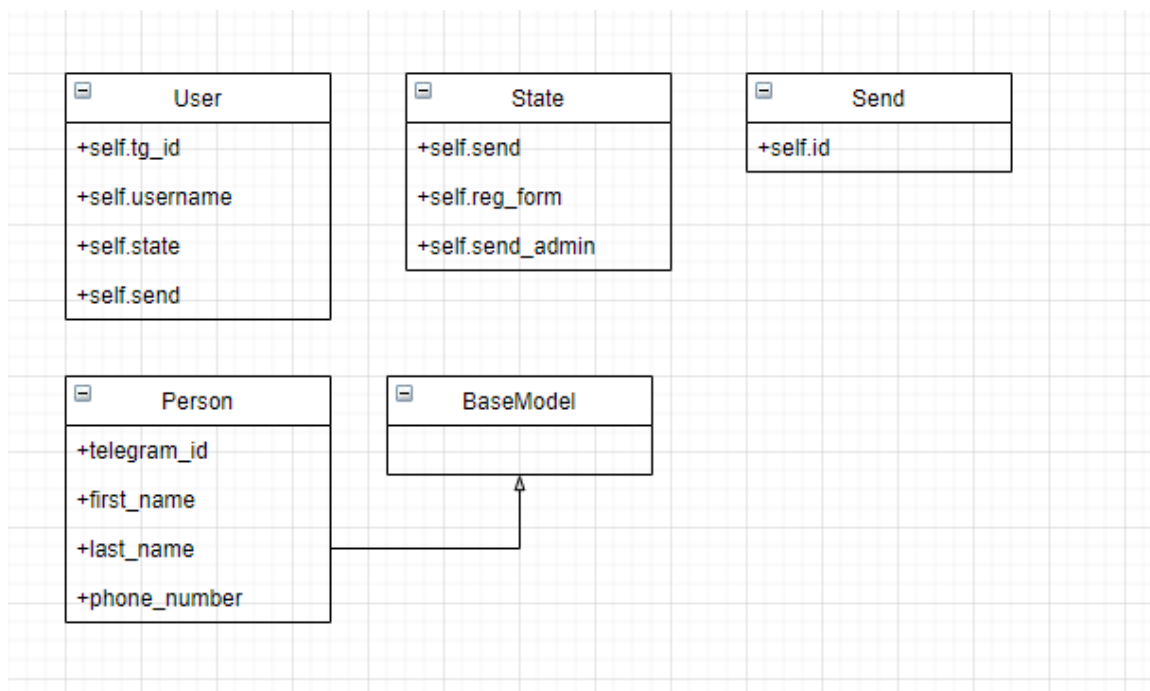


Рисунок 3.10 – Діаграма класів

3.4 Використання бази даних

3.4.1 Основа бази

SQL — декларативна мова програмування для взаємодії користувача з базами даних, що застосовується для формування запитів, оновлення і керування реляційними базами даних, створення схеми бази даних та її модифікації, системи контролю за доступом до бази даних. Сама по собі SQL не є ані системою керування базами даних, ані окремим програмним продуктом. На відміну від дійсних мов програмування (C або Pascal), SQL може формувати інтерактивні запити або, будучи вбудованою в прикладні програми, виступати як інструкції для керування даними. Окрім цього, стандарт SQL містить функції для визначення зміни, перевірки та захисту даних[17].

Для збереження даних анкети, яку заповнює користувач, використовується база даних на основі SQL. Для зручнішої роботи з базою

даних використовується reeewe. Це простий і невеликий ORM, що полегшує роботу з базою даних[14].

У створеної таблиці є чотири стовпці у які будуть записуватись данні:

- telegram_id – записується id аккаунта користувача, що виступає як Primary key. Тип даних стовбця – integer.

- first_name – записується ім'я, яке користувач вводить власноруч у сценарії запущеному при натисканні «Залишити Анкету ☐». Тип даних стовбця – varchar.

- last_name – записується прізвище, яке користувач вводить власноруч у тому самому сценарії, що і first_name. Тип даних – varchar.

- phone_number – записується цифри телефонного номеру користувача, які він вводить власноруч у тому самому сценарію, що й попередні дані.

3.4.2 SQLiteStudio

SQLite - полегшена реляційна система управління базами даних. Реалізована у вигляді бібліотеки, де втілено багато з стандарту SQL-92. Вихідний код SQLite поширюється як суспільне надбання, тобто може використовуватися без обмежень і безкоштовно з будь-якою метою. Особливістю SQLite є те, що вона не використовує парадигму клієнт-сервер, тобто рушій SQLite не є окремим процесом, з яким взаємодіє застосунок, а надає бібліотеку, з якою програма компілюється і рушій стає складовою частиною програми. Кілька процесів або потоків можуть одночасно без жодних проблем читати дані з однієї бази. Завдяк архітектурі рушія можливо використовуват SQLite як на вбудовуваних (embedded) системах, так і на виділених машинах з гігабайтними масивами даних[16].

SQLiteStudio використовується для менеджменту бази даних проекту. Вона дає можливість перегляду та ручного редагування бази даних, у якій користувачі бота-консультанта залишають свої дані коли заповнюють анкети.

Перегляд даних зображено на рисунку 3.11. Для окремих випадків можливо вручну редагувати структуру бази даних за допомогою SQLiteStudio у вкладці «Структура» (рисунок 3.12).

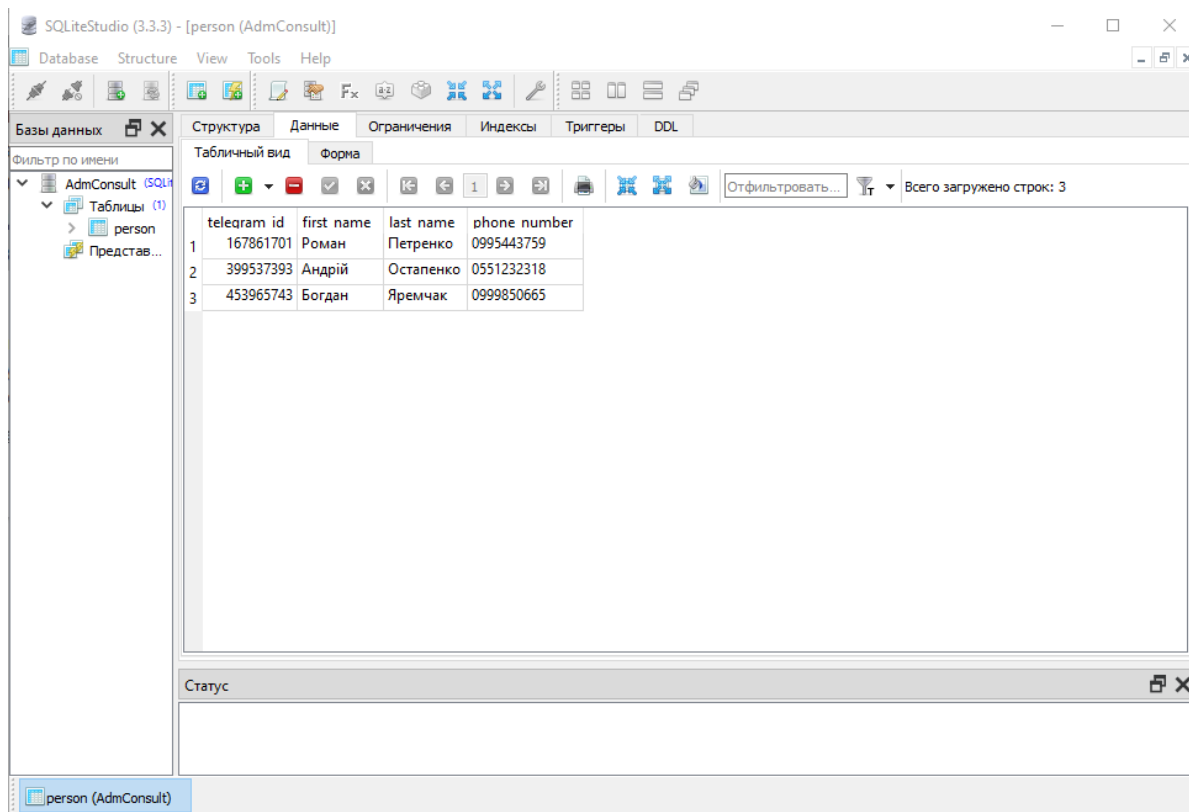


Рисунок 3.11 – Перегляд даних у SQLiteStudio

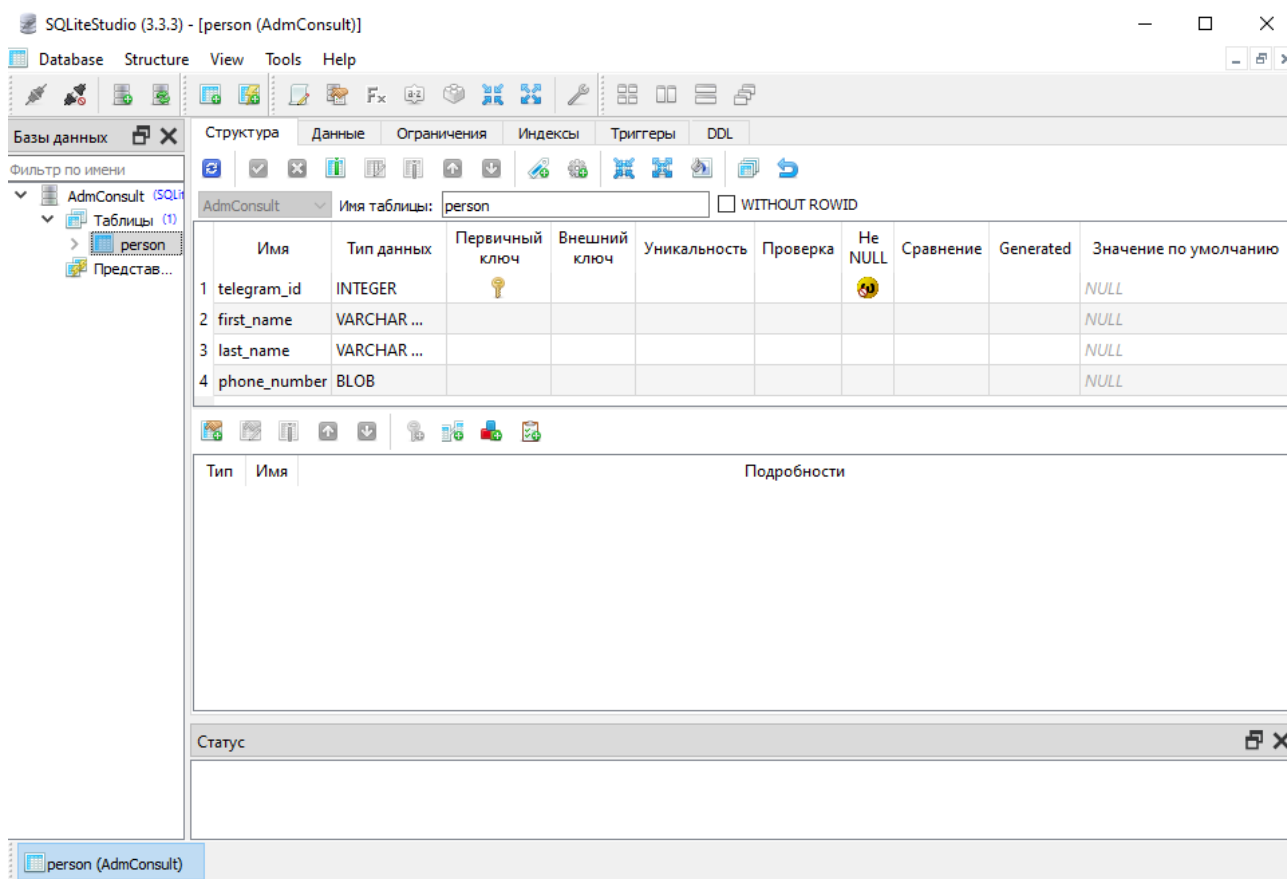


Рисунок 3.12 – Вкладка «Структура»

3.4.3 Работа з базою

Після створення бази даних та класу Person, який відповідає за таблицю, у процесі роботи з ботом-консультантом йде заповнення таблиці. Спочатку йде перевірка primary key у таблиці, якщо сходжень нема, тоді створю новий рядок. Так як індивідуальний id телеграм аккаунта користувача виступає як primary key, випадковий збіг у різних користувачів неможливий. Саме створення рядку має такий вид:

```
person, created = Person.get_or_create(telegram_id = sender.id).
```

Зберігання інших даних відбувається звертанням до відповідного рядку за допомогою primary key. Далі йде зміна значення у відповідному атрибуті та збереження у базі даних. Має ость такий вид:

- збереження first_name

```
person = Person.get(telegram_id = sender.id)
    person.first_name = message.text
    person.save();
```

- збереження last_name

```
person = Person.get(telegram_id = sender.id)
    person.last_name = message.text
    person.save();
```

- збереження phone_number

```
person = Person.get(telegram_id = sender.id)
    person.phone_number = message.text
    person.save().
```

4 КЕРІВНИЦТВО ПРОГРАМІСТА

4.1 Алгоритми послідовностей

4.1.1 Алгоритм основи

Для розуміння того, як працює бот-консультант, було розроблено алгоритм послідовностей, який демонструє логіку коду.

Алгоритм наведено на рисунку 4.1.

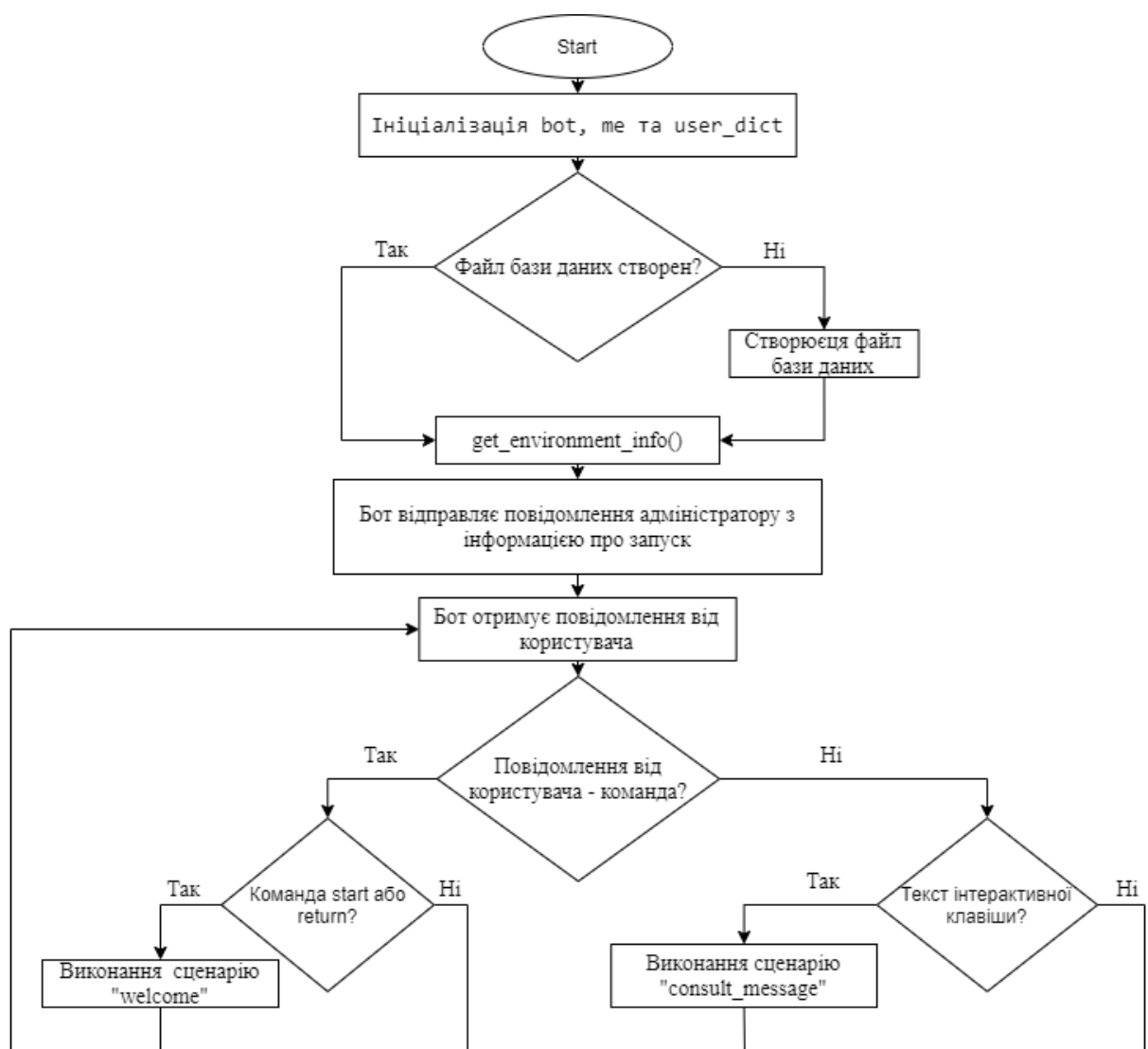


Рисунок 4.1– Алгоритм роботи бота

Опис алгоритму послідовностей:

Крок 1. Запуск бота.

Крок 2. Ініціалізується змінні `bot`, `me` та `user_dict`.

Крок 3. Запускається перевірка на наявність створеного файлу бази даних. Якщо файл відсутній, тоді створюється новий.

Крок 4. Виконання процедури `get_environment_info()`. В ній отримується інформація про ір-адресу та платформу з якої відбувся запуск бота-консультанта.

Крок 5. Далі бот відправляє адміністратору повідомлення, у якому міститься інформація отримана у минулому пункті(рисунок 4.2)

Крок 6. На цьому етапі бот переходить у режим очікування та чекає поки користувач відправить повідомлення або команду у чат бота.

Крок 7. Якщо це була команда із списку доступних, тоді відбувається сценарій цієї команди. Якщо бот отримує текст однієї з інтерактивних клавіш, тоді відбувається один із сценаріїв які запускає певна клавіша. Після завершення сценаріїв бот повертається до режиму очікування.

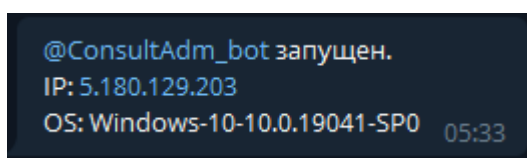


Рисунок 4.2 – Повідомлення про запуск бота

4.1.2 Алгоритми сценаріїв

Для наглядності логіки бота-консультанта, було наведено алгоритми сценаріїв «welcome» та «consult_message» на рисунках 4.3 та 4.4.

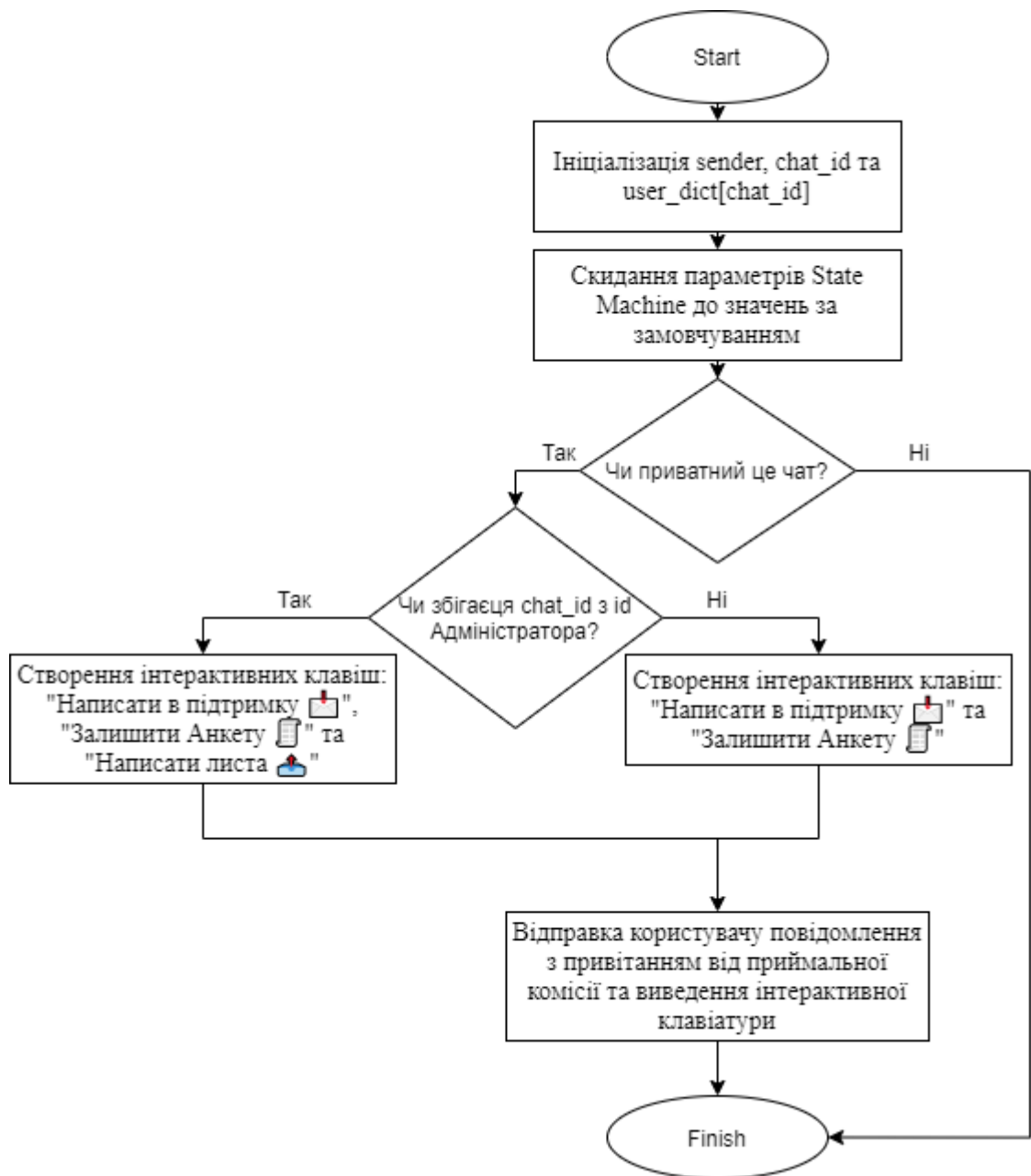


Рисунок 4.3 – Алгоритм сценарію «welcome»

Опис алгоритму послідовностей до рисунку 4.3:

Крок 1. Запуск сценарію.

Крок 2. Ініціалізуються змінні sender, chat_id та user_dict[chat_id].

Крок 3. Параметри для State Machine користувача скидаються до значень за замовчуванням:

- user_dict[chat_id].state.reg_form = 0;
- user_dict[chat_id].state.send = False;
- user_dict[chat_id].state.send_admin = 0.

Крок 4. Перевірка на приватність чату, якщо чат приватний, сценарій продовжується. Якщо чат виявиться не приватним, сценарій закінчується.

Крок 5. Перевірка аккаунту користувача. Перевіряється чи співпадає id користувача з id адміністратора, яке було введено у файлі config.json. Якщо так, тоді створюються три інтерактивні клавіши:

- "Написати в підтримку ✉";

- "Залишити Анкету □";

- "Написати листа 📧".

Якщо ні, тоді лише дві:

- "Написати в підтримку ✉";

- "Залишити Анкету □".

Крок 6. Бот відправляє користувачу повідомлення «Вас вітає підтримка приймальної комісії». Також у інтерфейсі користувача з'являється інтерактивна клавіатура з певним набором інтерактивних клавіш згідно з пунктом вище.

Крок 7. Сценарій закінчується.

Алгоритм роботи сценарію «consult_message» зображено на рисунку 4.4.

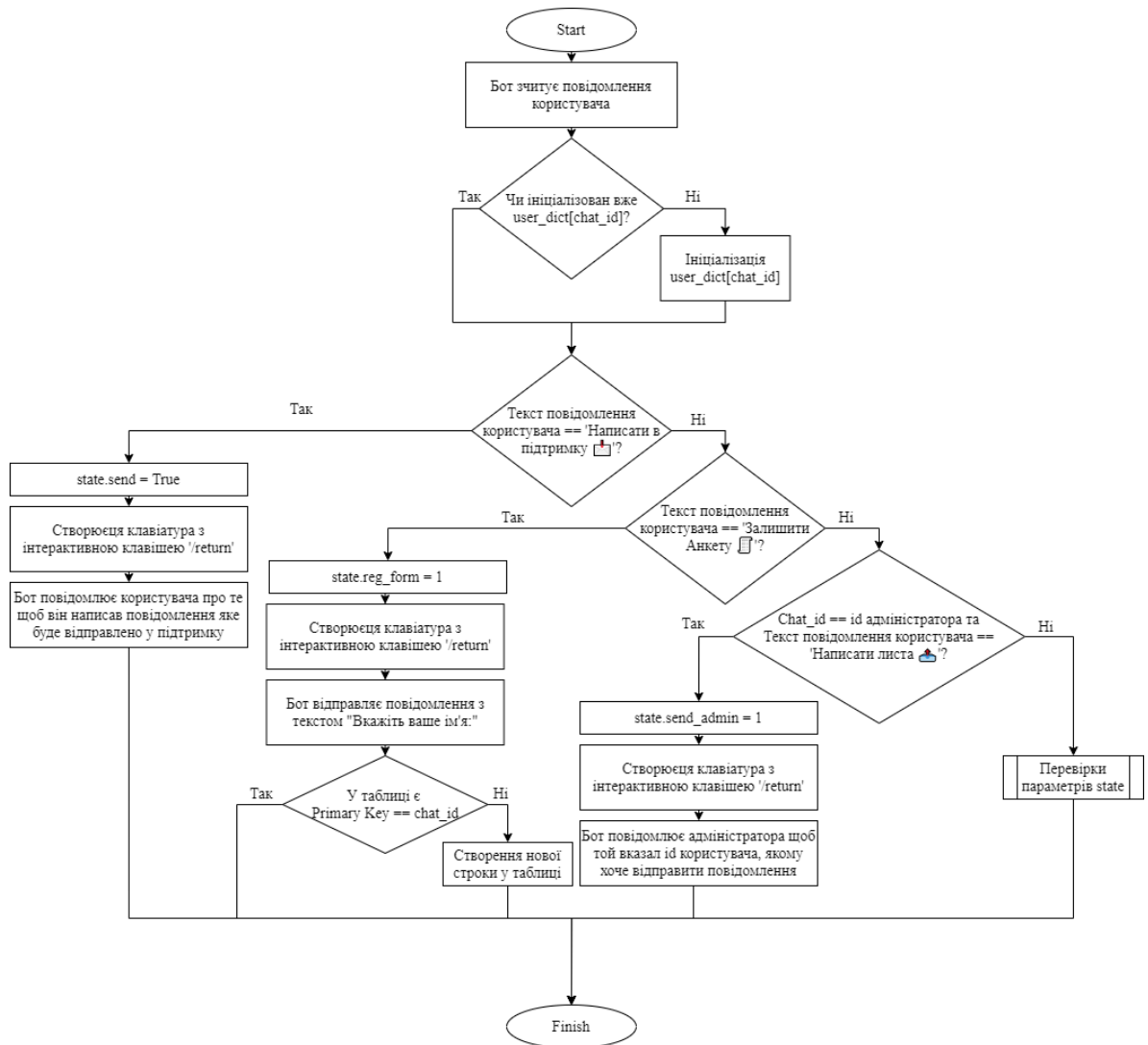


Рисунок 4.4 - Алгоритм сценарію «consult_message»

Опис алгоритму послідовностей до рисунку 4.4:

Крок 1. Запуск сценарію.

Крок 2. Бот зчитує повідомлення користувача.

Крок 3. Перевірка, чи ініціалізовано вже user_dict[chat_id]? Якщо ні, тоді ініціалізує.

Крок 4. Перевірка, чи співпадає текст у повідомленні користувача з «Написати в підтримку» ☒. Якщо так, тоді:

Крок 4.1. user_dict[chat_id].state.send = True.

Крок 4.2. У інтерфейсі користувача створюєся інтерактивна клавіша «/return».

Крок 4.3. Бот відправляє користувачу повідомлення з текстом «Напишіть ваше повідомлення, відправте його боту, а після цього натисніть на кнопку "Відправити". Підтримка незабаром відповість вам.».

Крок 5. Перевірка, чи співпадає текст у повідомленні користувача з «Залишити Анкету ☐». Якщо так, тоді:

Крок 5.1. `user_dict[chat_id].state.reg_form = 1.`

Крок 5.2. У інтерфейсі користувача створюється інтерактивна клавіша «`/return`».

Крок 5.3. Бот відправляє користувачу повідомлення з текстом «Вкажіть ваше ім'я:».

Крок 5.4. Виконання функції `Person.get_or_create(telegram_id=sender.id)`

Крок 6. Перевірка, чи співпадає id користувача з id Адміністратора та текст у повідомленні користувача з «Написати листа ». Якщо так, тоді:

Крок 6.1. `user_dict[chat_id].state.send_admin = True.`

Крок 6.2. У інтерфейсі користувача створиться інтерактивна клавіша «`/return`».

Крок 6.3. Бот відправляє користувачу повідомлення з текстом «Вкажіть id користувача якому хочете відправити повідомлення:».

Крок 7. Йдуть перевірки `user_dict[chat_id].state`. Алгоритм їх роботи наведено на рисунку 4.5.

Крок 8. Сценарій закінчується.

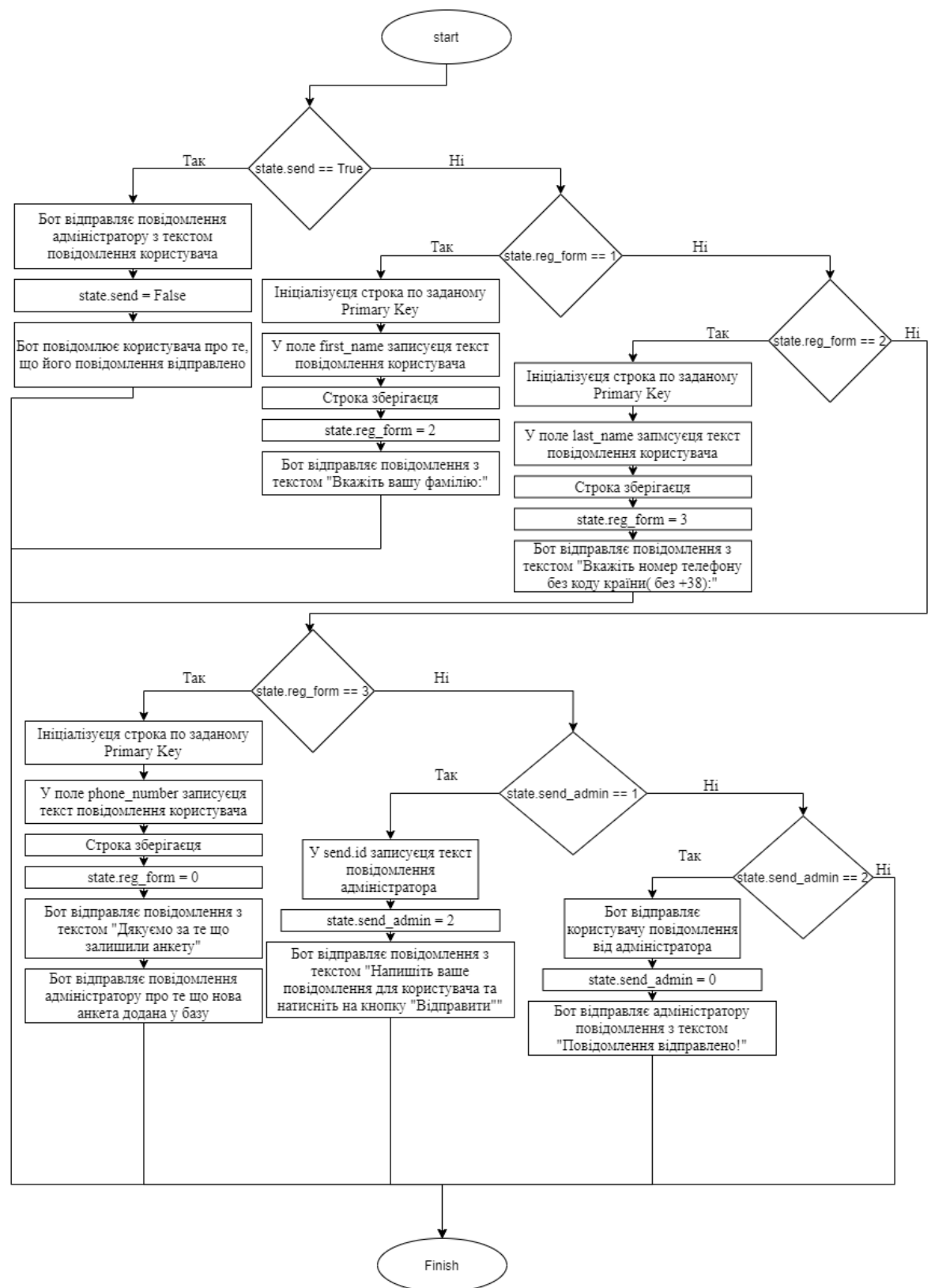


Рисунок 4.5 – Алгоритм роботи перевірок user_dict[chat_id].state

Опис алгоритму послідовностей до рисунку 4.5:

Крок 1. Запуск алгоритму перевірок.

Крок 2. Перевірка, `user_dict[chat_id].state.send == True`, якщо так:

Крок 2.1. Бот відправляє повідомлення адміністратору з текстом який написав користувач.

Крок 2.2. `user_dict[chat_id].state.send = False`.

Крок 2.3. Бот відправляє повідомлення користувачу з текстом «Ваш лист відправлено. Незабаром підтримка відповість вам.

Крок 3. Перевірка, `user_dict[chat_id].reg_form == 1`, якщо так:

Крок 3.1. Проходить ініціалізація певного рядку у таблиці бази даних. Пошук потрібного рядку відбувається по `id` користувача, який виступає за Primary Key.

Крок 3.2. `First_name` у ініціалізованого рядку змінюється на текст повідомлення, яке написав користувач.

Крок 3.3. Зберігання змін строки у базу даних.

Крок 3.4. `user_dict[chat_id].state.reg_form = 2`.

Крок 3.5. Бот відправляє користувачу повідомлення з текстом «Вкажіть ваше прізвище:».

Крок 4. Перевірка, `user_dict[chat_id].reg_form == 2`, якщо так:

Крок 4.1. Проходить ініціалізація певного рядку у таблиці бази даних. Пошук потрібного рядку відбувається по `id` користувача, який виступає за Primary Key.

Крок 4.2. `Last_name` у ініціалізованого рядку змінюється на текст повідомлення, яке написав користувач.

Крок 4.3. Зберігання змін строки у базу даних.

Крок 4.4. `user_dict[chat_id].state.reg_form = 3`.

Крок 4.5. Бот відправляє користувачу повідомлення з текстом «Вкажіть номер телефону без коду країни(без +38):».

Крок 5. Перевірка, `user_dict[chat_id].reg_form == 3`, якщо так:

Крок 5.1. Проходить ініціалізація певного рядку у таблиці бази даних. Пошук потрібного рядку відбувається по id користувача, який виступає за Primary Key.

Крок 5.2. Phone_number у ініціалізованого рядку змінюється на текст повідомлення, яке написав користувач.

Крок 5.3. Зберігання змін рядку у базу даних.

Крок 5.4. `user_dict[chat_id].state.reg_form = 0`.

Крок 5.5. Бот відправляє користувачу повідомлення з текстом «Дякуємо за те що залишили анкету».

Крок 5.6. Бот відправляє адміністратору повідомлення про те що нова анкета додана у базу(рисунок 4.6)

Крок 6. Перевірка, `user_dict[chat_id].send_admin == 1`, якщо так:

Крок 6.1. Запис у `user_dict[chat_id].send.id` текст повідомлення від адміністратора.

Крок 6.2. `user_dict[chat_id].send_admin = 2`.

Крок 6.3. Бот відправляє адміністратору повідомлення з текстом «Напишіть ваше повідомлення для користувача та натисніть на кнопку «Відправити»».

Крок 7. Перевірка, `user_dict[chat_id].send_admin == 2`, якщо так:

Крок 7.1. Бот відправляє користувачу, id якого було записано у `user_dict[chat_id].send.id`, повідомлення від адміністратора.

Крок 7.2. `user_dict[chat_id].send_admin = 0`.

Крок 7.3. Бот відправляє адміністратору повідомлення з текстом «Повідомлення відправлено!».

Крок 8. Завершення алгоритму перевірок.

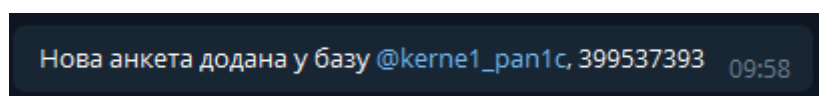


Рисунок 4.6 – Приклад повідомлення для адміністратора

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕСКПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

5.1 Тестування бота-консультанта

Було проведено повне тестування бота-консультанта, як зі сторони адміністратора, так і зі сторони звичайного користувача. Перевірен функціонал та працездатність усіх сценаріїв.

Спочатку було проведене тестування зі сторони звичайного користувача. Сценарій «welcome» працює справно, результат можливо побачити на рисунку 5.1.

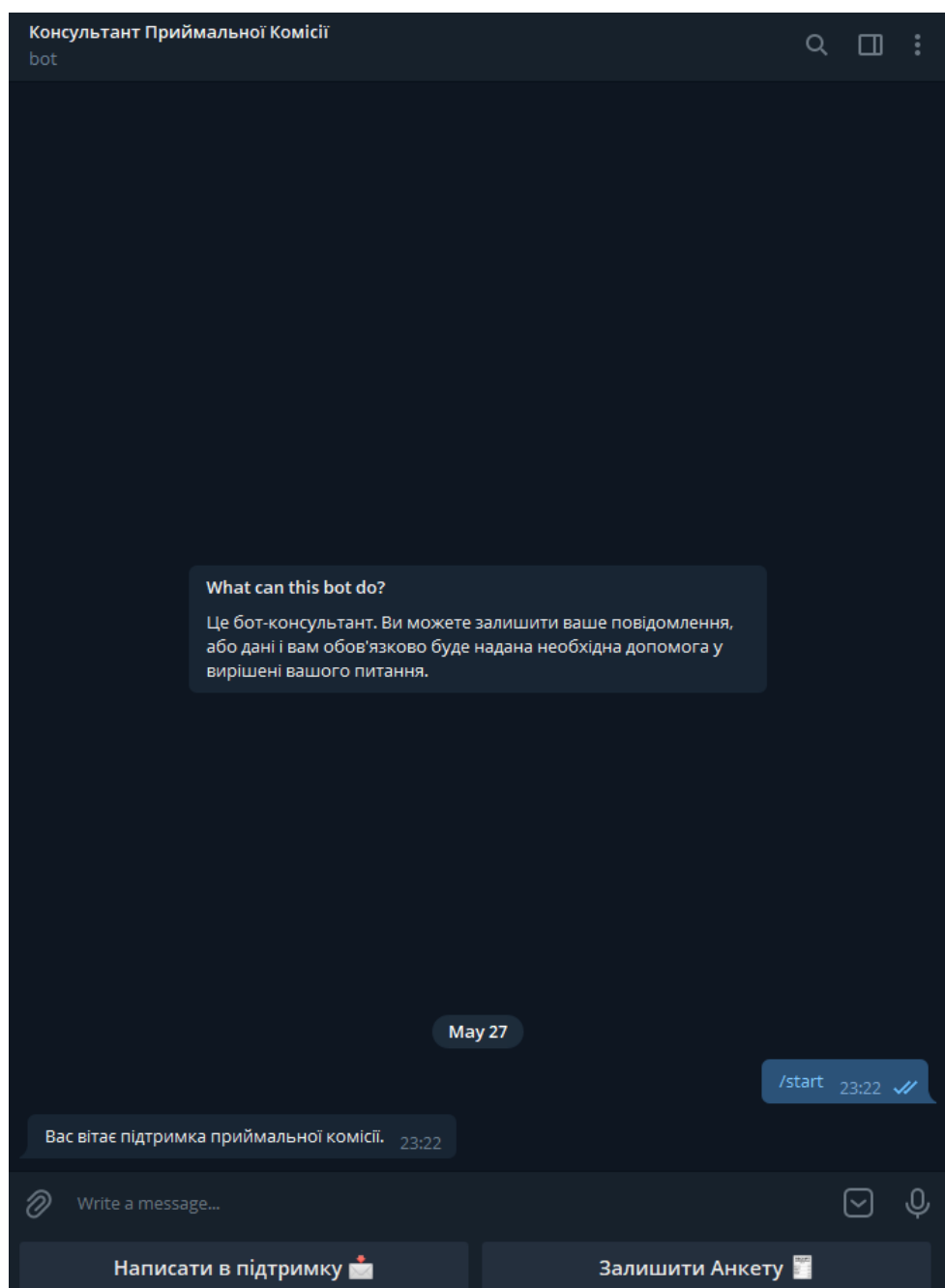


Рисунок 5.1 – Результат сценарію «welcome»

Далі було перевірено сценарій, який запускається інтерактивно клавішою «Залишити Анкету ☐». Сценарій також працює без помилок. Параметр `state.reg_form` змінюється вправно, просуваючи процес послідовно, як і було потрібно. У результаті чого усі введені дані було записано до бази даних. Також повідомлення про нову анкету у базі даних успішно доставлено адміністратору. Після виконання сценарію залишається доступна інтерактивна клавіша «/return», яка запускає сценарій «welcome», чим повертає користувача

до головного меню бота-консультанта. Результати зображено на рисунках 5.2 та 5.3.

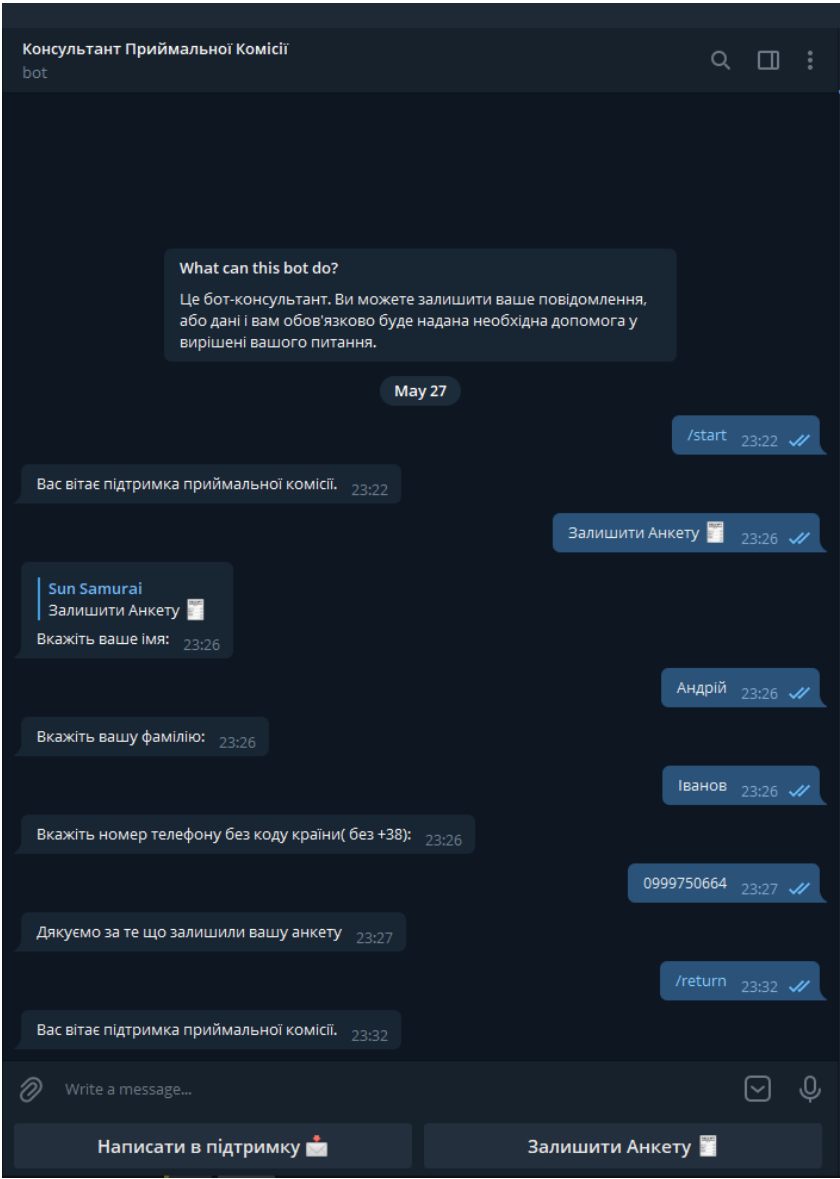


Рисунок 5.2 – Результат у вікні чату користувача

Структура					Данные					Ограничения					Индексы					Триггеры				
Табличный вид					Форма																			
1					telegram_id	first_name	last_name	phone_number																
					701138562	Андрій	Іванов	0999750664																

Рисунок 5.3 – Результат нового запису у базі даних

З акаунту звичайного користувача залишилось зробити ще один тест, сценарію, який запускає інтерактивна клавіша «Написати в підтримку ✉». У ході теста не було виявлено помилок. Повідомлення з текстом, доставленим користувачу, доставляється адміністратору, разом з id та username користувача. Результат зображено на рисунку 5.4.

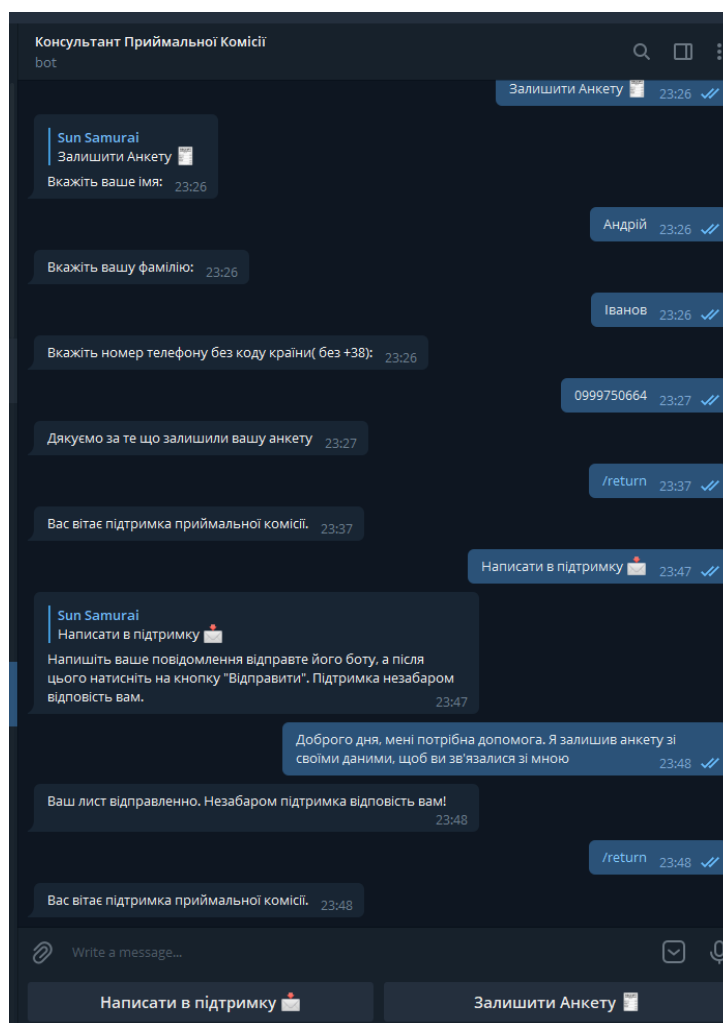


Рисунок 5.4 – Результат у чаті користувача

Наступним етапом тестування був вход у бота-консультанта з аккаунта адміністратора. Одразу видно повідомлення від бота, які підтверджують що минулі етапи тестування на аккаунті користувача пройшли успішно(рисунки 4.2.5). Після запуску команди «/start», створюється вже не дві інтерактивні

клавіши, а й також створюється додаткова «Написати листа », яка доступна лише адміністратору(рисунок 5.5).

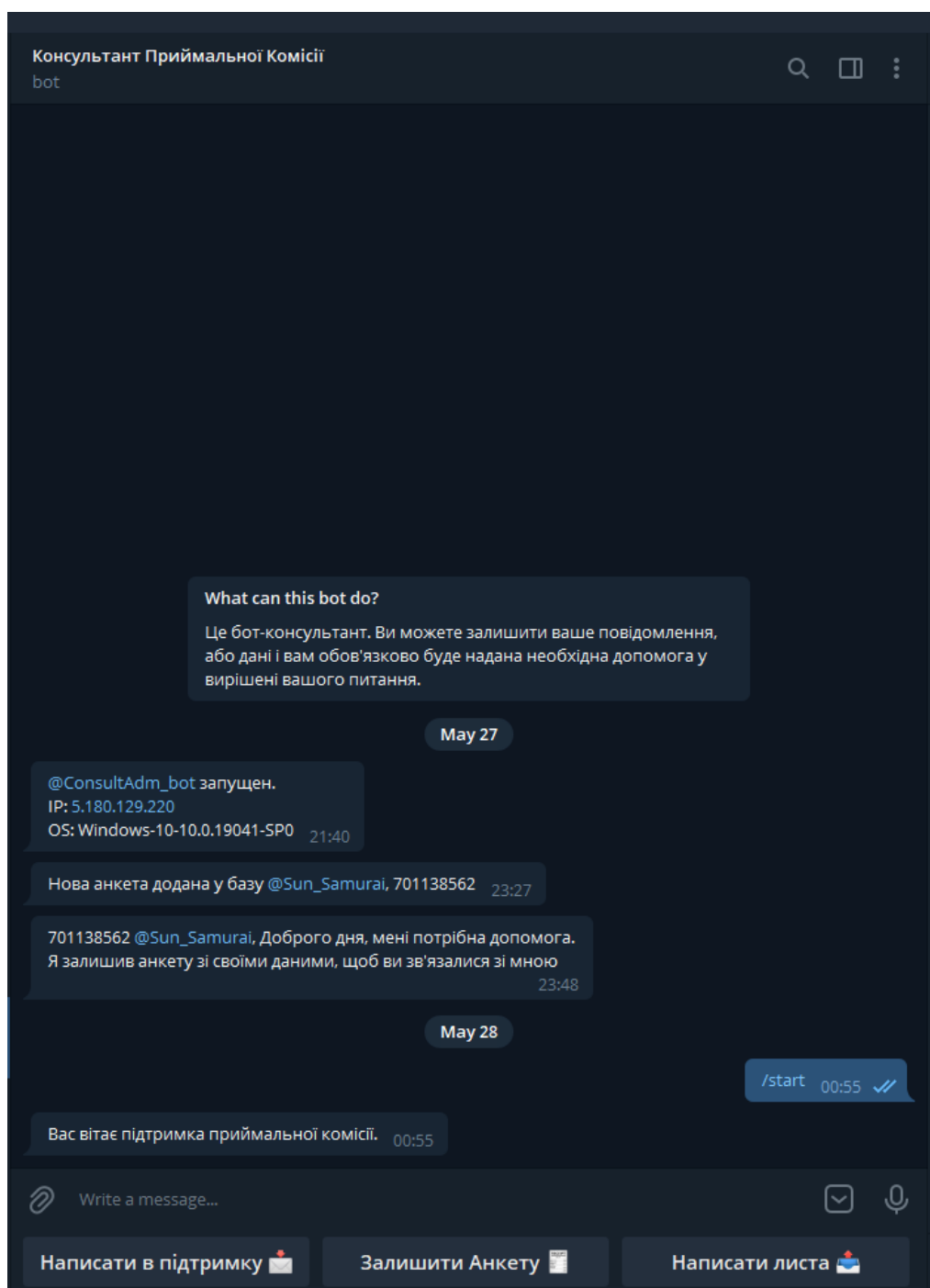


Рисунок 5.5 – Чат адміністратора

Наступним етапом тестування була перевірка сценарію при натисканні інтерактивної кнопки «Написати листа ». На останньому етапі також не було

виявлено помилок. Бот-консультант відправляє повідомлення від адміністратора певному користувачу, ід якого було взяте з технічних повідомлень вище та вказано адміністратором. Результат цього тестування зображено на рисунках 5.6 та 5.7.

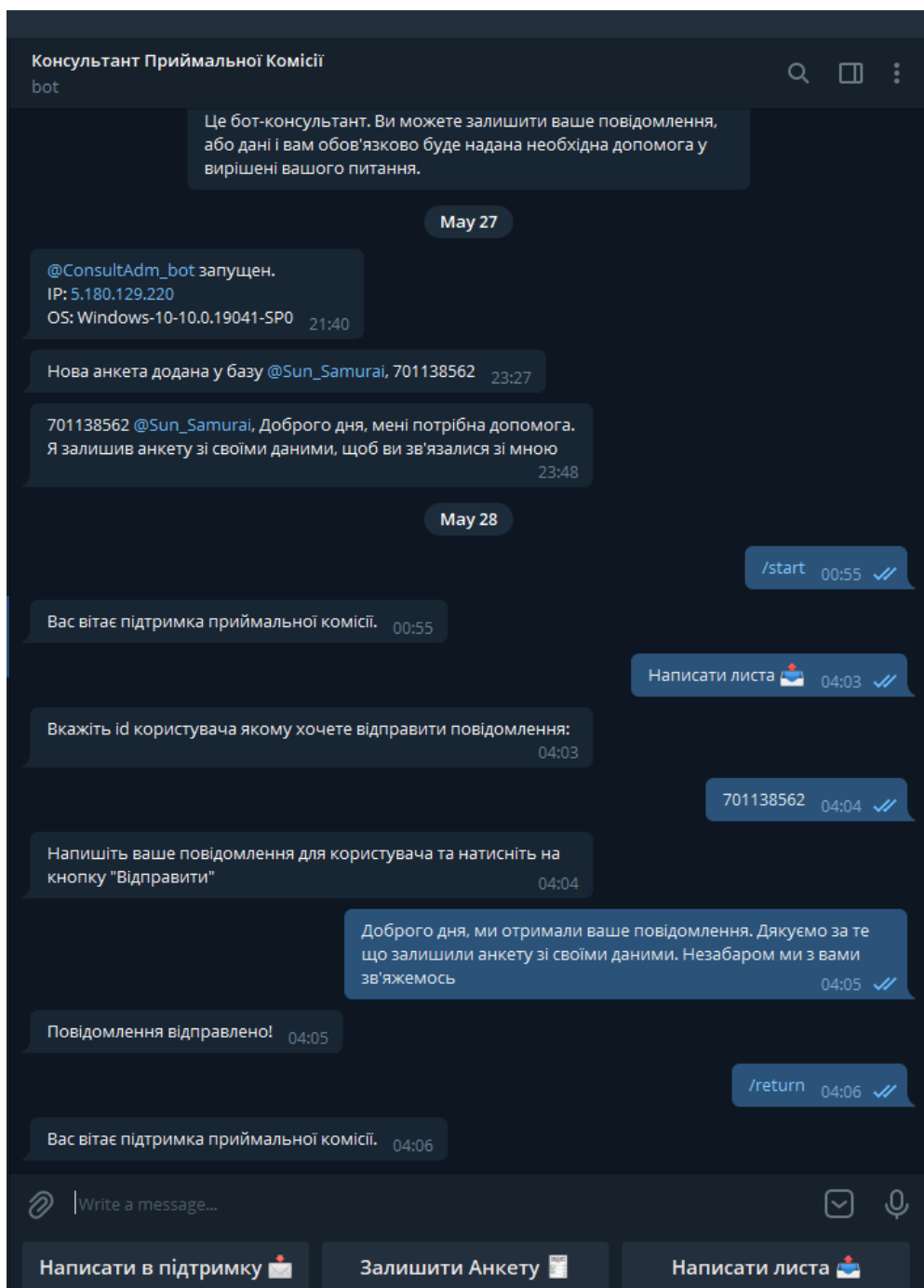


Рисунок 5.6 – Зображення чату адміністратора

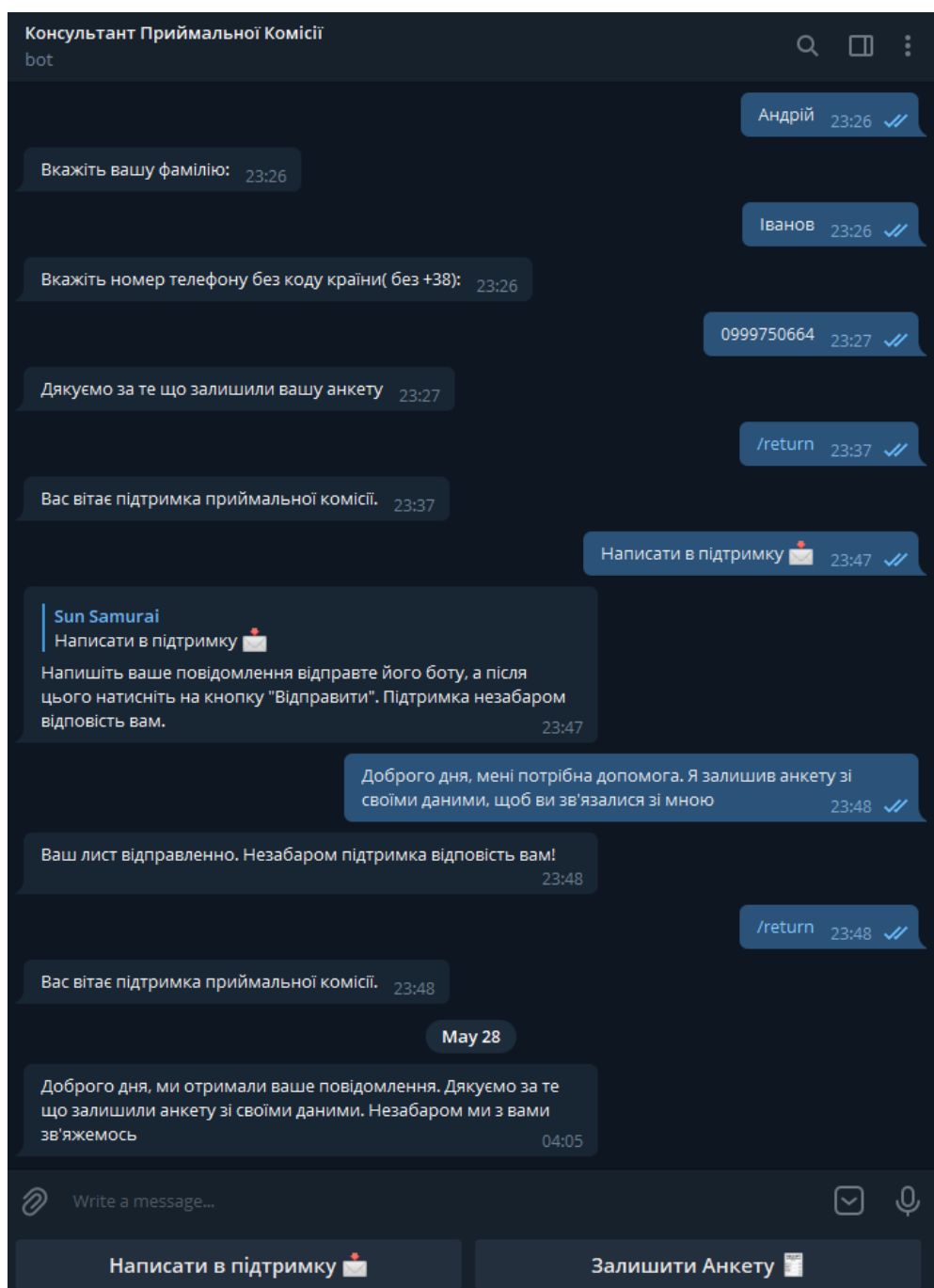


Рисунок 5.7 – Зображення чату користувача

Також була проведена перевірка одночасного користування ботом декількома користувачами, яка також пройшла без помилок, так як для кожного користувача створюється свій профіль всередині бота-консультанта.

Після усіх етапів тестування, які було пройдено без помилок, можна зробити висновок, що бот-консультант працює так, як і задумувався:

- усі сценарії є працездатними, та працюють без помилок;

- поділення функціоналу бота-консультанта для адміністратора та звичаного користувача зроблено;
- зберігання інформації у базу даних виконано;
- користування ботом-консультантом водночас декількома користувачами є можливим завдяки існуванню внутрішніх профілів для кожного окремого користувача.

Останнім етапом тестування була перевірка того як веде себе інтерфейс на різних платформах. Вигляд інтерфейсу під Windows та Android наведено на рисунках 5.8 та 5.9.

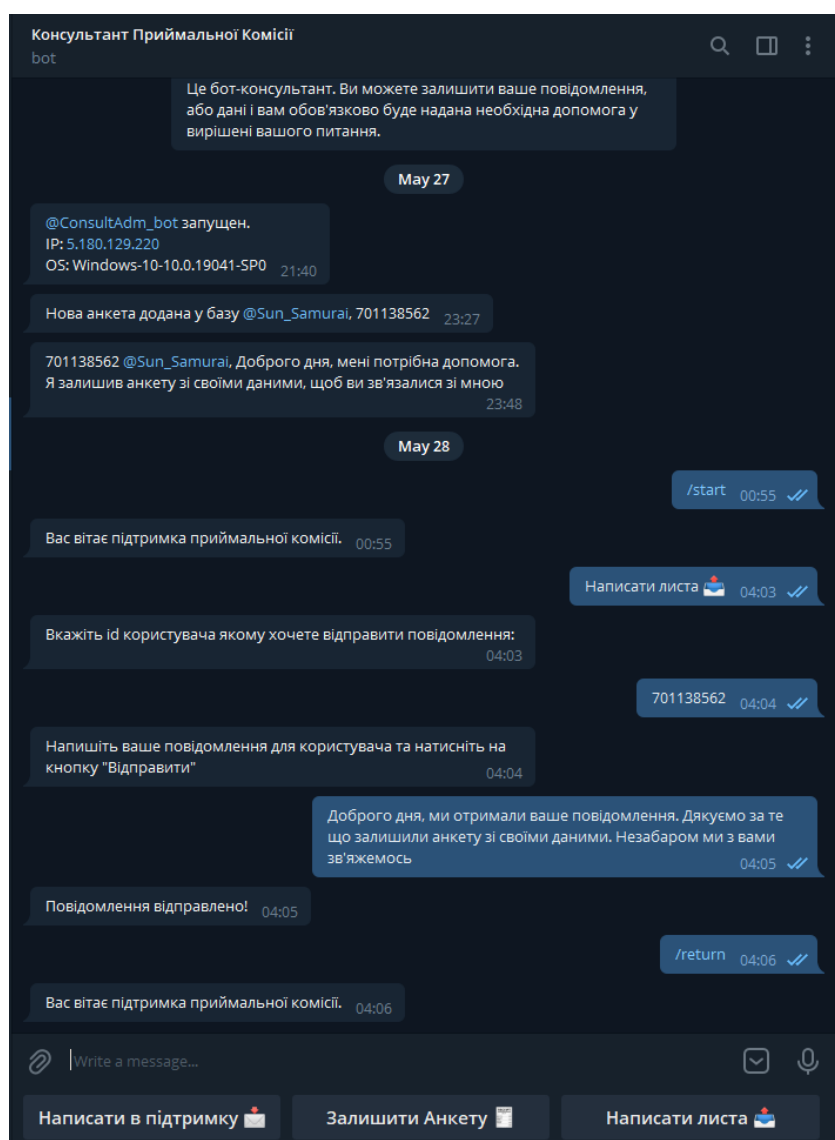


Рисунок 5.8 – Приклад інтерфейсу на Windows

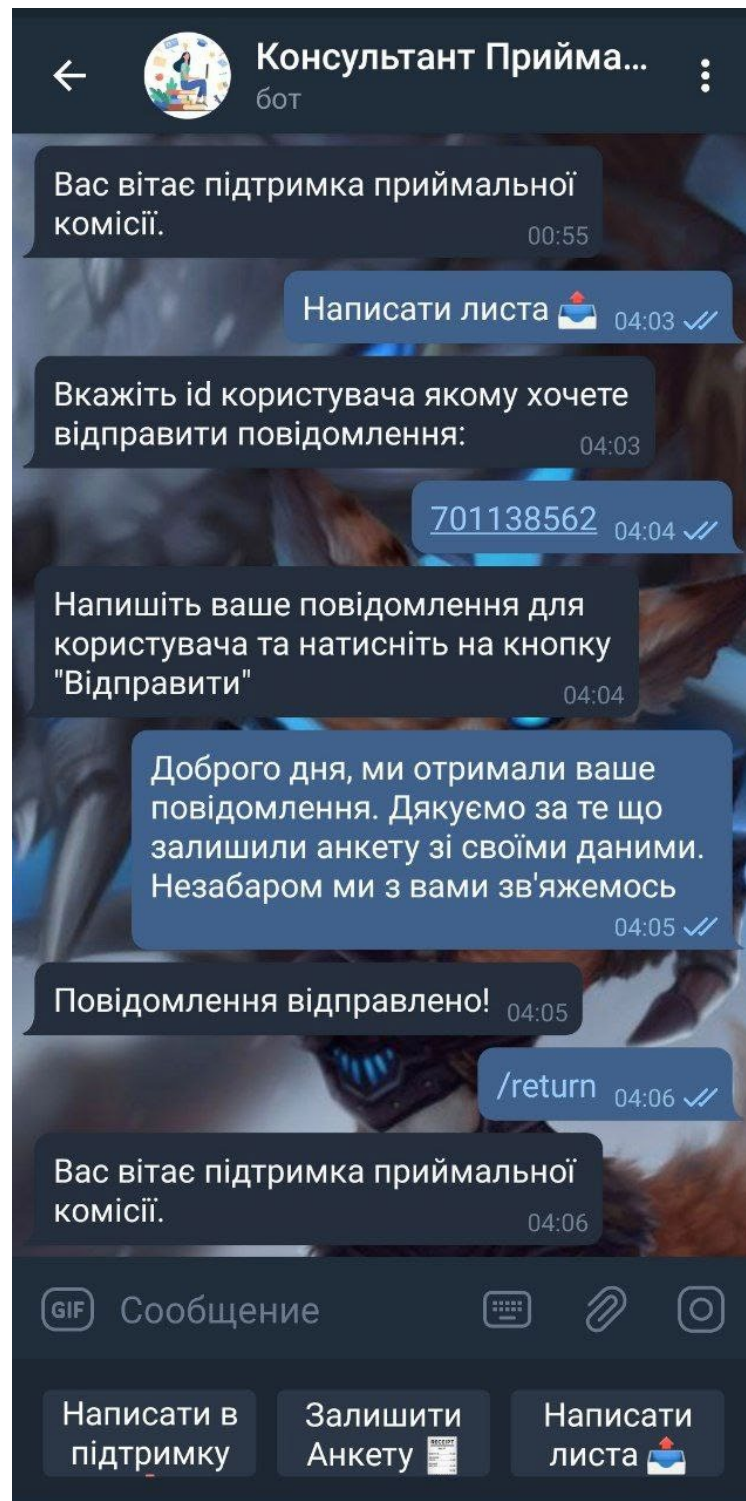


Рисунок 5.9 – Приклад інтерфейсу на Android

ВИСНОВКИ

Під час розробки бота-консультанта було ознайомлено з принципами та основами сучасного проектування ботів для месенджера Telegram за допомогою мови програмування Python. Було розширено існуючі знання та навички написання програмного коду.

Для розробленого бота-консультанта було створено репозиторій на GitHub: https://github.com/BoYaReN/ConsultAdm_bot

Одна з найголовніших цілей була ознайомитися та опанувати функції роботи з бібліотекою telebot, що й було виконано та продемонстровано у дипломній роботі під назвою “Бот-консультант приймальної комісії”.

Головна перевага проектування бота-консультанта для месенджера Telegram полягає у тому, що користуватися функціоналом бота-консультанта можливо на будь якій платформі, яка підтримує месенджер Telegram: Android, IOS, Windows Phone, Windows, macOS, Chrome OS, GNU/Linux.

Таким чином, якщо брати, наприклад, мобільні платформи та їх популярність у наш час, можливість скористуватися послугами бота-консультанта приймальної комісії є дуже доступною. Що до інтерфейсу на різних платформах, він адаптивен під кожен.

Під час розробки був більш ознайомлений у роботі з базами даних. А саме функціоналом реєстрації, що у разі спрощує роботу з базою даних на основі SQL. Основною перевагою є те, що реєстрація дуже зручно параметризує запити.

Удосконалено навички проектування діаграм класів та послідовностей.

Шляхи удосконалення застосування:

- реалізація обміну зображеннями, відео та аудіо матеріалами;
- реалізація можливості додати більше одного аккаунту адміністрації;
- реалізація автоматичного розсилу повідомлень від адміністрації усім користувачам залишившим анкету.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Бот-консультант Comfy.ua[Telegram бот]. - Режим доступа:
<https://t.me/ComfyuaBot>.
2. Бот-консультант monobank[Telegram бот]. - Режим доступа:
<https://t.me/monobankbot>.
3. Бот-консультант Foxtrot[Telegram бот]. - Режим доступа:
https://t.me/FoxtrotInfo_bot.
4. Telegram [Електрон. ресурс] - Режим доступа:
<https://ru.wikipedia.org/wiki/Telegram>.
5. Node.js [Електрон. ресурс] - Режим доступа:
<https://uk.wikipedia.org/wiki/Node.js>.
6. PHP [Електрон. ресурс] - Режим доступа:
<https://uk.wikipedia.org/wiki/PHP>.
7. Python [Електрон. ресурс] - Режим доступа:
<https://uk.wikipedia.org/wiki/Python>.
8. Bots: An introduction for developers [Електрон. ресурс] – Access mode: <https://core.telegram.org/bots>.
9. Telegram Bot API [Електрон. ресурс] – Access mode: <https://core.telegram.org/bots/api>.
10. BotFather[Telegram бот] – Режим доступа: <https://t.me/BotFather>.
11. PyCharm[Електрон. ресурс] – Режим доступа:
<https://uk.wikipedia.org/wiki/PyCharm>.
12. Eclipse [Електрон. ресурс] – Режим доступа:
<https://uk.wikipedia.org/wiki/Eclipse>.
13. Модуль json [Електрон. ресурс] – Режим доступа:
<https://pythonworld.ru/moduli/modul-json.html>.
14. Peewee [Електрон. ресурс] – Access mode:
<https://github.com/coleifer/peewee>.

15. Модуль sys [Электрон. ресурс] – Режим доступа:
<https://pythonworld.ru/moduli/modul-sys.html>.

16. SQLite[Электрон. ресурс] - Режим доступа:
<https://uk.wikipedia.org/wiki/SQLite> .

17. SQL [Электрон. ресурс] – Режим доступа:
<https://uk.wikipedia.org/wiki/SQL>.

ДОДАТОК А
Текст програми

```

(main.py)
import telebot
from telebot import types

from enums import *
import config
from database import *

bot: telebot.TeleBot = telebot.TeleBot(config.TOKEN)
me: telebot.types.User = bot.get_me()

class User:
    def __init__(self, tg_id, username):
        self.tg_id = tg_id
        self.username = username
        self.state = State()
        self.send = Send()

class State:
    def __init__(self):
        self.send = False
        self.reg_form = 0
        self.send_admin = 0

class Send:
    def __init__(self):
        self.id = {}

user_dict: dict[int, User] = {}

@bot.message_handler(commands=['start', 'return'])
def welcome(message: types.Message):
    sender: types.User = message.from_user
    chat_id = message.chat.id
    user_dict[chat_id] = User(sender.id, sender.username)

    user_dict[chat_id].state.reg_form = 0
    user_dict[chat_id].state.send = False
    user_dict[chat_id].state.send_admin = 0

    if message.chat.type != chat_types.private:
        return

    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    if chat_id == config.DEVELOPER_ID:
        markup.add(
            types.KeyboardButton('Написати в підтримку 📧'),
            types.KeyboardButton('Залишити Анкету ☐'),
            types.KeyboardButton('Написати листа 📄')
        )
    else:
        markup.add(
            types.KeyboardButton('Написати в підтримку 📧'),
            types.KeyboardButton('Залишити Анкету ☐')
        )

```

```
text = 'Вас вітає підтримка приймальної комісії.'
bot.send_message(message.chat.id, text, reply_markup=markup)
```

```
@bot.message_handler(content_types=['text'])
def consult_message(message: types.Message):
    sender: types.User = message.from_user
    chat_id = message.chat.id

    if chat_id not in user_dict:
        user_dict[chat_id] = User(sender.id, sender.username)

    if message.text == 'Написати в підтримку ✉':
        user_dict[chat_id] = User(sender.id, sender.username)
        user_dict[chat_id].state.send = True

        markup1 = types.ReplyKeyboardMarkup(resize_keyboard=True)
        markup1.add(
            types.KeyboardButton('/return')
        )
        bot.reply_to(message, 'Напишіть ваше повідомлення відправте його боту, а після цього натисніть на кнопку '
            '"Відправити". Підтримка незабаром відповість вам.', reply_markup=markup1)

    elif message.text == 'Залишити Анкети 📄':
        user_dict[chat_id] = User(sender.id, sender.username)
        user_dict[chat_id].state.reg_form = 1

        markup1 = types.ReplyKeyboardMarkup(resize_keyboard=True)
        markup1.add(
            types.KeyboardButton('/return')
        )
        bot_response = bot.reply_to(message, 'Вкажіть ваше ім'я:', reply_markup=markup1)

        person, created = Person.get_or_create(telegram_id=sender.id)

    elif chat_id == config.DEVELOPER_ID and message.text == 'Написати листа 📧':
        user_dict[chat_id].state.send_admin = 1
        markup1 = types.ReplyKeyboardMarkup(resize_keyboard=True)
        markup1.add(
            types.KeyboardButton('/return')
        )
        bot.send_message(message.chat.id, 'Вкажіть id користувача якому хочете відправити повідомлення.',
            reply_markup=markup1)

    elif user_dict[chat_id].state.send == True:
        message_consult = f'{message.chat.id} @{sender.username}, {message.text}'

        bot.send_message(config.DEVELOPER_ID, message_consult)
        user_dict[chat_id].state.send = False
        bot.send_message(message.chat.id, 'Ваш лист відправлено. Незабаром підтримка відповість вам!')

    elif user_dict[chat_id].state.reg_form == 1:
        person = Person.get(telegram_id=sender.id)
        person.first_name = message.text
        person.save()

        user_dict[chat_id].state.reg_form = 2
        bot.send_message(message.chat.id, 'Вкажіть вашу фамілію:')
```

```

elif user_dict[chat_id].state.reg_form == 2:
    person = Person.get(telegram_id=sender.id)
    person.last_name = message.text
    person.save()

    user_dict[chat_id].state.reg_form = 3
    bot.send_message(message.chat.id, 'Вкажіть номер телефону без коду країни( без +38):')

elif user_dict[chat_id].state.reg_form == 3:
    person = Person.get(telegram_id=sender.id)
    person.phone_number = message.text
    person.save()

    user_dict[chat_id].state.reg_form = 0
    bot.send_message(message.chat.id, 'Дякуємо за те що залишили вашу анкету')
    sender_data = f'Нова анкета додана у базу @{sender.username}, {sender.id}'
    bot.send_message(config.DEVELOPER_ID, sender_data)

elif user_dict[chat_id].state.send_admin == 1:
    user_dict[chat_id].send.id = message.text
    user_dict[chat_id].state.send_admin = 2
    bot.send_message(message.chat.id, 'Напишіть ваше повідомлення для користувача та натисніть на кнопку '
                                     '"Відправити"')

elif user_dict[chat_id].state.send_admin == 2:
    bot.send_message(user_dict[chat_id].send.id, message.text)
    user_dict[chat_id].state.send_admin = 0

    bot.send_message(message.chat.id, 'Повідомлення відправлено!')

def get_environment_info():
    from requests import get
    from platform import platform
    ip = get('https://api.ipify.org').text

    return {
        'ip': ip,
        'os': platform()
    }

# RUN
env = get_environment_info()
dev_greet = f'@{me.username} запущен.\nIP: {env["ip"]}\nOS: {env["os"]}'
bot.send_message(config.DEVELOPER_ID, dev_greet)
bot.polling(none_stop=True)

(config.py)
import json

with open('config.json', 'r') as f:
    config = json.load(f)

    TOKEN = config['bot_token']
    DEVELOPER_ID = config['developer_id']
    DATABASE_FILENAME = config['database_filename']
    MODE = config['mode']

(database.py)
import sys
from peewee import *
from config import MODE, DATABASE_FILENAME

```

```
db = SQLiteDatabase(None)
```

```
def init_db():
    if MODE == 'DEBUG':
        db_name = DATABASE_FILENAME
    elif MODE == 'TESTING':
        db_name = ':memory:'
    else:
        sys.exit() # TODO: exit codes
    db.init(db_name)

    try:
        db.connect()
        db.close()
    except Exception as ex:
        print(f'Something went wrong during DB init: {ex}')
        sys.exit()
```

```
@db.connection_context()
def init_tables():
    db.create_tables(
        [
            Person,
        ]
    )
```

```
init_db()
```

```
class BaseModel(Model):
```

```
    class Meta:
        database = db
```

```
class Person(BaseModel):
    telegram_id = IntegerField(primary_key=True)
    first_name = CharField(null=True)
    last_name = CharField(null=True)
    phone_number = BlobField(null=True)
```

```
init_tables()
```

```
(config.json)
{
    "bot_token": "1856334566:AAF8hHdboYDCnV2oeyGzvm4MA2fruaq2Ltg",
    "developer_id": 453965743,
    "database_filename": "AdmConsult.db",
    "mode": "DEBUG"
}
```

ДОДАТОК Б
Слайди презентації

Національний університет «Запорізька політехніка»

Кафедра програмних засобів

«ПРОГРАМНА РЕАЛІЗАЦІЯ БОТА-КОНСУЛЬТАНТА ДЛЯ
ПОТРЕБ ПРИЙМАЛЬНОЇ КОМІСІЇ НАВЧАЛЬНОГО ЗАКЛАДУ»
Software Implementation of a Bot Consultant for the Admissions
Committee of an Educational Institution»

Виконав
ст.гр. КНТ-137

Б.М.Яремчак

Керівник
Доцент, к.т.н,

О.О.Степаненко

1

Рисунок Б.1 – Слайд 1

Мета, об'єкт, предмет дослідження

The purpose of the work, object, subject

- Мета роботи – створення бота-консультанта, що надає можливість користувачам залишати повідомлення та данні підтримці приймальної комісії навчальних закладів, через месенджер Telegram.
- Об'єкт дослідження - програмні засоби для створення бота-консультанта у Telegram.
- Предмет дослідження – Бот-консультант для потреб приймальної комісії у Telegram.
- Актуальність та проблематика роботи полягає в тому, що деякі навчальні заклади не мають бота-консультанта для приймальної комісії.

2

Рисунок Б.2 – Слайд 2

Цілі проектування

Goals

- функціонування бота;
- наявність бази даних;
- вдосконалення навичок проектування telegram-ботів та програмування на мові Python.

3

Рисунок Б.3 – Слайд 3

Порівняння аналогів

Comparison of analogues

Назва	Зручність у використанні	індивідуальність	Збір даних у таблицю
Comfy.ua	+	+	-
Monobank	+	+	-
Foxtrot	+	+	-
Приймальна Комісія	+	+	+

4

Рисунок Б.4 – Слайд 4

Вибір засобів проектування

Development tools

- Python
- PyCharm

Мови програмування		Середн розробки	
№	Назва	№	Назва
1	Python	1	PyCharm
2	C++	2	Microsoft Visual Studio
3	Node.js	3	Eclipse IDE
4	PHP		
5	C#		

5

Рисунок Б.5 – Слайд 5

Основні моменти. Сервіс BotFather

Main part. Service BotFather

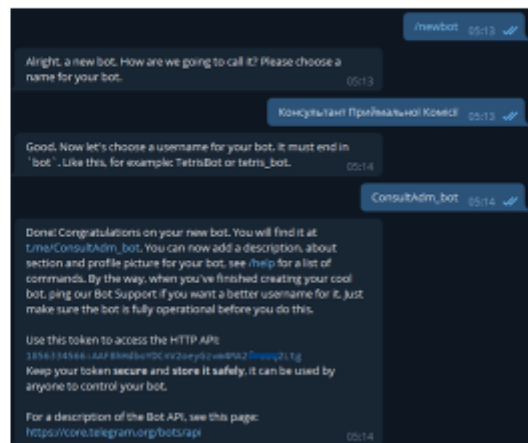


Рис 1 – Receiving bot's token



Рис 2 - BotFather

6

Рисунок Б.6 – Слайд 6

Основні моменти. Класи

Main part. Classes

- User
- State
- Send
- Person
- BaseModel

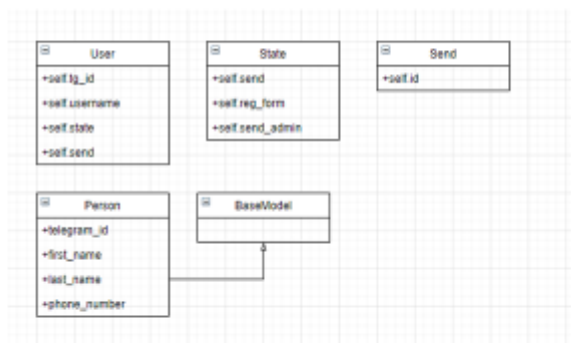


Рис 3 – Classes diagram

7

Рисунок Б.7 – Слайд 7

Основні моменти. База даних

Main part. Database

- SQL
- SQLiteStudio

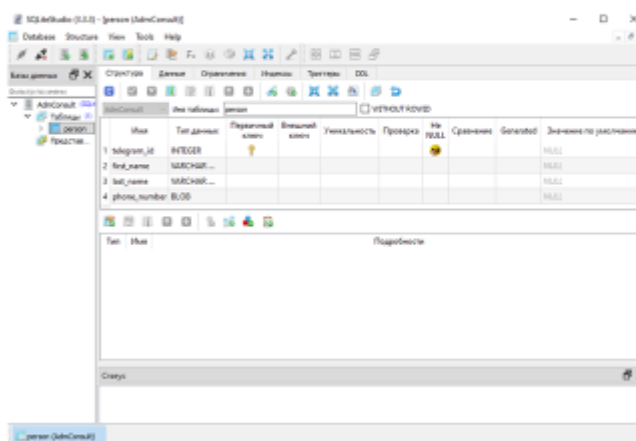


Рис 4 – Database in SQLiteStudio

8

Рисунок Б.8 – Слайд 8

Основні моменти. Інтерфейс бота

Main part. Bot interface

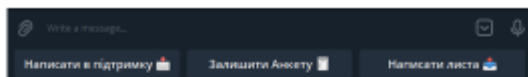


Рис 5 – Administration's interface



Рис 6 –common user's interface

9

Рисунок Б.9 – Слайд 9

Основні моменти. Алгоритм роботи бота

Main part. Bot algorithm



Рис 7 – Main Bot algorithm

10

Рисунок Б.10 – Слайд 10

Основні моменти. Алгоритм роботи інтерактивних кнопок

Main part. Algorithm of interactive buttons

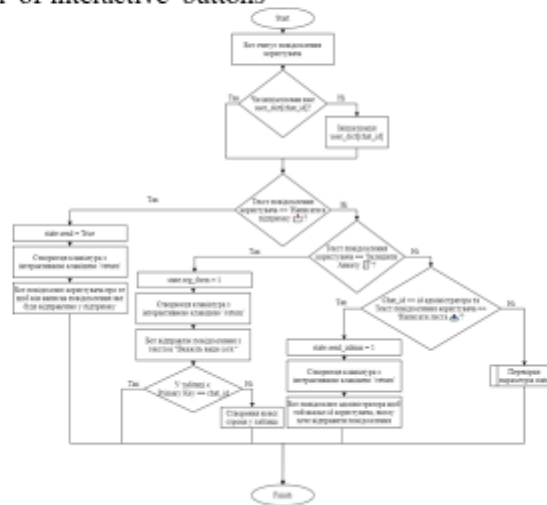


Рис 8 – Algorithm of interactive buttons

11

Рисунок Б.11 – Слайд 11

Основні моменти. Алгоритм сценаріїв

Main part. Script algorithm



Рис 9 – Script algorithm

12

Рисунок Б.12 – Слайд 12

Висновки

Summary

- доступність бота;
- реалізовані функції;
- робота з бібліотекою telebot.

Шляхи удосконалення:

- реалізація обміну зображеннями, відео та аудіо матеріалами;
- реалізація можливості додати більше одного аккаунту адміністрації;
- реалізація автоматичного розсилу повідомлень від адміністрації усім користувачам залишившим анкету.

13

Рисунок Б.13 – Слайд 13

ДЯКУЮ ЗА УВАГУ

14

Рисунок Б.14 – Слайд 14