МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Telegram-бот для органів студентського самоврядування СумДУ»

Завідувач випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Симоновський Ю.В.**

Студента групи ІН – 73 **Балаценко К.І.**

СУМИ 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедрою Довбиш А.С.

"_____" _____2021 р.

ЗАВДАННЯ
до випускної роботи

Студента четвертого курсу, групи ІН-73 спеціальності "Комп'ютерні науки" денної форми навчання Балаценка Кирила Ігоровича.

Тема: "Telegram-бот для органів студентського самоврядування СумДУ"

Затверджена наказом по СумДУ

№_____ від _____2021 р.

Зміст пояснювальної записки: 1) аналіз проблеми та постановка задачі; 2) вибір метода розв'язання задачі; 3) розробка інформаційного і програмного забезпечення системи

Дата видачі завдання "_____"_____2021 р.

Керівник випускної роботи _____ Симоновський Ю.В.

Завдання прийняв до виконання _____Балаценко К.І.

РЕФЕРАТ

Записка: 59 стор., 26 рис., 2 табл., 3 додатки, 6 джерел.

Об’єкт дослідження — чат-бот для месенджера Telegram

Мета роботи — інформаційне та програмне забезпечення чат-боту допомоги студентам з числа органів студентського самоврядування СумДУ для месенджера Telegram

Методи дослідження — технології створення чат ботів

Результати — розроблено інформаційну систему з надання першої невідкладної допомоги у вигляді чат боту для соціальної мережі Telegram. Створений бот зручний у користуванні, має декілька меню для пришвидшення отримання доступу до інформації та спрощення певних рутинних процесів. Розробка проводилась на базі мови програмування Python. У ході тестування проблем не виявлено.

TELEGRAM, BOT, PYTHON, АСИНХРОННЕ ПРОГРАМУВАННЯ

ЗМІСТ

ВСТУП

Сучасний світ дивовижний. Раніше, якщо людина чогось не знала, їй доводилося йти в бібліотеку та шукати відповідей там, або ж шукала експерта в тій області, з якої в неї питання. Складний ланцюжок, який займає чимало часу.

У сучасності, якщо ми чогось не знаємо, то все це можна знайти у декілька кліків, достатньо мати пристрій з виходом у мережу Інтернет. І тоді нас зустрічають об’ємні ресурси з відповідями на питання, базами шаблонів, спрощенним листуванням з експертами тощо. Просто та доступно.

Щоправда, в цьому є і свій мінус, а саме те, що інформації з кожною секундою стає все більше, а відповідно веб-ресурси стають об'ємними. Знайти потрібну інформацію стає дедалі складніше.

Для створення цієї дипломної роботи було обрано органи студентського самоврядування Сумського державного університету (ОСС СумДУ) як категорія, чию проблему перевантаження інформації мною буде вирішено.

ОСС існують в СумДУ лише з 2000 року, проте вони пережили чимало історичних подій в країні, а також в самому університеті. З кожним роком розвивалася їх діяльність, завдяки якій розвивається студентство, але паралельно з цим кожного року потрібно було знаходити шляхи для спрощення їх роботи, адже попит на їх активність росте, треба охоплювати якомога більше студентів новими різноманітними проектами - вечірки, лекції та курси, спортивні змагання, соціальні акції тощо.

Для якісної та безперебійної роботи, члени ОСС мають знати, як правильно оформити документацію для потрібного заходу, а саме: які службові записки або накази, як їх заповнювати та в кого підписувати.

Щоб це забезпечити, було створено відділ організаційно-методичної роботи ОСС, в якому співробітники консультують студентів з питань

оформлення потрібних документів для проведення заходів. Проте частіше за все відділ займається масштабними питаннями, а саме: накази про проведення заходів, підготовка проектів до змін в положення СумДУ з нововведеннями від членів ОСС та планами закупівель для органів студентського самоврядування. Із цього випливає, що ВОМР ОСС не має змоги постійно консультувати з шаблонних речей на кшталт службової записки про надання аудиторії для проведення заходу.

Окрім цього, кожного року голови органів студентського самоврядування різних рівнів звітують за свою каденцію перед студентами на конференціях. В даному випадку команди повинні зібрати інформацію про всі заходи, які вони проводили - фотографії, назви цих заходів та створити з цього презентацію. На цей процес витрачається як мінімум дві години, що не є зручним.

Саме тому метою моєї дипломної роботи є створення асистента члена ОСС у вигляді чат-боту в месенджері Telegram, який допомагатиме правильно оформлювати документацію для проведення заходів, а також створить презентацію з вашими заходами для звіту перед студентами.

1 АНАЛІЗ ВІДОМИХ РІШЕНЬ

Для створення ефективного асистента активістів ОСС, потрібно оглянути всі можливі канали комунікації та інструменти, на недоліках яких базуватиметься наше рішення.

1.1 Огляд існуючих рішень

1.1.1 Веб-ресурс “Шаблони документів СумДУ”

Даний веб-ресурс знаходиться за адресою document.sumdu.edu.ua.

Зайшовши на нього, ми бачимо обширний список шаблонів документів для різних сфер діяльності (рис. 1.1).

РЕЄСТР
форм документів сервісу системи управління якістю діяльності
«Шаблони документів СумДУ»

ПЕРЕЛІК ШАБЛОНІВ ДОКУМЕНТІВ РЕЄСТРУ 1) Пошук документу

№ з/п	Назва документу	Версія шаблону	Дата затвердження	Ступінь автоматизації
	1. Питання кадрового забезпечення			
	1.1. Загальні кадрові питання. Інформація щодо персональних даних. Штатний розпис			
	1.2. Прийом та звільнення працівників (крім категорії науково-педагогічних працівників)			
	1.3. Цивільно-правові договори та договори підряду			
	1.4. Організація конкурсного відбору науково-педагогічних працівників			
	1.5. Прийняття, переведення, продовження роботи за посадами науково-педагогічних працівників та на умовах погодинної оплати; укладання відповідних контрактів			
	1.6. Підвищення кваліфікації співробітників			
	1.7. Присвоєння вчених звань			
	1.8. Вибори ректора			
	2. Навчальна діяльність			
	2.1. Прийняття та поновлення на навчання			
	2.2. Рух контингенту			
	2.3. Академічна відпустка			
	2.4. Відрахування осіб, які навчаються. Випуск			
	2.5. Спеціальності. Освітні програми			
	2.6. Ліцензійна та акредитаційна експертиза			
	2.7. Організація та методичне забезпечення навчального процесу			
	2.8. Організація проходження практики			
	2.9. Конкурси інноваційно-педагогічної спрямованості			
	2.10. Індивідуальна підготовка за навчальним планом із поглибленою науковою складовою			
	2.11. Навчання за індивідуальним графіком			
	2.12. Організація контролю знань (крім атестації здобувачів вищої освіти)			
	2.13. Атестація здобувачів вищої освіти			
	2.14. Документи студента та випускника			
	2.15. Процедурні питання щодо документів студента та випускника; інформація щодо персональних даних			

Рис.1.1 Веб-ресурс “Шаблони документів СумДУ”

На даному ресурсі ми обираємо категорію, яка нам потрібна і в ній обираємо той документ, який нам треба оформити. З'являється форма (рис.1.2), яку ми заповнюємо і потім на основі введених даних, веб-ресурс генерує повноцінну службову записку, яку залишається роздрукувати та підписати.

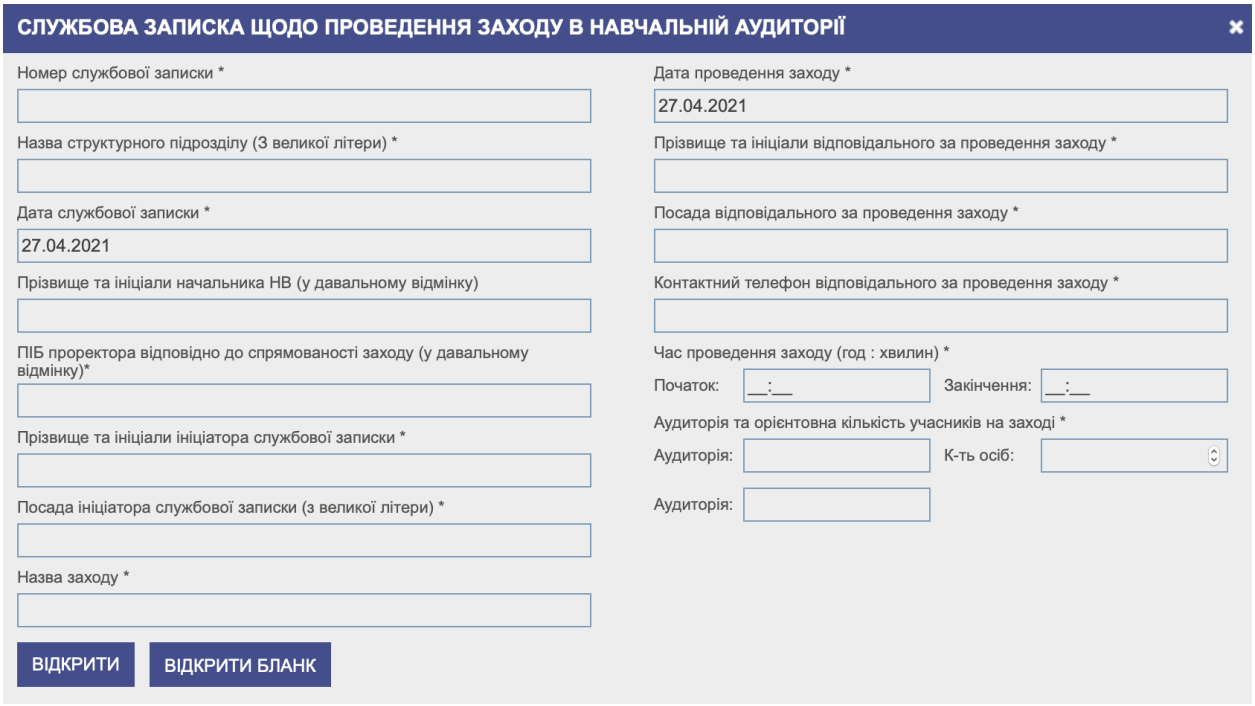

Рис.1.2 Форма для оформлення службової записки щодо проведення заходу в навчальній аудиторії

Перевагою такого ресурсу є те, що він доступний цілодобово та кожен член ОСС може звертатися до нього у будь-який час.

Недоліком є те, що активісти ОСС з цієї бази використовують лише 20% документів, а так як вона є чималою, то заплутатися в ній дуже просто, після чого студенти звертаються до відділу організаційно-методичної роботи органів студентського самоврядування (ВОМР ОСС).

1.1.2 Відділ організаційно-методичної роботи органів студентського самоврядування

Відділ організаційно-методичної роботи органів студентського самоврядування (ВОМР ОСС) існує з 2017 року. Мета структури – всебічно сприяти ОСС в документальній та організаційній частині.

Штат відділу є малочисельним, тому в нього є можливість сприяти під час організації масштабних проектів або державних закупівель. Якщо, наприклад, ОСС мають на меті проводити звичайну лекцію або зустріч зі студентами, то в цьому їм доведеться розбиратися самому, адже штат відділу доступний не у цілодобовому режимі.

У результаті аналізу існуючих рішень, було складено порівняльну таблицю параметрів, що допоможе під час розробки проекту.

Таблиця 1.1 Порівняльна характеристика існуючих аналогів

Функція	Веб-ресурс “Шаблони документів СумДУ”	Відділ організаційно-методичної роботи ОСС	Telegram-бот для органів студентського самоврядування
Цілодобовий доступ	+	-	+
Документи, що потрібні лише для ОСС	-	+	+
Створення звітів	-	-	+

1.2 Постановка задачі

У результаті проведеного аналізу, поставлено мету цієї роботи – проектування та розробка telegram-боту зі зручним інтерфейсом користувача та швидким доступом до всієї інформації. Для здобуття цієї мети необхідно завершити наступні завдання:

1) Вивчення теорії програмування чат-боту для месенджера Telegram;
2) Проектування зв’язків;
3) Реєстрація бота в системі;
4) Розробка наповнення боту;
5) Провести тестування сторонніми користувачами;

2 ВИБІР ОСНОВНИХ КОМПОНЕНТІВ БОТУ ТА ІХ РЕАЛІЗАЦІЯ

2.1 Створення MindMap для проектування боту

MindMap - діаграма зв’язків, яка допомагає структурувати ідеї за допомогою графічного запису. Вона представляє собою деревовидну структуру та відображає зв’язки між фрагментами інформації.

За допомогою MindMap я хочу спроектувати логіку бота, яка буде імплементована під час розробки сервісу.

За основу своєї карти я взяв команди на які буде відповідати бот (рисунок 2.1).

Після початку роботи із ботом по команді /start, користувач отримає вітальне повідомлення, яке розповість про бот та команди для роботи з ним.

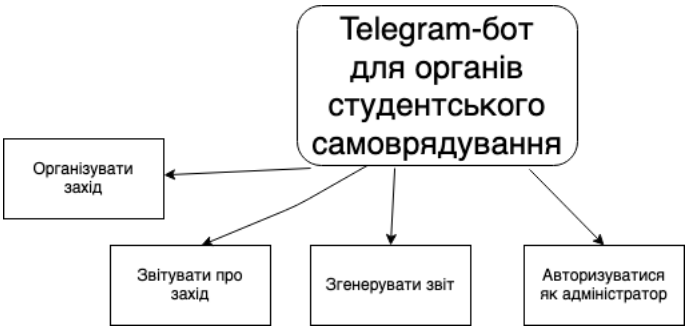

Рисунок 2.1 - Основа діаграми зв’язків

При натисканні кнопки «Організувати захід» користувач отримає питання, що саме потрібно йому для заходу (рисунок 2.2), після відповіді на яке він отримує:

- Назви документів, які потрібно оформити для проведення заходу
- Посилання на шаблони документів
- Посилання на приклади заповнення документів
- Список уповноважених осіб, в яких потрібно підписати пакет документів

Рисунок 2.2 – Принцип роботи бота після натискання на кнопку «Організувати захід»

Функція «Звітувати про захід» доступна при натисканні на кнопку в меню. Вона дозволяє залишати інформацію про заходи, які були проведені на факультеті. Після натискання на відповідну кнопку, бот попросить авторизуватися. Пароль буде наданий адміністратором, який створить користувача у базі даних. Це потрібно для того, щоб розмежувати інформацію від кожного факультету або інституту та не дозволити будь-якому користувачу залишати недостовірну інформацію про заходи на тому чи іншому структурному підрозділі.

Потім бот запитає назву заходу. Після відповіді користувача, він запитає про фото з заходу. Так як у Telegram не існує такого поняття як альбом, бот проситиме по одному фото до того моменту, поки користувач не натисне на кнопку «Готово», після чого дані про захід збережуться у базі даних.

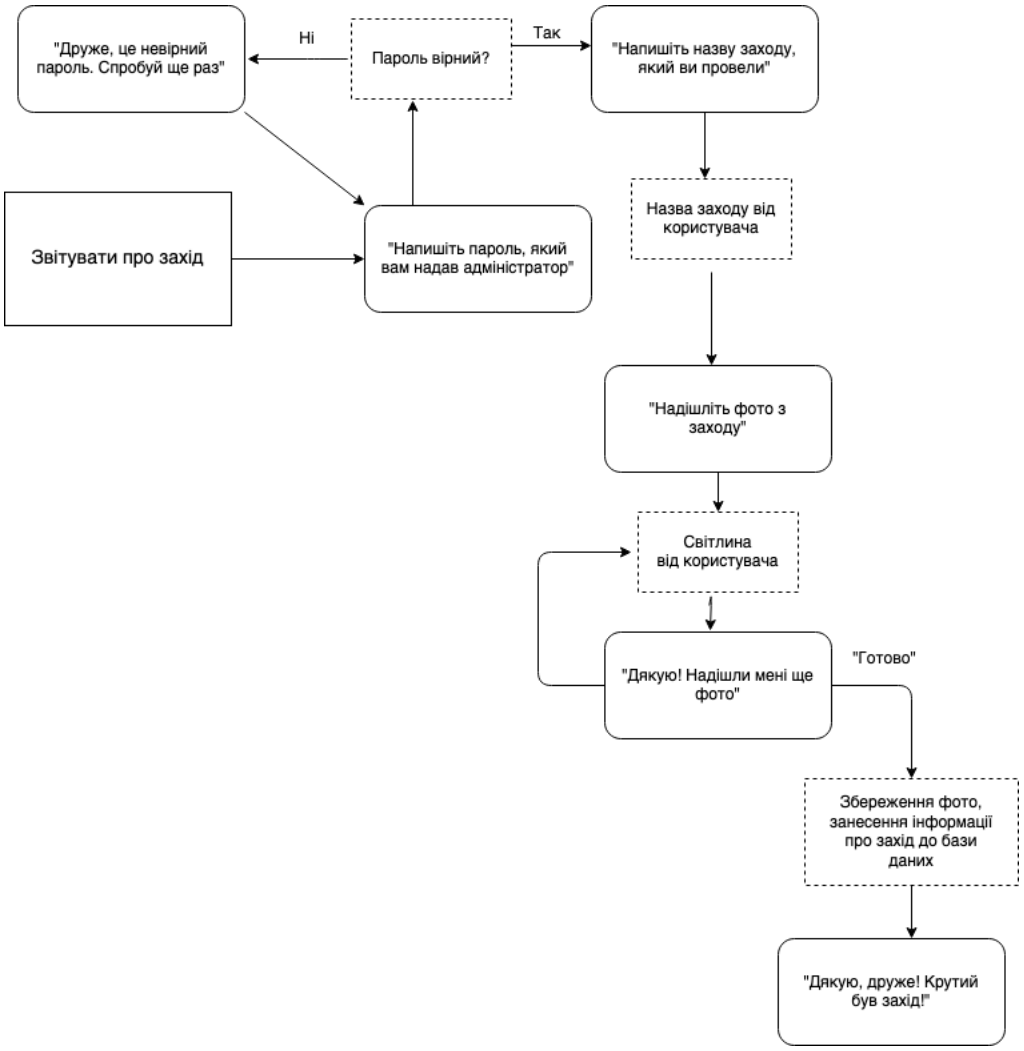

Рисунок 2.3 – Принцип роботи функції «Звітувати про захід»

Функція «Сформувати звіт» дозволяє згенерувати презентацію про проведені заходи або проекти на основі тих даних, які ви надавали боту протягом певного проміжку часу. Дана функція корисна для студентських деканів або директорів, адже замість самостійного створення презентації для звіту в кінці каденції, що займає від 2 годин, голова ОСС за допомогою telegram-боту зможе згенерувати такий звіт протягом 1-2 хвилин. На рисунку 2.4 показано принцип роботи функції «Сформувати звіт»

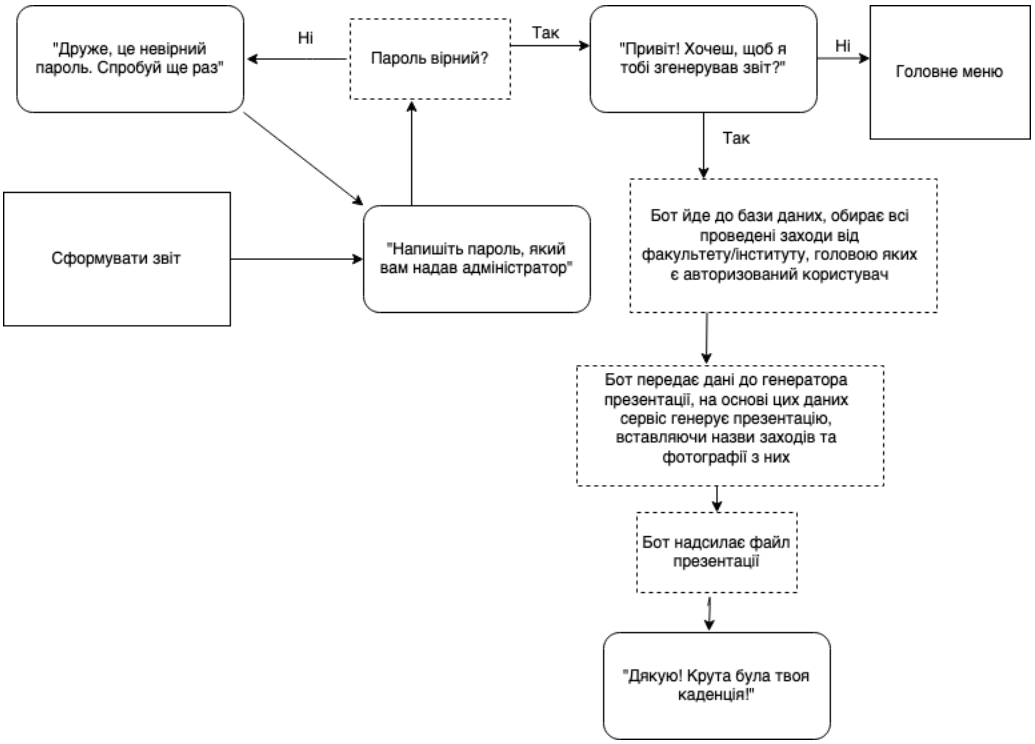

Рисунок 2.4 – Принцип роботи функції «Сформувати звіт»

Функція «Авторизуватися як адміністратор» надає доступ до адміністраторської панелі, яка дозволяє додавати, видаляти та переглядати список користувачів. Для того, щоб потрапити в панель адміністратора, пароль не потрібний. У telegram-боті є список user id тих користувачів, які є адміністраторами. При натисканні на кнопку «Авторизуватися як адміністратор» відбувається перевірка, чи є user id користувача, що натиснув кнопку, у списку user id адміністраторів. Якщо так, тоді він отримує доступ до адміністраторських функцій, якщо ні – користувач отримує повідомлення «Друже, тобі сюди не можна». Логіка роботи продемонстрована на рис.2.6.

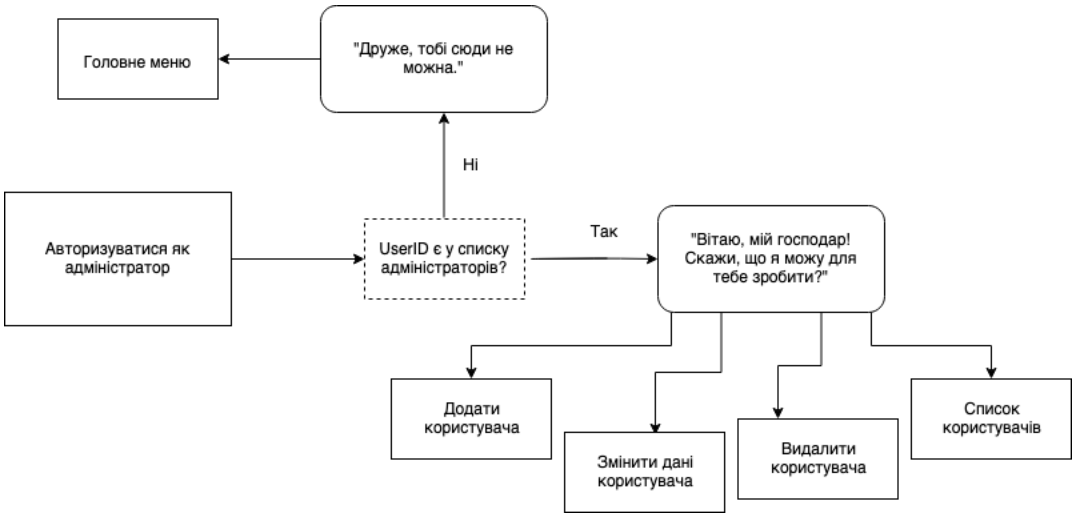

Рисунок 2.5 – Логіка роботи функції «Авторизуватися як адміністратор»

2.2 Реєстрація бота в системі

Перед програмною реалізацією бота в месенджері Telegram, потрібно його зареєструвати. Цей процес дуже простий і повністю автоматизований.

В мережі Telegram для створення та реалізації ботів існує окремий аккаунт @BotFather, що допоможе вам при розробці власного боту. На рисунку 2.6 зображено початок роботи із цим ботом.

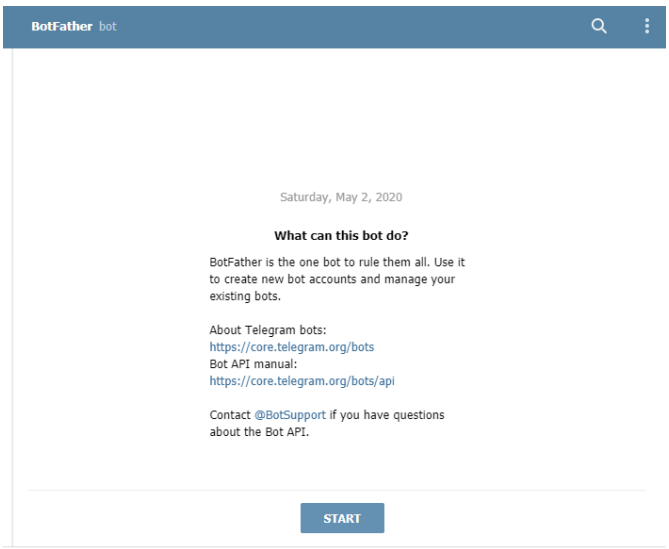

Рисунок 2.6 - Початок роботи з BotFather

Після отримання списку базових команд для роботи із @BotFather натискаємо /newbot для створення нового боту (рисунок 2.7).

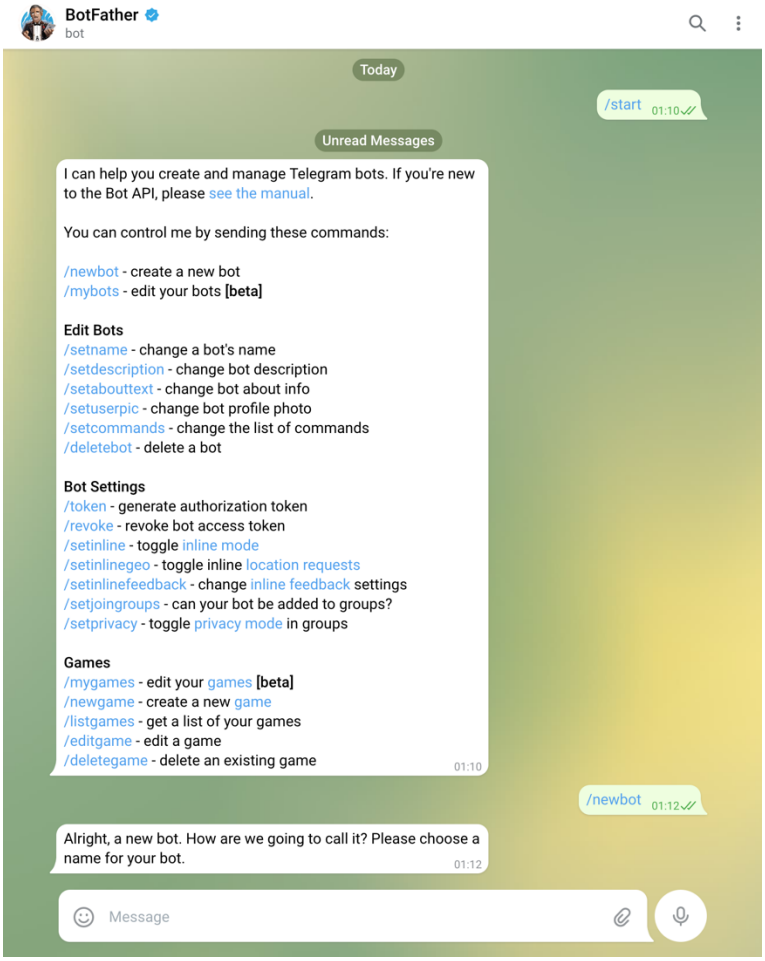

Рисунок 2.7 - Створення нового бота

Відправляєте бажане вами ім'я для нового бота і його юзернейм, який обов'язково повинен закінчуватися на "bot". Для свого проекту я обрав назву “Khaba”, на честь голови ВОМР ОСС Хаби Анни Петрівни з 2017 року.

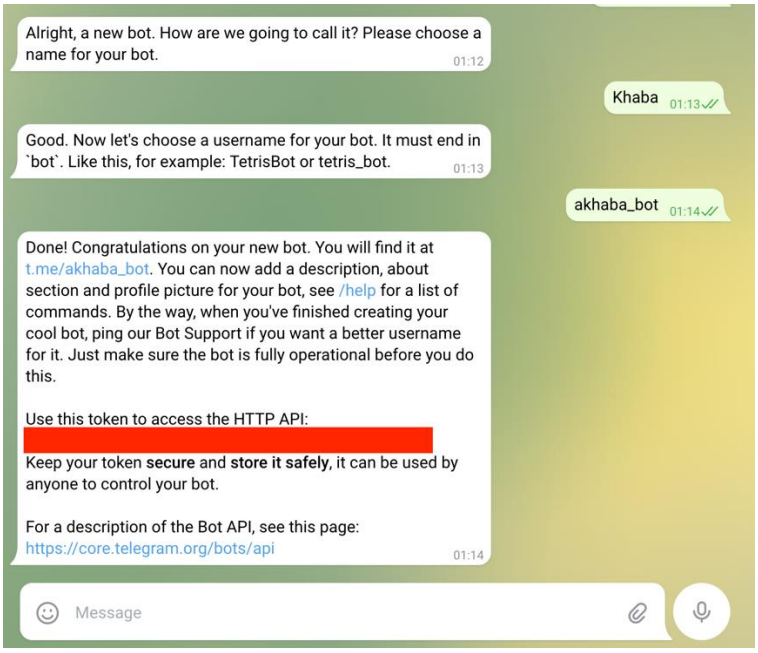

Рисунок 2.8 - Іменування бота та отримання ун

Отримуєте унікальний токен, що надасть вам доступ до HTTP API

Токен – цифровий ключ, що допоможе вам пройти ідентифікацію в який-небудь важливий сервіс. Відмінність від звичайного пароля полягає в тому, що цифровий ключ весь час оновлюється (постійно генерує нову послідовність символів), він відомий лише вам, в момент ідентифікації.

HTTP – гіпертекстовий протокол передачі даних прикладного рівня моделі OSI.

API – набір готових інтерфейсів, що надаються додатком (бібліотекою, сервісом) для використання в зовнішніх програмних продуктах.

Також можна додати опис бота, список його команд і відповідне зображення. На рисунку 2.9 зображено чат після вибору команди /setdescription, де ми вводимо інформацію для опису боту «Телеграм-бот, який допоможе тобі у тому, щоб робити життя в університеті цікавішим»

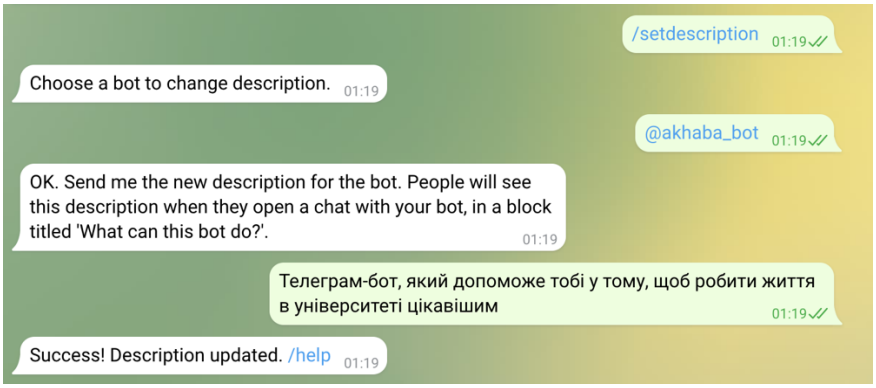

Рисунок 2.9 - Створення опису телеграм бота

Далі я ввожу /setuserpic для завантаження фото профілю (рисунок 2.10).

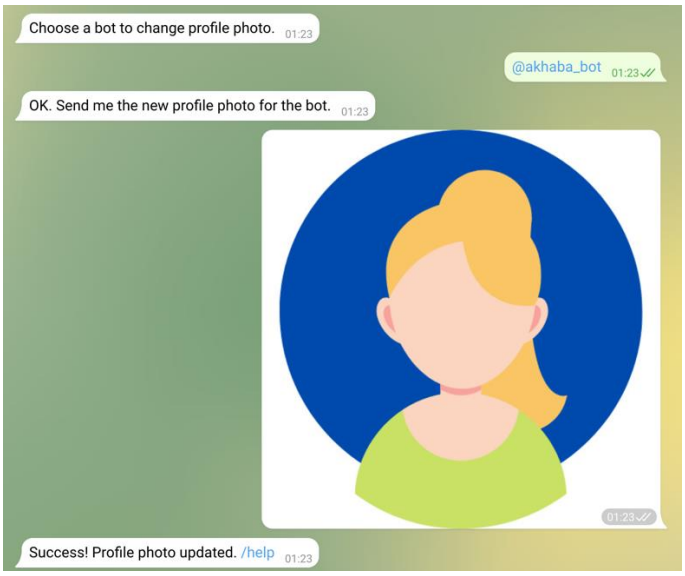

Рисунок 2.10 - Додавання фото до профілю бота

На рисунку 2.11 ми бачимо зображення готового профілю боту після вище зазначених дій.

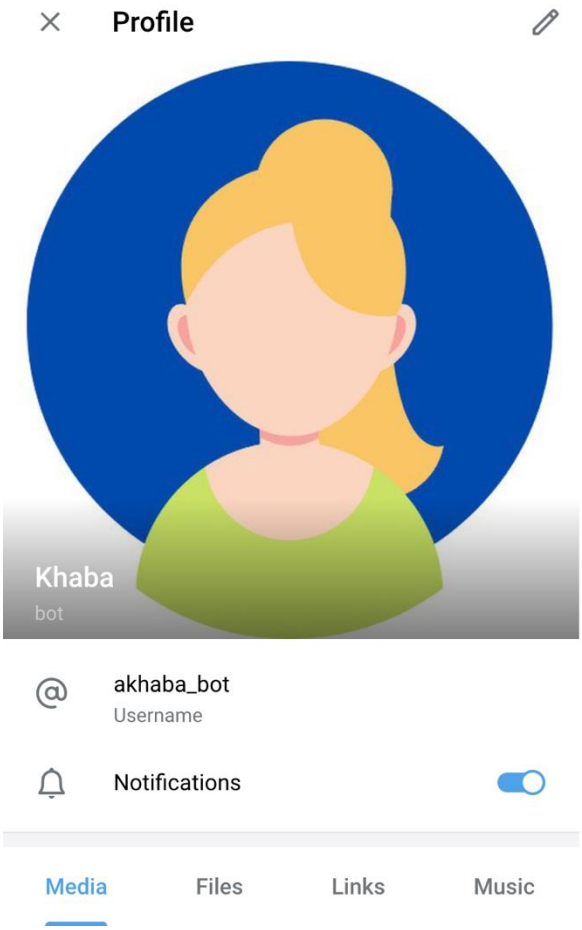

Рисунок 2.11 - Профіль боту

Таким чином ми отримали основу для боту із описом та фотографією.

3 КОМП'ЮТЕРНА РЕАЛІЗАЦІЯ ПРОЕКТУ ТА ТЕСТУВАННЯ

3.1 Вибір середовища розробки

Існує декілька шляхів втілення функціональності Telegram-бота. Розберемо кожен з них.

- Використовуючи Microsoft Visual Studio, доцільно вибрати C # або Node.js. Це дозволить пришвидшити розробку, адже можна використовувати готові бібліотеки, що спрощують створення бота –Bot Application, Bot Framework Emulator і Bot Dialog. Розробнику потрібно тільки налаштувати шаблони під свої потреби.
- Іншим не менш корисним та зручним варіантом є мова програмування PHP. Він також дає багато можливостей для написання коду. Наявність готових бібліотек, серед яких - Telegram Bot SDK, схиляє розробників на свою сторону, адже це дозволяє звести зусилля до мінімуму. Недоліки такого підходу у відсутності принципів асинхронного програмування, що впливає на продуктивність та стресостійкість бота.
- Python є одним з найбільш провідних рішень. Дана мова програмування має великі можливості, що відкриваються при використанні стандартних бібліотек. Не менш важливим є можливість користуватися спеціальними бібліотеками розрахованими на роботу з Telegram. Наприклад, telebot, яка є найпоширенішою серед розробників Telegram-ботів.

Таблиця 3.1. Порівняльний аналіз мов програмування для створення ботів мережі Telegram

Параметри	C#	PHP	Python
Шаблони	+	-	+
Бібліотеки	+	+	+
Розповсюдженість мови	+	+	+
Читабельність	+	+	+

Проаналізувавши дані підходи, для розробки бота було обрано мову Python та бібліотеку aiogram, що базується на принципах асинхронного програмування, а також має таку особливість як «машина станів», яка дозволяє зберігати в оперативній пам’яті дані, що надаються користувачем. Це покращує роботу telegram-бота, адже якщо на якомусь етапі користувач відмовляється від виконання дії, то дані видаляються та не зберігаються у базі даних.

Із СУБД було обрано PostgreSQL, з якою Python дозволяє працювати за допомогою Object-relational mapping (ORM), що спрощує моделювання та створення бази даних, а також роботу з нею, адже для цих дій не потрібно відходити від синтаксису Python та згадувати SQL-синтаксис – достатньо ознайомитися з документацією ORM, щоб розуміти її функціонал та знати базові принципи побудови бази даних та роботи з нею.

В якості ORM було обрано GINO, що базується на SQLAlchemy та дозволяє гнучку роботу з СУБД.

Для зберігання даних про документи, буде використано JSON (JavaScript Object Notation) – текстовий формат обміну даними, що є читабельним для розробника.

3.2 Проектування бази даних

Коли було визначено зовнішню та внутрішню структуру telegram-боту, необхідно було вирішити наступне питання – спосіб збереження інформації: дані користувачів, заходи та фото заходів, до яких вони відносяться. Для повноцінної та коректної роботи telegram-бота необхідно спроектувати базу даних та виконати її підключення до telegram-бота. База даних telegram-бота «Khaba» складається із таких таблиць: user, photos, event. На рисунку 3.1 зображена структура бази даних telegram-боту.

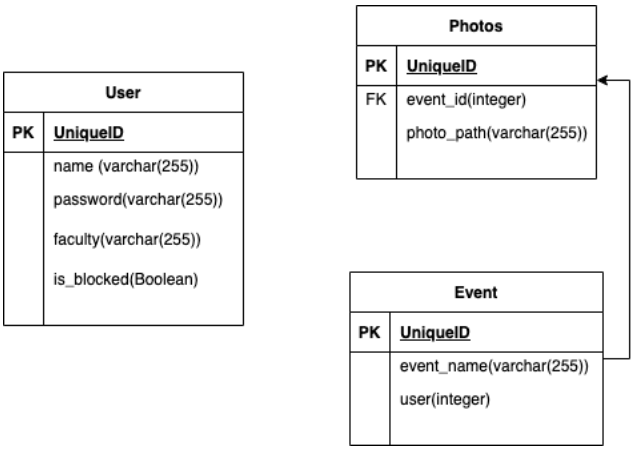

Рисунок 3.1 - ERD-діаграма бази даних telegram-бота

Підключення до бази даних та робота з нею відбуватиметься за допомогою ORM GINO, що дозволить не відходити від синтаксису Python та забезпечувати ефективну роботу між telegram-ботом та базою даних.

3.2 Програмна реалізація бота

Структура telegram-бота виглядає так:

- Bot – основа боту, яка поділена на ланцюги (chains) та проміжні засоби (middle wares). Ланцюги дозволяють розбити монолітний проект на декілька сервісів, що забезпечує більш зручне читання коду та спрощену розробку. В telegram-боті Khaba є такі ланцюги, як admin_panel (сервіс адміністраторської панелі), create_event (сервіс отримання інформації про документацію, що потрібна для створення

заходу), record_event (сервіс, що дозволяє користувачу звітувати про захід), get_report (сервіс, що дозволяє користувачу отримати звіт за певний проміжок часу), base (сервіс, що викликається під час натискання кнопки start). Окрім цього, в кожному з сервісів прописані властивості клавіатур для кожного з випадків. Також є проміжні засоби, такий як throttling (забезпечує захист від спаму). Bot має описані методи роботи з JSON, файл конфігурації та core.py, що має в собі команди, які викликаються при запуску бота в файлі __main__.py.

- db – директорія, що містить в собі опис таблиць бази даних у вигляді моделей та методи, що неодноразово викликаються у telegram-боті.
- Presentation – сервіс, що викликається у боті та генерує звіт по запиту користувача. Даний сервіс реалізовано за допомогою бібліотеки python-pptx

Для початку розробки, у директорії проекта ми створюємо файл .env, в якому зберігатимуться такі дані, як токен бота, user id адміністратора та дані про базу даних (хост, порт, назва бази даних, логін, пароль).

Структура .env файлу виглядатиме так:

```
TELEGRAM_BOT_TOKEN = #Токен телеграм бота
ADMIN_CHAT_ID = #User id користувача, якого призначено адміністратором
DB_HOST=
DB_PORT=
DB_NAME=
DB_USERNAME=
DB_PASSWORD=
```

Коли ми виконали цей крок, ми встановлюємо такі бібліотеки, як:

- Aiogram, асинхронна бібліотека для створення телеграм-ботів
- SQLAlchemy, ORM за допомогою якою написані моделі бази даних
- GINO, легка ORM, що базується на SQLAlchemy, для легких асинхронних запитів до БД
- python-pptx - бібліотека, що відповідатиме за генерацію презентацій

- Celery, асинхронна черга задач

Встановлення відбувається у терміналі за допомогою команди pip install.

Після цього, ми створюємо директорію bot, в якій ми також створюємо директорії chains та middlewares.

У директорії chains ми створюємо субдиректорії base, admin_panel, create_event, get_report, record_event. В кожній з цих директорій буде розроблено функціонал бота на кожний з випадків. У директорії base ми створюємо файли handlers.py, що містить в собі вихідний код обробки випадку, коли користувач натискає на кнопку start; kb.py, що містить в собі стартову клавіатуру для бота, кнопки в якій перенаправлятимуть на функції бота.

Вихідний код handlers.py можна переглянути на рис.3.2

```
#Імпорт бібліотек та файлів
from aiogram import types
from aiogram.dispatcher import FSMContext
from aiogram.types import User as TgUser

from bot.chains.base.kb import start_kb
from bot.core import dp


@dp.message_handler(commands="start", state="*") #Обробник, що викликає функцію start_command при натисканні на кнопку start
async def start_command(msg: types.Message, state: FSMContext): #Функція, що викликається після натискання кнопки start
    await state.finish()
    # Повідомлення, яким бот зустрічає користувача
    await msg.answer(
        f"Привіт,  {TgUser().get_current().first_name}!\n\n"
        f"Я - той, хто допоможе тобі робити вітер у студентському самоврядуванні :)\n\n"
        f"Ти мені пишеш, який захід хочеш організувати, що для цього треба, "
        f"а я тобі розповідаю як це все втілити у життя.",
        reply_markup=start_kb, #Клавіатура, яка викликається разом з повідомленням
    )
```

Рисунок 3.2 – handlers.py у base, що виконується при натисканні на кнопку start

Після цього, ми створюємо стартову клавіатуру, що дозволяє отримати доступ до потрібного функціоналу бота. Її вихідний код можна переглянути на рис.3.3.

```
from aiogram.types import ReplyKeyboardMarkup, KeyboardButton # Імпортуємо методи, які дозволяють створити клавіатуру

start_kb = ReplyKeyboardMarkup(
    # Масив кнопок стартової клавіатури
    keyboard=[
        [
            KeyboardButton(text="Організувати захід 🎉"),
            KeyboardButton(text="Захід"),
            KeyboardButton(text="Залогінитися як адмін 😎"),
            KeyboardButton(text="Отримати звіт"),
        ]
    ],
    resize_keyboard=True, #Параметр, що робить клавіатуру адаптивною для будь-якого типу екрану
)
```

Рисунок 3.3 – kb.py у base, в якому прописана стартова клавіатура

Для того, щоб обробник працював, у автоматично створеному файлі __init__.py ми імпортуємо start_command з файлу handlers.py.

Вихідний код інших ланцюгів бота можна переглянути у додатку А.

Також у директорії middlewares ми створюємо такі файли, як:

- Throttling.py – файл, що захищатиме бота від спаму

Вихідний код middlewares можна переглянути у додатку Б.

Щоб бот коректно працював, переходимо у директорію bot та створюємо такі файли, як:

- core.py – ядро бота, в якому прописуються певні налаштування бота
- config.py – файл конфігурації, який встановлює константний шлях для збереження фото або доступу до файлів
- __main__.py – файл, в якому прописується вихідний код для запуску бота
- files.py – файл, в якому прописано функції для відкриття та читання файлів
- tree.py – файл конфігурації для доступу до JSON, в якому зберігається інформація про документацію

Вихідний код кожного з файлів можна переглянути у додатку В.

Для роботи з базою даних, ми використаємо SQLAlchemy ORM (для моделювання таблиць бази даних) та GINO ORM (для запитів до БД).

Переходимо у корінь проекту та створюємо директорію db, а в ній субдиректорію models.

Вихідний код файлів директорії db можна переглянути у додатку Д.

Для генерації презентацій ми використаємо бібліотеку python-pptx. Вона є синхронною, але так як бот використовує принципи асинхронного програмування, ми додамо до генератора презентацій Celery – асинхронну чергу задач, яка дозволяє синхронні задачі перетворювати на асинхронні, що забезпечує ефективну роботу.

Вихідний код генератора презентацій можна переглянути на рис.3.4.

```
import os

from pptx import Presentation
from pptx.util import Inches
from db.config import BASE_DIR
from celery import Celery

app = Celery() # Ініціалізуємо Celery


@app.task #Декоратор, який вносить задачу створення презентації до асинхронної черги задач
def create_presentation(name, image_list, faculty): # Виклик функції, що генерує презентацію
    prs = Presentation()
    steps = [Inches(1), Inches(4), Inches(6.5)] # Кроки для переміщення зображень на слайді
    for i in range(len(image_list)): # Цикл, що ітерується по довжині масиву списку зображень
        slide = prs.slides.add_slide(prs.slide_layouts[1])
        title_shape = slide.shapes.title
        title_shape.text = name[i] # Додавання назви заходу на слайд
        for image in image_list[i]: # Цикл, що додає зображення на слайд
            left = steps[image_list[i].index(image)] #На скільки кроків рухається зображення
            # Розміри зображення
            top = Inches(2)
            width = Inches(3)
            height = Inches(3)
            img = slide.shapes.add_picture(image, left, top, width, height) # Додавання зображення згідно заданих параметрів

    prs.save(os.path.join(BASE_DIR, f"presentations/{faculty}_report.pptx")) # Збереження презентації з назвою факультету, для якого генерується звіт


if __name__ == "__main__":
    app.worker_main() # Запуск задачі задачі у Celery
```

Рисунок 3.4 – presentation_creator.py у директорії presentation

Структура JSON, в якому зберігатимуться дані про документи, виглядатиме так:

«потрібно_для_заходу»: {

“link”: “посилання_на_шаблон_та_приклад_заповнення_документу”,

“name”: «назва_документу»,

“significants”: [“список_тих_у_кого_треба_підписати”]

}

3.4 Тестування Telegram боту

Після реалізації боту було проведення тестування різними сторонніми користувачами. В ході тестування проблем не виявлено.

На початку роботи бота користувач бачить опис того чим займається бот та кнопку «Start» (рисунок 3.5), після натискання якої боту відправиться повідомлення із командою /start (рисунок 3.6).

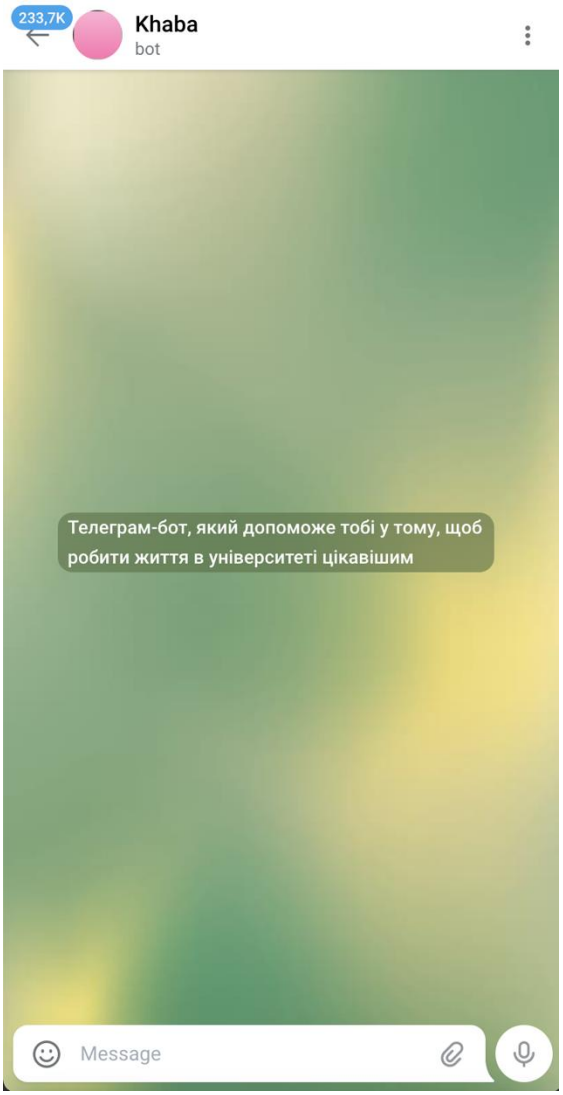

Рисунок 3.5 – Початок роботи із ботом

На команду /start бот відповідає вітальним повідомленням та клавіатурою з доступним функціоналом (рисунок 3.6).

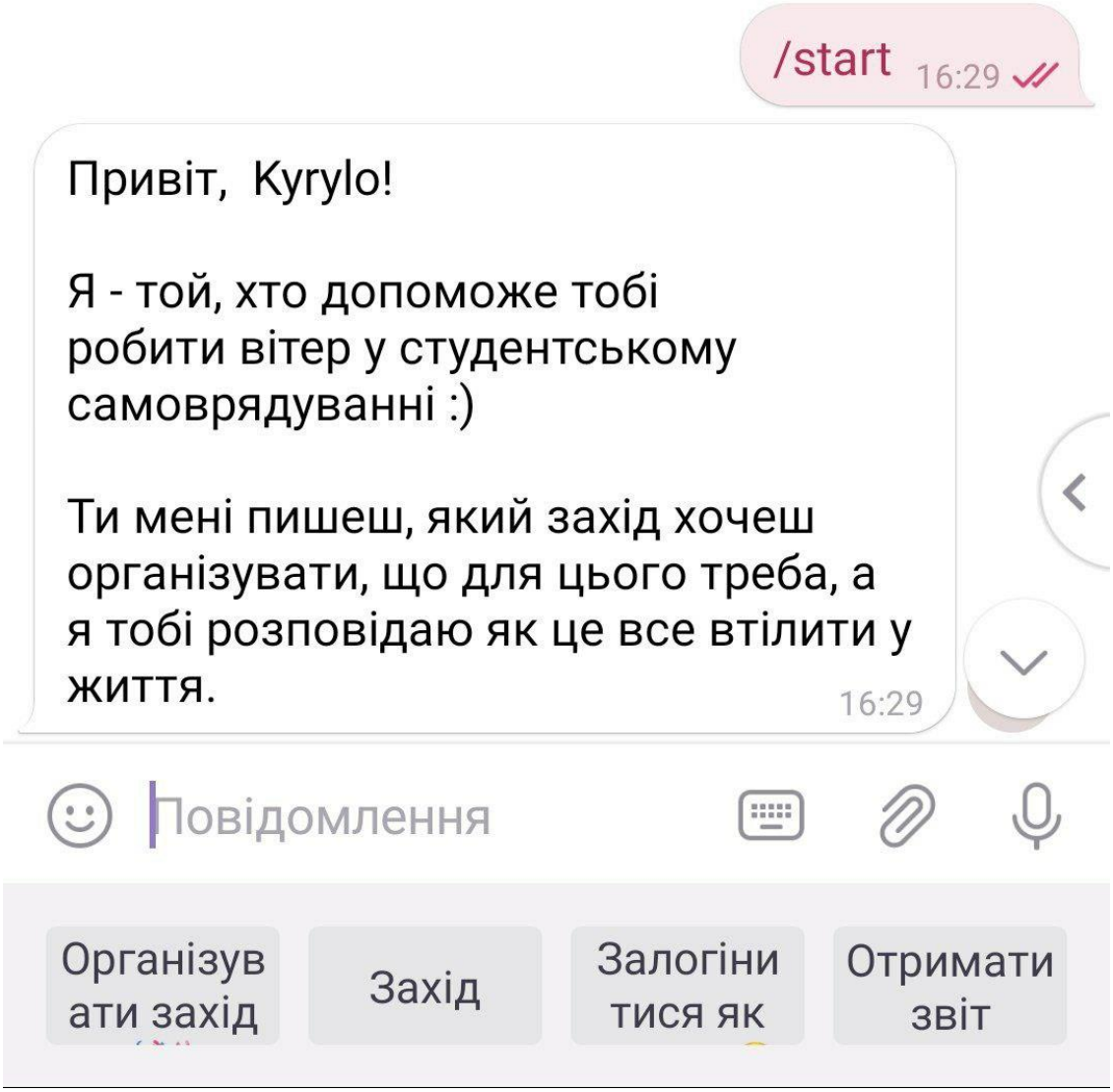

Рисунок 3.6 – Чат із ботом після команди /start

При натисканні на кнопку «Організувати захід», бот питає що нам потрібно для його організації та надає приклад, в якому вигляді надати інформацію. Після нашої відповіді, він надає повну інформацію по документації: посилання на шаблон та приклад, назва документу та у кого підписувати (рис.3.7).

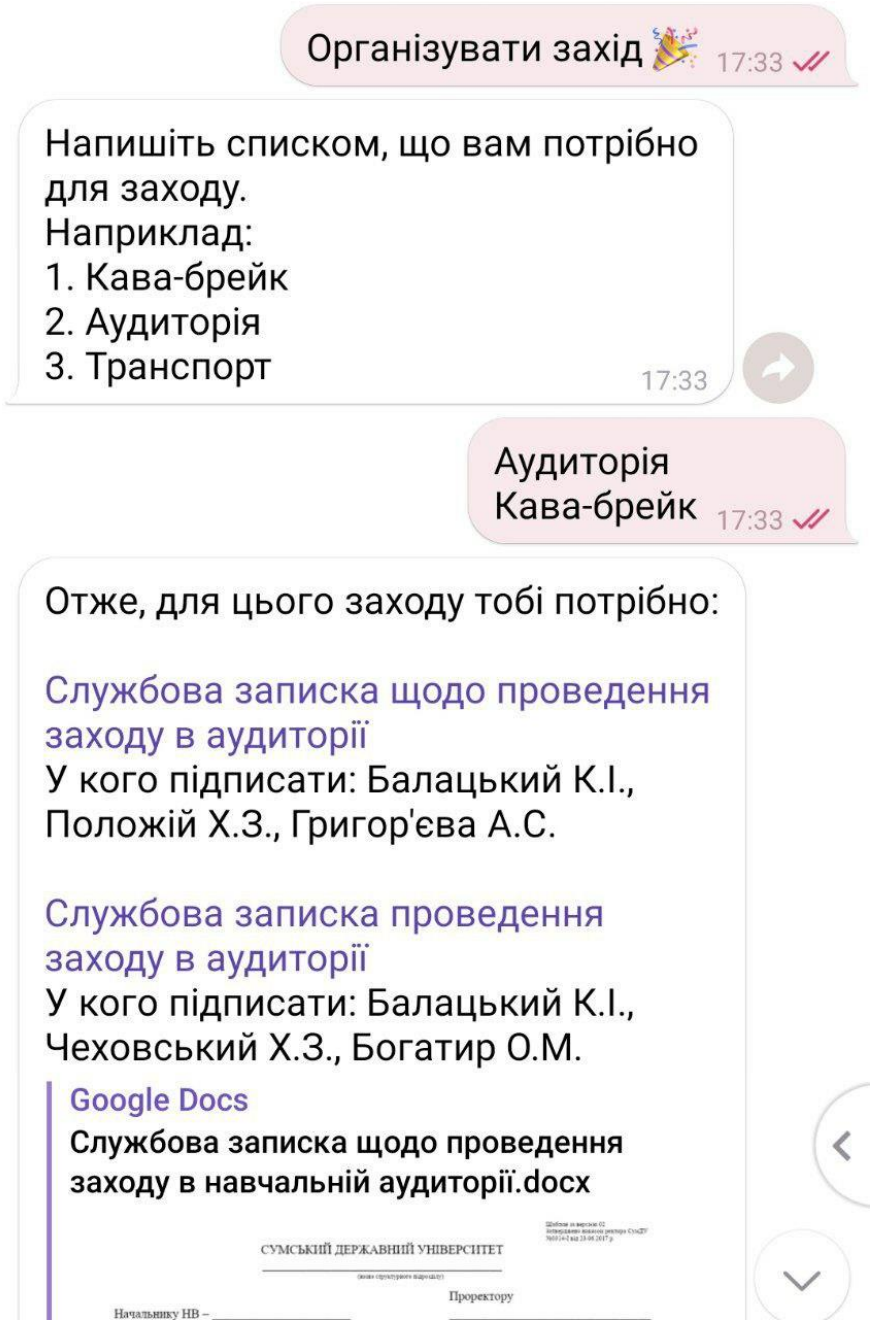

Рисунок 3.7 – Чат із ботом після натискання кнопки «Організувати захід»

Якщо користувач є адміністратором, він може створювати, видаляти, змінювати або отримувати список користувачів. Для цього йому достатньо натиснути на кнопку «Залогінитися як адмін». Функція створення користувача представлена на рис.3.8.

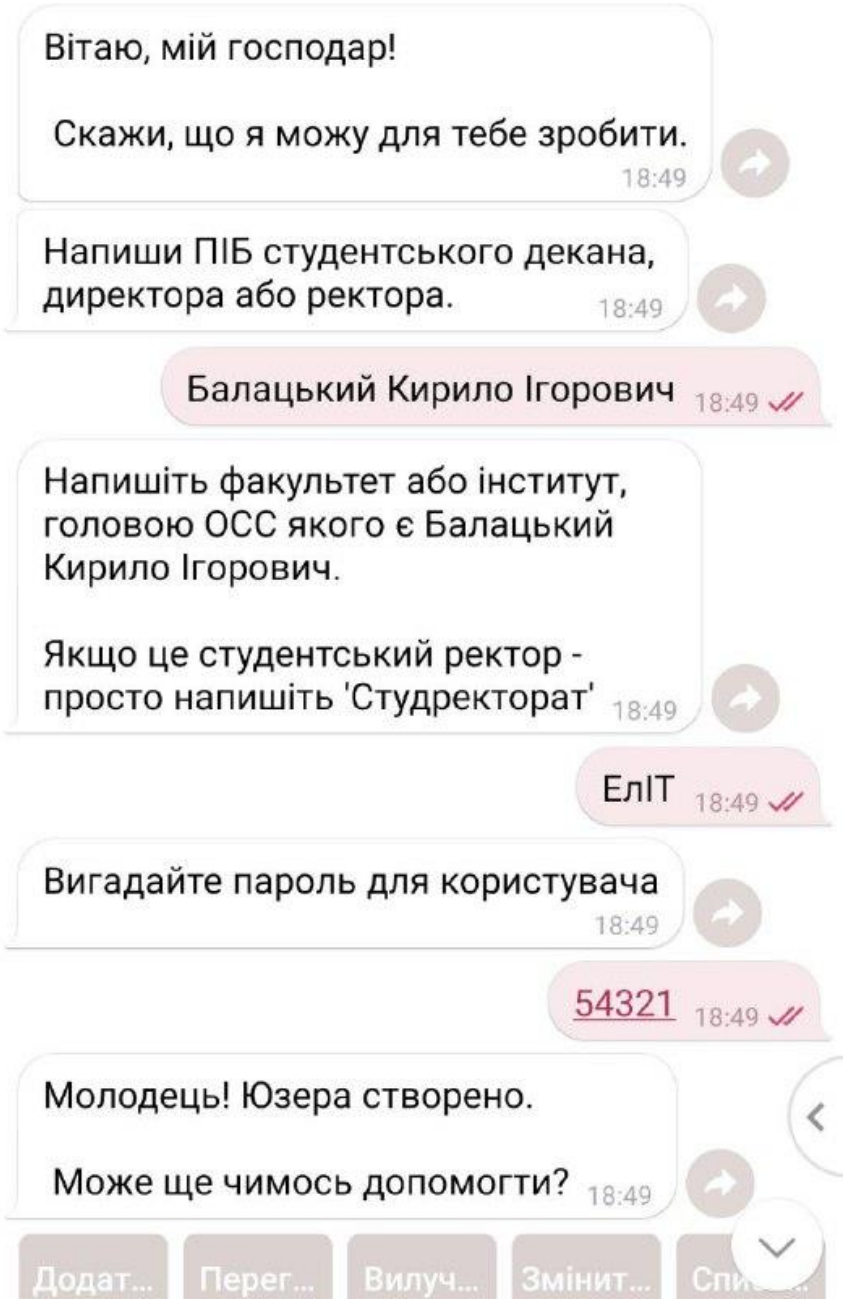

Рисунок 3.8 – Чат із ботом після натискання кнопки «Залогінитися як адмін» та обрання функції «Додати користувача»

На рис.3.9 представлено функції «Видалити користувача» та «Список користувачів»

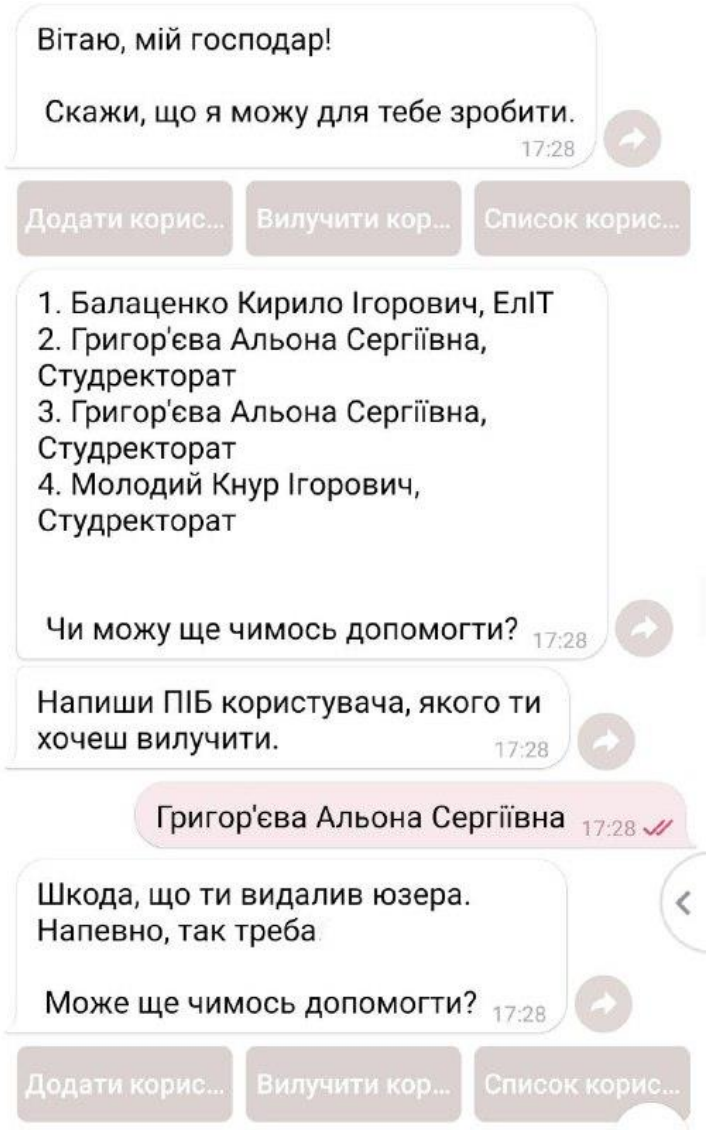

Рисунок 3.9 – Чат із ботом після натискання кнопки «Залогінитися як адмін» та обрання функцій «Список користувачів», «Видалити користувача»

При натисканні на кнопку «Захід», ми отримуємо доступ до функції звітування про захід. Бот питає в нас пароль, по якому визначає з якого ми факультету та як нас звати. Якщо пароль вірний – він вітає нас та питає назву заходу, після чого він питає в нас перше фото (рис.3.10).

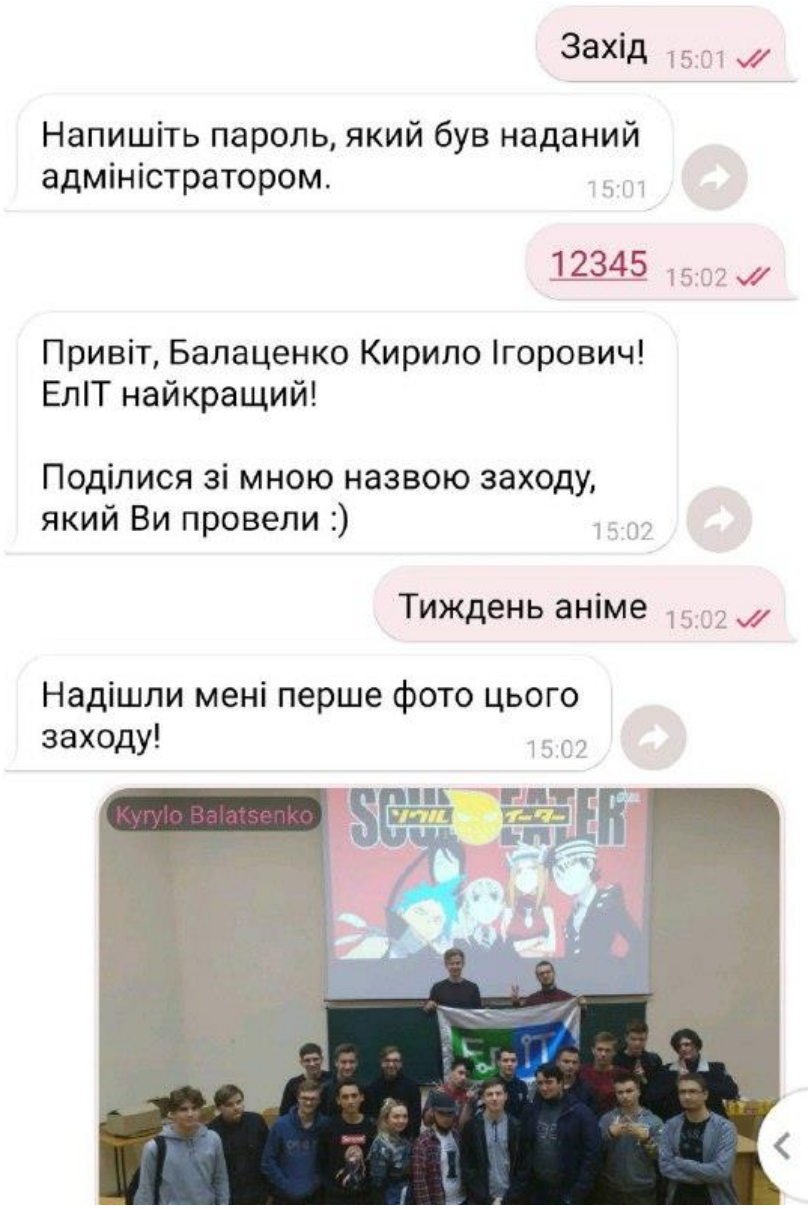

Рисунок 3.10 – Чат із ботом після натискання кнопки «Захід»

Після чого він питає, чи є ще фото з заходу. Ми надсилаємо ще одне фото, він ставить теж саме питання і ми надсилаємо фото до того моменту, поки не натиснемо кнопку «Готово» (рис.3.11).

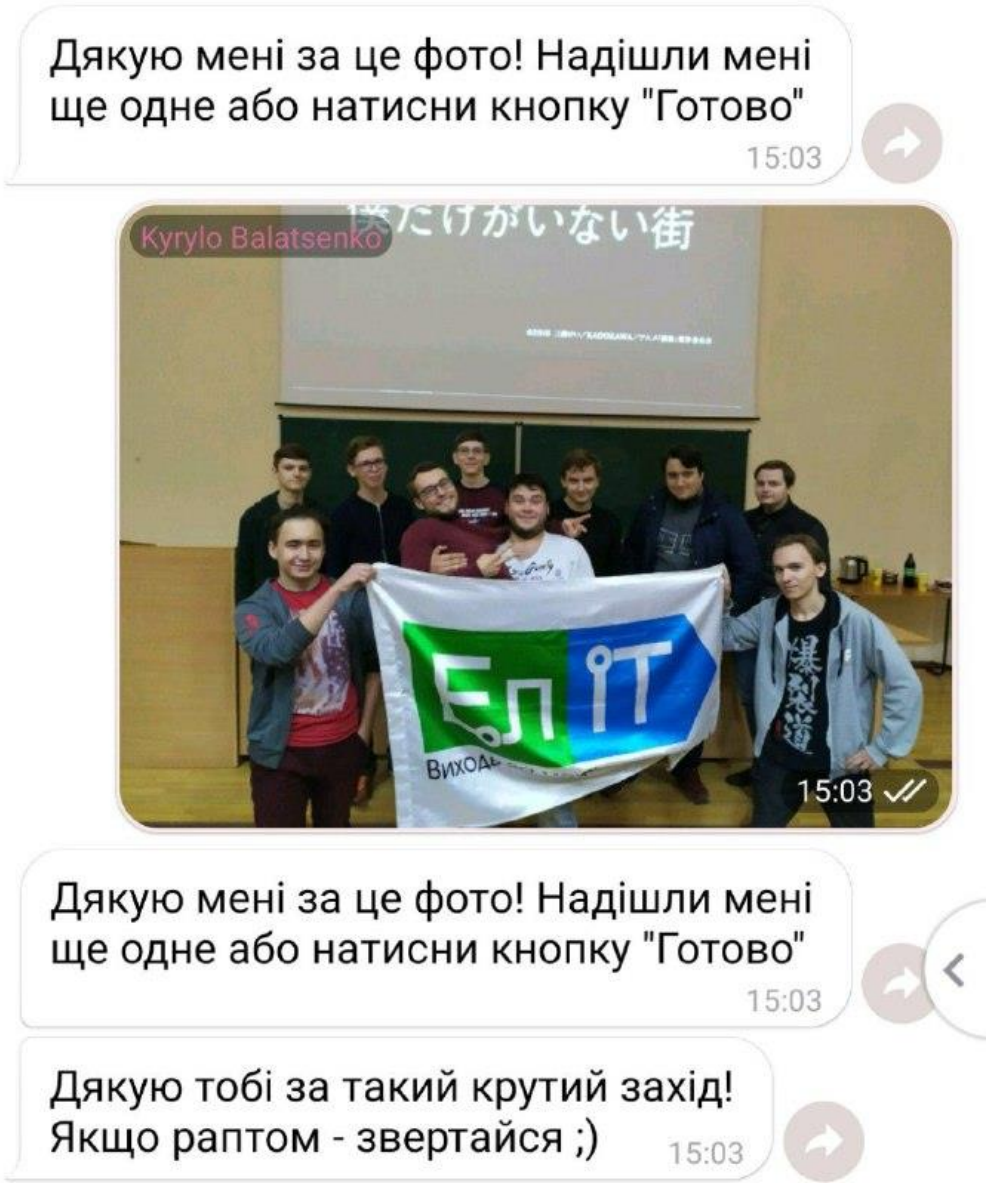

Рисунок 3.11 – Чат із ботом перед та після натискання кнопки «Готово»

Кнопка «Отримати звіт» дозволяє нам отримати презентацію із усіма заходами, про які ми звітували перед ботом. Ми натискаємо на кнопку і знову авторизуємося, після чого бот запитує, чи дійсно ми хочемо отримати презентацію. При натисканні на кнопку «Так», бот збирає дані згідно вашого факультету, передає їх до генератора презентацій і на основі цих даних створюється звіт у форматі .pptx, який потім надсилається ботом і його можна завантажити.

На рисунку 3.12 представлено чат з ботом після використання функції «Отримати звіт».

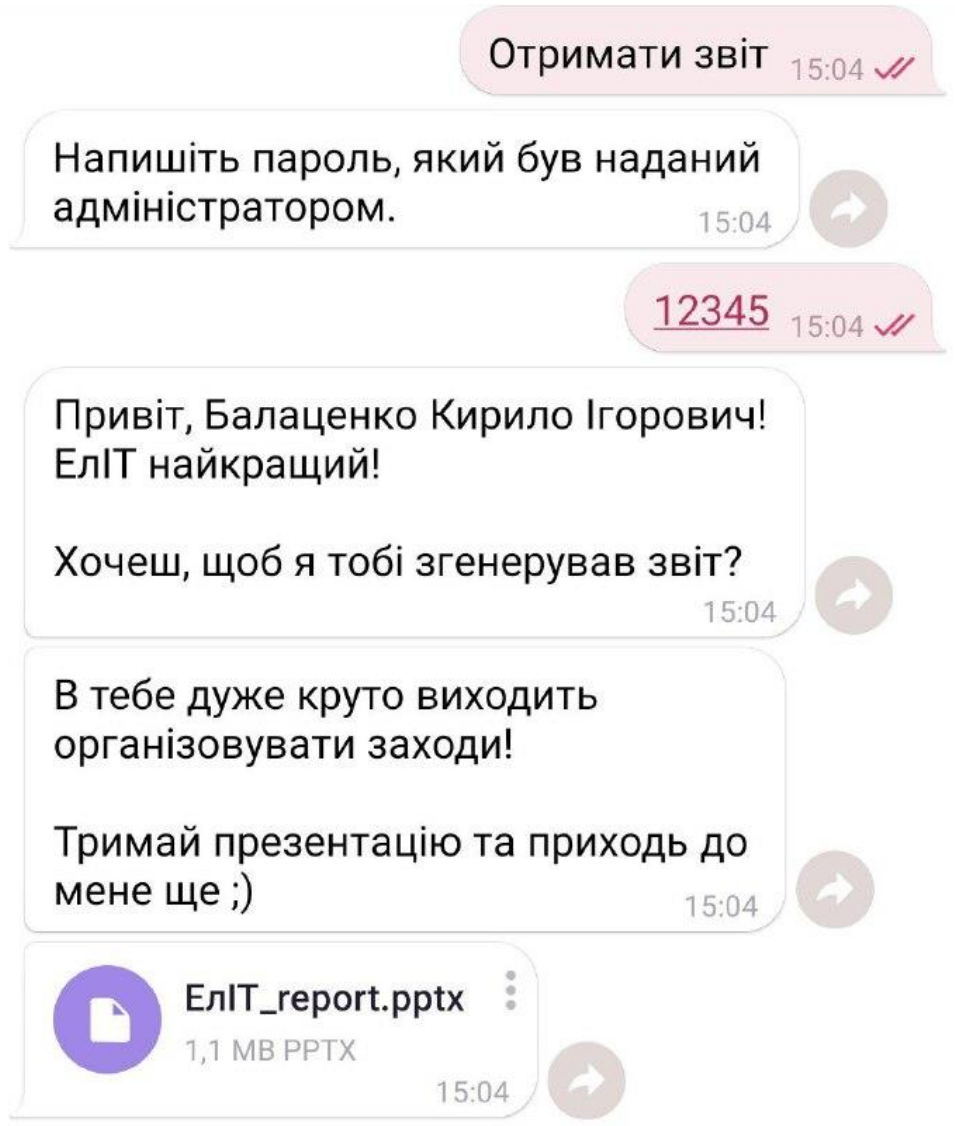

Рисунок 3.12 – Чат з ботом після натискання кнопки «Отримати звіт»

На рисунку 3.13 представлено приклад звітної презентації, яка генерується ботом.

Рисунок 3.13 – приклад звітної презентації, згенерованої ботом

ВИСНОВКИ

В ході виконання роботи були вирішені наступні задачі:

- розглянуті та проаналізовані сучасні мови програмування для створення telegram-бота
- проведено аналіз тематичної літератури та огляд існуючих рішень
- створено та налаштовано роботу бота
- проведено тестування працездатності бота

Результатом виконання роботи є створений функціональний telegram-бот “Khaba”, який допоможе членам органів студентського самоврядування у проведенні заходів, а головам ОСС – у спрощенні рутинної задачі, пов’язаної зі звітуванням в кінці каденції.

Розроблений чат бот має ряд особливостей:

- інтуїтивно зручний та зрозумілий інтерфейс
- меню з кнопками замість команд
- цілодобова робота

СПИСОК ЛІТЕРАТУРИ

1. Шаблони документів СумДУ [Електронний ресурс] – Режим доступу: document.sumdu.edu.ua
2. Положення про організацію відділу організаційно-методичної роботи органів студентського самоврядування [Електронний ресурс] – Режим доступу : https://normative.sumdu.edu.ua/?task=getfile&tmpl=component&id=10595837-2629-e811-89d6-001a4be6d04a&kind=1
3. Документація Aiogram [Електронний ресурс] – Режим доступу :
4. Регистрация бота в Telegram – BotFather [Електронний ресурс] – Режим доступу : https://way23.ru/%D1%80%D0%B5%D0%B3%D0%B8%D1%81%D1%82%D1%80%D0%B0%D1%86%D0%B8%D1%8F-%D0%B1%D0%BE%D1%82%D0%B0-%D0%B2-telegram/
5. Создание бота в Telegram. Основы [Електронний ресурс] – Режим доступу : https://wibe.team/sozdanie-bota-v-telegram/
6. Документація по створенню Telegram-ботів [Електронний ресурс] – Режим доступу : https://core.telegram.org/bots

ДОДАТОК А

Програмна реалізація ланцюгів

create_event (“Організувати захід”)

handlers.py

```
import re

from aiogram import types
from aiogram.dispatcher import FSMContext

from bot import files
from bot.chains.base.kb import start_kb
from bot.chains.create_event.kb import cancel_kb
from bot.chains.create_event.state import CreateEvent
from bot.core import dp, bot
from bot.tree import DocsPath

Docs = files.loadFile(DocsPath)


def mapping_json(nd, p):
    return Docs[0][nd][f"{p}"]


@dp.callback_query_handler(lambda x: x.data == "event_creation_cancel",
state="*")
async def cancel(c: types.CallbackQuery, state: FSMContext):
    await c.message.delete_reply_markup()
    await state.finish()
    await bot.send_message(c.from_user.id, "Скасовано",
reply_markup=start_kb)
    await c.answer("Скасовано")


@dp.message_handler(regexp="Організувати захід □", state="*")
async def list_event(msg: types.Message, state: FSMContext):
    answer = await msg.answer(
        "Напишіть списком, що вам потрібно для заходу.\n"
        "Наприклад:\n"
        "1. Кава-брейк\n"
        "2. Аудиторія\n"
        "3. Транспорт\n",
        reply_markup=cancel_kb,
    )
```

```
    await CreateEvent.wait_list.set()
    await state.update_data({"message": answer})


@dp.message_handler(state=CreateEvent.wait_list)
async def docs_for_event(msg: types.Message, state: FSMContext):
    data_state = await state.get_data()
    await data_state.get("message").delete_reply_markup()
    await state.update_data({"description": msg.text})
    need_docs = re.findall(r"\S+", msg.text)
    message_pattern = [
        f"<a href='{mapping_json(need_docs[i].lower(), 'link')}'>"
        f"{mapping_json(need_docs[i].lower(), 'name')}</a>\n"
        f"У кого підписати: {', '.join(mapping_json(need_docs[i].lower(),
'significants'))} \n"
        for i in range(len(need_docs))
    ]
    message = "\n".join(message_pattern)
    answer = await msg.answer(
        f"Отже, для цього заходу тобі потрібно: \n\n" f"{message}",
        reply_markup=cancel_kb,
        parse_mode="HTML",
    )
```

kb.py

```
from aiogram.types import InlineKeyboardMarkup, InlineKeyboardButton

cancel_kb = InlineKeyboardMarkup(
    inline_keyboard=[
        [InlineKeyboardButton("Скасувати",
callback_data="event_creation_cancel")]
    ],
    resize_keyboard=True,
)
```

state.py

```
from aiogram.dispatcher.filters.state import StatesGroup, State


class CreateEvent(StatesGroup):
    wait_event = State()
    wait_list = State()
```

get_report (“Отримати звіт”)

handlers.py

```
import os

from aiogram import types
from aiogram.dispatcher import FSMContext

from bot.chains.base.kb import start_kb
from bot.chains.get_report.kb import choose, cancel_kb
from bot.chains.get_report.state import ReportEvents
from bot.chains.record_event.state import RecordEvent
from bot.core import dp, bot
from db.config import BASE_DIR
from db.models.user import User
from presentation.presentation_creator import create_presentation
from db.models.events import Event


@dp.callback_query_handler(lambda x: x.data == "event_report_cancel",
state="*")
async def cancel(c: types.CallbackQuery, state: FSMContext):
    await c.message.delete_reply_markup()
    await state.finish()
    await bot.send_message(
        c.from_user.id, "Скасовано створення звіту про захід",
reply_markup=start_kb
    )
    await c.answer("Скасовано")


@dp.message_handler(regexp="Отримати звіт", state="*")
async def auth(msg: types.Message, state: FSMContext):
    await ReportEvents.wait_for_password.set()
    data_state = await msg.answer(
        "Напишіть пароль, який був наданий адміністратором.",
reply_markup=cancel_kb
    )
    await state.update_data({"message": data_state})


@dp.message_handler(state=ReportEvents.wait_for_password)
async def report_event_choose(msg: types.Message, state: FSMContext):
    data = await state.get_data()
    await data.get("message").delete_reply_markup()
    check_password = await User.check_password(msg.text)
    if not check_password:
```

```
        await msg.answer("Друже, це невірний пароль.",
reply_markup=cancel_kb)
    else:
        user_data = await User.user_data(msg.text)
        await state.update_data(
            {
                "username": user_data["name"],
                "faculty": user_data["faculty"],
                "user": user_data["id"],
            }
        )
        answer = await msg.answer(
            f'Привіт, {user_data["name"]}! {user_data["faculty"]} найкращий!
\n\n'
            f"Хочеш, щоб я тобі згенерував звіт?",
            reply_markup=choose,
        )
        await RecordEvent.wait_event_name.set()
        await state.update_data({"message": answer})


@dp.callback_query_handler(lambda x: x.data == "event_report_approve",
state="*")
async def generate_presentation(c: types.CallbackQuery, state: FSMContext):
    await c.message.delete_reply_markup()
    data = await state.get_data()
    events = await Event.query.where(User.id == data.get("user")).gino.all()
    create_presentation(
        [events[k].event_name for k in range(len(events))],
        [await events[j].photo for j in range(len(events))],
        data.get("faculty"),
    )
    pres_file = open(
        os.path.join(BASE_DIR,
f'presentations/{data.get("faculty")}_report.pptx'), "rb"
    )
    await bot.send_message(
        c.from_user.id,
        "В тебе дуже круто виходить організовувати заходи! \n\n"
        "Тримай презентацію та приходь до мене ще ;)",
    )
    await bot.send_document(c.from_user.id, pres_file)
    await state.finish()
```

kb.py

```
from aiogram.types import InlineKeyboardMarkup, InlineKeyboardButton

choose = InlineKeyboardMarkup(
    inline_keyboard=[
        [
            InlineKeyboardButton("Так",
callback_data="event_report_approve"),
            InlineKeyboardButton("Ні", callback_data="event_report_cancel"),
        ]
    ],
    resize_keyboard=True,
)

cancel_kb = InlineKeyboardMarkup(
    inline_keyboard=[
        [InlineKeyboardButton("Скасувати",
callback_data="event_report_cancel")]
    ]
)
```

state.py

```
from aiogram.dispatcher.filters.state import StatesGroup, State


class ReportEvents(StatesGroup):
    wait_for_password = State()
```

record_event (“Захід”)

handlers.py

```
import os
import random

from aiogram import types
from aiogram.dispatcher import FSMContext

from bot.chains.base.kb import start_kb
from bot.chains.record_event.kb import cancel_record, done_kb
from bot.chains.record_event.state import RecordEvent
from bot.core import dp, bot
from db.models.user import User
from db.models.events import Event
```

```
from db.models.photos import Photos
from db.config import UPLOAD_DIR


@dp.callback_query_handler(lambda x: x.data == "event_record_cancel",
state="*")
async def cancel(c: types.CallbackQuery, state: FSMContext):
    await c.message.delete_reply_markup()
    await state.finish()
    await bot.send_message(
        c.from_user.id, "Скасовано створення звіту про захід",
reply_markup=start_kb
    )
    await c.answer("Скасовано")


@dp.message_handler(regexp="Захід", state="*")
async def auth(msg: types.Message, state: FSMContext):
    await RecordEvent.wait_password.set()
    data_state = await msg.answer(
        "Напишіть пароль, який був наданий адміністратором.",
reply_markup=cancel_record
    )
    await state.update_data({"message": data_state})


@dp.message_handler(state=RecordEvent.wait_password)
async def record_event_start(msg: types.Message, state: FSMContext):
    data = await state.get_data()
    await data.get("message").delete_reply_markup()
    check_password = await User.check_password(msg.text)
    if not check_password:
        await msg.answer("Друже, це невірний пароль.",
reply_markup=cancel_record)
    else:
        user_data = await User.user_data(msg.text)
        await state.update_data(
            {
                "username": user_data["name"],
                "faculty": user_data["faculty"],
                "user": user_data["id"],
            }
        )
        answer = await msg.answer(
            f'Привіт, {user_data["name"]}! {user_data["faculty"]} найкращий!
\n\n'
            f"Поділися зі мною назвою заходу, який Ви провели :)",
            reply_markup=cancel_record,
        )
```

```
        await RecordEvent.wait_event_name.set()
        await state.update_data({"message": answer})


@dp.message_handler(state=RecordEvent.wait_event_name)
async def record_event_name(msg: types.Message, state: FSMContext):
    data = await state.get_data()
    await data.get("message").delete_reply_markup()
    await state.update_data({"event_name": msg.text})
    answer = await msg.answer(
        f"Надішли мені перше фото цього заходу!", reply_markup=cancel_record
    )
    event = await Event.create(event_name=msg.text, user=data.get("user"))
    await state.update_data({"message": answer, "event_id": event.id})
    await RecordEvent.wait_event_photos.set()


@dp.message_handler(state=RecordEvent.wait_event_photos,
content_types=["photo"])
async def event_photos(msg: types.Message, state: FSMContext):
    data = await state.get_data()
    await data.get("message").delete_reply_markup()
    media_path = os.path.join(
        UPLOAD_DIR, f'{data.get("event_name")}_{random.randint(0,
19999)}.jpg'
    )
    await msg.photo[-1].download(media_path)
    answer = await msg.answer(
        'Дякую мені за це фото! Надішли мені ще одне або натисни кнопку
"Готово"',
        reply_markup=done_kb,
    )
    await Photos.create(photo_path=media_path,
parent_id=data.get("event_id"))
    await state.update_data({"message": answer})
    await RecordEvent.wait_event_photos.set()


@dp.callback_query_handler(lambda x: x.data == "done_photos", state="*")
async def cancel(c: types.CallbackQuery, state: FSMContext):
    await c.message.delete_reply_markup()
    await state.finish()
    await bot.send_message(
        c.from_user.id,
        "Дякую тобі за такий крутий захід! Якщо раптом - звертайся ;)",
        reply_markup=start_kb,
    )
    await c.answer("Готово")
```

kb.py

```
from aiogram.types import InlineKeyboardMarkup, InlineKeyboardButton

cancel_record = InlineKeyboardMarkup(
    inline_keyboard=[
        [InlineKeyboardButton("Скасувати",
callback_data="event_record_cancel")]
    ],
    resize_keyboard=True,
)

done_kb = InlineKeyboardMarkup(
    inline_keyboard=[[InlineKeyboardButton("Готово",
callback_data="done_photos")]],
    resize_keyboard=True,
)
```

state.py

```
from aiogram.dispatcher.filters.state import StatesGroup, State


class RecordEvent(StatesGroup):
    wait_password = State()
    wait_event_name = State()
    wait_event_photos_count = State()
    wait_event_photos = State()
```

admin_panel (“Залогінитися як адмін”)

handlers.py

```
import hashlib

from aiogram import types
from aiogram.dispatcher import FSMContext
from aiogram.types import User as TgUser

from bot.chains.admin_panel.kb import admin_start_kb, cancel_kb
from bot.chains.admin_panel.state import AdminPanel
from bot.config import ADMIN_CHAT_ID
from bot.core import dp, bot
from db.models.user import User
```

```
@dp.callback_query_handler(lambda x: x.data == "cancel_activity", state="*")
async def cancel(c: types.CallbackQuery, state: FSMContext):
    await c.message.delete_reply_markup()
    await state.finish()
    await bot.send_message(c.from_user.id, "Скасовано",
reply_markup=admin_start_kb)
    await c.answer("Скасовано")


@dp.message_handler(regexp="Залогінитися як адмін □", state="*")
async def admin_start(msg: types.Message, state: FSMContext):
    if TgUser.get_current().id != int(ADMIN_CHAT_ID):
        await msg.answer("Тобі сюди не можна, друже.")
    else:
        await msg.answer(
            "Вітаю, мій господар! \n\n Скажи, що я можу для тебе зробити.",
            reply_markup=admin_start_kb,
        )


@dp.callback_query_handler(lambda x: x.data == "user_creation", state="*")
async def name_head(c: types.CallbackQuery, state: FSMContext):
    await c.message.delete_reply_markup()
    await AdminPanel.wait_name.set()
    answer = await bot.send_message(
        c.from_user.id,
        "Напиши ПІБ студентського декана, директора або ректора.",
        reply_markup=cancel_kb,
    )
    await state.update_data({"message": answer})


@dp.message_handler(state=AdminPanel.wait_name)
async def faculty_head(msg: types.Message, state: FSMContext):
    data = await state.get_data()
    await data.get("message").delete_reply_markup()
    answer = await msg.answer(
        f"Напишіть факультет або інститут, головою ОСС якого є {msg.text}.
\n\n"
        f"Якщо це студентський ректор - просто напишіть 'Студректорат'",
        reply_markup=cancel_kb,
    )
    await state.update_data({"name": msg.text, "message": answer})
    await AdminPanel.wait_faculty.set()


@dp.message_handler(state=AdminPanel.wait_faculty)
async def password_head(msg: types.Message, state: FSMContext):
    data = await state.get_data()
```

```
    await data.get("message").delete_reply_markup()
    answer = await msg.answer(
        "Вигадайте пароль для користувача", reply_markup=cancel_kb
    )
    await state.update_data({"faculty": msg.text, "message": answer})
    await AdminPanel.wait_password.set()


@dp.message_handler(state=AdminPanel.wait_password)
async def creating_user(msg: types.Message, state: FSMContext):
    await state.update_data({"password": msg.text})
    data = await state.get_data()
    await data.get("message").delete_reply_markup()
    await User.create(
        name=data.get("name"),
        faculty=data.get("faculty"),
        password=hashlib.md5(data.get("password").encode("utf-
8")).hexdigest(),
    )
    await msg.answer(
        "Молодець! Юзера створено. \n\n Може ще чимось допомогти?",
        reply_markup=admin_start_kb,
    )
    await state.finish()


@dp.callback_query_handler(lambda x: x.data == "user_delete", state="*")
async def delete_user_start(c: types.CallbackQuery, state: FSMContext):
    await c.message.delete_reply_markup()
    await AdminPanel.wait_name_delete.set()
    answer = await bot.send_message(
        c.from_user.id,
        "Напиши ПІБ користувача, якого ти хочеш вилучити.",
        reply_markup=cancel_kb,
    )
    await state.update_data({"message": answer})


@dp.message_handler(state=AdminPanel.wait_name_delete)
async def deleting(msg: types.Message, state: FSMContext):
    await state.update_data({"name": msg.text})
    data = await state.get_data()
    await data.get("message").delete_reply_markup()
    await User.delete.where(User.name == data.get("name")).gino.first()
    await msg.answer(
        "Шкода, що ти видалив юзера. Напевно, так треба."
        "\n\n Може ще чимось допомогти?",
        reply_markup=admin_start_kb,
    )
```

```
    await state.finish()


@dp.callback_query_handler(lambda x: x.data == "list_user", state="*")
async def get_list(c: types.CallbackQuery):
    users = await User.query.gino.all()
    message_pattern = [
        f"{users[i].id}. {users[i].name}, {users[i].faculty} \n"
        for i in range(len(users))
    ]

    await bot.send_message(
        c.from_user.id,
        f'{"".join(message_pattern)}\n\n ' f"Чи можу ще чимось допомогти?",
        reply_markup=admin_start_kb,
    )
```

kb.py

```
from aiogram.types import InlineKeyboardMarkup, InlineKeyboardButton

admin_start_kb = InlineKeyboardMarkup(
    inline_keyboard=[
        [
            InlineKeyboardButton("Додати користувача",
callback_data="user_creation"),
            InlineKeyboardButton("Вилучити користувача",
callback_data="user_delete"),
            InlineKeyboardButton("Список користувачів",
callback_data="list_user"),
        ]
    ],
    resize_keyboard=False,
)

cancel_kb = InlineKeyboardMarkup(
    inline_keyboard=[
        [InlineKeyboardButton("Скасувати", callback_data="activity_cancel")]
    ]
)
```

state.py

```
from aiogram.dispatcher.filters.state import StatesGroup, State


class AdminPanel(StatesGroup):
    wait_name = State()
    wait_faculty = State()
    wait_password = State()
    wait_faculty_report = State()
    wait_name_delete = State()
```

ДОДАТОК Б

Програмна реалізація проміжних засобів

throttling.py

```
import asyncio

from aiogram import Dispatcher, types
from aiogram.dispatcher.handler import current_handler, CancelHandler
from aiogram.dispatcher.middlewares import BaseMiddleware
from aiogram.utils.exceptions import Throttled

from bot.config import DEFAULT_RATE_LIMIT


def rate_limit(limit: int, key=None):
    """
    Decorator for configuring rate limit and key in different functions.
    :param limit:
    :param key:
    :return:
    """

    def decorator(func):
        setattr(func, "throttling_rate_limit", limit)
        if key:
            setattr(func, "throttling_key", key)
        return func

    return decorator


class ThrottlingMiddleware(BaseMiddleware):
    """
    Simple middleware
    """

    def __init__(self, limit=DEFAULT_RATE_LIMIT, key_prefix="antiflood_"):
        self.rate_limit = limit
        self.prefix = key_prefix
        super(ThrottlingMiddleware, self).__init__()

    async def on_process_message(self, message: types.Message, data: dict):
        """
        This handler is called when dispatcher receives a message
        :param data:
        :param message:
```

```
        """
        # Get current handler
        handler = current_handler.get()

        # Get dispatcher from context
        dispatcher = Dispatcher.get_current()
        # If handler was configured, get rate limit and key from handler
        if handler:
            limit = getattr(handler, "throttling_rate_limit",
self.rate_limit)
            key = getattr(
                handler, "throttling_key",
f"{self.prefix}_{handler.__name__}"
            )
        else:
            limit = self.rate_limit
            key = f"{self.prefix}_message"

        # Use Dispatcher.throttle method.
        try:
            await dispatcher.throttle(key, rate=limit)
        except Throttled as t:
            # Execute action
            await self.message_throttled(message, t)

            # Cancel current handler
            raise CancelHandler()

    async def message_throttled(self, message: types.Message, throttled:
Throttled):
        """
        Notify user only on first exceed and notify about unlocking only on
last exceed
        :param message:
        :param throttled:
        """
        handler = current_handler.get()
        dispatcher = Dispatcher.get_current()
        if handler:
            key = getattr(
                handler, "throttling_key",
f"{self.prefix}_{handler.__name__}"
            )
        else:
            key = f"{self.prefix}_message"

        # Calculate how many time is left till the block ends
        delta = throttled.rate - throttled.delta
```

```
        # Prevent flooding
        if throttled.exceeded_count <= 10:
            await message.reply("Не поспішай, не встигаю читати 🙃")

        # Sleep.
        await asyncio.sleep(delta)

        # Check lock status
        thr = await dispatcher.check_key(key)

        # If current message is not last with current key - do not send
message
        if thr.exceeded_count == throttled.exceeded_count:
            await message.reply("Все, дочитав")
```

ДОДАТОК В

Програмна реалізація основи бота

tree.py

```
import os

from db.config import BASE_DIR

DocsPath = os.path.join(BASE_DIR, "data/docs.json")
```

config.py

```
import os

from envparse import env

BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

env.read_envfile(os.path.join(BASE_DIR, "sgassistant/.env"))

TELEGRAM_BOT_TOKEN = env.str("TELEGRAM_BOT_TOKEN")

ADMIN_CHAT_ID = env.str("ADMIN_CHAT_ID")

DEFAULT_RATE_LIMIT = 2
```

files.py

```
import json

import codecs


def loadFile(FilePath):
    with codecs.open(FilePath, encoding="utf-8") as file:
        data = json.loads(file.read())

    return data


def saveFile(data, FilePath):
    with codecs.open(FilePath, "w", encoding="utf-8") as file:
        json.dump(data, file)
```

core.py

```
from aiogram import Bot, Dispatcher
from aiogram.contrib.fsm_storage.memory import MemoryStorage
```

```
from aiogram.utils.executor import Executor

from bot.config import TELEGRAM_BOT_TOKEN
from middlewares.throttling import ThrottlingMiddleware

bot = Bot(token=TELEGRAM_BOT_TOKEN, parse_mode="HTML")

dp = Dispatcher(bot=bot, storage=MemoryStorage())

dp.middleware.setup(ThrottlingMiddleware())

executor = Executor(dp, skip_updates=True)
```

__main__.py

```
import logging

from aiogram import Dispatcher

from bot.core import executor
from db import db
from db.config import POSTGRES_URI

from bot.chains import *


async def on_startup(dp: Dispatcher):
    logging.info("Setup PostgreSQL Connection")
    await db.set_bind(POSTGRES_URI)


async def on_shutdown(dp: Dispatcher):
    bind = db.pop_bind()
    if bind:
        logging.info("Close PostgreSQL Connection")
        await bind.close()


def main():
    executor.on_startup(on_startup)
    executor.on_shutdown(on_shutdown)

    executor.start_polling()


if __name__ == "__main__":
    main()
```

ДОДАТОК Д

Програмна реалізація роботи з БД

db/models

user.py

```
import hashlib

from db.core import db
from db.models.base import TimedBaseModel


class User(TimedBaseModel):
    __tablename__ = "users"

    id = db.Column(db.Integer, primary_key=True, index=True, unique=True)
    name = db.Column(db.String(255), nullable=False)
    password = db.Column(db.String(255), nullable=False)
    faculty = db.Column(db.String(50), nullable=False)
    is_blocked = db.Column(db.Boolean, default=False)

    @classmethod
    async def check_password(cls, password):
        password = await User.query.where(
            User.password == hashlib.md5(password.encode("utf-
8")).hexdigest()
        ).gino.first()
        return bool(password)

    @classmethod
    async def user_data(cls, password):
        usr = (
            await User.select("id", "name", "faculty")
            .where(User.password == hashlib.md5(password.encode("utf-
8")).hexdigest())
            .gino.first()
        )
        return usr
```

photos.py

```
from db.core import db
from db.models.base import TimedBaseModel
```

```
class Photos(TimedBaseModel):
    __tablename__ = "photos"

    id = db.Column(db.Integer, primary_key=True, index=True, unique=True)

    # event = db.Column(db.String(50), nullable=False)
    event_id = db.Column(db.Integer, db.ForeignKey("events.id"))
    photo_path = db.Column(db.String(255), nullable=False)
```

events.py

```
from db.core import db
from db.models.base import TimedBaseModel
from .photos import Photos
from .user import User


class Event(TimedBaseModel):
    __tablename__ = "events"

    id = db.Column(db.Integer, primary_key=True, index=True, unique=True)

    event_name = db.Column(db.String(50))

    user = db.Column(db.Integer, nullable=False)

    def __init__(self, **kw):
        super().__init__(**kw)
        self._photo = set()

    @property
    async def photo(self):
        image = (
            await Photos.select("photo_path")
            .where(Photos.parent_id == self.id)
            .gino.all()
        )
        return [image[i].photo_path for i in range(len(image))]
```

base.py

```
import datetime
from typing import List

import sqlalchemy as sa
```

```
from db.core import db


class BaseModel(db.Model):
    __abstract__ = True

    def __str__(self):
        model = self.__class__.__name__
        table: sa.Table = sa.inspect(self.__class__)
        primary_key_columns: List[sa.Column] = table.primary_key.columns
        values = {
            column.name: getattr(self, self._column_name_map[column.name])
            for column in primary_key_columns
        }
        values_str = " ".join(f"{name}={value!r}" for name, value in
values.items())
        return f"<{model} {values_str}>"


class TimedBaseModel(BaseModel):
    __abstract__ = True

    created_at = db.Column(db.DateTime(True), server_default=db.func.now())
    updated_at = db.Column(
        db.DateTime(True),
        default=datetime.datetime.utcnow,
        onupdate=datetime.datetime.utcnow,
        server_default=db.func.now(),
    )
```

db

config.py

```
import os

from envparse import env

BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

env.read_envfile(os.path.join(BASE_DIR, ".env"))
UPLOAD_DIR = os.path.join(BASE_DIR, "uploads/")

POSTGRES_HOST = env.str("DB_HOST", default="localhost")

POSTGRES_PORT = env.int("DB_PORT", default=5432)
POSTGRES_PASSWORD = env.str("DB_PASSWORD", default="")
POSTGRES_USER = env.str("DB_USERNAME")
POSTGRES_DB = env.str("DB_NAME")
POSTGRES_URI =
f"postgresql://{POSTGRES_USER}:{POSTGRES_PASSWORD}@{POSTGRES_HOST}:{POSTGRES_
PORT}/{POSTGRES_DB}"
```

core.py

```
from gino_aiohttp import Gino

db = Gino()
```