НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО

В.о. завідувача кафедри

_____Олександр КОВАЛЬ

«_____»_____202_р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем в енергетиці»
спеціальності 121 Інженерія програмного забезпечення
на тему: «Автоматизована система комунікації між випускниками на базі Telegram-бота»

Виконала:
студентка IV курсу, групи ТВ-13
Сейкаускайте Аліна Андріївна _____
(прізвище, ім'я, по батькові) (підпис)

Керівник:
д.ф. ст.викл. Бандурка О.І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент:
д.ф. ст. викл. Некрашевич О. В._____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студентка_____
(підпис)

Київ – 2025

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення інтелектуальних кібер - фізичних систем в енергетиці»

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

_____ Олександр КОВАЛЬ (підпис)

«____»_____202_р.

ЗАВДАННЯ

на дипломну роботу студенту

Сейкаускайте Аліні Андріївні
(прізвище, ім'я, по батькові)

1. Тема роботи
Автоматизована система комунікації між випускниками кафедри на базі телеграм-бота

керівник роботи д.ф. ст. викл. Бандурка Олена Іванівна, _____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від "__"_____2025 року №_____
2. Строк подання студентом роботи _____
3. Вихідні дані до роботи мова програмування Python, Aiogram, Telegram Bot API, середовище розробки Visual Studio Code
4. Зміст (дипломної роботи) пояснювальної записки (перелік завдань, які потрібно розробити) розробити основні функції бота, такі як створення груп для комунікації, можливість публікування новин та оголошень, створення нагадувань про зустрічі, пошук і зв'язок між випускниками за інтересами або спеціалізацією, надсилання файлів, фото- та відеоматеріалів; розробити архітектуру бота за допомогою Python з урахуванням безпеки та зручності, а також налаштування бази даних для зберігання інформації про випускників; розробити основні модулі: можливість керування ботом з боку адміністратора, впровадження функціоналу для розсилки повідомлень, автентифікація випускників; тестування бота на різних пристроях та в різних умовах, для перевірки функціональності та продуктивності.

5. Перелік ілюстративного матеріалу діаграми акторів, діаграма об’єктів предметної області, діаграма принципу життєвого циклу EC2 та потоку даних FSM.

6. Дата видачі завдання «01» __жовтня__ 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Строки виконання етапів роботи	Примітка
1	Отримання завдання	01.10.2024	Виконано
2	Дослідження предметної області	02.10.2024 -01.11.2024	Виконано
3	Постановка вимог до проєктування системи	02.11.2024 – 01.12.2024	Виконано
4	Дослідження існуючих рішень	02.12.2024 – 03.03.2025	Виконано
5	Розробка програмного продукту	04.04.2025 – 18.05.2025	Виконано
6	Тестування	04.04.2025 – 18.05.2025	Виконано
7	Захист програмного продукту	15.05.2025	Виконано
8	Оформлення дипломної роботи	14.04.2025 – 06.06.2025	Виконано
9	Передзахист	02.06.2025 – 06.06.2025	Виконано
10	Захист	16.06.2025 – 27.06.2025	Виконано

Студент _____ Аліна СЕЙКАУСКАЙТЕ
(підпис) (ім’я, прізвище)

Керівник роботи _____ Олена БАНДУРКА
(підпис) (ім’я, прізвище)

РЕФЕРАТ

Структура та обсяг дипломної роботи: робота містить 59 сторінок, 25 рисунків, 3 додатки та 23 посилання.

Мета роботи: створити Telegram-бота, що забезпечує автоматизоване спілкування між випускниками з можливістю отримання логістичних даних адміністрацією кафедри. Основною проблемою є відсутність зручного та єдиного інструменту для обміну інформацією, реєстрації, організації подій та підтримки зв'язку з випускниками. Розроблений бот забезпечує простий та інтуїтивно зрозумілий інтерфейс для взаємодії, дозволяє створювати новини, події, спільні чати, надсилати медіа-матеріали, а також редагувати профілі користувачів.

Для досягнення поставленої мети було виконано наступні завдання:

- Аналіз вимог**:** Досліджено потреби адміністрації кафедри та очікування випускників щодо функціональності системи. Визначено ключові сценарії взаємодії, потребу в авторизації та реєстрації, фільтрації за роком випуску, групою тощо.
- Проектування системи**:** Розроблено загальну структуру Telegram-бота, побудовано структуру бази даних, обрано Telegram API, SQLite та бібліотеку Aiogram для реалізації логіки взаємодії з користувачем.
- Розробка програмного забезпечення**:** Створено основні модулі для реєстрації, входу в систему, перегляду та редагування профілю, адміністрування, публікації новин, створення подій, спілкування у чатах за групами та спеціальностями.
- Тестування і валідація**:** Проведено тестування бот-системи з реальними користувачами та адміністраторами, перевірено відповідність функціональності очікуванням та коректність збереження даних у базі.

Розроблена система буде корисною для кафедр або навчальних закладів та випускників, які бажають підтримувати зв'язок, покращити обмін інформацією та залучити їх до участі в подіях факультету.

Практичне значення одержаних результатів полягає у впровадженні якісного інструменту комунікації, що дозволяє спростити управління даними випускників, забезпечити актуальність контактної інформації та організувати ефективну розсилку матеріалів. Рішення протестовано на прикладі кафедри АПЕПС та готове до масштабування на інші підрозділи. Система дозволяє зменшити витрати на організацію заходів, забезпечити точне інформування та підвищити залученість випускників у життя кафедри.

Ключові слова: Telegram-бот, комунікація, випускники, кафедра, адміністрування, спілкування, новини, події, база даних, автоматизація, FSM-моделі.

ABSTRACT

Structure and scope of the thesis: the work contains 59 pages, 25 figures, 3 appendices and 23 references.

The purpose of the work is to create a Telegram bot that provides automated communication between graduates with the ability to obtain logistical data by the department administration. The main problem is the lack of a convenient and unified tool for exchanging information, registering, organizing events, and maintaining contact with graduates. The developed bot provides a simple and intuitive interface for interaction, allows you to create news, events, joint chats, send media materials, and edit user profiles.

To achieve the goal, the following tasks were performed:

• Requirement's analysis: The needs of the department administration and the expectations of graduates regarding the functionality of the system were studied. Key interaction scenarios were identified, the need for authorization and registration, filtering by year of graduation, group, etc.

• System design: The general structure of the Telegram bot was developed, the database structure was built, the Telegram API, SQLite and the Aiogram library were selected to implement the logic of interaction with the user.

• Software development: The main modules for registration, login, profile viewing and editing, administration, news publishing, event creation, and chat communication by groups and specialties were created.

• Testing and validation: The bot system was tested with real users and administrators, and the functionality was verified to meet expectations and the correctness of data storage in the database.

The developed system will be useful for departments or educational institutions and graduates who want to maintain contact, improve information exchange, and involve them in participation in the events of the faculty.

The practical significance of the results obtained lies in the implementation of a high-quality communication tool that allows simplifying the management of alumni data,

ensuring the relevance of contact information, and organizing effective distribution of materials. The solution was tested on the example of the APEPS department and is ready to be scaled to other departments. The system allows reducing the costs of organizing events, providing accurate information, and increasing the involvement of graduates in the life of the department.

Keywords: Telegram-bot, communication, graduates, department, administration, communication, news, events, database, automation, FSM-models.

ЗМІСТ

ВСТУП

В умовах глобалізації комунікація між людьми сильно змінилася. Зручність доступу до новин та своїх контактів, швидкість обміну інформацією та використання різних інтерактивних платформ у повсякденному житті значно полегшили взаємодію людей у суспільстві. Але навіть за наявності численної кількості різних месенджерів, пошт або навіть простих SMS-повідомлень, підтримка зв'язку між випускниками навчальних заходів, а особливо університетів та коледжів, залишається проблематичною. Після закінчення навчання випускники часто їдуть у інші міста, регіони або навіть країни, в той самий час часто втрачаючи контакт з одногрупниками, викладачами, а як наслідок і новини з життя кафедри.

Актуальність даної теми помітна при порівнянні з минулими роками. Наприклад, ще 70 років тому, у 1950-х роках, основними засобами зв'язку тоді були паперові листи, які надсилалися поштою, телефонні дзвінки за стаціонарними номерами, та, у найкращому випадку, відбувалися зустрічі під час різних заходів і, в основному, лише тоді, коли люди знаходилися в одному місті. Організація будь-якої зустрічі або повідомлення про зміни в житті кафедри могли зайняти дні, а інколи навіть і тижні. Таким чином випускники часто втрачали контакт назавжди, бо просто не мали зручного способу для підтримки регулярної комунікації. З появою мобільних телефонів ситуація значно змінилась і у кращу сторону: легший та швидший метод комунікації, можливість дзвінків на далекі відстані, проте проблеми на цьому не скінчилися. При втраті номеру телефону, зазвичай зникав і контакт з іншими людьми, відстань все ще була проблемою і спілкування досі залишалося обмеженим. Соціальні медіа платформи, як наприклад, Facebook або Instagram, допомогли з комунікацією, проте враховуючи появу нікнеймів, було важно знайти саме тих людей, кого треба.

Проте, дані труднощі можливо подолати, попри це, відсутність організованого та структурованого середовища для взаємодії між випускниками все ще залишається проблемою. Звичайні канали, як наприклад, електронна пошта

чи групи в соціальних мережах, не забезпечують того рівня надійності та охоплення аудиторії, що допомогло би змінити ситуацію. Саме тому актуальним ж впровадження автоматизованої системи на базі телеграм-бота, який дозволить не лише об’єднати випускників кафедри, але й створити умови для безперервної комунікації.

Основна проблема, яку покликана вирішити дана система — це відсутність каналу для комунікації випускників та труднощі спілкування один із одним. Сучасні месенджери, не зважаючи на популярність, не завжди є ефективними без спеціальних інструментів. Відсутність організованих груп, розсилок, управління та підтвердження належності до спільноти зменшує інтерес випускників не лише у комунікації, але й у розвитку їх кафедри.

Створення такої системи сприятиме полегшенню отримання відгуків про навчальний процес для кафедри, а також допоможе налагодити зв’язки між випускниками

1 ПОСТАНОВКА ЗАДАЧІ ТА ОСНОВНІ ЗАВДАННЯ

Мета: створення та адаптація зручної автоматизованої системи комунікації між випускниками кафедри на базі телеграм-бота з можливістю отримання логістичних даних для покращення сервісів кафедри. Система повинна спряти підтриманню зв’язків, ефективному обміну новинами та матеріалами, як наприклад, фото- та відео ресурси, організації спільних заходів, а також надати можливість інтерактивної взаємодії між випускниками в зручному та інтуїтивно зрозумілому форматі.

Для реалізації проекту було поставлено низку завдань. Саме розбиття мети на складові дає зрозуміти основні проблеми, що повинна вирішити система, і таким чином ми отримуємо наступні завдання:

- розробити основний функціонал бота: створення груп, публікація та перегляд новин та оголошень, створення нагадувань про зустрічі, пошук зв’язку для випускників за роком випуску, спеціальністю та групою, поширення файлів, фото- та відеоматеріалів;
- розробити надійну архітектуру системи, що базується на мові програмування Python із забезпеченням безпеки та зручного використання;
- створити та імплементувати базу даних для зберігання всієї необхідної інформації;
- реалізувати функціонал для модерації, розсилки та управління подіями;
- провести тестування бота на різних пристроях, написати юніттести для перевірки надійності, функціональності та продуктивності.

До основних функціональних можливостей можна віднести реєстрацію та автентифікацію користувачів, створення тематичних груп за роком випуску та спеціальністю, оповіщення про новини та події з можливістю їх додавання у календар, надсилання й отримання файлів, фото та відео, модерація та адміністрування чатів, а також автоматичні нагадування та повідомлення. Таким чином, випускники зможуть реєструватися через бот підтвердивши свою

приналежність до кафедри, а також отримають можливість для зміни та редагування своїх персональних даних; вони також отримають можливість переглядати новини та події кафедри з можливістю їх додавання у календар, отримання нагадувань та тощо. До основних завдань дипломної роботи, що допомогли у реалізації саме даного програмного продукту, можна віднести:

- проведення аналізу предметної області для кращого розуміння особливостей спілкування випускників;
- аналіз вже існуючих ботів та схожих систем;
- формування вимог для телеграм-бота;
- розробку структури СУБД для зберігання даних про користувачів, події, новини та файли, використовуючи SQLite;
- побудову архітектури системи мовою Python3.x;
- розробку адміністративного модулю бота з обмеженим і повним доступом;
- використання JSON-файлів для забезпечення надійності системи;
- тестування створеного функціоналу системи на різних пристроях, а саме десктоп, мобільна та веб-версії застосунку Telegram;
- підготовка звіту щодо використання бота з детальними поясненнями для користувачів;
- хостинг системи для її безперервної роботи для забезпечення автоматизації.

Дана система також допоможе сервісам кафедри отримувати дані для аналізу їх продуктивності, оцінку якості освіти, можливість отримання логістичних даних про кар’єру студентів після випуску та їх досягнення.

Висновок до розділу 1

У цьому розділі була визначена основна мета дипломної роботи — розробити Telegram-бот для автоматизації процесів взаємодії між випускниками кафедри та її адміністрацією. В результаті формалізовано наступні завдання: проєктування

системи реєстрації випускників з фільтрацією за роком випуску, спеціальністю та кафедрою; створення механізму адміністрування з розмежуванням прав доступу (звичайні адміністратори та супер-адміністратор); реалізація функцій публікації новин, організації подій, спілкування за спеціальностями та групами, а також збереження інформації у базі даних SQLite та JSON-файлах. Чітке формулювання цих завдань дозволило структуровано реалізувати проєкт та зрозуміти основні напрямки розробки програмного продукту.

2 АНАЛІЗ ІСНУЮЧИХ СИСТЕМ ТА ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

2.1 Існуючі системи комунікації між випускниками

Комунікація випускників залишається важливою частиною підтримки зв’язків. З розвитком сучасних інформаційних технологій та месенджерів, спілкування значно полегшилося та стало більш різноманітним. Мається на увазі, що стало можливим спілкуватися навіть перебуваючи в різних країнах, надсилати не лише фотографії, але й відео та аудіо, що стало невід’ємною частиною нашого життя. Для цього використовуються різноманітні платформи. Розпочнемо з соціальних мереж, а саме Facebook та LinkedIn. Дані платформи дозволяють створювати групи, поширювати новини, ділитися подіями. Наприклад, LinkedIn містить вже вбудований функціонал пошуку колишніх студентів за університетом, роком випуску або спеціальністю [1].

Групи Facebook заробили багато популярності в минулі роки, адже вони запропонували надійний спосіб для створення спільнот та комунікації онлайн. Однак, він має кілька досить вагомих проблем у його використанні, що змусило користувачів шукати альтернативи. Окрім цього, вони помітили, що після приєднання до груп вони перестали бачити їх нові публікації та оновлення у стрічці, якщо безпосередньо не перейти шукану сторінку. Алгоритми Facebook ставить у пріоритет саме той контент, що базується на заохочуванні користувача, що означає, що не всі члени групи можуть побачити нові публікації [2]. Таким чином, велика кількість інформації, як наприклад, про події може бути просто непомічена зацікавленою аудиторією, що значно погіршує можливості у спілкуванні та у зручній комунікації онлайн.

Наступним способом комунікації є електронна пошта. Це доволі традиційний, проте не дуже ефективний, спосіб поширення інформації між

випускниками, але він не забезпечує достатньої інтерактивності, обмежений у швидкості відповіді та у знаходженні інших людей, та листи часто можуть не лише просто не дійти до отримувача, але й просто потрапити до спаму [3].

Месенджери, в даному випадку Viber та WhatsApp, стали невід'ємною частиною нашого життя, а особливо комунікації. Вони забезпечують швидкий зв'язок проте не мають розвиненої системи керування ботами, погано підтримують модерацію великих груп і не передбачають гнучкого управління доступом до інформації. WhatsApp відомий своєю великою базою користувачів та безперешкодною інтеграцією з телефонним контактами [4], проте має гіршу надійність та конфіденційність, а Viber більше зосереджений на голосових повідомлень та відео дзвінків. Основними перевагами Telegram порівняно з іншими месенджерами є його унікальні функції, як самознищення повідомлень, хмарне сховище, велика ємність груп та більше налаштувань конфіденційності, він також відомий своїм ефективним використанням даних, хоча WhatsApp його дещо опереджає в даному плані. Telegram має кращі можливості для обміну даними, що дозволяє його користувачам надсилати файли до 2 ГБ.

Одними із найкращих систем для спілкування із випускниками є спеціалізовані платформи для випускників, наприклад, Graduway, AlumniFire, Almabase. Вони пропонують великий набір необхідних функцій, таких як база даних, події, розсилка та аналітика, проте зазвичай вони є комерційними продуктами, які можуть бути недоступними для освітніх закладів через високу вартість підписки. Дані системи набирають високу популярність у Сполучених Штатах Америки та в багатьох країнах Європейського Союзу[16; 21; 22; 23].

2.2 Методи автоматизації комунікації

Для автоматизованої комунікації між великою кількістю користувачів необхідне використання перевірених та надійних методів програмної інженерії, баз даних, хмарних або локальних серверів, а також доступних та зрозумілих форматів

збереження даних, що легко інтегрувати та як і мають низький поріг доступу. В ході реалізації даного програмного продукту було використано декілька сервісів, що допомогло автоматизувати дану систему. До них можна віднести Finite State Machine (FSM) [5], Telegram Bot API, SQLite, JSON та CSV-файли, а також хостинг через віртуальну машину VPS.

Станова машина є одним із найважливіших методів для розробки бота, адже вона дозволяє будувати логіку покрокової взаємодії користувача із системою, наприклад:

- під час реєстрації;
- під час заповнення даних свого профілю;
- під час створення новини або подій.

FSM реалізовано за допомогою бібліотеки aiogram, що підтримує асинхронне програмування і дозволяє контролювати бота в залежності від введених користувачем даних.

Telegram Bot API [9] це основний інструмент для створення ботів, що базується на HTTP-інтерфейсі створений для розробників, які заохочені в будуванні ботів Телеграм. Він надає доступ до подій (кнопки, повідомлення, callback-запити), обробку файлів, управління групами та чатами, модерацію та призначення прав доступу, а також підтримку клавіатур та inline-режимів при роботі з ботом.

База даних в даному проекті була створена за допомогою SQLite для зберігання даних про користувачів, події, новини, адміністраторів, файли, чати та для заявок на реєстрацію членів адміністрації. Дана СУБД має нескладну структуру, що легка та швидка в роботі, добре працює з Python та з Visual Studio Code. Підключитися до даної бази даних при роботі можна через програмний модуль sqlite3.

JSON є зручним для зберігання внутрішніх даних системи та для кращого контролю подій, новин та реєстрацій. Дані файли є невід’ємною частиною у синхронізації подій, збереження реєстраційних даних та кешування новин та

реєстрацій у зручному форматі для експорту, редагування та просто перегляду даних.

CSV-файли використовуються як проміжний спосіб збереження та обміну даними, зокрема в даному проекті вони використовуються для валідації номерів телефону в залежності від країни, збереження даних про групові чати та їх імпорт до бази даних. Використання даних файлів допомагає значно полегшити контроль над даними та їх експорт.

Для того, щоб зробити цю систему автоматизованою було використано також хостинг через віртуальну машину. Даний бот було розміщено на віртуальній машині, що надає багато варіантів щодо використання та запуску бота. Так створення віртуальної машини Linux Ubuntu дозволяє запускати бот у фоновому режимі, що дозволяє значно зменшити навантаження на пристрій, було встановлено необхідні бібліотеки для роботи з ботом, а також забезпечено копіювання файлів та є підтримка постійної доступності бота без прив’язки до локального серверу.

Таким чином, використання різних засобів для автоматизації системи допомогло створити її цілісною та готовою до інтеграції.

2.3 Опис предметної області

Предметна область цього проекту охоплює взаємодію випускників із кафедрою вищого навчального закладу. Відомо, що із завершенням навчання багато студентів втрачають зв'язок зі своїми колегами та викладачами, проте попит на підтримку зростає і не лише зі сторони студентів, але й кафедри [11; 17]. Це стосується як академічного аспекту, збір інформації про задоволення якістю освіти, професійні характеристики тощо, а також й кар’єрного аспекту, як наприклад, випускники могли би поширювати свій досвід із студентами молодших курсів, допомагати з місцями проходження практики або тренінгами [12; 13]. Дана система

допомогла би долучити людей, що могли б із цим допомогти до колективу кафедри та покращити стосунки між випускниками.

До ключових ролей у даній системі можна віднести:

- випускники (або користувачі) — це основні користувачі;
- кафедра (або адміністрація) — регулюють роботу бота з користувачами;
- супер-адміністратор — затверджує нових адміністраторів та має доступ до всіх функцій системи.

Адміністрація боту з обмеженим доступом може працювати лише з даними користувачів та виконувати весь можливий функціонал, а саме видалення або блокування користувача, зміна їх даних, тощо.

Супер-адміністратор має доступ до усіх адміністративних методів бота. Він може переглядати надіслані файли, працювати з користувачами, створювати та публікувати новини та події та переглядати реєстрації на них, затверджує нових адміністраторів та надсилає оголошення. Візуальне представлення можливостей ролей адміністраторів можна побачити на діаграмі акторів (рисунок 2.1).

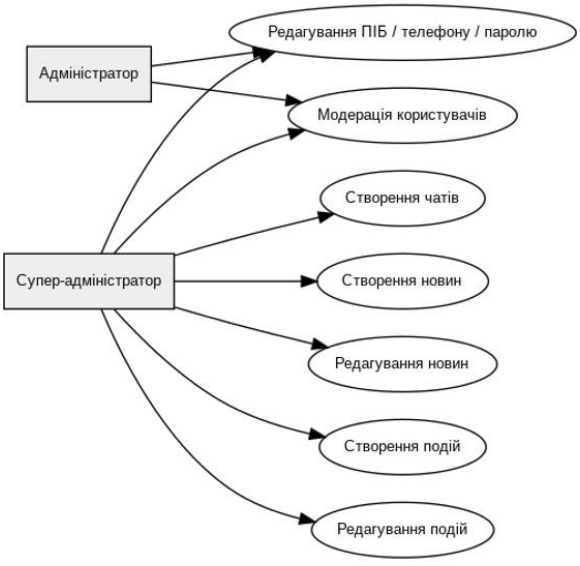


Рисунок 2.1 — Діаграма акторів для адміністратора та супер-адміністратора

Доступними для користування функціями у випускників є реєстрація з підтвердженням своєї приналежності до кафедри (за групою, роком вступу та

випуску, спеціальністю, тощо), отримувати новини та доступні події від кафедри, знаходити чати з випускниками за фільтрами, надсилання фото- та відео-матеріалів, а також запис на події та редагування власного профілю. Візуальне представлення можливостей користувача можна побачити на рисунку 2.2.

Рисунок 2.2 — Діаграма акторів для користувача

Дані діаграми дають краще зрозуміти обов’язки ролей у даній системі для проектування її функціональних можливостей та характеристик.

Інформаційними об’єктами предметної області є:

- профілі випускників, що містять дані про ПІБ, рік вступу, рік випуску, групу, спеціальність, день народження та телефон;
- події, вони мають назву, дату, опис, кількість місць та адресу;
- новини, що також мають повний та короткий опис, дату публікації та посилання;
- медіа, а саме фото та відео, що надсилаються користувачем;
- чати, що мають посилання та назву.

Для візуального представлення об’єктів предметної області було створену діаграму об’єктів предметної області (рисунок 2.3).

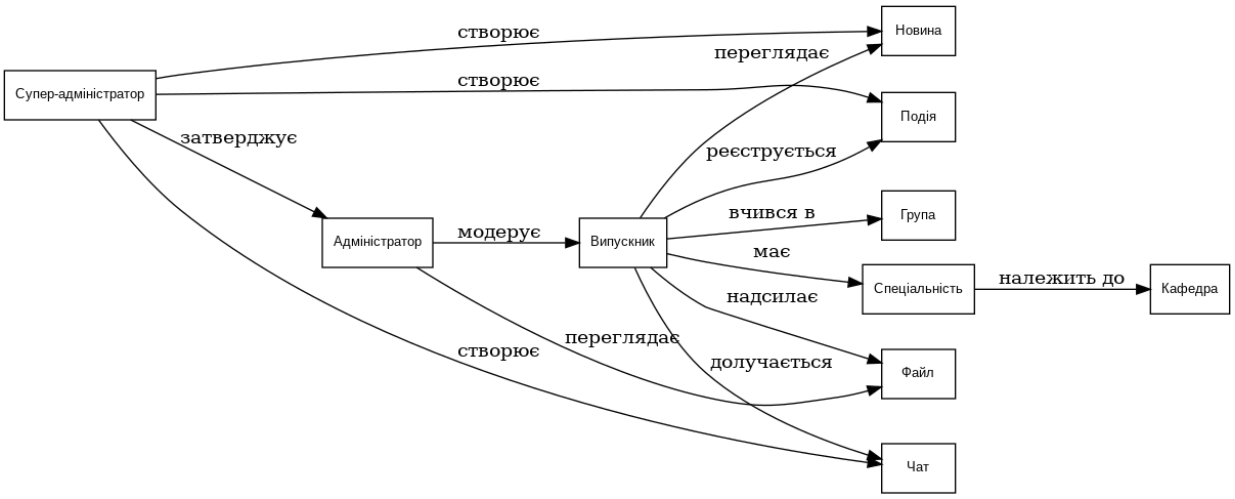


Рисунок 2.3 — Діаграма об’єктів предметної області

Також для оцінки якості розроблюваної системи, її сильні та слабкі сторони, а також можливості та загрози, проведемо так званий SWOT-аналіз на рисунку 2.4.. Даний аналіз це аналітичний метод, який використовується для оцінки, пов’язаний із конкуренцією, що впливають на проект або бізнес.

S - Сильні сторони	**W - Слабкі сторони**
1. Популярність Телеграму серед молоді 2. Гнучкість і масштабованість 3. Миттєве розповсюдження інформації 4. Простота використання	1. Залежність від Телеграму 2. Не всі випускники користуються 3. Потреба у підтримці та супроводі 4. Початкова складність для не технічних адміністраторів
T - Загрози	**O - Можливості**
1. Можливе блокування Телеграму в деяких країнах 2. Зміни в політиці API Telegram 3. Ризики втрати даних без резервного копіювання 4. Низький рівень залученості випускників	1. Інтеграція з веб-інтерфейсами, Firebase, Google Calendar 2. Масштабування на інші кафедри 3. Залучення випускників до менторства та подій 4. Автоматизація опитувань

Рисунок 2.4 — SWOT-аналіз

Таким чином, можна зробити висновок, що використання саме наших технологій для створення програмного продукту дає змогу задовольнити

специфічні потреби випускників, адаптувати систему до розвитку та мінімізувати можливі ризики та бар’єри

Висновок до розділу 2

У цьому розділі було здійснено порівняльний аналіз застосунків, шо використовуються у навчальних закладах для комунікації випускників. Аналіз показав, що наявні рішення переважно реалізують лише окремі функції та зазвичай є недоступними у безкоштовному варіанті — реєстрацію студентів, розсилку повідомлень або перегляд розкладу.

Водночас жодна з систем не поєднує повний спектр можливостей, які були потрібні в межах нашого проєкту: покрокова реєстрація з валідацією даних, чат-групи за роком вступу або спеціальністю, підтвердження прав адміністратора, генерація коротких описів новин за допомогою моделі Hugging Face, обробка подій із нагадуванням, перегляд і фільтрація файлів користувачів. Це обґрунтувало необхідність розробки власного рішення, що було спроєктовано в рамках дипломної роботи.

3 ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір мови програмування та бібліотек

Telegram це не просто популярний сервіс для комунікації користувачів, це також надзвичайно корисна та універсальна платформа для створення чат-ботів. Дана платформа містить багато допоміжних фреймворків, що можуть значно допомогти при створенні та автоматизації ботів. BotFather це спеціальний бот на платформі Telegram, який дозволяє користувачам створювати та управляти ботом. За його допомогою можна встановлювати опис та головні команди бота, відізвати токен, видалити бота, установити його фотографію та тощо.

Для створення автоматизованої системи комунікації на базі телеграм-бота, було використано мову програмування Python через простоту синтаксису, швидкість обробки, наявність великої кількості бібліотек для роботи з чатботами, читабельність коду та його популярність загалом у використанні [6; 7], що можна побачити на рисунку 3.1 за загальним рейтингом мов програмування.

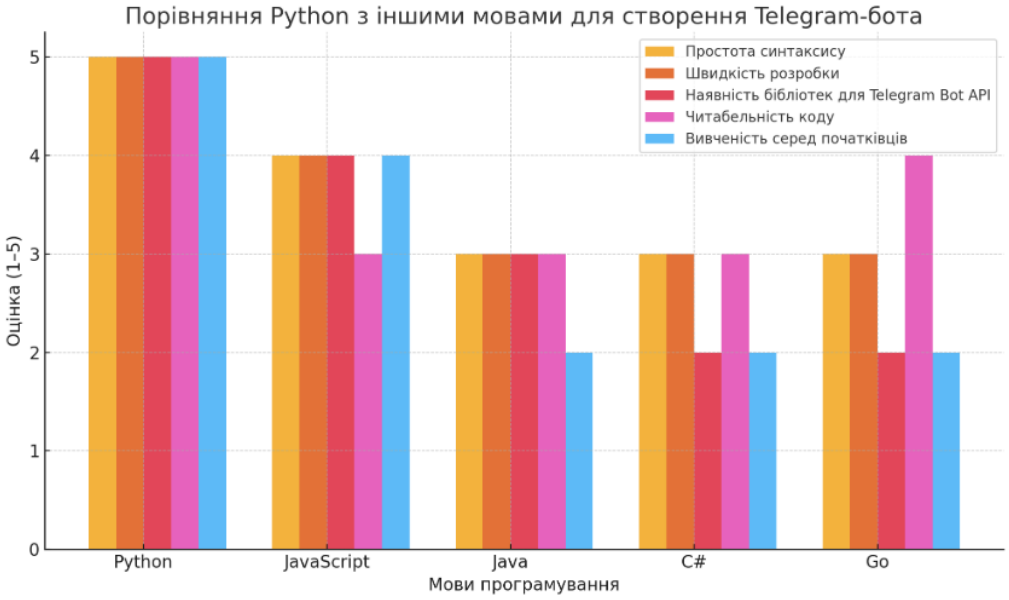


Рисунок 3.1 — Рейтинг мов програмування на базі даних від StackOverflow

Для порівняння, JavaScript часто використовують для розробки веб-додатків, проте в нього обмежена робота з кінцевими становими машинами. Java і C# є доволі потужними середовищами, проте через складність синтаксису та громіздкість вони рідше використовуються для підтримки телеграм. Новітня мова програмування Go є швидкою та функціональною, проте має значно менше підтримки Telegram-ботів та явно складніший синтаксис.

До найпопулярніших бібліотек для створення ботів відносять aiogram [14], Python-telegram-bot та TeleBot. Використання бібліотек допомагає значно полегшити створення та інтеграцію бота як готової системи. Для створення бота для випускників було використано бібліотеку aiogram. Дана бібліотека є асинхронною та створена для роботи саме з Телеграм, вона дозволяє реалізувати ботів, що можуть працювати паралельно з декількома користувачами, не очікуючи відповіді від кожного із них, це надає ботам більше автоматизації та можливості створити більш незалежний діалог із користувачем.

Одночасно, бібліотека Python-telegram-bot пропонує більш традиційний підхід до розробки, що заснований вже на синхронній взаємодії в стилі «питання-відповідь». Така бібліотека використовується швидше для створення асистентів, де кожна дія користувача потребує негайної відповіді від бота.

TeleBot підтримує різні стратегії для взаємодії з користувачем, що робить дану бібліотеку корисним інструментом для широкого спектру застосунків. Вона містить переваги асинхронної та синхронної розробки, завдяки чому вона дає гнучкість у розробці, для створення як простих так і складних ботів [7].

3.2 Середовище розробки Visual Studio Code

Для розробки системи комунікації для випускників було обране середовище Visual Studio Code. Проте це був не єдиний варіант для розробки. На вибір середовища впливали декілька факторів: тоді як VS Code є абсолютно безкоштовним, PyCharm має безкоштовну версію, проте в такому випадку має

значно менший функціонал; Студія має простий користувацький інтерфейс та може бути адаптована з різними доступними темами, що робить його більш привабливим для розробників. PyCharm має меншу екосистему, проте в той самий час має більше корисного функціоналу в платній версії програми; обидва ресурси добре інтегрується з GitHub, що значно полегшує роботу із системою керування версіями. Він має глибоку інтеграцію із Flask, Django та зами даних, а також вбудовані тести та інспекцію коду.

Visual Studio Code це редактор вихідного коду, він містить 15 різних мов інтерфейсу для його налаштування та підтримує великий ряд мов програмування, підсвітку синтаксису у випадку виникнення помилок, рефакторинг, відладку, навігацію та підтримку Git. Дане середовище може бути встановлене на Linux, Windows, macOS, що робить його функціональним для багатьох користувачів. Дане IDE засноване на Electron, що дозволяє використовувати такий самий компонент редактора, що є в Azure DevOps, і реалізується через веб-редактор, що був розроблений спеціально під Visual Studio Online. Даний редактор дуже легко інтегрувати із віртуальними середовищами, що значно допомогло в автоматизації системи комунікації. Використовуючи кілька розширень FTP , користувачі можуть синхронізувати код між сервером і редактором без необхідності завантажувати додаткове ПЗ [13].

Окрім легкості та швидкості запуску, Visual Studio Code має доповнення коду IntelliSense для змінних, методів та імпортованих модулів, графічне налагодження, лінтування, графічне налагодження, редагування кількома курсорами та потужні функції редагування; швидка навігація та рефакторинг коду. Більшість даних функцій було імпортовано саме з технологій Visual Studio.

3.3 Використання SQLite

SQLite — це вбудована бібліотека, яка реалізує безсерверний механізм бази даних SQL без конфігурації. Код для SQLite знаходиться у відкритому доступі,

тому його можна безкоштовно використовувати для будь-яких цілей [2; 8; 18]. На відміну від більшості інших баз даних SQL, SQLite не має окремого серверного процесу. SQLite читає та записує безпосередньо у звичайні дискові файли. Повна база даних із кількома таблицями, індексами, тригерами та представленнями даних міститься в одному дисковому файлі.

Це дуже компактна бібліотека, враховуючи весь її функціонал вона займатиме всього 750кБ пам’яті, але чим більше пам’яті виділено тим швидше вона працюватиме.

Використання саме даної СУБД було обумовлено декількома основними причинами. До них можна віднести:

- SQLite — це програмне забезпечення з відкритим кодом, що не потребує ліцензії після встановлення;
- безперервність, оскільки для роботи не потрібен інший серверний процес або система;
- він полегшує роботу з кількома базами даних під час одного сеансу одночасно, що робить його дуже гнучким;
- кросплатформеність, тобто, може працювати на всіх операційних системах включаючи macOS, Linux, Windows;
- не потребує конфігурації, налаштування та адміністрування.

Окрім цього, SQLite використовується для розробки вбудованого програмного забезпечення для таких пристроїв, як телевізори, мобільні телефони, камери тощо. Він може керувати HTTP-запитами з низьким і середнім трафіком, а також перетворювати файли на архіви меншого розміру з меншою кількістю метаданих [2].

3.4 Хостинг та запуск VPS

Для забезпечення цілодобової роботи бота, готову систему було розгорнуто на віртуальному приватному сервері (VPS). Це дозволило отримати повний

контроль над середовищем виконання, автоматизувати його запуск, забезпечити гнучке масштабування та незалежність від сторонніх хостингових обмежень. На відміну від безкоштовних рішень, наприклад PythonEverywhere, використання такого серверу не має обмежень на запити, зберігання файлів чи тривалість сесії, дозволяє використати власну базу даних та зберігати дані від файлів-обробників, забезпечує автозапуск бота навіть після перевантаження та має фіксовану адресу, яку можна використати для Webhook та API. На рисунку 3.2 можна побачити принцип його життєвого циклу.

Виділений хостинг дозволяє приміняти відповідні ліцензії на ПЗ від Microsoft та Oracle. Виділений хост від Amazon EC2 інтегрований із AWS License Manager, що є сервісом, який допомагає керувати ліцензіями на програмне забезпечення включаючи Microsoft Windows Server та Microsoft SQL Server.

У цьому проекті використовується Polling, а не Webhook, оскільки це простіше для налаштування і не потребує SSl-сертифікату. Polling — це регулярне звернення до Telegram-серверів для перевірки нових повідомлень.

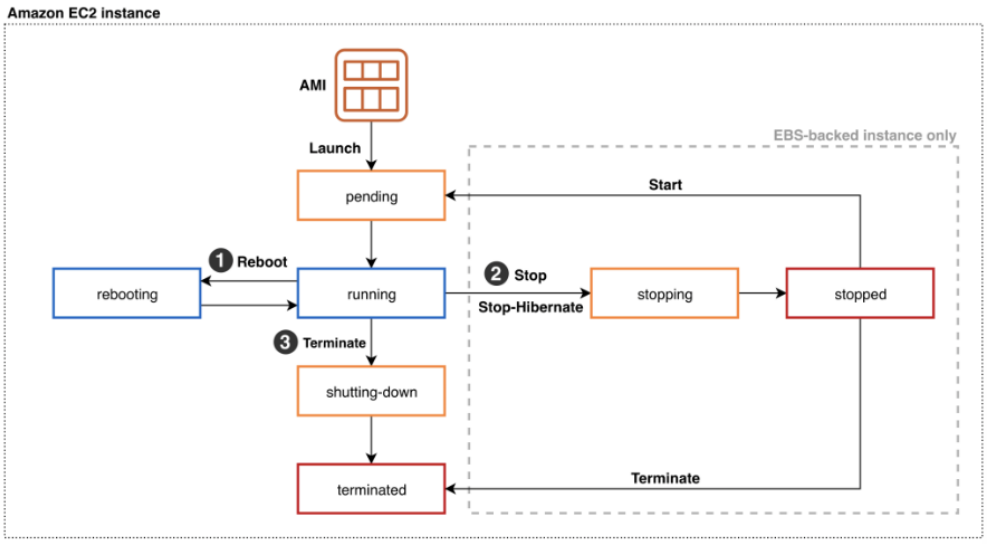


Рисунок 3.2 — Принцип життєвого циклу EC2

Для конфігурації було використано операційну систему Ubuntu Server 22.04 LTS. Хостинг провайдером став Amazon EC2 через його безкоштовний тариф та велику кількість можливостей та його надійність.

3.5 GitHub

Під час розробки даного бота для автоматизованої системи комунікації було використано систему контролю версій Git та онлайн-репозиторій GitHub для проміжних етапів в розробці програмного продукту.

GitHub — це веб-інтерфейс, який використовує Git, програмне забезпечення для керування версіями з відкритим кодом, яке дозволяє кільком людям одночасно вносити окремі зміни до веб-сторінок.

GitHub дозволяє декільком розробникам працювати над одним проектом одночасно, зменшує ризик повторюваної або конфліктної роботи та може допомогти скоротити час виробництва. За допомогою GitHub розробники можуть одночасно створювати код, відстежувати зміни та інноваційні рішення проблем, які можуть виникнути під час процесу розробки застосунку або сайту.

Git — це система контролю версій, яку використовує GitHub. Git має відкритий вихідний код і його можна безкоштовно використовувати для малих і великих проектів. Це система, яка відстежує кожну зміну, яку ви вносите в GitHub. Тим часом GitHub — це місце, де люди можуть ділитися файлами, які вони створили, і співпрацювати з ними, що також дуже допомагає при реалізації великих групових проєктів. GitHub дозволяє використовувати Git без необхідності попередньо вивчати коди команд [1; 8].

В межах цього проєкту він був використаний для:

- зберігання коду Telegram-бота;
- керування історією розробки;
- швидкого доступу до проекту з будь-якого пристрою;
- документації у вигляді README.md;
- резервного копіювання;
- відстежування змін та контрою версій.

3.6 Формати збереження: JSON та CSV

Окрім основної бази даних, при розробці даного проекту також використовевалися формати JSON та CSV для збереження, передачі та резервного копіювання інформації.

JSON (JavaScript Object Notation) — це легкий формат для зберігання та транспортування даних, який часто використовується, коли дані надсилаються із сервера на веб-сторінку. Він «самоописується» і його легко зрозуміти не тільки розробникам з досвідом, але і програмістам, що лише починають свій шлях. Незважаючи на те, що формат JSON походить від структури об’єктів JavaScript, він є лише текстовим і може використовуватися незалежно від JavaScript. Основними причинами використання даного формату є простота використання та отримання даних від серевера, краща схема підтримки, можливість самоописуватись та швидкість його роботи. Формат файлу JSON може легко обробляти неструктуровані складні дані, що неможливо з форматом CSV. Крім того, ієрархічні дані можна легко представити за допомогою JSON, на відміну від формату CSV, тобто він є більш універсальним [6, 10].

Файли CSV («файли зі значеннями, розділеними комами» або «файли, розділені комами») є одним із найпоширеніших форматів програм для роботи з електронними таблицями. Файли CSV — це звичайні текстові файли, у яких коми використовуються для розділення даних. Оскільки вони забезпечують простий і стандартизований формат, який можна легко читати та обробляти більшістю програмних програм, вони широко використовуються в багатьох галузях для обміну даними [11].

Оскільки файли CSV розроблені як відносно прості, їх часто використовують у багатьох галузях для передачі складних даних між програмами. За допомогою таких файлів можна експортувати складні дані з однієї програми у формат CSV, а потім імпортувати експортовані дані в іншу програму, де їх можна використовувати [12].

Ще одним із найпоширеніших випадків використання файлу CSV є завантаження списку контактів у програму. Це можна використовувати для надсилання маркетингових кампаній через платформу доставки електронної пошти, як-от SendGrid, або для синхронізації інформації про клієнтів у CRM, як-от Salesforce.

В даній програмі файли використовувалися для збереження кодів країн, список зареєстрованих на події, самі події та новини. Це дозволило краще розуміти процеси роботи бота та перевіряти точність його роботи, а також зменшити кількість помилок та покращити структуру коду.

3.7 Використання FSM

Finite State Machine (FSM) або ще скінчений автомат — це обчислювальна модель, яка використовується для опису поведінки системи шляхом поділу її на скінченну кількість станів разом із переходами між цими станами на основі певних умов або подій [14].

За своєю суттю FSM складається із станів, переходів та вхідних даних. Стани представляють різні умови або режими, в яких може перебувати система в будь-який момент часу. Кожен стан інкапсулює певний набір поведінки або дій, які система може виконувати в цьому стані. Переходи визначають умови або події, які викликають зміну стану. Коли відбувається перехід, система переходить з одного стану в інший, потенційно змінюючи свою поведінку або внутрішню конфігурацію. Вхідні дані/події — це стимули або сигнали, які керують переходами між станами. Вхідні дані можуть надходити з різних джерел, наприклад від взаємодії користувача, показань датчиків або зовнішніх подій [5; 12; 20].

В Aiogram 3 інтеграція автоматів надає розробникам можливість легко керувати складним потоком введених користувачами даних, як зображено на рисунку 3.3.

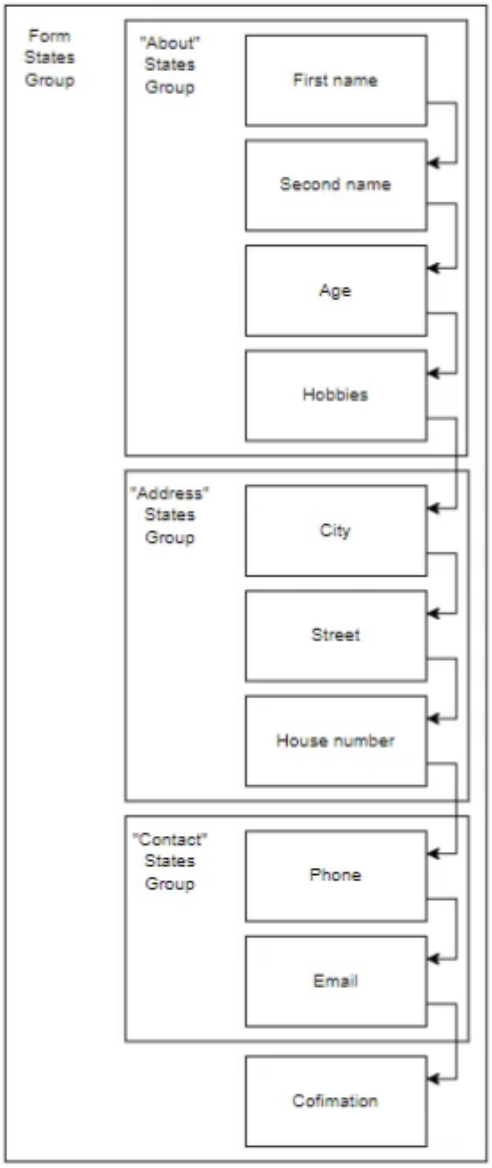


Рисунок 3.3 — Приклад потоку даних FSM

Проводячи користувачів через послідовність визначених станів і обробляючи кожен стан окремо, FSM дають змогу розробникам створювати ботів Telegram, які якісно працюють і переміщуються в складних сценаріях взаємодії, що забезпечує безперебійну розмову для користувачів [19]. В Aiogram FSM представлені як StateGroups, які служать основою для організації поведінки бота в окремі стани та керування переходами між ними [5; 14].

Висновки до розділу 3

В даному розділі було розглянуто основні засоби та інструменти, використані для створення Telegram-бота автоматизованої системи комунікації між випускниками. Інтеграція мови програмування Python, бібліотеки Aiogram 3.x, SQLite3 для зберігання даних, а також JSON- і CSV-файлів для взаємодії з подіями та чатами дозволила забезпечити ефективну, гнучку та надійну архітектуру системи. Застосування асинхронного програмування та FSM-моделі забезпечило зручну реалізацію складних сценаріїв взаємодії з користувачем. Застосовані засоби сприяли побудові функціонального, стабільного та масштабованого програмного рішення для вирішення поставлених завдань.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Програмна реалізація Telegram-бота для автоматизованої системи зв’язку між випускниками побудована за модульною структурою, що зображена на рисунку 4.1, що дозволяє легко розширювати функціонал, підтримувати код та розмежовувати логіку за функціональними напрямками. В основі проекту лежить бібліотека aiogram, а також FSM (скінченна автоматична модель) для покрокової взаємодії з користувачем.

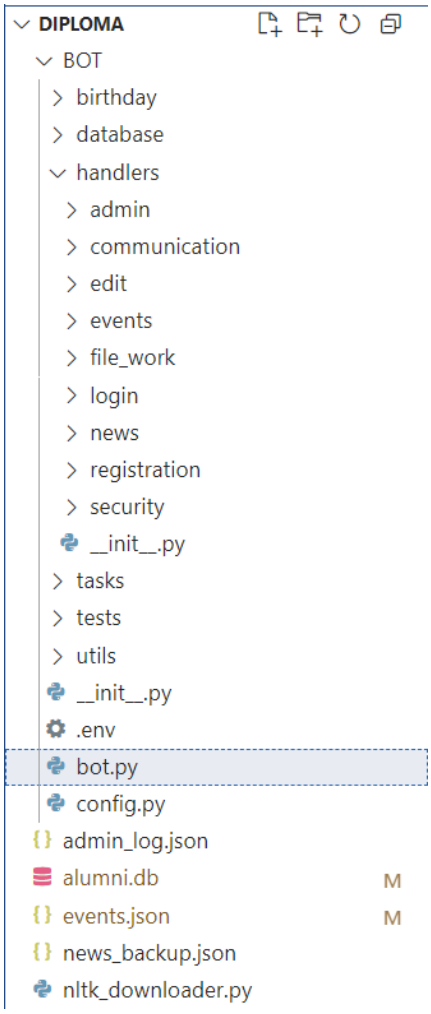


Рисунок 4.1 — Модулі системи

При проектуванні було дотримано наступних принципів: модульність, масштабованість, використання станів для складних сценаріїв та поділ логіки за ролями, що допомогло покращити читабельність та зрозумілість коду.

До основних модулів програми можна віднести:

- main.py — точка входу: ініціалізує бота, диспетчер, підключає маршрути та запускає FSM;
- config.py — містить токен, шлях до БД, інші змінні середовища;
- папка database/ — містить логіку ініціалізації юази даних, SQL-запити, а також заповнення початкових даних при ініціації програми(default.py);
- папка handlers/ — обробники користувацьких дій: реєстрація, вхід, редагування, адміністративна панель;
- папки news/, events/, communication/, file_work/ — модулі, що реалізують відповідний функціонал у роздільному вигляді для користувача та адміністратора;
- utils/ — допоміжні функції: побудова клавіатур, нотифікації, перевірка телефону;
- tasks/reminders.py — фонові задачі (нагадування про події);
- keyboard.py — файл, що містить всі функціональні кнопки програми.

Головний файл запуску ініціалізує базу даних, бота, диспетчер, маршрутизатори, фонові задачі та логування.

4.1 Реєстрація користувача

Реєстрація випускника в даному боті є одним з найголовніших етапів, адже саме на цьому кроці кафедра матиме змогу отримати логістичні дані та мати можливість допомогти у спілкуванні між студентами. Це покроковий процес збору персональних даних користувача, що реалізований за допомогою машини станів із бібліотеки aiogram, де кожен наступний крок активується лише після правильного виконання попереднього. Такий підхід дозволяє керувати контекстом діалогу із кожним користувачем окремо.

Функціонал реєстрації користувача реалізовано в модулі handlers/registration/register_user.py на основі машини станів **(**FSMContext**)**

бібліотеки aiogram (рисунок 4.2), яка дозволяє поетапно обробляти дані від користувача.

```python
# Стан машини для реєстрації
class Registration(StatesGroup):
    full_name = State()
    phone_number = State()
    old_phone_number_check = State()
    old_phone_number = State()
    enrollment_year = State()
    graduation_year = State()
    department_id = State()
    specialty_input = State()
    specialty_select = State()
    group_name = State()
    birth_date = State()
    ask_survey = State()
```

Рисунок 4.2 — Машина станів для Реєстрації користувача

Усі кроки збережено в класі UserRegistrationState, де кожен стан відповідає конкретному полю профілю. Кожен стан активує відповідну функцію-хендлер, яка отримує повідомлення, перевіряє його, зберігає в контекст і перемикає FSM на наступний крок.

Реалізована логіка включає перевірку дублікатів, обробку номера телефона, уточнення зміни номеру:

Реалізована логіка включає перевірку дублікатів, обробку номера телефона, уточнення зміни номеру, валідацію кожного введеного поля та автоматичний запис у базу. До переваг обраного підходу розробки можна віднести розділення кожного кроку на окрему функцію, спільні частини логіки для реєстрації користувача та адміністратора винесено в окремий файл, що створює уніфікованість, є можливість змінювати порядок полів та додавати нові етапи без порушення роботи інших модулів та усі поля з FSM відповідають відповідним полям у таблиці баз даних.

4.2 Реєстрація та авторизація адміністратора

Реєстрація та вхід адміністратора реалізовані як окремий сценарій з розділенням доступу: адміністратор спочатку надсилає запит на реєстрацію, який

повинен бути підтверджений супер-адміністратором, після чого адміністратор отримує доступ до функціоналу згідно з рівнем доступу. Варто зауважити, що адміністратор, обмежений у правах доступу, може працювати лише з даними користувачів. Уся логіка реалізована в модулях register_admin.py, login_admin.py та admin_panel.py.

Механізм реєстрації адміністратора реалізований за тією ж схемою, що й реєстрація користувача — за допомогою машини станів (FSMContext), з повторним використанням загальних методів, винесених у register_start.py.

Адміністратор вводить ПІБ, номер телефону та пароль входу. Після введення даних, бот формує запис у таблиці admin_requests, який чекає підтвердження від супер-адміністратора. Це дозволяє уникнути несанкціонованого створення адміністраторських обліковок. Такий функціонал допомагає знизити кількість можливих помилок та розмежування доступу. Заявки автоматично виводяться супер-адміністратору у зручному вигляді.

Контроль доступу реалізовано через перевірку access_level при натисканні адміністративних кнопок. Функція get_admin_access_level() викликається перед виконанням дії (рисунок 4.3).

```python
def get_admin_access_level(telegram_id: str) -> str | None:
    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT access_level FROM admins WHERE telegram_id = ?", (telegram_id,))
    result = cursor.fetchone()
    conn.close()
    return result[0] if result else None
```

Рисунок 4.3 — Реалізація методу get_admin_access_level

Для входу адміністратора встановлені обмеження на кількість спроб у разі введення неправильного паролю для забезпечення вищого рівня безпеки та надійності застосунку. Паролі зберігаються у відкритому вигляді для спрощення верифікації, у майбутньому є сенс додати хешування.

4.3 Адміністративна панель

Після авторизації адміністратор (звичайний або супер-адміністратор) отримує доступ до адміністративної панелі Telegram-бота**.** Усі елементи реалізовано у вигляді інтерактивного меню з кнопками (inline-клавіатура) з урахуванням рівня доступу. Панель організована в модулі handlers/admin/admin_panel.py, де зберігається основна логіка маршрутизації дій адміністраторів.

Для випадку з адміністратором, що має обмежений функціонал створена валідація, що опирається на його тип доступу та можливі функції. admin_limited має доступ до перегляду користувачів, редагування профілю та блокування користувачів, в той час як admin_super виконує модерацію, публікацію та редагування новин та подій, створення чатів за спеціальністю та підтвердження заявок. Це реалізується шляхом перевірки значення поля access_level у таблиці admins.

Більшість складних дій реалізовано через FSM, що дозволяє обробляти дії поетапно та зберігати контекст між повідомленнями. Всі дії адміністратора перевіряються по telegram_id, що є унікальним значенням, що присвоєне самим сервером Telegram API. Кожна функція перед виконання перевіряє чи є достатній рівень доступу за допомогою встановлення обмежень на методи через @super_admin_only.

4.4 Новини

Новини реалізовано повністю через роботу з становою машиною, що дозволяє обробити багатокроковий сценарій, кожна дія обробляться в окремих функціях згідно SRP (Siingle Responsibility). Користувачі працюють лише з інтерфейсною частиною, тоді як адміністратори мають доступ до CRUD-можливостей.

Функціонал роботи з новинами реалізовано в окремому модулі news/, що містить два компоненти: news_admin.py — функції для створення, редагування та видалення новин адміністраторами та news_user.py — функції для перегляду новин користувачами.

Всі новини зберігаються у таблиці news в базі даних SQLite, що дозволяє швидко їх фільтрувати, відображати або оновлювати. Кожна новина містить її унікальний номер, короткий та повний опис, дату у форматі ISO та посилання.

Додавання новини відбувається через FSM-послідовність: введення короткого опису, повного опису, посилання та підтвердження збереження.

Адміністратор може переглянути список новин з кнопками для редагування або видалення. При редагуванні новина відкривається як FSM-послідовність, де можна змінити опис, дату та посилання.

Для генерації короткого опису новини було використано модель NLP, а саме d0p3/03ap-sm (рисунок 4.4) [15]. Це модель створена українськими розробниками, що допомагає у створенні заголовків на базі отриманого повного опису новини, що була тренована на невеликій архітектурі T5. Використаним датасетом є Ukrainian Corpus CCMatrix. Як і кожної моделі в неї є обмеження, наприклад, помилкова генерація заголовка.

```python
# Ініціалізація моделі для генерації короткого опису
tokenizer = AutoTokenizer.from_pretrained("d0p3/O3ap-sm")
model = AutoModelForSeq2SeqLM.from_pretrained("d0p3/O3ap-sm")

JSON_FILE = "news_backup.json"
DB_NAME = "alumni.db"

def generate_short_title(full_desc: str, max_length: int = 25) -> str:
    input_ids = tokenizer(full_desc, return_tensors="pt", truncation=True, max_length=512).input_ids
    output_ids = model.generate(input_ids, max_length=max_length, num_beams=4, early_stopping=True)
    return tokenizer.decode(output_ids[0], skip_special_tokens=True)
```

Рисунок 4.4 — Ініціалізація моделі та функція генерації

Для використання даного інтерпритатора треба підключити бібліотеку transformers. Для генерації було використано метод generate_short_title(). При кожному етапі додавання або редагування новини, система може зберігати новини

також у news.json, а при запуску зчитувати наявні новини та синхронізувати їх з базою даних.

Кожен із методів має свої обробки на помилки, щоб програма працювала відповідно до завдання та мала підказки для користувачів у випадку виникнення збоїв або помилок вводу.

4.5 Події та системи їх реєстрації

Функціонал керування подіями — один із ключових модулів Telegram-бота. Він забезпечує: створення, редагування та видалення подій адміністраторами (модуль events_admin.py); перегляд подій і реєстрацію на них користувачами (модуль events_user.py); резервне збереження у events.json та registrations.json; підтримку синхронізації між JSON та SQLite.

Супер-адміністратор може ініціювати створення події за допомогою машини станів, що містить працює за наступною послідовністю: назва події, опис, дата та час та кількість місць. Для кількості місць відбувається перевірка, щоб вона не була порожньою та мала хоча б 3 символи, опис події мінімум 20 символів, дата та час очікується у форматі ДД.ММ.РРРР ГГ:ХХ та валідується через datetime.strptime() та порівнюється із datetime.now(). Для кількості місць перевіряється, щоб воно було числом та кількість місць була невід’ємним числом. Після отримання цих даних бот рахує кількість доступних місць, надсилає запит до бази даних та викликає метод sync_db_to_json() для оновлення events.json.

Кожну із подій можна також редагувати. Дане редагування працює за тим самим принципом, що і редагування профілю користувача та адміністратора.

Для кожного користувача при реєстрації на подію перевіряється чи є вже такий запис для цього користувача та чи залишилися доступні місця, у разі успіху додається запис у registrations, оновлюється available_seats -=1 у events, та оновлюється registrations.json.

Модуль подій реалізовано як двосторонній канал, де адміністратор управляє даними, а користувач взаємодіє лише через кнопки. Уся логіка побудована на асинхронній обробці, сценаріях та перевірках, а структура таблиць підтримує масштабованість, облік місць та резервування.

4.6 Реалізація комунікації

Комунікаційний модуль забезпечує розподілення користувачів за спільними ознаками (група, спеціальність, рік вступу або випуску) та надає доступ до відповідних Telegram-чатів. Усі дії поділено між адміністраторами (створення) та користувачами (перегляд). Реалізація охоплює два окремі модулі: communication_admin.py — створення чатів супер-адміністратором та communication_user.py — отримання доступу до чатів користувачами.

Чати зберігаються одночасно у CSV-файлі (chat_links.csv) та у двох таблицях SQLite: communication_chats, та chats**.** Таблиця chats використовується для перевірки унікальності комбінацій, щоб не допустити дублювання чатів. Дані з неї синхронізуються у CSV для зручної модифікації вручну.

FSM-сценарій ChatCreationState реалізує 3 етапи: вибір типу чату (група / спеціальність / рік вступу / рік випуску), введення значення (наприклад, "ІПЗ-21" або "2021") та введення посилання на Telegram-групу.

FSM автоматично викликається при натисканні кнопки "Створити чат", а callback_data відповідає за вибір параметра (create_year_chat, create_specialty_chat тощо).

На кожному з етапів є валідація на тип чату, значення, посилання та унікальність вводу.

Після успішного створення запис додається до communication_chats (для історії), запис додається до chats (для перевірки дублювання) та файл chat_links.csv оновлюється відповідною інформацією.

CSV-файл оновлюється автоматично після кожного створення чату, використовується при візуальному перегляді або експорту (наприклад, у Excel), а при зчитується через csv.DictReader.

Модуль комунікації реалізований через FSM (створення), callback-кнопки (перегляд), подвійне збереження (SQLite та CSV), унікальні перевірки та контекстний підбір чатів на основі профілю користувача. Це забезпечує централізовану, але гнучку систему зв'язку між випускниками.

4.7 Обмін файлами та нагадування

Модуль обміну медіа забезпечує користувачам можливість надсилати фото, відео та документи, які зберігаються в базі даних і надалі доступні для перегляду адміністраторами. Реалізація поділена на два файли: user_upload.py — обробка медіа, які надсилає користувач; files_admin.py — перегляд файлів у вигляді списку з можливістю фільтрації.

Функціонал активується через callback-кнопки: «Надіслати фото», «Надіслати відео» та «Надіслати документ».

Після натискання бот переходить у відповідний режим і чекає на тип повідомлення. Потім виконується збереження: запис у таблицю user_files, підтвердження користувачу про успішне завантаження та питання, чи бажає він надіслати ще один файл (через InlineKeyboardMarkup). На кожному етапі є валідація для типу контенту, унікальність та реєстрація.

Адміністратор відкриває меню «Медіа» або «Файли користувачів». Йому доступні фільтри: за типом файлу (фото, відео, документ), за користувачем (введення номера або Telegram ID) та пагінація — по 5 або 10 елементів за сторінку.

Кожен елемент виводиться через InputMediaPhoto, InputMediaVideo або InputMediaDocument (залежно від file_type). Повідомлення містить: медіа, опис (caption) та дату.

Telegram API дозволяє повторно надсилати файли за file_id, тому не зберігається фізичний файл. Обмеження на обсяг з боку Telegram не контролюються ботом, але враховуються в UX (наприклад, великі відео не підтримуються). Асинхронна обробка, інтеграція з базою та callback-керовані кнопки роблять систему зручною як для користувача, так і для адміністратора.

У Telegram-боті реалізовано механізм фонових автоматичних задач, який відповідає за надсилання нагадувань про майбутні події та персональних повідомлень, зокрема привітань з днем народження.

Уся логіка зосереджена у модулі tasks/reminders.py, який запускається разом з main.py і працює на основі асинхронного планувальника.

Бот використовує асинхронний цикл з періодичною перевіркою умов: чи є події, які починаються в найближчі 24 години та чи є користувачі, у яких сьогодні день народження. Це реалізується через нескінченний цикл із asyncio.sleep(...), запущений паралельно з основною обробкою aiogram.

Функція check_upcoming_events(bot) виконує SQL-запит до таблиці events, щоб знайти події, які відбудуться через 12–24 годин, для кожної події отримує список зареєстрованих користувачів з таблиці registrations та надсилає кожному повідомлення. Нагадування також можна додавати у свій календар за допомогою файлу з розширенням .ics.

Висновок до розділу 4

У розділі детально описано код розробки бота згідно основним принципам. Розроблено роздільні сценарії для користувача та адміністратора. Реєстрація включає введення ПІБ, номеру телефону, року вступу та випуску, кафедри, спеціальності, групи, а також уточнення старого номера у разі зміни. Адміністратор має можливість керувати користувачами, блокувати їх, змінювати рівень доступу, публікувати або видаляти новини, створювати чати для спілкування та події для подій — збереження максимальної кількості місць і списку зареєстрованих.

5 РОБОТА КОРИСТУВАЧА ІЗ СИСТЕМОЮ

5.1 Реєстрація та редагування

Реєстрація користувача — перший і обов'язковий етап роботи з Telegram-ботом. Без неї випускник не має доступу до функціоналу: перегляду новин, подій, чатів, обміну файлами тощо. Весь процес проходить у вигляді діалогу між ботом і користувачем, з покроковим введенням даних. Після натискання кнопки «Реєстрація» або запуску /start, бот привітно звертається до користувача та пояснює, що необхідно пройти реєстрацію (рисунок 5.1). Усі наступні кроки супроводжуються короткими інструкціями й прикладами введення:

1. Введення ПІБ, де користувач вводить своє повне ім'я (наприклад, *Іван Петренко*) і бот перевіряє, щоб ім'я складалося з 2 або більше слів.
2. Номер телефону, де користувач вводить номер вручну або через кнопку «Поділитися контактом», підтримуються формати для країн, що зазначені у файлі phone_rules.csv і якщо бот виявляє, що номер було змінено після випуску — запитує старий номер для ідентифікації.
3. Рік вступу та рік випуску, де бот просить ввести роки у форматі, наприклад: *2020, 2024*. Ці значення використовуються для пошуку чатів та подій за роком навчання.
4. Вибір кафедри, де бот надсилає кнопки зі списком доступних кафедр (дані з бази), користувач обирає одну зі списку (наприклад, *Теплоенергетичний факультет*).
5. Вибір спеціальності, де залежно від кафедри, бот пропонує список спеціальностей, підтримується також ручне введення, якщо потрібно.
6. Назва групи, де користувач вводить назву групи вручну (наприклад, *ТВ-21*) і цей параметр використовується для формування групових чатів.
7. Дата народження має очікуваний формат: ДД.ММ.РРРР і ці дані використовуються для персоналізованих привітань.

Всі дані проходять їх перевірку та у випадку неправильного введення користувача буде проінформовано.

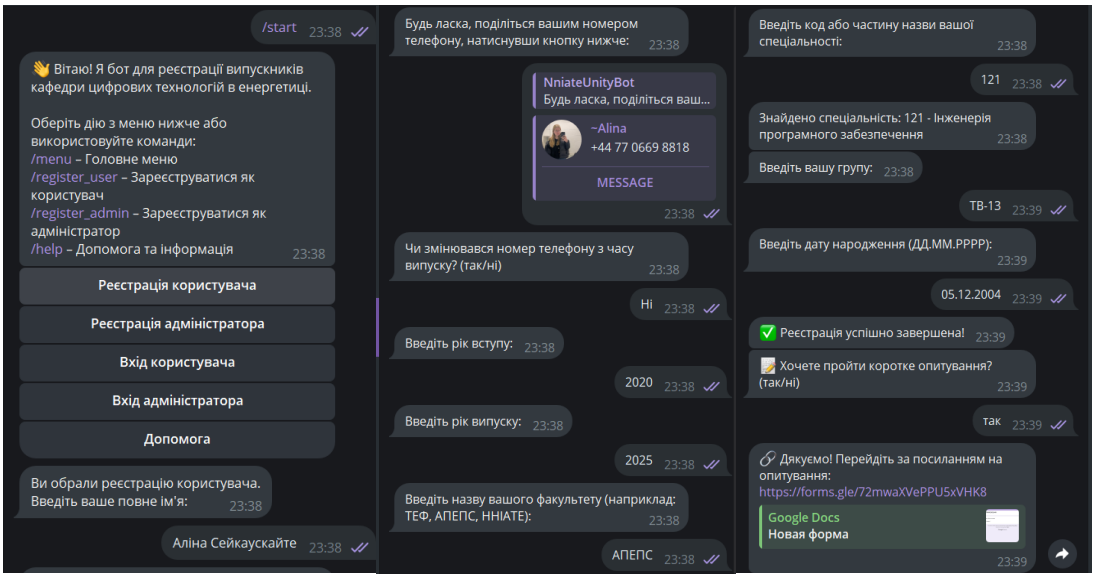


Рисунок 5.1 — Реєстрація користувача

Після введення всіх даних, користувач має змогу пройти професійне опитування у вигляді GoogleForm. Після підтвердження інформація записується в базу, і користувач отримує доступ до головного меню.

Після проходження реєстрації користувач автоматично авторизується у системі. Telegram-бот зберігає Telegram ID кожного зареєстрованого користувача, тому при наступній взаємодії бот одразу впізнає його і надає доступ до всіх дозволених функцій.

Випускнику не потрібно вводити логін або пароль — Telegram ID є унікальним ключем, що ідентифікує користувача. При кожному зверненні до бота (наприклад, через команду /start або натискання кнопки) система перевіряє: чи Telegram ID є в таблиці users; якщо знайдено — користувач одразу отримує головне меню; якщо не знайдено — запускається процедура реєстрації з нуля. У випадку, якщо користувач вже був зареєстрований, бот не дозволить повторно пройти реєстрацію. Це захищає дані від перезапису та зберігає цілісність профілю.

Після успішної авторизації бот надає кнопкове меню (рисунок 5.2) з наступними пунктами: редагувати профіль, Новини, Події, Спілкування, Надіслати фото, Надіслати відео, Надіслати документ.

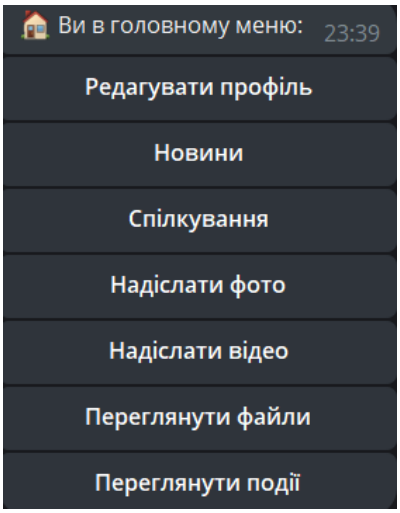


Рисунок 5.2 — Меню користувача

Меню відображається у вигляді Inline-клавіатури, зручно структурованої під екран користувача. При натисканні на кожну кнопку запускається відповідна логіка взаємодії.

Адміністратор після підтвердження супер-адміністратором має пройти авторизацію за паролем (рисунок 5.3).

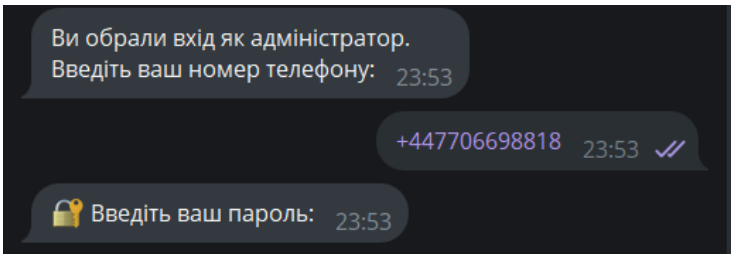


Рисунок 5.3 — Вхід адміністратора

Йому потрібно вибрати пункт «Увійти як адміністратор», ввести пароль та у разі успішного входу — отримує доступ до адміністративної панелі (рисунок 5.4).

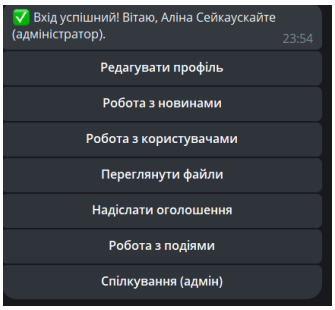


Рисунок 5.4 — Адміністративна панель

Адміністраторів розділено за рівнем доступу:

- Обмежений (admin_limited) — лише модерація користувачів;
- Повний (admin_super) — повне керування всіма розділами.

Система авторизації повністю автоматизована, базується на Telegram ID, і не вимагає зайвих дій від користувача. Це забезпечує швидкий і зручний доступ до всіх функцій одразу після реєстрації, а також гарантує безпеку через контроль ID, паролів (для адмінів) і ролей.

Кнопка «Редагування профілю» для користувачів дозволяє змінити ПІБ, номер телефону, спеціальність, групу, рік вступу та випуску, а також дату народження. Перед редагуванням користувачеві буде показано його введені дані (рисунок 5.5).

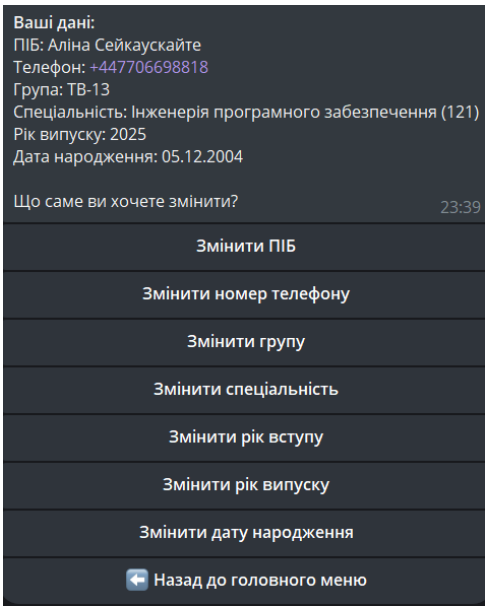


Рисунок 5.5 — Редагування профілю користувача

При редагуванні профілю, адміністратор може змінити своє ім’я, номер телефону та пароль (рисунок 5.6)

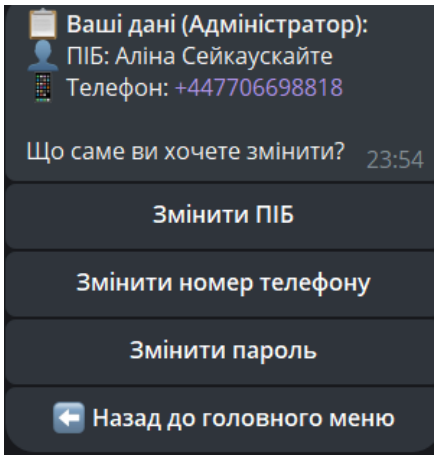


Рисунок 5.6 — Редагування профілю адміністратора

Щоб повернутися у до головного меню, користувачу необхідно натиснути кнопку «Назад до головного меню».

5.2 Файли

Функціонал обміну файлами забезпечує передачу **фото**, **відео** та **документів** між користувачами та системою з подальшим доступом до цих матеріалів адміністраторами. Кожен надісланий файл фіксується в базі даних із відповідним Telegram ID, типом, часом і коментарем.

Після авторизації або реєстрації користувач бачить у меню кнопки: «Надіслати фото» та «Надіслати відео».

Натискання однієї з кнопок запускає відповідний режим очікування повідомлення з потрібним типом контенту. Користувач надсилає файл відповідного типу: фото → message.photo, відео → message.video

Бот в цей час визначає тип контенту, отримує file_id, file_unique_id, та виконує запис у таблицю user_files. Після надсилання файлу, бот вимагає або завершення даного методу або продовжити надсилання, для цього він використовує вбудовані клавіші (рисунок 5.7).

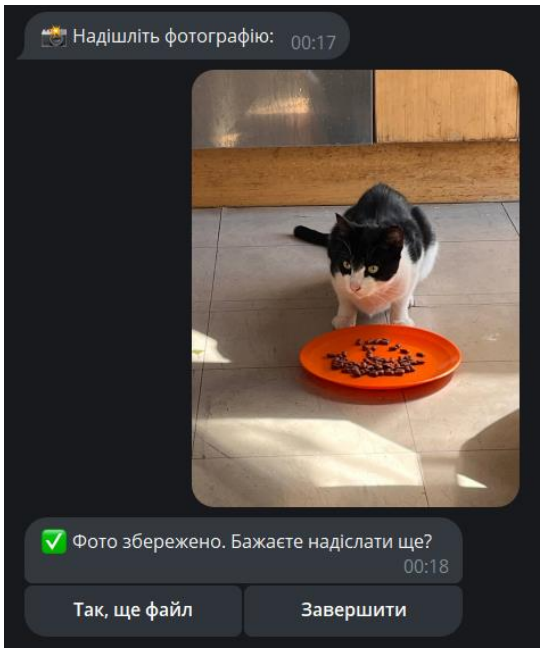


Рисунок 5.7 — Надсилання фото

В адмін-панелі є кнопка «Переглянути файли користувачів». Адміністратор отримує список записів з бази user_files (рисунок 5.8), кожен з яких містить: file_type — тип медіа, file_id — ідентифікатор Telegram, telegram_id користувача, дату надсилання (upload_time).

Рисунок 5.8 — Перегляд файлів адміністратора

Бот повторно надсилає файл за допомогою Telegram API без потреби у фізичному завантаженні: для фото → InputMediaPhoto(file_id), для відео → InputMediaVideo(file_id), для документа → InputMediaDocument(file_id).

5.3 Новини та події

Новини та події — ключові інструменти інформування та організації спільної активності випускників. Telegram-бот забезпечує інтерактивний інтерфейс перегляду новин і подій з боку користувача та розширені можливості створення, редагування і керування для адміністратора.

Всі дії реалізовані через зручні кнопки, діалоги, а також синхронізуються з базою даних і JSON-файлами для резервного збереження. Після натискання кнопки **«Новини»**, користувачеві треба обрати за який саме час він би хотів переглянути новини: «Новини за тиждень» або «Обрати дату». Для кожної новини виводиться короткий опис, дата публікації та кнопка «Детальніше». Користувач може обрати перегляд новин за конкретною датою або за останній тиждень (рисунок 5.9).

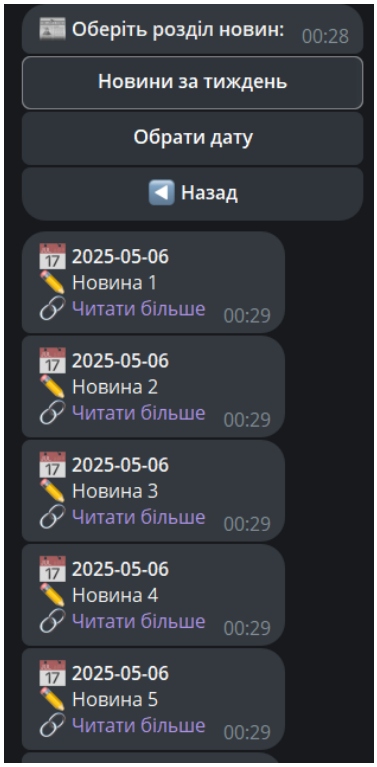


Рисунок 5.9 — Інтерфейс користувача для роботи з новинами

Для роботи з новинами треба здійснити вхід, як супер адміністратор. Через адмін-панель запускається процес додавання новини (через FSM), де послідовно вводяться: короткий опис, повний опис, дата публікації та посилання (рисунок 5.10).

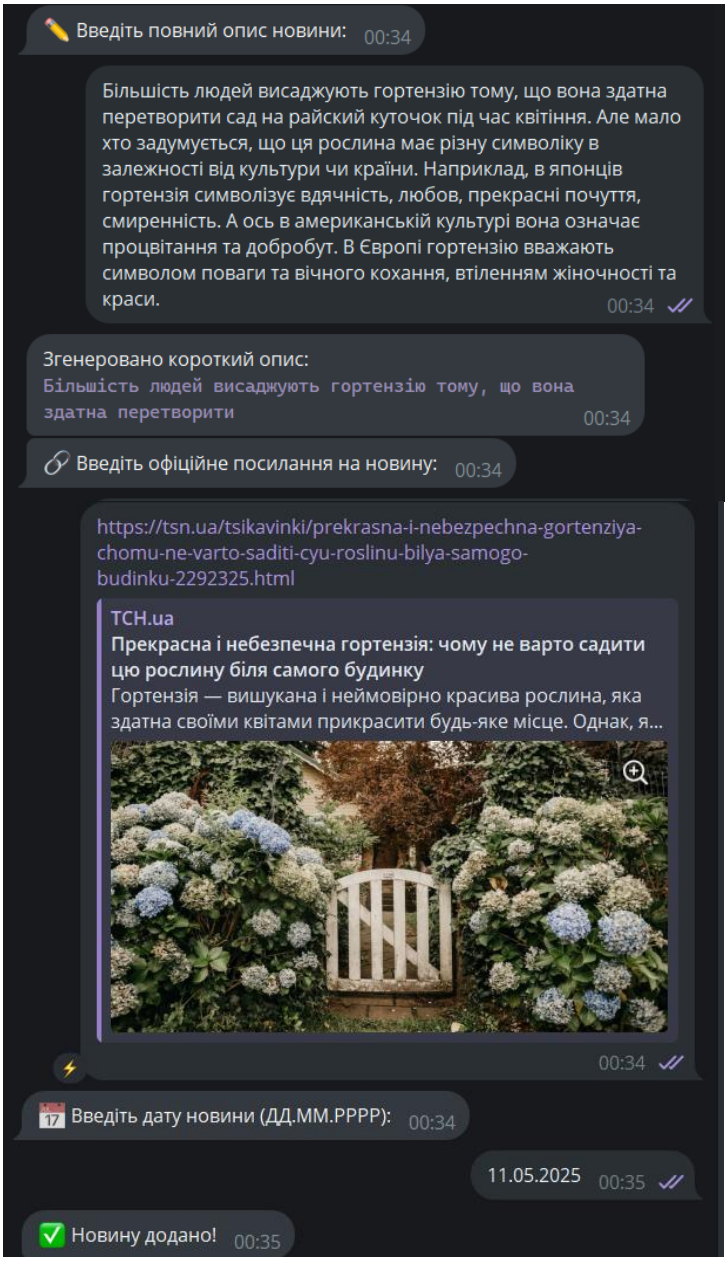


Рисунок 5.10 — Додавання новини

Адміністратор може не лише додавати, але й редагувати новини**.** Зі списку адміністратор обирає новину для редагування, а у випадку видалення новини, воно

підтверджується окремим повідомленням. Після кожної дії новини автоматично зберігаються в news.json.

У розділі «Події» користувач бачить перелік заходів, запланованих на майбутнє (рисунок 5.11).

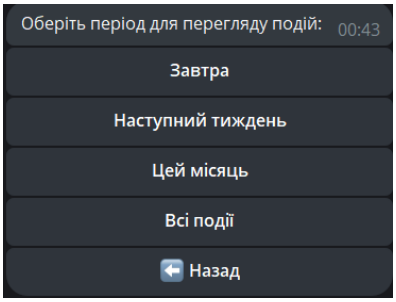


Рисунок 5.11 — Меню подій

Для кожної події вказано: назву, дату та час, короткий опис, кількість доступних місць (рисунок 5.12). При натисканні **«Зареєструватися»** в меню користувача бот перевіряє: чи ще є доступні місця та чи користувач уже зареєстрований. У разі успіху надсилається підтвердження реєстрації. За 12–24 години до початку події користувач отримує персональне нагадування з усіма деталями.

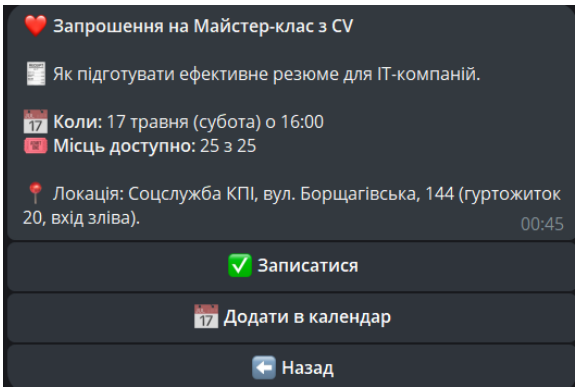


Рисунок 5.12 — Запрошення на подію

FSM-послідовність дозволяє поетапно ввести: назву події, опис, дату та час (із валідацією формату) та максимальну кількість місць. Подію можна змінити повністю або частково (назва, дата, місця тощо) або видаляти. Адміністратор може

переглядати список зареєстрованих користувачів на кожну подію: ПІБ, групу, телефон. Події синхронізуються з events.json, а реєстрації — з registrations.json.

Робота з новинами та подіями побудована так, щоб забезпечити простоту взаємодії для користувача та гнучке керування для адміністратора. FSM дозволяє зручно працювати з багатокроковими діями, а синхронізація з базою та JSON — забезпечує цілісність і відновлення даних у разі потреби.

5.4 Спілкування

Система спілкування в Telegram-боті реалізована через доступ до спеціально створених Telegram-чатів. Це дозволяє випускникам зберігати зв’язок у межах своєї групи, спеціальності, року вступу або року випуску.

Користувач отримує лише посилання, що релевантні саме для його профілю (рисунок 5.13), а супер-адміністратор має змогу створювати нові чати та зберігати їх у системі.

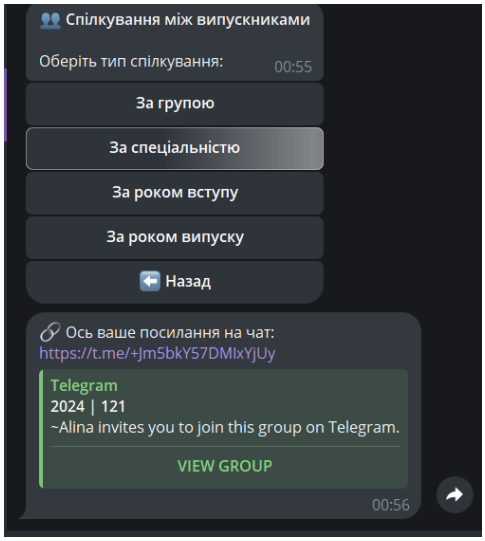


Рисунок 5.13 — Отримання посилань користувача

Після вибору одного з пунктів бот зчитує значення з профілю користувача (наприклад, група = ІПЗ-21), виконує SQL-запит до таблиці, якщо знайдено — бот

надсилає кнопку з посиланням, якщо не знайдено — бот повідомляє, що чат відсутній.

У панелі адміністратора доступний пункт «Керування чатами», який веде до підменю з опціями керування чатами. Створення чату реалізоване як покроковий процес (FSM): вибір типу чату (через кнопки), введення значення (наприклад, "ІПЗ-21" або "2020"), введення Telegram-посилання (https://t.me/назва_чату) та підтвердження збереження (рисунок 5.14).

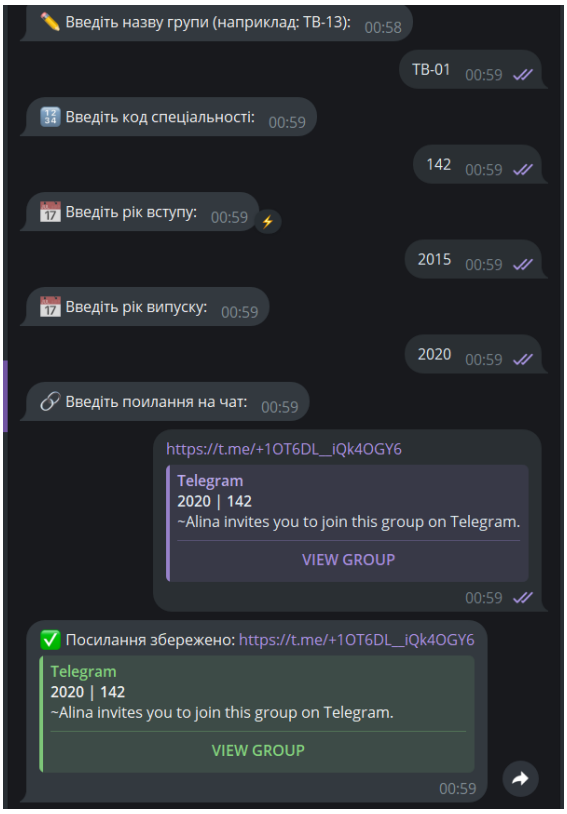


Рисунок 5.14 — Створення групи

Функціонал спілкування створений таким чином, щоб користувачі могли швидко знайти відповідну спільноту; адміністратор мав повний контроль над структурою чатів; бот гарантував унікальність і доступність посилань відповідно до профілю.

Це дозволяє випускникам зберігати зв’язки навіть через роки після завершення навчання, а кафедрі — ефективно модерувати спільноту

Висновок до розділу 5

Реалізований Telegram-бот забезпечує зручну взаємодію з користувачем. Після старту бота користувач проходить реєстрацію, під час якої дані перевіряються на коректність. У головному меню доступні функції перегляду новин (з коротким описом та повною версією), спілкування через чати (відфільтровані за групою, спеціальністю або роком вступу), надсилання фото чи відео, а також запис на події. Адміністратор має доступ до панелі управління, яка адаптується відповідно до рівня доступу. Передбачено також функцію автоматичного нагадування користувачам про події, на які вони зареєструвались. Такий підхід забезпечує інтуїтивну логіку взаємодії з ботом і покриває всі типові сценарії використання.

.

ВИСНОВКИ

Під час написання дипломної роботи спроєктовано та реалізовано Telegram-бот для автоматизованої системи комунікації між випускниками кафедри. Метою проєкту було створення зручної, функціональної системи, яка б дозволяла підтримувати зв’язок між випускниками, обмінюватися інформацією, новинами, організовувати спільні заходи та взаємодіяти в межах спільноти.

У процесі реалізації розроблено повноцінну структуру програмного застосунку з використанням мови програмування Python та асинхронної бібліотеки Aiogram для роботи з Telegram Bot API [9]. Всі основні сценарії — такі як реєстрація, редагування профілю, обробка подій, перегляд новин, надсилання медіа, доступ до групових чатів — реалізовані за допомогою FSM-моделі (Finite State Machine), що дозволяє зберігати контекст взаємодії з кожним користувачем.

У якості бази даних використано SQLite, що забезпечує локальну та швидку обробку запитів. Для резервного копіювання та зручного обміну даними були додатково реалізовані JSON- і CSV-файли для подій, реєстрацій і чатів. Система підтримує розмежування ролей користувачів (звичайний користувач, обмежений адміністратор, супер-адміністратор) з відповідною перевіркою доступу до окремих функцій.

Telegram-бот охоплює такі функціональні можливості:

- покрокова реєстрація та валідація даних користувача;
- створення, редагування та перегляд новин і подій;
- система реєстрації на події з контролем кількості місць;
- автоматичні нагадування та привітання;
- обмін файлами (фото, відео, документи);
- система групових чатів за профілем;
- повна адмін-панель з контролем реєстрацій, контенту та користувачів.

У процесі розробки застосовувалися принципи модульності, асинхронності, валідації введених даних та інтерфейсної зручності через кнопкову навігацію. Інтерфейс реалізовано відповідно до практик UX для мобільного середовища.

Серед можливих напрямків подальшого розвитку:

- реалізація панелі адміністратора в web-інтерфейсі;
- додавання підтримки push-сповіщень у браузері;
- впровадження штучного інтелекту для фільтрації контенту або генерації більш точних коротких описів;
- підтримка декількох мов інтерфейсу (наприклад, додати ще англійську).

Таким чином, реалізована система є завершеним рішенням, яке вже може бути впроваджене у практичну діяльність нашої кафедри та адаптоване під інші спеціальності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Chacon S., Straub B. Pro Git. Berkeley, CA : Apress, 2014. URL: https://doi.org/10.1007/978-1-4842-0076-6 (дата звернення: 02.05.2025).

2. Allen G., Owens M. Introducing SQLite. *The Definitive Guide to SQLite*. Berkeley, CA, 2010. P. 1—16. URL: https://doi.org/10.1007/978-1-4302-3226-1_1 (дата звернення: 04.05.2025).

3. Bassett L. Introduction to JavaScript Object Notation: A to-The-Point Guide to JSON. O'Reilly Media, Incorporated, 2015. 126 p.

4. van den Burg G. J. J., Nazábal A., Sutton C. Wrangling messy CSV files by detecting row and type patterns. *Data mining and knowledge discovery*. 2019. Т. 33, № 6. С. 1799—1820. URL: https://doi.org/10.1007/s10618-019-00646-y (дата звернення: 08.05.2025).

5. Taylor J. T., Taylor W. T. Finite state machines. *Patterns in the machine*. Berkeley, CA, 2021. С. 137—152. URL: https://doi.org/10.1007/978-1-4842-6440-9_10 (дата звернення: 04.05.2025).

6. Json / P. Bourhis та ін. *SIGMOD/PODS'17: international conference on management of data*, м. Chicago Illinois USA. New York, NY, USA, 2017. URL: https://doi.org/10.1145/3034786.3056120 (дата звернення: 03.05.2025).

7. Browning J. B., Alchin M. Documentation. *Pro python 3*. Berkeley, CA, 2019. С. 331—347. URL: https://doi.org/10.1007/978-1-4842-4385-5_8 (дата звернення: 01.05.2025).

8. SQLite Documentation. URL: https://www.sqlite.org/docs.html (дата звернення: 01.05.2025).

9. Telegram Bot API. Офіційна документація. URL: https://core.telegram.org/bots/api (дата звернення: 02.05.2025).

10. Концепція цифрової трансформації освіти і науки на період до 2026 року. Міністерство освіти і науки України. URL:

https://mon.gov.ua/news/kontseptsiya-tsifrovoi-transformatsii-osviti-i-nauki-mon-zaproshue-do-gromadskogo-obgovorennya (дата звернення: 09.05.2025).

11. Білан С. В. Методи автоматизації взаємодії в освітніх онлайн-середовищах *Інформаційні технології і засоби навчання*. 2021. №3(83). С. 45–58.

12. Огієнко О. І., Андрушенко Т. І. Застосування месенджерів у системах організації комунікації між учасниками освітнього процесу // *Наукові записки НУ «Острозька академія». Серія: Психологія і педагогіка*. 2022. №1(17). С. 101–106.

13. Microsoft. *Visual Studio Code Documentation*. URL: https://code.visualstudio.com/docs (дата звернення: 05.05.2025).

14. Aiogram Documentation. *Finite State Machine*. URL: https://docs.aiogram.dev/en/latest/dispatcher/finite_state_machine/index.html (дата звернення: 08.05.2025).

15. Hugging Face. *d0p3/03ap-sm: Ukrainian summarization model*. URL: https://huggingface.co/d0p3/03ap-sm (дата звернення: 11.05.2025).

16. Ammar A., Al-Samarraie H., Alzahrani A. *Chatbot Use Cases in Education: A Systematic Review* // *Education and Information Technologies*. – 2023. – Vol. 28. – P. 2583–2605. DOI: https://doi.org/10.1007/s10639-022-11394-1 (дата звернення: 09.05.2025).

17. Радченко Т. В., Бондаренко І. С. Інформаційні системи та технології в управлінні закладом вищої освіти // *Науковий вісник УжНУ. Серія: Педагогіка*. – 2022. – Вип. 1(48). – С. 56–61.

18. Kulkarni S. *Use of SQLite in Mobile and Embedded Systems* // *International Journal of Computer Applications*. – 2019. – Vol. 178(26). – P. 1–5. DOI: https://doi.org/10.5120/ijca2019919495 (дата звернення: 05.05.2025).

19. Müller F., Mildner T., Neumayr B., Schrefl M. *Modeling Behavior with FSM* // *Behavior Modeling—Foundations and Applications*. – Springer, 2011. – P. 45–72. DOI: https://doi.org/10.1007/978-3-642-18272-0_3 (дата звернення: 05.05.2025).

20. Griol D., Molina J. M. *A Proposal for the Development of Chatbot Interfaces Using FSM and NLP* // *Expert Systems with Applications*. – 2020. – Vol. 160. – Article

113731. DOI: https://doi.org/10.1016/j.eswa.2020.113731 (дата звернення: 05.05.2025).

21. Shah S., Shrivastava R. *Role of Chatbots in Education: A Review* // *International Journal of Creative Research Thoughts*. – 2020. – Vol. 8(6). – P. 950–957. URL: https://ijcrt.org/papers/IJCRT2006208.pdf (дата звернення: 06.05.2025).

22. Натрошвілі О. М., Довгополова І. І. Аналіз використання чат-ботів у дистанційному навчанні // *Молодий вчений*. – 2021. – №12(102). – С. 207–210. URL: http://molodyvcheny.in.ua/files/journal/2021/12.102/43.pdf (дата звернення: 07.05.2025).

23. Pashchenko R. B., Melnyk S. V. Telegram bots as a tool for educational process optimization // *Scientific Bulletin of Uzhhorod National University*. – 2021. – № 2(48). – P. 155–159.

ДОДАТОК А

«Автоматизована система комунікації між випускниками на базі Телеграм-бота»

Презентація

Аркушів: 6

Київ 2025

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

АВТОМАТИЗОВАНА СИСТЕМА КОМУНІКАЦІЇ МІЖ ВИПУСКНИКАМИ КАФЕДРИ НА БАЗІ ТЕЛЕГРАМ-БОТА

Студентка: Сейкаускайте Аліна Андріївна, ТВ-13

Керівник: Бандурка Олена Іванівна

2025

Актуальність теми

Ефективна комунікація між випускниками є ключовим елементом підтримки професійних зв'язків, обміну досвідом та формування спільнот. Відсутність централізованої платформи ускладнює ці процеси.

Telegram-бот створює умови для зручного спілкування між випускниками, автоматизує взаємодію, забезпечує збереження контактної інформації та підтримує організацію спільних ініціатив, а також допомагає в автоматизації багатьох процесів.

Мета та завдання роботи

Мета: створити систему комунікації між випускниками кафедри на базі телеграм-бота з можливістю отримання логістичних даних для покращення сервісів кафедри

Завдання: спроектувати систему реєстрації випускників можливістю фільтрації; створити механізм адміністрування з розмежуванням прав доступу; реалізувати функції публікації новин та подій, а також спілкування за спеціальностями та групами

Проаналізувати схожі системи для комунікації між студентами та розробити основну архітектуру бота

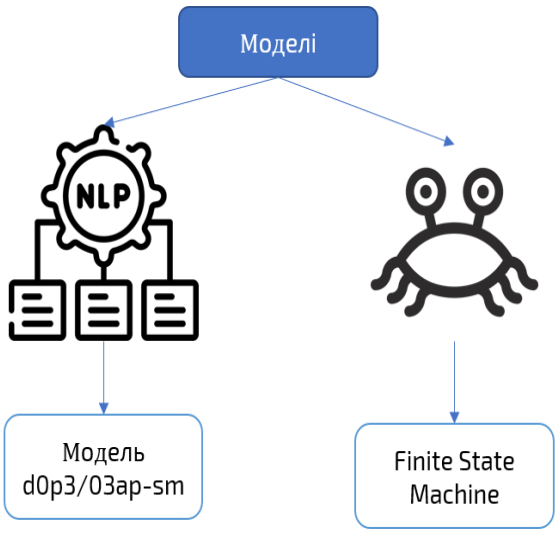
В результаті отримуємо програмне забезпечення, що допомагає сервісам кафедри та випускникам у спілкуванні між собою

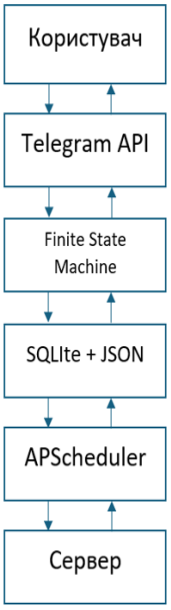
Аналіз існуючих інструментів та систем

	Наявність груп	Можливість пошуку	Надійність інформації	Ефективність та швидкість	Конфіденційність	Модерація	Підтримка ботів	Безкоштовність
Facebook	+	+-	-	+	-	-	-	+
LinkedIn	+	+	+-	-	+	-	-	+
Viber/WhatsApp	+	-	+-	+	+-	-	-	+
Telegram	+	+	+	+	+	+	+	+
Ел. пошта	-	-	-	-	+	-	-	+
Graduway etc	+	+	+	+	+	+	+	-

Проєктування системи

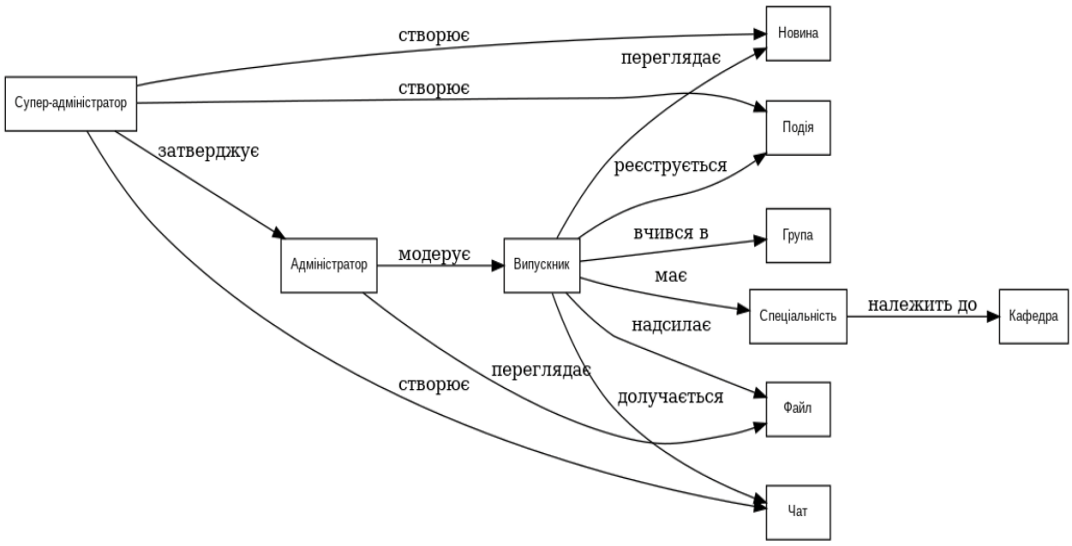
Основні компоненти при проєктуванні системи





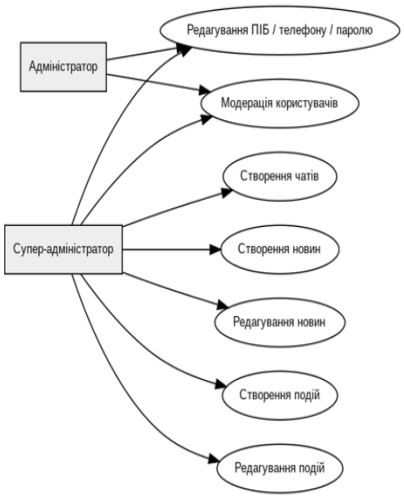
Проєктування системи

Діаграма об'єктів предметної області



Проєктування системи

Діаграми акторів



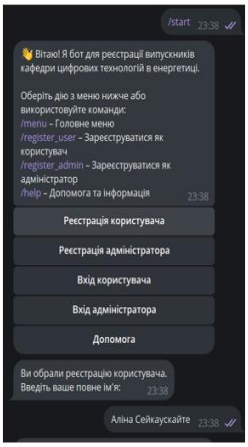
Реалізація системи

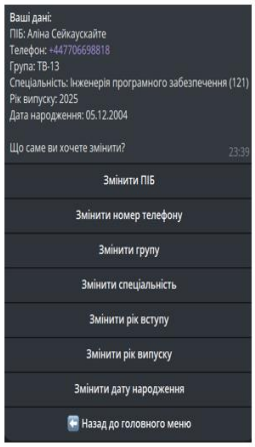
Використані засоби

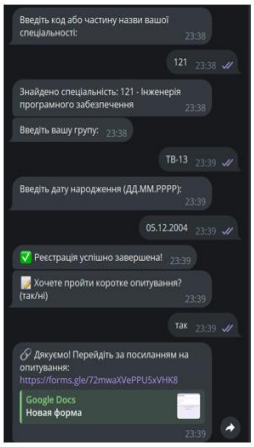
- Мова програмування: Python
- Бібліотека для Телеграму: aiogram 3.x
- Схема станів: Finite State Machine
- База даних: SQLite + JSON / CSV
- Планувальник: APScheduler
- Редактор: Visual Studio Code
- Хостинг: AWS EC2
- Тестування: pytest

Реалізація системи

Інтерфейс користувача



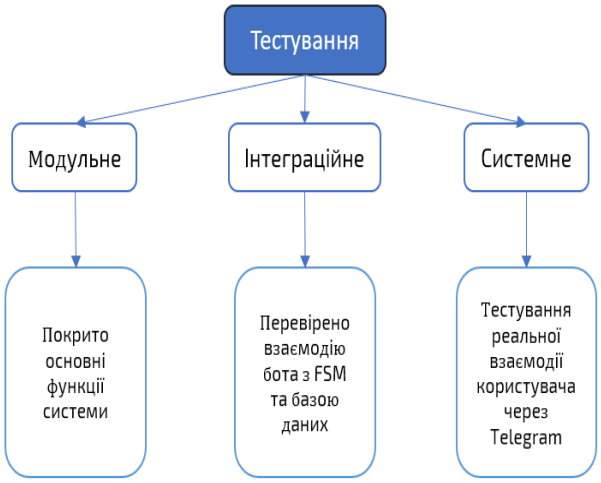




Тестування системи

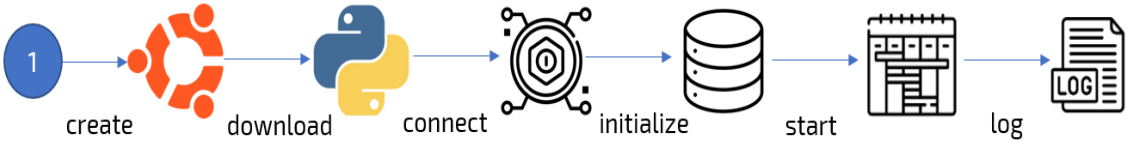
Приклад unit-тесту

```python
@pytest.mark.asyncio
async def test_admin_creates_same_year_twice(monkeypatch):
    """
    ➡️ Якщо у CSV вже є чат для року '2020',
    бот має попередити про дублікат і завершити FSM.
    """

    # CSV з уже існуючим роком 2020
    CHAT_CSV_PATH.write_text("year,link,specialty\n2020,https://t.me/exist,\n", encoding="utf-8")

    context_data = {"chat_type": "create_year_chat", "chat_value": "2020"}
    called = {}

    class DummyFSMContext:
        async def get_data(self): return context_data
        async def clear(self): called["cleared"] = True
        async def update_data(self, **kwargs): called["updated"] = kwargs
        async def set_state(self, state): called["state_set"] = state

    class DummyMessage:
        text = "2020"
        async def answer(self, text): called["answer"] = text

    await receive_chat_value(DummyMessage(), DummyFSMContext())

    assert "вже існує" in called["answer"]
    assert called.get("cleared") is True
```



Впровадження та супровід

Процес розгортання системи



Можливі покращення системи

Висновки

Під час написання дипломної роботи спроєктовано та реалізовано Telegram-бот для автоматизованої системи комунікації між випускниками кафедри.

Для досягнення поставленої мети було виконано наступні завдання:

- **Аналіз вимог:** Досліджено потреби адміністрації кафедри та очікування випускників щодо функціональності системи. Визначено ключові сценарії взаємодії, потребу в авторизації та реєстрації, фільтрації за роком випуску, групою тощо.
- **Проектування системи:** Розроблено загальну структуру Telegram-бота, побудовано структуру бази даних, обрано Telegram API, SQLite та бібліотеку Aiogram для реалізації логіки взаємодії з користувачем.
- **Розробка програмного забезпечення:** Створено основні модулі для реєстрації, входу в систему, перегляду та редагування профілю, адміністрування, публікації новин, створення подій, спілкування у чатах за групами та спеціальностями.
- **Тестування і валідація:** Проведено тестування бот-системи з реальними користувачами та адміністраторами, перевірено відповідність функціональності очікуванням та коректність збереження даних у базі.

ДОДАТОК Б

«Автоматизована система комунікації між випускниками на базі Телеграм-бота»

Лістинг адміністратора

Аркушів: 12

Київ 2025

Admin_panel.py:

```
import secrets
import sqlite3
import string
from aiogram import Router
from aiogram.types import CallbackQuery, Message
from aiogram.fsm.context import FSMContext
from aiogram.fsm.state import State, StatesGroup
from aiogram.exceptions import TelegramBadRequest
from ...utils.phone_validator import is_valid_phone
from ...utils.keyboard import admin_main_menu_keyboard, view_users_sort_keyboard,
user_management_keyboard, limited_admin_menu_keyboard
from ...database.queries import get_connection

router = Router()

# Стани для адмін дій
class AdminPanelStates(StatesGroup):
    waiting_phone_to_delete = State()
    waiting_phone_to_block = State()
    waiting_phone_to_change_access = State()
    waiting_new_access_level = State()

def get_admin_access_level(telegram_id: str) -> str | None:
    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT access_level FROM admins WHERE telegram_id = ?", (telegram_id,))
    result = cursor.fetchone()
    conn.close()
    return result[0] if result else None

def super_admin_only(func):
    async def wrapper(callback: CallbackQuery, *args, **kwargs):
        access_level = get_admin_access_level(str(callback.from_user.id))
        if access_level != "admin_super":
            await callback.message.answer("🚫 У вас немає прав для цієї дії.")
            return
        return await func(callback, *args, **kwargs)
    return wrapper

@router.callback_query(lambda c: c.data == "user_management_menu")
async def open_user_management_menu(callback_query: CallbackQuery, state: FSMContext):
    await callback_query.message.edit_text(
        "👥 <b>Робота з користувачами:</b>\nОберіть дію нижче:",
        reply_markup=user_management_keyboard,
        parse_mode="HTML"
    )

# Після входу – показати адмін-панель
@router.callback_query(lambda c: c.data == "login_admin_menu")
async def show_admin_menu(callback: CallbackQuery, state: FSMContext):
    telegram_id = str(callback.from_user.id)
    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT access_level FROM admins WHERE telegram_id = ?", (telegram_id,))
    result = cursor.fetchone()
    conn.close()

    if result:
        access_level = result[0]
```

```
            if access_level == "admin_super":
                await callback.message.answer("🔧 Адмін-панель (повний доступ):",
reply_markup=admin_main_menu_keyboard)
            else:
                await callback.message.answer("👮 Адмін-панель (обмежений доступ):",
reply_markup=limited_admin_menu_keyboard)
        else:
            await callback.message.answer("❌ Ви не авторизовані як адміністратор.")

    # 📊 Показати варіанти сортування користувачів
    @router.callback_query(lambda c: c.data == "view_users")
    async def view_users_sort_menu(callback_query: CallbackQuery, state: FSMContext):
        try:
            await callback_query.message.edit_text("📊 Виберіть спосіб сортування користувачів:",
reply_markup=view_users_sort_keyboard)
        except TelegramBadRequest as e:
            if "message is not modified" not in str(e):
                raise

    # 🎓 Сортування за групою
    @router.callback_query(lambda c: c.data == "view_users_group")
    async def view_users_by_group(callback_query: CallbackQuery, state: FSMContext):
        conn = sqlite3.connect("alumni.db")
        c = conn.cursor()
        c.execute("SELECT full_name, phone_number, group_name, role FROM users ORDER BY group_name
ASC")
        users = c.fetchall()
        conn.close()

        text = format_user_list(users, "🎓 Користувачі за групами")

        try:
            await callback_query.message.edit_text(text, parse_mode="HTML",
reply_markup=view_users_sort_keyboard)
        except TelegramBadRequest as e:
            if "message is not modified" not in str(e):
                raise

    # 🗓️ Сортування за роком випуску
    @router.callback_query(lambda c: c.data == "view_users_year")
    async def view_users_by_year(callback_query: CallbackQuery, state: FSMContext):
        conn = sqlite3.connect("alumni.db")
        c = conn.cursor()
        c.execute("SELECT full_name, phone_number, graduation_year, role FROM users ORDER BY
graduation_year DESC")
        users = c.fetchall()
        conn.close()

        if not users:
            text = "❌ Жодного користувача не знайдено."
        else:
            text = "\n".join([f"👤 {u[0]} | 📱 {u[1]} | 🗓️ {u[2]} | 🔑 {u[3]}" for u in users])
            text = f"🗓️ <b>Користувачі за роком випуску:</b>\n\n{text}"

        try:
            await callback_query.message.edit_text(text, parse_mode="HTML",
reply_markup=view_users_sort_keyboard)
        except TelegramBadRequest as e:
            if "message is not modified" not in str(e):
```

```
            raise

    # 📘 Сортування за спеціальністю + роком
    @router.callback_query(lambda c: c.data == "view_users_specialty")
    async def view_users_by_specialty(callback_query: CallbackQuery, state: FSMContext):
        conn = sqlite3.connect("alumni.db")
        c = conn.cursor()
        c.execute('''
            SELECT u.full_name, u.phone_number, s.name, u.graduation_year, u.role
            FROM users u
            JOIN specialties s ON u.specialty_id = s.id
            ORDER BY s.name ASC, u.graduation_year DESC
        ''')
        users = c.fetchall()
        conn.close()

        if not users:
            text = "❌ Жодного користувача не знайдено."
        else:
            text = "\n".join([f"👤 {u[0]} | 📱 {u[1]} | 📘 {u[2]} | 🗓️ {u[3]} | 🔑 {u[4]}" for u in users])
            text = f"📘 <b>Користувачі за спеціальністю:</b>\n\n{text}"

        try:
            await callback_query.message.edit_text(text, parse_mode="HTML",
reply_markup=view_users_sort_keyboard)
        except TelegramBadRequest as e:
            if "message is not modified" not in str(e):
                raise

    # Форматування списку
    def format_user_list(users, title):
        if not users:
            return "❌ Жодного користувача не знайдено."
        user_list = "\n".join([f"👤 {u[0]} | 📱 {u[1]} | 🎓 {u[2]} | 🔑 {u[3]}" for u in users])
        return f"👥 <b>{title}:</b>\n\n{user_list}"

    # ❌ Видалити користувача
    @router.callback_query(lambda c: c.data == "delete_user")
    async def delete_user_prompt(callback_query: CallbackQuery, state: FSMContext):
        await callback_query.message.answer("📱 Введіть номер телефону користувача, якого потрібно
видалити:")
        await state.set_state(AdminPanelStates.waiting_phone_to_delete)

    @router.message(AdminPanelStates.waiting_phone_to_delete)
    async def delete_user_confirm(message: Message, state: FSMContext):
        phone = message.text.strip()
        if not is_valid_phone(phone):
            await message.answer("❌ Невірний формат телефону.")
            return

        if not user_exists_by_phone(phone):
            await message.answer("❌ Користувача з таким номером не знайдено.")
        else:
            conn = sqlite3.connect("alumni.db")
            c = conn.cursor()
            c.execute("DELETE FROM users WHERE phone_number = ?", (phone,))
            conn.commit()
            conn.close()
            await message.answer("☑️ Користувача успішно видалено.")
```

```
    await show_admin_menu(message, state)

# 🚫 Заблокувати
@router.callback_query(lambda c: c.data == "block_user")
async def block_user_prompt(callback_query: CallbackQuery, state: FSMContext):
    await callback_query.message.answer("📱 Введіть номер телефону користувача, якого потрібно заблокувати:")
    await state.set_state(AdminPanelStates.waiting_phone_to_block)

@router.message(AdminPanelStates.waiting_phone_to_block)
async def block_user_confirm(message: Message, state: FSMContext):
    phone = message.text.strip()
    if not is_valid_phone(phone):
        await message.answer("❌ Невірний формат телефону.")
        return

    if not user_exists_by_phone(phone):
        await message.answer("❌ Користувача з таким номером не знайдено.")
    else:
        conn = sqlite3.connect("alumni.db")
        c = conn.cursor()
        c.execute("UPDATE users SET role = 'blocked' WHERE phone_number = ?", (phone,))
        conn.commit()
        conn.close()
        await message.answer("🚫 Користувача заблоковано.")

    await show_admin_menu(message, state)


# 🔐 Змінити доступ
@router.callback_query(lambda c: c.data == "change_access")
@super_admin_only
async def change_access_prompt(callback_query: CallbackQuery, state: FSMContext, **kwargs):
    await callback_query.message.answer("📱 Введіть номер телефону користувача, чий доступ потрібно змінити:")
    await state.set_state(AdminPanelStates.waiting_phone_to_change_access)


@router.message(AdminPanelStates.waiting_phone_to_change_access)
@super_admin_only
async def change_access_level(message: Message, state: FSMContext, **kwargs):
    phone = message.text.strip()
    await state.update_data(phone=phone)
    if not user_exists_by_phone(phone):
        await message.answer("❌ Користувача не знайдено.")
        await show_admin_menu(message, state)
    else:
        await message.answer("🔐 Введіть новий рівень доступу ('user', 'admin_limited', 'admin_super'):")
        await state.set_state(AdminPanelStates.waiting_new_access_level)


def generate_password(length: int = 8) -> str:
    characters = string.ascii_letters + string.digits
    return ''.join(secrets.choice(characters) for _ in range(length))


@router.message(AdminPanelStates.waiting_new_access_level)
@super_admin_only
```

```
async def confirm_access_change(message: Message, state: FSMContext, **kwargs):
    access = message.text.strip()
    data = await state.get_data()
    phone = data.get("phone")

    if access not in {"user", "admin_limited", "admin_super"}:
        await message.answer("❌ Недійсний рівень доступу. Спробуйте ще раз.")
        return

    conn = sqlite3.connect("alumni.db")
    cursor = conn.cursor()

    # Отримуємо telegram_id користувача за номером
    cursor.execute("SELECT telegram_id, full_name FROM users WHERE phone_number = ?", (phone,))
    user_row = cursor.fetchone()

    if not user_row:
        await message.answer("❌ Користувача не знайдено.")
        conn.close()
        return

    telegram_id, full_name = user_row

    # Оновлюємо рівень доступу
    cursor.execute("UPDATE users SET access_level = ? WHERE phone_number = ?", (access, phone))

    if access == "user":
        cursor.execute("DELETE FROM admins WHERE telegram_id = ?", (telegram_id,))
        await message.answer("👤 Користувача переведено до звичайних користувачів.")
    else:
        cursor.execute("SELECT 1 FROM admins WHERE telegram_id = ?", (telegram_id,))
        if cursor.fetchone() is None:
            password = generate_password()
            cursor.execute(
                "INSERT INTO admins (telegram_id, phone_number, full_name, password, access_level) VALUES (?,
?, ?, ?, ?)",
                (telegram_id, phone, full_name, password, access)
            )
            await message.bot.send_message(
                chat_id=telegram_id,
                text=f"🔐 Вас призначено адміністратором.\nВаш пароль для входу: <code>{password}</code>",
                parse_mode="HTML"
            )
        else:
            cursor.execute("UPDATE admins SET access_level = ? WHERE telegram_id = ?", (access, telegram_id))
        conn.commit()
        conn.close()

        await message.answer(f"☑ Доступ користувача оновлено до: {access}")
        await show_admin_menu(message, state)

# Перевірка існування користувача за телефоном
def user_exists_by_phone(phone_number: str) -> bool:
    conn = sqlite3.connect("alumni.db")
    c = conn.cursor()
    c.execute("SELECT 1 FROM users WHERE phone_number = ?", (phone_number,))
    exists = c.fetchone() is not None
    conn.close()
    return exists
```

```
@router.callback_query(lambda c: c.data == "view_uploaded_files")
@super_admin_only
async def view_uploaded_files(callback: CallbackQuery, state: FSMContext, **kwargs):
    conn = get_connection()
    cursor = conn.cursor()

    cursor.execute('''
        SELECT telegram_id, file_id, file_type, upload_time
        FROM user_files
        ORDER BY upload_time DESC
        LIMIT 10
    ''')
    files = cursor.fetchall()
    conn.close()

    if not files:
        await callback.message.answer("❌ Жодного файлу ще не надіслано.")
        return

    for idx, (user_id, file_id, file_type, upload_time) in enumerate(files, start=1):
        caption = f"#{idx} 🗓 {upload_time}\n👤 Користувач: <code>{user_id}</code>"

        try:
            if file_type == "photo":
                await callback.message.bot.send_photo(chat_id=callback.from_user.id, photo=file_id, caption=caption,
parse_mode="HTML")
            else:
                await callback.message.bot.send_message(chat_id=callback.from_user.id, text=f"{caption}\n🔗 Файл:
{file_id}", parse_mode="HTML")
        except TelegramBadRequest as e:
            await callback.message.answer(f"⚠️ Помилка при відправленні файлу #{idx}.")

    await callback.message.answer("⬅️ Повертаємось до адмін-панелі.",
reply_markup=admin_main_menu_keyboard)
```

communication_admin.py:

```
from aiogram import Router
from aiogram.types import CallbackQuery, Message, InlineKeyboardMarkup, InlineKeyboardButton
from aiogram.fsm.context import FSMContext
from aiogram.fsm.state import State, StatesGroup
import csv
from pathlib import Path

from ...utils.keyboard import communication_admin_menu_keyboard, filter_chats_keyboard
from ...utils.notify import notify_users_about_chat

router = Router()
CHAT_CSV_PATH = Path("BOT/handlers/communication/chat_links.csv")

class ChatCreationState(StatesGroup):
    entering_group = State()
    entering_specialty = State()
    entering_enrollment_year = State()
    entering_graduation_year = State()
    waiting_for_link = State()

class ChatFilterState(StatesGroup):
    waiting_filter_value = State()
```

```
@router.callback_query(lambda c: c.data == "admin_communication_menu")
async def open_admin_communication_menu(callback: CallbackQuery, state: FSMContext):
    await callback.message.edit_text(
        "<b>Меню спілкування:</b>\nОберіть дію нижче.",
        reply_markup=communication_admin_menu_keyboard,
        parse_mode="HTML"
    )

@router.callback_query(lambda c: c.data == "create_new_chat")
async def start_chat_creation(callback: CallbackQuery, state: FSMContext):
    await callback.message.answer("🖍️ Введіть назву групи (наприклад: ТВ-13):")
    await state.set_state(ChatCreationState.entering_group)

@router.message(ChatCreationState.entering_group)
async def get_group(message: Message, state: FSMContext):
    await state.update_data(group=message.text.strip())
    await message.answer("🔢 Введіть код спеціальності:")
    await state.set_state(ChatCreationState.entering_specialty)

@router.message(ChatCreationState.entering_specialty)
async def get_specialty(message: Message, state: FSMContext):
    await state.update_data(specialty=message.text.strip())
    await message.answer("🗓️ Введіть рік вступу:")
    await state.set_state(ChatCreationState.entering_enrollment_year)

@router.message(ChatCreationState.entering_enrollment_year)
async def get_enrollment_year(message: Message, state: FSMContext):
    await state.update_data(enrollment_year=message.text.strip())
    await message.answer("🗓️ Введіть рік випуску:")
    await state.set_state(ChatCreationState.entering_graduation_year)

@router.message(ChatCreationState.entering_graduation_year)
async def get_graduation_year(message: Message, state: FSMContext):
    await state.update_data(graduation_year=message.text.strip())
    await message.answer("🔗 Введіть поилання на чат:")
    await state.set_state(ChatCreationState.waiting_for_link)

@router.message(ChatCreationState.waiting_for_link)
async def save_chat_link(message: Message, state: FSMContext):
    data = await state.get_data()
    link = message.text.strip()

    if not (link.startswith("https://t.me/") or link.startswith("https://telegram.me/")):
        await message.answer("❌ Посилання повинно починатися з https://t.me/ або https://telegram.me/")
        return

    new_row = {
        "group": data["group"],
        "specialty": data["specialty"],
        "enrollment_year": data["enrollment_year"],
        "graduation_year": data["graduation_year"],
        "link": link
    }

    rows = []
    updated = False

    if CHAT_CSV_PATH.exists():
```

```
        with CHAT_CSV_PATH.open("r", encoding="utf-8") as f:
            reader = csv.DictReader(f)
            rows = list(reader)

    for row in rows:
        if (
            row["group"] == new_row["group"] and
            row["specialty"] == new_row["specialty"] and
            row["enrollment_year"] == new_row["enrollment_year"] and
            row["graduation_year"] == new_row["graduation_year"]
        ):
            row["link"] = link
            updated = True
            break

    if not updated:
        rows.append(new_row)

    with CHAT_CSV_PATH.open("w", newline="", encoding="utf-8") as f:
        writer = csv.DictWriter(f, fieldnames=["group", "specialty", "enrollment_year", "graduation_year", "link"])
        writer.writeheader()
        writer.writerows(rows)

    await message.answer(f"☑ Посилання {'оновлено' if updated else 'збережено'}: {link}")

    await notify_users_about_chat(
        bot=message.bot,
        chat_type="group",
        match_value=f"{data['group']} | {data['specialty']} | {data['enrollment_year']}-{data['graduation_year']}",
        link=link
    )

    await state.clear()

@router.callback_query(lambda c: c.data == "view_all_chats")
async def view_all_chats(callback: CallbackQuery, state: FSMContext):
    if not CHAT_CSV_PATH.exists():
        await callback.message.answer("📪 Ще не збережено жодного чату.")
        return

    with CHAT_CSV_PATH.open("r", encoding="utf-8") as f:
        reader = csv.DictReader(f)
        for row in reader:
            description = (
                f"👥 Група: {row['group']}\n"
                f"🧪 Спеціальність: {row['specialty']}\n"
                f"🗓 {row['enrollment_year']} – {row['graduation_year']}\n"
            )
            keyboard = InlineKeyboardMarkup(inline_keyboard=[
                [InlineKeyboardButton(text="🔗 Перейти до чату", url=row['link'])]
            ])
            await callback.message.answer(
                f"📌 <b>Чат</b>\n{description}🔗 {row['link']}",
                parse_mode="HTML",
                reply_markup=keyboard
            )

@router.callback_query(lambda c: c.data == "communication_filter_chats")
async def show_filter_menu(callback: CallbackQuery, state: FSMContext):
```

```
        await callback.message.edit_text("🔍 Оберіть критерій для фільтрації чатів:",
reply_markup=filter_chats_keyboard)

    @router.callback_query(lambda c: c.data.startswith("communication_filter_"))
    async def start_filtering(callback: CallbackQuery, state: FSMContext):
        filter_type = callback.data.replace("communication_filter_", "")
        await state.update_data(filter_type=filter_type)
        prompt = {
            "group": "Введіть назву групи:",
            "specialty": "Введіть код спеціальності:",
            "enrollment_year": "Введіть рік вступу:",
            "graduation_year": "Введіть рік випуску:"
        }[filter_type]
        await callback.message.answer(prompt)
        await state.set_state(ChatFilterState.waiting_filter_value)

    @router.message(ChatFilterState.waiting_filter_value)
    async def apply_filter(message: Message, state: FSMContext):
        data = await state.get_data()
        filter_type = data["filter_type"]
        value = message.text.strip()

        if not CHAT_CSV_PATH.exists():
            await message.answer("📭 Файл з чатами не знайдено.")
            await state.clear()
            return

        found = False
        with CHAT_CSV_PATH.open("r", encoding="utf-8") as f:
            reader = csv.DictReader(f)
            for row in reader:
                if row[filter_type] == value:
                    found = True
                    description = (
                        f"👥 Група: {row['group']}\n"
                        f"🧪 Спеціальність: {row['specialty']}\n"
                        f"🗓 {row['enrollment_year']} – {row['graduation_year']}\n"
                    )
                    keyboard = InlineKeyboardMarkup(inline_keyboard=[
                        [InlineKeyboardButton(text="🔗 Перейти до чату", url=row['link'])]
                    ])
                    await message.answer(
                        f"📌 <b>Знайдено чат:</b>\n{description}🔗 {row['link']}",parse_mode =
"HTML",reply_markup=keyboard
                    )

        if not found:
            await message.answer("❌ Жодного чату за вказаним критерієм не знайдено.")

        await state.clear()
```

register_admin.py:

```
import sqlite3
import random
import string
import re
from aiogram import Router, Bot
```

```
from aiogram.types import Message, CallbackQuery, InlineKeyboardMarkup, InlineKeyboardButton,
ReplyKeyboardRemove
from aiogram.filters import Command
from aiogram.fsm.context import FSMContext
from aiogram.fsm.state import State, StatesGroup

from ...config import SUPER_ADMIN_ID
from ...database.queries import get_connection
from ...utils.phone_validator import is_valid_phone
from ...utils.keyboard import request_phone_keyboard

router = Router()

# Стани для реєстрації адміна
class AdminRegistration(StatesGroup):
    full_name = State()
    phone_number = State()

# Генерація рандомного паролю
def generate_random_password(length=8):
    chars = string.ascii_letters + string.digits
    return ''.join(random.choice(chars) for _ in range(length))

# Старт реєстрації адміна
@router.message(Command('register_admin'))
async def start_admin_register(message: Message, state: FSMContext):
    await message.answer("Реєстрація адміністратора.\nВведіть ваше повне ім'я:")
    await state.set_state(AdminRegistration.full_name)

@router.message(AdminRegistration.full_name)
async def admin_full_name(message: Message, state: FSMContext):
    full_name = message.text.strip()
    if not re.match(r"^[A-Яа-яІіЇїЄєҐґA-Za-z]+ [A-Яа-яІіЇїЄєҐґA-Za-z]+$", full_name):
        await message.answer("❌ Ім'я повинно містити лише літери і складатися з двох слів (наприклад: Іван
Петренко). Спробуйте ще раз.")
        return
    await state.update_data(full_name=full_name)
    await message.answer("📞 Введіть номер телефону у міжнародному форматі (наприклад, +380501234567):",
reply_markup=request_phone_keyboard)
    await state.set_state(AdminRegistration.phone_number)

@router.message(AdminRegistration.phone_number)
async def admin_phone_number(message: Message, state: FSMContext, bot: Bot):
    phone = message.contact.phone_number if message.contact else message.text.strip() if message.text else ""

    if not is_valid_phone(phone):
        await message.answer("❌ Невалідний номер. Введіть ще раз у форматі +380501234567:")
        return

    await message.answer("☑ Номер прийнято.", reply_markup=ReplyKeyboardRemove())

    await state.update_data(phone_number=phone)
    data = await state.get_data()
    password = generate_random_password()
    telegram_id = str(message.from_user.id)

    conn = get_connection()
    cursor = conn.cursor()

    # Чи вже є така заявка?
```

```
    cursor.execute("SELECT 1 FROM admin_requests WHERE telegram_id = ?", (telegram_id,))
    if cursor.fetchone():
        await message.answer("🕓 Ваша заявка вже на розгляді.")
        return

    try:
        cursor.execute("""
            INSERT INTO admin_requests (telegram_id, full_name, phone_number, password)
            VALUES (?, ?, ?, ?)
        """, (telegram_id, data["full_name"], phone, password))
        conn.commit()
    except sqlite3.IntegrityError:
        await message.answer("⚠️ Ви вже подали заявку або зареєстровані.")
        return

    await message.answer("☑️ Заявку на реєстрацію адміністратора надіслано. Очікуйте підтвердження.")

    await bot.send_message(
        SUPER_ADMIN_ID,
        f"📝 Нова заявка на адміністратора:\n"
        f"👤 Ім'я: {data['full_name']}\n📱 Телефон: {phone}\n🆔 Telegram ID: {telegram_id}",
        reply_markup=InlineKeyboardMarkup(inline_keyboard=[
            [InlineKeyboardButton(text="☑️ Прийняти", callback_data=f"approve_admin_{telegram_id}")],
            [InlineKeyboardButton(text="❌ Відхилити", callback_data=f"reject_admin_{telegram_id}")]
        ])
    )
    await state.clear()

# ☑️ Схвалення суперадміністратором
@router.callback_query(lambda c: c.data.startswith("approve_admin_"))
async def approve_admin(callback: CallbackQuery, bot: Bot):
    telegram_id = callback.data.split("_")[-1]
    conn = get_connection()
    cursor = conn.cursor()

    cursor.execute("SELECT full_name, phone_number, password FROM admin_requests WHERE telegram_id = ?",
(telegram_id,))
    row = cursor.fetchone()
    if not row:
        await callback.message.answer("❌ Заявку не знайдено.")
        return

    full_name, phone_number, password = row

    cursor.execute("SELECT 1 FROM admins WHERE telegram_id = ?", (telegram_id,))
    if cursor.fetchone():
        await callback.message.answer("⚠️ Адміністратор вже існує.")
        return

    cursor.execute("""
        INSERT INTO admins (telegram_id, full_name, phone_number, password)
        VALUES (?, ?, ?, ?)
    """, (telegram_id, full_name, phone_number, password))
    cursor.execute("DELETE FROM admin_requests WHERE telegram_id = ?", (telegram_id,))
    conn.commit()

    await callback.message.answer("☑️ Адміністратора підтверджено.")
    await bot.send_message(
        chat_id=telegram_id,
```

```
        text=(
            "🎉 Вас підтверджено як адміністратора!\n\n"
            f"👤 ПІБ: {full_name}\n📱 Телефон: {phone_number}\n"
            f"🔐 Ваш пароль: <code>{password}</code>\n\n"
            "Використовуйте цей пароль для авторизації.",
        ),
        parse_mode="HTML"
    )


# ❌ Відхилення заявки
@router.callback_query(lambda c: c.data.startswith("reject_admin_"))
async def reject_admin(callback: CallbackQuery, bot: Bot):
    telegram_id = callback.data.split("_")[-1]
    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("DELETE FROM admin_requests WHERE telegram_id = ?", (telegram_id,))
    conn.commit()
    await callback.message.answer("🚫 Заявку відхилено.")
    await bot.send_message(telegram_id, "❌ Ваша заявка на адміністратора була відхилена.")
```

ДОДАТОК В

«Автоматизована система комунікації між випускниками на базі Телеграм-бота»

Лістинг користувача

Аркушів: 9

Київ 2025

Communication_user.py:

```
from aiogram import Router
from aiogram.types import CallbackQuery
from aiogram.fsm.context import FSMContext
from ...utils.keyboard import communication_user_menu_keyboard, user_main_menu_keyboard
from ...database.queries import get_connection
from pathlib import Path
import csv

router = Router()
CHAT_CSV_PATH = Path("BOT/handlers/communication/chat_links.csv")

@router.callback_query(lambda c: c.data == "user_communication_menu")
async def open_communication_menu(callback: CallbackQuery, state: FSMContext):
    await callback.message.edit_text(
        "<b>👥 Спілкування між випускниками</b>\n\n"
        "Оберіть тип спілкування:",
        reply_markup=communication_user_menu_keyboard,
        parse_mode="HTML"
    )


@router.callback_query(lambda c: c.data == "chat_by_group")
async def send_group_chat_link(callback: CallbackQuery, state: FSMContext):
    telegram_id = str(callback.from_user.id)

    conn = get_connection()
    cursor = conn.cursor()

    cursor.execute("SELECT group_name FROM users WHERE telegram_id = ?", (telegram_id,))
    result = cursor.fetchone()
    conn.close()

    if not result:
        await callback.message.answer("❌ Не вдалося отримати вашу групу.")
        return

    group_name = result[0]

    # Пошук у CSV
    if not CHAT_CSV_PATH.exists():
        await callback.message.answer("📭 Жодного чату ще не створено.")
        return

    with CHAT_CSV_PATH.open("r", encoding="utf-8") as f:
        reader = csv.DictReader(f)
        for row in reader:
            if row["group"] == group_name:
                await callback.message.answer(f"🔗 Ось ваше посилання на чат:\n{row['link']}")
                return

    await callback.message.answer("❌ Чат для вашої групи ще не створено адміністратором.")


@router.callback_query(lambda c: c.data in ["chat_by_specialty", "chat_by_enrollment", "chat_by_graduation"])
async def send_communication_link(callback: CallbackQuery, state: FSMContext):
    telegram_id = str(callback.from_user.id)

    conn = get_connection()
```

```
    cursor = conn.cursor()

    if callback.data == "chat_by_specialty":
        cursor.execute("SELECT specialty_id FROM users WHERE telegram_id = ?", (telegram_id,))
        result = cursor.fetchone()
        if not result:
            await callback.message.answer("❌ Не вдалося отримати вашу спеціальність.")
            return
        specialty_id = result[0]
        cursor.execute("SELECT code FROM specialties WHERE id = ?", (specialty_id,))
        code_result = cursor.fetchone()
        if not code_result:
            await callback.message.answer("❌ Не вдалося знайти код спеціальності.")
            return
        user_value = code_result[0]
        chat_type = "specialty"
    elif callback.data == "chat_by_enrollment":
        cursor.execute("SELECT enrollment_year FROM users WHERE telegram_id = ?", (telegram_id,))
        result = cursor.fetchone()
        if not result:
            await callback.message.answer("❌ Не вдалося отримати рік вступу.")
            return
        user_value = str(result[0])
        chat_type = "enrollment_year"
    elif callback.data == "chat_by_graduation":
        cursor.execute("SELECT graduation_year FROM users WHERE telegram_id = ?", (telegram_id,))
        result = cursor.fetchone()
        if not result:
            await callback.message.answer("❌ Не вдалося отримати рік випуску.")
            return
        user_value = str(result[0])
        chat_type = "graduation_year"
    else:
        await callback.message.answer("❌ Невідомий тип чату.")
        conn.close()
        return

    conn.close()

    if not CHAT_CSV_PATH.exists():
        await callback.message.answer("📭 Жодного чату ще не створено.")
        return

    with CHAT_CSV_PATH.open("r", encoding="utf-8") as f:
        reader = csv.DictReader(f)
        for row in reader:
            if row.get(chat_type) == user_value:
                await callback.message.answer(f"🔗 Ось ваше посилання на чат:\n{row['link']}")
                return

    await callback.message.answer("❌ Чат ще не створено адміністратором.")
```

register_user.py:

```
import sqlite3
import re
from aiogram import Router
from aiogram.types import Message, CallbackQuery, ReplyKeyboardRemove
from aiogram.filters import Command
```

```
from aiogram.fsm.context import FSMContext
from aiogram.fsm.state import State, StatesGroup

from ...utils.phone_validator import is_valid_phone
from ...utils.specialties import SPECIALTIES, search_specialty
from ...utils.department_recogniser import normalize_department
from ...utils.keyboard import request_phone_keyboard, main_menu_keyboard, user_main_menu_keyboard
from ...database.queries import get_connection

from datetime import datetime

# Підключення до БД
conn = get_connection()
cursor = conn.cursor()

router = Router()

# Стан машини для реєстрації
class Registration(StatesGroup):
    full_name = State()
    phone_number = State()
    old_phone_number_check = State()
    old_phone_number = State()
    enrollment_year = State()
    graduation_year = State()
    department_id = State()
    specialty_input = State()
    specialty_select = State()
    group_name = State()
    birth_date = State()
    ask_survey = State()

@router.callback_query(lambda c: c.data == 'register_user')
async def callback_register_user(callback_query: CallbackQuery, state: FSMContext):
    await callback_query.message.answer("Ви обрали реєстрацію користувача.\nВведіть ваше повне ім'я:")
    await state.set_state(Registration.full_name)

@router.callback_query(lambda c: c.data == 'register_admin')
async def callback_register_admin(callback_query: CallbackQuery):
    await callback_query.message.answer("Використовуйте команду /register_admin для реєстрації адміністратора.")

@router.callback_query(lambda c: c.data == 'help_info')
async def callback_help(callback_query: CallbackQuery):
    help_text = (
        "👋 <b>Цей бот створений для випускників кафедри цифрових технологій в енергетиці</b>.\n\n"
        "Тут ви можете:\n"
        " 🔹 Пройти реєстрацію або увійти в акаунт\n"
        " 🔹 Редагувати свій профіль\n"
        " 🔹 Отримувати актуальні новини\n"
        " 🔹 Переглядати майбутні події та реєструватися на них\n"
        " 🔹 Надсилати фото й відео (наприклад, з минулих подій)\n"
        " 🔹 Знаходити чати для спілкування з одногрупниками або спеціальністю\n\n"
        "🔽 Використовуйте кнопки або команди меню для навігації.\n"
        "Допомога: alina.seikauskaite3@gmail.com"
    )
    await callback_query.message.answer(help_text)
```

```
@router.message(Command('start'))
async def cmd_start(message: Message, state: FSMContext):
    help_text = (
        "👋 Вітаю! Я бот для реєстрації випускників кафедри цифрових технологій в енергетиці.\n\n"
        "Оберіть дію з меню нижче або використовуйте команди:\n"
        "/menu – Головне меню\n"
        "/register_user – Зареєструватися як користувач\n"
        "/register_admin – Зареєструватися як адміністратор\n"
        "/help – Допомога та інформація\n"
    )
    await message.answer(help_text, reply_markup=main_menu_keyboard)


@router.message(Registration.full_name)
async def process_full_name(message: Message, state: FSMContext):
    full_name = message.text.strip()

    # Перевірка: тільки букви + один пробіл
    if not re.match(r"^[А-Яа-яЁёІіЇїЄєҐґA-Za-z]+ [А-Яа-яЁёІіЇїЄєҐґA-Za-z]+$", full_name):
        await message.answer("❌ Ім'я повинно містити тільки літери і складатися з двох слів (наприклад: Іван Петренко). Введіть ще раз:")
        return

    await state.update_data(full_name=full_name)
    await message.answer("Будь ласка, поділіться вашим номером телефону, натиснувши кнопку нижче:", reply_markup=request_phone_keyboard)
    await state.set_state(Registration.phone_number)


@router.message(Registration.phone_number)
async def process_phone_number(message: Message, state: FSMContext):
    contact = message.contact
    if contact:
        phone = contact.phone_number
    else:
        phone = message.text

    if not is_valid_phone(phone):
        await message.answer("Неправильний номер телефону. Введіть ще раз або натисніть кнопку:", reply_markup=request_phone_keyboard)
        return

    await state.update_data(phone_number=phone)
    await message.answer("Чи змінювався номер телефону з часу випуску? (так/ні)", reply_markup=ReplyKeyboardRemove())
    await state.set_state(Registration.old_phone_number_check)

@router.message(Registration.old_phone_number_check)
async def process_old_phone_check(message: Message, state: FSMContext):
    if message.text.lower() == 'так':
        await message.answer("Введіть старий номер телефону:")
        await state.set_state(Registration.old_phone_number)
    else:
        data = await state.get_data()
        await state.update_data(old_phone_number=data['phone_number'])
        await message.answer("Введіть рік вступу:")
        await state.set_state(Registration.enrollment_year)

@router.message(Registration.old_phone_number)
async def process_old_phone_number(message: Message, state: FSMContext):
```

```
    await state.update_data(old_phone_number=message.text)
    await message.answer("Введіть рік вступу:")
    await state.set_state(Registration.enrollment_year)

@router.message(Registration.enrollment_year)
async def process_enrollment_year(message: Message, state: FSMContext):
    year_str = message.text.strip()
    if not year_str.isdigit():
        await message.answer("Рік вступу має бути числом. Введіть ще раз:")
        return

    year = int(year_str)
    current_year = datetime.today().year
    if year > current_year or year < 1975:
        await message.answer(f"Рік вступу має бути між 1975 та {current_year}. Введіть ще раз:")
        return

    await state.update_data(enrollment_year=year)
    await message.answer("Введіть рік випуску:")
    await state.set_state(Registration.graduation_year)


@router.message(Registration.graduation_year)
async def process_graduation_year(message: Message, state: FSMContext):
    year_str = message.text.strip()

    if not year_str.isdigit():
        await message.answer("Рік випуску має бути числом. Введіть ще раз:")
        return

    graduation_year = int(year_str)
    current_year = datetime.today().year
    if graduation_year > current_year or graduation_year < 1975:
        await message.answer(f"Рік випуску має бути між 1975 та {current_year}. Введіть ще раз:")
        return

    data = await state.get_data()
    enrollment_year = data.get("enrollment_year")

    if enrollment_year and graduation_year <= enrollment_year:
        await message.answer("Рік випуску не може бути меншим або рівним року вступу. Введіть ще раз:")
        return

    await state.update_data(graduation_year=graduation_year)
    await message.answer("Введіть назву вашого факультету (наприклад: ТЕФ, АПЕПС, ННІАТЕ):")
    await state.set_state(Registration.department_id)


@router.message(Registration.department_id)
async def process_department(message: Message, state: FSMContext):
    department = normalize_department(message.text)

    if not department:
        await message.answer(
            "На жаль, ми маємо інформацію лише для Теплоенергетичного факультету, кафедра цифрових
технологій в енергетиці.\n"
            "Будь ласка, введіть назву вашого факультету ще раз (наприклад: ТЕФ, АПЕПС, ННІАТЕ):"
        )
        return  # Не змінюємо стан, даємо ще одну спробу
    await state.update_data(department_id=department)
```

```
    await message.answer("Введіть код або частину назви вашої спеціальності:")
    await state.set_state(Registration.specialty_input)

@router.message(Registration.specialty_input)
async def process_specialty_input(message: Message, state: FSMContext):
    user_input = message.text.strip()
    results = search_specialty(user_input)

    if not results:
        await message.answer("Не знайдено спеціальність. Спробуйте ще раз або уточніть код/назву:")
        return

    if len(results) == 1:
        specialty = results[0]
        await state.update_data(specialty_id=specialty['id'])
        await message.answer(f"Знайдено спеціальність: {specialty['code']} - {specialty['name']}")
        await message.answer("Введіть вашу групу:")
        await state.set_state(Registration.group_name)
    else:
        reply = "Знайдено кілька спеціальностей:\n"
        for idx, s in enumerate(results, start=1):
            reply += f"{idx}. {s['code']} - {s['name']}\n"
        reply += "\nНапишіть номер відповідної спеціальності:"
        await state.update_data(specialty_options=results)
        await message.answer(reply)
        await state.set_state(Registration.specialty_select)

@router.message(Registration.specialty_select)
async def process_specialty_selection(message: Message, state: FSMContext):
    data = await state.get_data()
    options = data.get('specialty_options', [])

    try:
        choice = int(message.text.strip()) - 1
        specialty = options[choice]
        await state.update_data(specialty_id=specialty['id'])
        await message.answer(f"Обрано: {specialty['code']} - {specialty['name']}")
        await message.answer("Введіть вашу групу:")
        await state.set_state(Registration.group_name)
    except (ValueError, IndexError):
        await message.answer("Некоректний вибір. Спробуйте ще раз:")

@router.message(Registration.group_name)
async def process_group_name(message: Message, state: FSMContext):
    group = message.text.strip().upper()

    if not re.match(r"^[А-ЯA-Z]{2}-\d{2}$", group):
        await message.answer("Група повинна бути у форматі: 2 літери, тире, 2 цифри (наприклад: ТВ-12).
Введіть ще раз:")
        return

    await state.update_data(group_name=group)
    await message.answer("Введіть дату народження (ДД.ММ.РРРР):")
    await state.set_state(Registration.birth_date)

@router.message(Registration.ask_survey)
async def handle_survey_response(message: Message, state: FSMContext):
    text = message.text.lower().strip()

    if text in ["так", "так", "yes", "y", "ага", "да"]:
```

```
            await message.answer("🔗 Дякуємо! Перейдіть за посиланням на опитування:\nhttps://forms.gle/72mwaXVePPU5xVHK8")
        else:
            await message.answer("🙌 Добре, можливо пізніше!")

        # Завершення реєстраційного циклу
        await message.answer("🏠 Ви в головному меню:", reply_markup=user_main_menu_keyboard)
        await state.clear()

    @router.message(Registration.birth_date)
    async def process_birth_date(message: Message, state: FSMContext):
        birth_date_str = message.text.strip()

        try:
            # Перевірка формату та дійсності дати
            birth_date = datetime.strptime(birth_date_str, "%d.%m.%Y")

            # Перевірка віку: мінімум 16 років
            today = datetime.today()
            age = today.year - birth_date.year - ((today.month, today.day) < (birth_date.month, birth_date.day))

            if age < 16 or age > 95:
                await message.answer("Вам має бути щонайменше 16 років та, на жаль, не більше 95 для реєстрації. Введіть іншу дату народження:")
                return

        except ValueError:
            await message.answer("Неправильний формат дати. Введіть дату народження у форматі ДД.ММ.РРРР:")
            return

        await state.update_data(birth_date=birth_date_str)
        data = await state.get_data()

        conn = get_connection()
        cursor = conn.cursor()

        try:
            cursor.execute('''
                INSERT INTO users (
                    telegram_id, full_name, phone_number, old_phone_number, enrollment_year,
                    graduation_year, department_id, specialty_id, group_name, birth_date
                ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
            ''', (
                str(message.from_user.id),
                data['full_name'],
                data['phone_number'],
                data['old_phone_number'],
                data['enrollment_year'],
                data['graduation_year'],
                data['department_id'],
                data['specialty_id'],
                data['group_name'],
                data['birth_date']
            ))
            conn.commit()
            await message.answer("☑️ Реєстрація успішно завершена!")
            await message.answer("📝 Хочете пройти коротке опитування? (так/ні)")
            await state.set_state(Registration.ask_survey)
```

```
    except sqlite3.IntegrityError:
        await message.answer("⚠️ Ви вже зареєстровані.")
    finally:
        conn.close()
```

register_start.py:

```
from aiogram import Router
from aiogram.types import Message
from aiogram.filters import Command
from ...utils.keyboard import main_menu_keyboard
from aiogram.types import CallbackQuery
from aiogram.fsm.context import FSMContext
from ...handlers.login.login_user import UserLogin
from ...handlers.login.login_admin import AdminLogin

router = Router()

@router.message(Command("start", "menu"))
async def show_main_menu(message: Message):
    text = (
        "👋 Вітаю! Я бот для реєстрації випускників кафедри цифрових технологій в енергетиці.\n\n"
        "Оберіть дію з меню нижче або використовуйте команди:\n"
        "/register_user – Зареєструватися як користувач\n"
        "/register_admin – Зареєструватися як адміністратор\n"
        "/help – Допомога та інформація\n"
        "/login_user - Увійти як користувач\n"
        "/login_admin - Увійти як адміністратор\n"
    )
    await message.answer(text, reply_markup=main_menu_keyboard)

@router.message(Command("help"))
async def show_help(message: Message):
    help_text = (
        "ℹ Допомога:\n"
        "- Якщо ви студент або випускник – використовуйте /register_user, /login_user або кнопку.\n"
        "- Якщо ви адміністрація – використовуйте /register_admin, /login_admin або кнопку.\n"
        "- Для повернення до меню – /menu.\n\n"
        "Підтримка: alina.seikauskaite3@gmail.com"
    )
    await message.answer(help_text)

@router.callback_query(lambda c: c.data == 'login_user')
async def callback_login_user(callback_query: CallbackQuery, state: FSMContext):
    await callback_query.message.answer("Ви обрали вхід як користувач.\nВведіть ваш номер телефону:")
    try:
        await state.set_state(UserLogin.phone_number)
    except Exception:
        await callback_query.message.answer("⚠️ Сталася помилка при встановленні стану. Спробуйте ще
раз.")

@router.callback_query(lambda c: c.data == 'login_admin')
async def callback_login_admin(callback_query: CallbackQuery, state: FSMContext):
    await callback_query.message.answer("Ви обрали вхід як адміністратор.\nВведіть ваш номер телефону:")
    try:
        await state.set_state(AdminLogin.phone_number)
    except Exception:
        await callback_query.message.answer("⚠️ Сталася помилка при встановленні стану. Спробуйте ще
раз.")
```