МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«ІНФОРМАЦІЙНА СИСТЕМА ДЛЯ ЗДОБУВАЧІВ ЗВО НА ОСНОВІ ТЕЛЕГРАМ-БОТА»

Рівень «бакалавр»
Галузь знань 12 «Інформаційні технології»,
спеціальність 121 «Інженерія програмного забезпечення»
Виконав здобувач 4 курсу
Сорокін Владислав Сергійович

Керівник к.т.н., доцент
Манаков Сергій Юрійович

Рецензент к.пед.н., доцент
Логінова Наталія Іванівна

Одеса – 2024

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»
Факультет кібербезпеки та інформаційних технологій
Кафедра інформаційних технологій
Галузь знань: 12 Інформаційні технології
Спеціальність: 121 Інженерія програмного забезпечення
Назва освітньої програми: Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри інформаційних технологій
доцент Наталія Логінова _____
«____»_____ 20___ року

З А В Д А Н Н Я
на кваліфікаційну (бакалаврську) роботу
здобувачу Сорокіну Владиславу Сергійовичу

1. Тема роботи: «Інформаційна система для здобувачів ЗВО на основі телеграм-бота»
   Керівник роботи: к.т.н, доцент Манаков Сергій Юрійович
   затверджені наказом НУ «ОЮА» № 809-06 від «02» квітня 2024 року.
2. Строк подання здобувачем роботи «20» травня 2024 року.
3. Дата видачі завдання «15» вересня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області та вибір засобів розробки	01.12.2023	
2	Проєктування інформаційної системи на основі телеграм-бота	26.01.2024	
3	Розробка програмного продукту на основі телеграм-бота	15.03.2024	
4	Тестування системи	26.04.2024	
5	Оформлення звітної частини	19.05.2024	

Здобувач _____
(підпис) (прізвище та ініціали)

Керівник роботи _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сорокін В. С. Інформаційна система для здобувачів ЗВО на основі телеграм-бота. – Кваліфікаційна робота бакалавра за спеціальністю 121 «Інженерія програмного забезпечення», освітньою програмою «Інженерія програмного забезпечення».

Кваліфікаційна робота викладена на 55 сторінках загального тексту, містить 3 розділи, 28 рисунків, 2 таблиць, 1 додаток, 16 використаних джерел.

Метою роботи є розробка інформаційної системи для здобувачів вищої освіти на основі телеграм-бота.

Об’єктом дослідження є процеси автоматизації інформаційного забезпечення студентів.

Предмет дослідження – методи та засоби розробки телеграм-ботів для інформаційного забезпечення здобувачів вищої освіти.

У кваліфікаційній роботі розроблено Telegram-бот, який надає студентам доступ до розкладу занять, перегляду завдань, оновлення статусу, взаємодії з соціальними мережами навчального закладу, а також інформування через інтеграцію з базою даних. Проведено аналіз подібних Telegram-ботів, на основі чого визначено ключові вимоги до розробки власного рішення. Проведено функціональне тестування розробленого бота, що підтвердило його коректну роботу та відповідність вимогам користувачів.

У першому розділі проведено аналіз предметної області та вибір засобів розробки. У другому розділі здійснено проєктування інформаційної системи на основі телеграм-бота. У третьому розділі відбувається демонстрація функціональності бота. За результатами роботи розроблено висновки щодо виконаних та поставлених задач.

Ключові слова: SQLite, телеграм-бот, база даних, Activity Diagram, Python, Telegram.

ABSTRACT

Sorokin V. S. Information system for university applicants based on a telegram bot – Bachelor's thesis in specialty 121 "Software Engineering", educational program "Software Engineering".

The qualification work is set out on 55 total pages, contains 3 chapters, 28 figures, 2 tables, 1 appendix, 16 references.

The purpose of the work is to develop an information system for higher education students based on bot telegrams.

The object of research is the processes of automation of information support for students.

The subject of research is methods and tools for developing bot telegrams for information support of higher education students.

In the qualification work, a Telegram bot was developed that provides students with access to class schedules, viewing assignments, status updates, interaction with social networks of the educational institution, as well as information through integration with the database. An analysis of similar Telegram bots was conducted, based on which the key requirements for developing our own solution were identified. Functional testing of the developed bot was conducted, which confirmed its correct operation and compliance with user requirements.

The first section analyzes the subject area and selects development tools. The second section describes the design of an information system based on the Telegram bot. The third section demonstrates the functionality of the bot. Based on the results of the work, conclusions are drawn on the accomplished and set tasks.

Keywords: SQLite, Telegram bot, database, Activity Diagram, Python, Telegram.

ЗМІСТ

ВСТУП

У цій роботі представлено інформаційну систему для здобувачів ЗВО, розроблену на основі телеграм-бота. Система пропонує достатньо широкий спектр функцій, спрямованих на покращення комунікації та організації навчання студентів.

У зв'язку з обставинами сьогодення та ситуацією в країні, велика кількість студентів змушена перебувати закордоном. Це створює нові виклики для організації навчального процесу та підтримки зв'язку зі студентами. Інформаційна система на основі телеграм-бота може стати цінним інструментом для вирішення проблем.

Метою роботи є розробка інформаційної системи для здобувачів вищої освіти на основі телеграм-бота.

Об’єктом дослідження є процеси автоматизації інформаційного забезпечення студентів.

Предмет дослідження – методи та засоби розробки телеграм-ботів для інформаційного забезпечення здобувачів вищої освіти.

Даний бот сприятиме покращенню комунікації та організації навчального процесу. Деканат зможе використовувати систему для надсилання повідомлень студентам, пошуку студентів та збору інформації. Студенти зможуть використовувати систему для перегляду розкладу та засобів зв’язку з деканатом, зміни статусу своєї присутності, надання особистих даних, отримання SMS-повідомлень від працівників університету.

Система на основі телеграм-боту проста у використанні, ефективна та пропонує широкий спектр функцій, які можуть бути корисними як для студентів, так і для деканату.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ЗАСОБІВ РОЗРОБКИ

1.1 Специфіка розробки інформаційної системи на основі телеграм-бота

Інформаційна система – це сукупність організаційних і технічних засобів для збереження та обробки інформації з метою забезпечення інформаційних потреб користувачів.

Розробка інформаційної системи для здобувачів вищої освіти на основі телеграм-бота вимагає комплексного підходу та використання специфічних технологій.

1.2 Аналіз наявних аналогів

Для аналізу було обрано три програмних продукти, які мають схожі функції та цільову аудиторію з пропонованою інформаційною системою на основі Telegram-бота:

UniverBot – це багатофункціональний бот Telegram, який використовується студентами університетів для отримання інформації про розклад занять, оцінки, завдання, віртуальні класи та інші послуги.

Недоліки: складність в налаштуванні та помилки в інтеграції.

Переваги: широкий спектр функцій, інтеграція з університетськими системами, зручний інтерфейс.

EdBot – це бот Telegram, який використовується викладачами для створення та проведення вікторин, опитувань та тестів, а також для спілкування з студентами.

Недоліки: обмежена функціональність, орієнтований на викладачів.

Переваги: простота використання, гнучкість налаштування, висока ефективність.

CampusBot**:** – це бот Telegram, який використовується університетами для надання студентам інформації про події, новини, послуги та інші ресурси.

Недоліки: невеликий спектр функцій, не інтегрований з університетськими системами.

Переваги: зручний доступ до інформації, можливість персоналізації.

На рис. 1.1 показані результати проведеного SWOT-аналізу конкурентів.

За результатами аналізу наявних аналогів програмного забезпечення, можна зробити висновок про доцільність та затребуваність розробки програмного продукту "NU OLA" для студентів на основі телеграм-бота. Даний проєкт має на меті стати комплексним рішенням, який задовільнить потреби студентів у зручному та ефективному доступі до інформації про навчання.

Даний бот буде надавати широкий спектр функцій, буде зручний та інтуїтивно зрозумілий інтерфейс що дозволить студентам користуватися ним навіть без попереднього досвіду використання телеграм-ботів.

АНАЛОГ	СИЛЬНІ СТОРОНИ	СЛАБКІ СТОРОНИ	МОЖЛИВОСТІ
UNIVERBOT	ШИРОКИЙ СПЕКТР ФУНКЦІЙ, ІНТЕГРАЦІЯ З УНІВЕРСИТЕТСЬКИМИ СИСТЕМАМИ, ЗРУЧНИЙ ІНТЕРФЕЙС	СКЛАДНІСТЬ НАЛАШТУВАННЯ, МОЖЛИВІ ПОМИЛКИ ПРИ ІНТЕГРАЦІЇ	ЗРОСТАННЯ ПОПУЛЯРНОСТІ TELEGRAM-БОТІВ, ІНТЕГРАЦІЯ З ІНШИМИ ПЛАТФОРМАМИ
EDBOT	ПРОСТОТА ВИКОРИСТАННЯ, ГНУЧКІСТЬ НАЛАШТУВАННЯ, ВИСОКА ЕФЕКТИВНІСТЬ	ОБМЕЖЕНА ФУНКЦІОНАЛЬНІСТЬ, ОРІЄНТОВАНИЙ НА ВИКЛАДАЧІВ	ЗРОСТАННЯ ПОПУЛЯРНОСТІ ОНЛАЙН-НАВЧАННЯ, ІНТЕГРАЦІЯ З СИСТЕМАМИ LMS
CAMPUSBOT	ЗРУЧНИЙ ДОСТУП ДО ІНФОРМАЦІЇ, МОЖЛИВІСТЬ ПЕРСОНАЛІЗАЦІЇ	НЕВЕЛИКИЙ СПЕКТР ФУНКЦІЙ, НЕ ІНТЕГРОВАНИЙ З УНІВЕРСИТЕТСЬКИМИ СИСТЕМАМИ	ЗРОСТАННЯ ПОПУЛЯРНОСТІ МОБІЛЬНИХ ДОДАТКІВ, ІНТЕГРАЦІЯ З СОЦІАЛЬНИМИ МЕРЕЖАМИ

Рисунок 1.1 – SWOT-аналіз конкурентів

"NU OLA" буде побудований з урахуванням можливості майбутнього розширення функціоналу, щоб задовольняти потребам студентів та працівникам деканату.

1.3 Функціональні вимоги до бота

Функціональна вимога – це формулювання того, як система повинна поводитися. Він визначає, що повинна робити система, щоб задовольнити потреби

або очікування користувача. Функціональні вимоги можна розглядати як функції, які виявляє користувач. Вони відрізняються від нефункціональних вимог, які визначають, як система повинна працювати всередині (наприклад, продуктивність, безпека тощо).

Функціональні вимоги складаються з двох частин: функції та поведінки. Функція – це те, що виконує система (наприклад, «розрахувати податок з продажу»). Поведінка залежить від того, як система це робить (наприклад, «Система розраховує податок із продажу, помноживши ціну покупки на ставку податку») [3]. В таблиці 1.1 можна наглядно переглянути, що таке функція і поведінка.

Таблиця 1.1 – Функціональні вимоги

Функція	Поведінка
Реєстрація	Система дозволить користувачеві пройти реєстрацію за його особистими даними
Перегляд особистої інформації	Система надає доступ користувачеві до особистої інформації, яка була введена при реєстрації
Редагування особистої інформації	Система дозволяє редагувати особисту інформацію
Перегляд контактів деканату	Система відправляє контакти користувачу
Встановлення та зміна статусу	Система дає змогу користувачеві встановити свій статус та за потреби його змінити
Перегляд розкладу	Система дає змогу переглянути розклад, який був завантажений адміністратором

Звідси можна зробити висновок, що функціональними можливостями даної системи є:

1) Реєстрація. Студенти можуть зареєструватися в системі, використовуючи свої особисті дані.

2) Зміна статусу. Студенти можуть змінювати свій статус, щоб повідомити про свою доступність для навчання, наприклад, "Вдома", "Хворий", "Працюю", "Закордоном" тощо. Ця інформація може бути корисною для деканату.

3) Перегляд розкладу. Здобувачі освіти можуть переглядати свій розклад занять, включаючи час, місце проведення та назву курсу.

4) SMS-повідомлення. Деканат може надсилати SMS-повідомлення студентам, щоб інформувати їх про важливі події, зміни в розкладі тощо.

5) Пошук студентів. Працівник деканату може використовувати систему для пошуку студентів за прізвищем. Це може бути корисно для швидкого зв'язку або перегляду необхідної інформації.

1.4 Інтеграція з базою даних

Для зберігання та керування інформацією було використано базу даних SQLite3. Це дозволило забезпечити ефективне зберігання даних про користувачів. Використання SQLite3 забезпечило швидкий доступ до даних через Python.

SQLite – полегшена реляційна система керування базами даних. Втілена у вигляді бібліотеки, де реалізовано багато зі стандарту SQL-92. Початковий код SQLite поширюється як суспільне надбання (англ. public domain), тобто може використовуватися без обмежень та безоплатно з будь-якою метою. Фінансову підтримку розробників SQLite здійснює спеціально створений консорціум, до якого входять такі компанії, як Adobe, Oracle, Mozilla, Nokia, Bentley[en] і Bloomberg.

Створення та обслуговування БД можуть здійснюватись через текстову консоль SQL-командами або через спеціальні інструменти, у тому числі – з графічним інтерфейсом користувача [4].

Для роботи з даними в SQLite використовується мова SQL (Structured Query Language). За допомогою SQL-запитів можна виконувати операції додавання, зміни, видалення та вибору даних з таблиць. База даних підтримує транзакції, що дозволяють виконувати групу операцій як єдине ціле. Транзакції забезпечують цілісність даних і можуть бути скасовані, якщо виникає помилка. Вся база даних SQLite зберігається в одному файлі на диску. Він має розширення .sqlite або .db і містить всі таблиці, індекси, схему та дані. Його можна копіювати, переміщати та резервувати, як звичайний файл. База містить одну чи кілька таблиць, які використовують для зберігання даних. Вони визначаються схемою, яка включає імена стовпців, типи даних та інші обмеження [5].

1.5 Вибір технологічного стека

При розробці інформаційної системи для студентів було вирішено використовувати платформу Telegram для зручного доступу до інформації та взаємодії з користувачами. Телеграм-боти надають можливість автоматизованої обробки запитів користувачів і забезпечують миттєвий зв'язок.

1.5.1 Середовище розробки

В якості інтегрованого середовища розробки було використано Visual Studio Code. Це потужне середовище, яке можна використовувати для виконання всього циклу розробки в одному місці. Це комплексне інтегроване середовище розробки (IDE), яке можна використовувати для запису, редагування, налагодження та складання коду, а потім розгортання програми. Крім редагування та налагодження коду Visual Studio включає компілятори, засоби завершення коду, керування версіями, розширення та багато іншого, щоб покращити кожен етап процесу розробки програмного забезпечення [2].

Вибір Visual Studio Code обумовлений такими факторами:

Безкоштовність: VS Code є безкоштовним редактором коду, що робить його доступним для будь-якого розробника.

Підтримка мови Python: Visual Studio Code має вбудовану підтримку мови Python, що робить його зручним інструментом для розробки Telegram-ботів.

Розширення: має безліч розширень, які можуть допомогти розробникам у їхній роботі, наприклад, розширення для форматування коду.

Для інформаційної системи була вибрана база даних SQLite.

Вибір цієї бази даних залежить від конкретних вимог та обмежень проекту. Для цього також треба знати більше про плюси та мінуси СКБД.

Переваги:

– це дуже легка база даних, яка не потребує окремого сервера. Вона працює швидко та ефективно, особливо на системах з обмеженими ресурсами;

– SQLite відносно проста у використанні та не вимагає складних налаштувань. Взаємодіяти з базою даних можна, використовуючи стандартні мови програмування і мову запитів SQL;

– СКБД можна вбудовувати безпосередньо в програмне забезпечення, що полегшує розповсюдження та розгортання додатків;

– всі дані зберігаються в одному файлі бази даних, що робить резервне копіювання і перенесення простою справою;

–SQLite підтримує транзакції, і це дозволяє забезпечити цілісність та безпеку даних;

– СКБД має багато різних оболонок і бібліотек, які роблять SQLite доступним для багатьох мов програмування: Python, Java, C# і багато інших [6].

Нижче на рис. 1.2 наведено приклад простого SQL запиту для створення таблиці.

```
CREATE TABLE Users (
    ID INTEGER PRIMARY KEY,
    Name TEXT,
    Age INTEGER
);
```

Рисунок 1.2 – SQL запит для створення таблиці

1.5.2 Мова програмування та вибір бібліотеки

Мовою програмування для реалізації бота було обрано Python через його простоту та широкі можливості для роботи з API Telegram.

Python – це мова програмування високого рівня загального призначення. Його філософія дизайну наголошує на читабельності коду з використанням значних відступів. Він підтримує кілька парадигм програмування, включаючи

структуроване (зокрема процедурне), об’єктно-орієнтоване та функціональне програмування [1].

Python було обрано для розробки Telegram-бота з вагомих причин. По-перше, мова програмування Python відома своєю великою бібліотекою стандартних модулів, що дозволяє ефективно вирішувати різноманітні завдання. По-друге, Python має безліч сторонніх бібліотек, які розширюють його функціональні можливості і полегшують розробку. Також ця мова програмування вважається однією з найбільш оптимальних мов програмування для створення ботів, завдяки своїй простоті та читабельності коду.

Для розробки Telegram-бота використовувалась бібліотека TeleBot.

TeleBot – це простий та зручний фреймворк для створення Telegram-ботів на Python. Він дозволяє швидко розробляти прості боти, але має обмежений функціонал порівняно з іншими фреймворками [7].

Бібліотека є простотою та зручною у використанні. Крім того, вона має широкий спектр функцій, які дозволяють створювати ботів з різноманітними можливостями. Має детальну і чітку документацію.

1.4.3 Система керування базами даних

Система керування базами даних (СКБД, англ. Database Management System, DBMS) – набір взаємопов'язаних даних (база даних) і програм для доступу до цих даних. Надає можливості створення, збереження, оновлення та пошуку інформації в базах даних (БД) з контролем доступу до даних [8].

Є трирівнева система організації СКБД ANSI-SPARC, при якій є незалежний рівень для ізоляції програми від особливостей подання даних на нижчому рівні.

Рівні:

Зовнішній – подання БД з точки зору користувача.

Концептуальний – узагальнене подання БД, описує які дані зберігаються в БД і зв'язки між ними. Підтримує зовнішні подання, підтримується внутрішнім рівнем.

Внутрішній – фізичне подання БД в комп'ютері.

Логічна незалежність – повна захищеність зовнішніх моделей від змін, що вносяться в концептуальну модель.

Фізична незалежність – захищеність концептуальної моделі від змін, які вносяться у внутрішню модель [9].

SQLite – полегшена реляційна система керування базами даних. Втілена у вигляді бібліотеки, де реалізовано багато зі стандарту SQL-92. Початковий код SQLite поширюється як суспільне надбання (англ. public domain), тобто може використовуватися без обмежень та безоплатно з будь-якою метою [10].

Для зберігання даних Telegram-бота вирішено було використовувати систему керування базами даних SQLite3.

SQLite3 є простою у використанні системою керування базами даних. SQLite3 не потребує встановлення сервера бази даних, що робить її зручним рішенням для розробки Telegram-ботів.

Дана БД є швидкою системою керування базами даних, що робить її оптимальною у використанні для Telegram-ботів.

1.6 Висновки до розділу

Розробка Telegram-бота на основі інформаційної системи для здобувачів вищої освіти є актуальним та перспективним завданням. Використання Telegram-платформи та сучасних технологій дозволить створити зручний та ефективний інструмент для доступу до інформації та взаємодії з користувачами.

Для реалізації проекту було обрано технологічний стек, який включає:

Мову програмування**:** Python

Інтегроване середовище розробки: Visual Studio Code

Бібліотека для розробки Telegram-ботів: TeleBot

Система керування базами даних: SQLite3

Функціональні вимоги до Telegram-бота ґрунтуються на потребах користувачів та включають:

1) Отримання інформації про розклад занять, події, новини та інші послуги університету.

2) Можливість інформування працівників деканату про свій стан здоров’я та місцезнаходження.

3) Можливість перегляду інформації про студентів.

4) Пошук осіб в системі за індивідуальними даними.

Вибір засобів розробки обумовлений їх простотою, зручністю використання, широкими можливостями та відповідністю потребам проекту.

Реалізація даного проекту дозволить:

1) Забезпечити зручний та ефективний доступ до інформації для здобувачів вищої освіти.

2) Покращити комунікацію між студентами та працівниками деканату.

3) Автоматизувати ряд завдань, пов'язаних з навчальним процесом.

В подальшому планується розширити функціонал Telegram-бота та інтегрувати його з іншими системами університету.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ТЕЛЕГРАМ-БОТА

2.1 Проєктування бази даних «Students»

З самого початку розробки бота ми ґрунтувалися на принципах мінімалізму, вибираючи дизайн бази даних з однією таблицею. Дане рішення зменшує навантаження на систему, роблячи роботу бота швидшою, полегшує розуміння та обслуговування бази даних, роблячи її більш гнучкою та масштабованою.

Також важливим є визначити основні етапи проєктування бази даних:

– Концептуальне проєктування
– Логічне проєктування
– Фізичне проєктування

Мета етапу концептуального проектування – створення концептуальної моделі даних виходячи з уявлень користувачів про предметну область. Для її досягнення виконується низка послідовних процедур:

1) Визначення сутностей та їхнє документування. Для ідентифікації сутностей визначаються об'єкти, які існують незалежно від інших. Такі об'єкти є суттю. Кожній сутності надається осмислене ім'я, зрозуміле користувачам. Імена та описи сутностей заносяться до словника даних. Якщо можливо, то встановлюється очікувана кількість примірників кожної сутності.

2) Визначення зв'язків між сутностями та їх документування. Визначаються ті зв'язки між сутностями, які необхідні для задоволення вимог до проєкту бази даних. Встановлюється тип кожної їх. Виявляється клас власності сутностей. Зв'язкам надаються осмислені імена, виражені дієсловами. Розгорнутий опис кожного зв'язку із зазначенням її типу та класу належності сутностей, що беруть участь у зв'язку, заноситься до словника даних.

3) Створення ER-моделі предметної області. Для подання сутностей та зв'язків між ними використовуються ER-діаграми. На їх основі створюється єдиний наочний образ предметної області, що моделюється – ER-модель предметної області.

4) Визначення атрибутів та їх документування. Виявляються всі атрибути, що описують сутність створеної моделі ER. Кожному атрибуту надається осмислене ім'я, зрозуміле користувачам.

Мета етапу логічного проектування – перетворення концептуальної моделі на основі обраної моделі даних на логічну модель, не залежну від особливостей використовуваної надалі СКБД для фізичної реалізації бази даних. Для її досягнення виконуються такі процедури:

1) Вибір моделі даних. Найчастіше вибирається реляційна модель даних у зв'язку з наочністю табличного подання даних та зручності роботи з ними.

2) Визначення набору таблиць, виходячи з ER-моделі та їх документування. Для кожної сутності ER-моделі створюється таблиця. Ім'я сутності – ім'я таблиці. Встановлюються зв'язки між таблицями за допомогою механізму первинних та зовнішніх ключів. Структури таблиць та встановлені зв'язки між ними документуються.

3) Нормалізація таблиць. Для правильного виконання нормалізації проектувальник повинен глибоко вивчити семантику та особливості використання даних. На цьому кроці він перевіряє коректність структури таблиць, створених на попередньому кроці через застосування до них процедури нормалізації.

4) Перевірка логічної моделі даних щодо можливості виконання всіх транзакцій, передбачених користувачами.

5) Визначення вимог підтримки цілісності даних та їх документування. Ці вимоги є обмеженнями, які запроваджуються, щоб запобігти розміщенню в базі даних суперечливих даних. На цьому етапі питання цілісності даних висвітлюються безвідносно до конкретних аспектів її реалізації.

5) Створення остаточного варіанта логічної моделі даних та обговорення його з користувачами. На цьому кроці готується остаточний варіант ER-моделі, що представляє логічну модель даних. Сама модель та оновлена документація, включно зі словником даних та реляційною схемою зв'язку таблиць, надаються для перегляду та аналізу користувачам, які повинні переконатися, що вона точно відображає предметну область. Саме така діаграма наведена нижче на рис. 2.2.1.

Мета етапу фізичного проєктування – опис конкретної реалізації бази даних, що розміщується у зовнішній пам'яті комп'ютера. Це опис структури зберігання даних та ефективних методів доступу до даних бази. У логічному проєктуванні відповідають на питання – що треба зробити, а у фізичному – обирається спосіб, як це зробити. Процедури фізичного проектування такі:

1) Проєктування таблиць бази даних засобами обраної СКБД. Здійснюється вибір реляційної СКБД, яка використовуватиметься для створення бази даних, що розміщується на машинних носіях. Глибоко вивчаються її функціональні можливості щодо проектування таблиць. Потім виконується проектування таблиць і схеми їхнього зв'язку серед СКБД. Підготовлений проект бази даних описується у документації, що супроводжує.

2) Реалізація бізнес-правил у середовищі обраної СКБД. Оновлення інформації в таблицях може бути обмежене бізнес-правилами. Спосіб реалізації залежить від обраної СКБД. Одні системи реалізації вимог предметної області пропонують більше можливостей, інші – менше. У деяких системах взагалі немає підтримки реалізації бізнес-правил. У такому разі розробляються додатки для реалізації їх обмежень.

3) Проєктування фізичної організації бази даних. На цьому кроці обирається найкраща файлова організація для таблиць. Виявляються транзакції, які будуть виконуватися в базі даних, що проєктується, і виділяються найважливіші з них. Аналізується пропускна спроможність транзакцій: кількість транзакцій, які можна обробити за вказаний інтервал часу, і час відповіді – проміжок часу, необхідний виконання однієї транзакції. Прагнуть до підвищення пропускної спроможності транзакцій і зменшення часу відповіді.

4) Розробка стратегії захисту бази даних. База даних є цінним корпоративним ресурсом, і організації її захисту приділяється велика увага. Для цього проєктувальники повинні мати повне та ясне уявлення про всі засоби захисту, що надаються обраною СКБД.

5) Організація моніторингу функціонування бази даних та її налаштування. Після створення фізичного проєкту бази даних організується

безперервне стеження за її функціонуванням. Отримані відомості про рівень продуктивності бази даних використовуються для її налаштування. Для цього залучаються засоби обраної СКБД [11].

Опис таблиці "Students":

– Первинний ключ: user_id

Атрибути:

– name: Ім'я студента
– surname: Прізвище студента
– birthday: Дата народження студента
– department: Назва факультету
– studentGroup: Назва групи
– course: Курс навчання
– phoneNumber: Номер телефону
– email: Адреса електронної пошти
– status: Статус

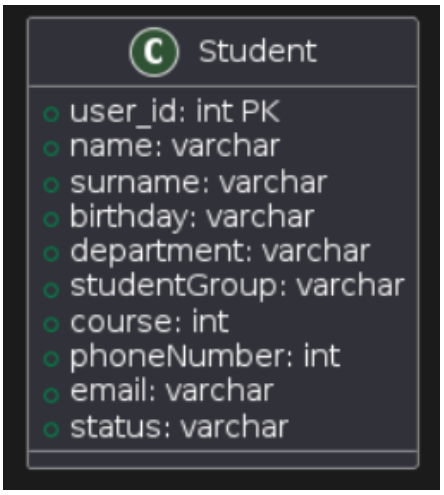

Рисунок 2.1 – Сутність Student бази даних

2.2 Моделювання програмної системи

UML (Unified Modeling Language, уніфікована мова моделювання) — це стандарт для опису та візуалізації систем програмного забезпечення. UML

використовується розробниками програмного забезпечення для створення діаграм, які описують різні аспекти системи, такі як її структура, поведінка та взаємодія з користувачами [12].

Діаграми прецедентів (Use Cases diagrams) – це тип UML-діаграм, які використовуються для опису функціональних можливостей системи та акторів, які з нею взаємодіють. Діаграми прецедентів допомагають розробникам зрозуміти, що система повинна робити, і як користувачі зможуть з нею взаємодіяти [13].

Основні елементи діаграми прецедентів:

Актори: Це люди, пристрої або інші об’єкти, які взаємодіють з системою.

Прецеденти: це функціональні можливості системи, які надають користувачам певну цінність.

Зв'язки: це зв'язки між акторами та прецедентами.

Діаграми прецедентів можуть бути корисними для визначення вимог до системи, планування розробки та документування системи [14].

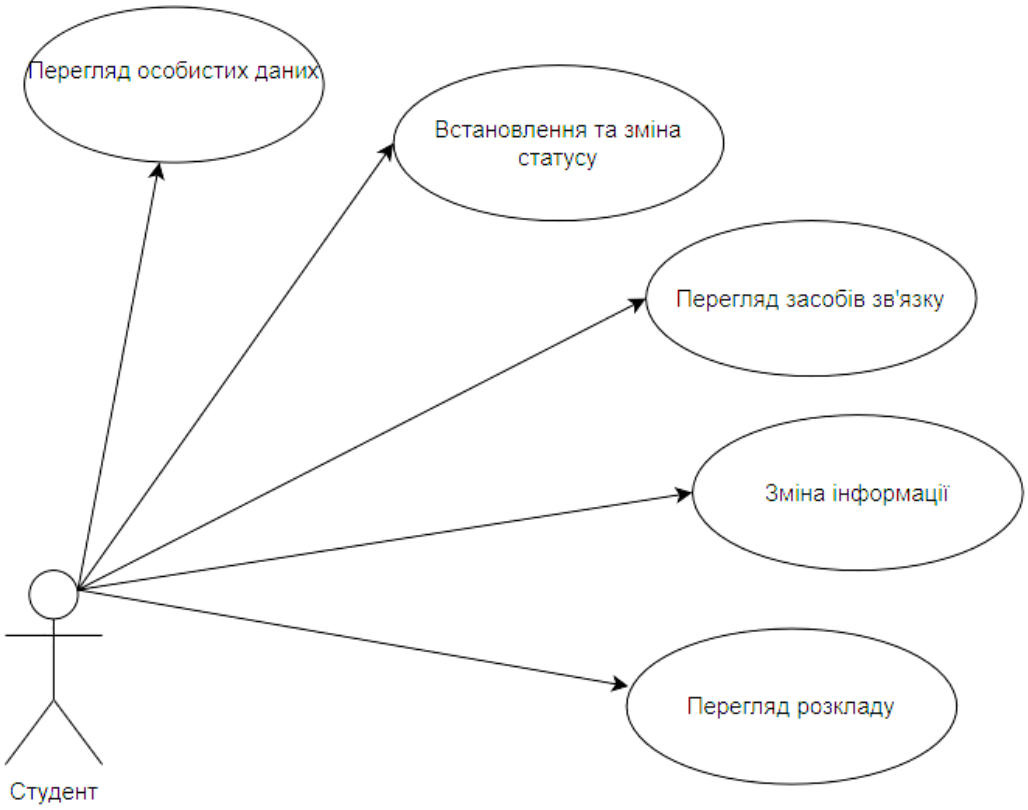

Рисунок 2.2 – Use Cases Diagram для студента

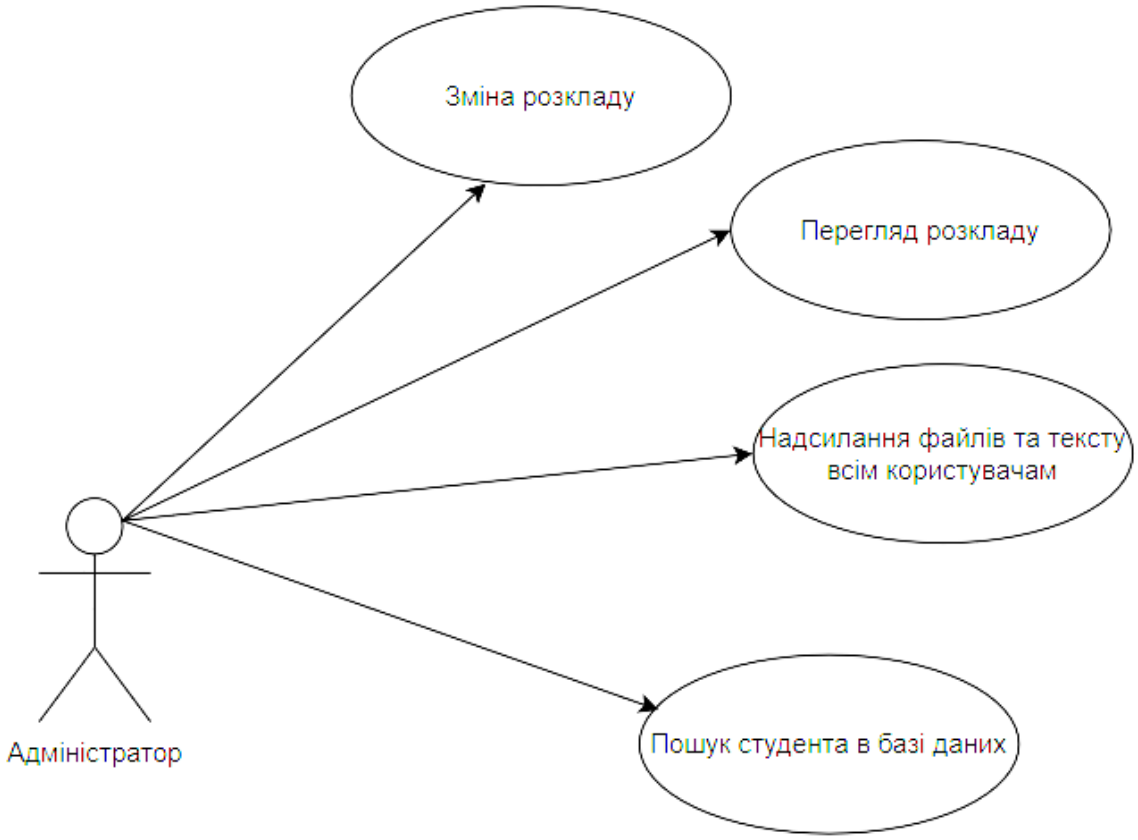

Рисунок 2.3 – Use Cases Diagram для адміністратора

На рис. 2.4 показана діаграма активності. На діаграмі показано процес зміни статусу користувача в телеграм-боті.

Activity Diagram – це блок-схема, яка показує, як одна діяльність призводить до іншої. Цю дію можна назвати системною операцією. Одна операція веде до наступної в потоці керування. Цей потік може бути паралельним, одночасним або розгалуженим. Діаграми діяльності використовують багато функцій, таких як fork, join тощо, щоб справлятися з усіма типами керування потоком. Подібно до інших діаграм, діаграми діяльності служать схожим фундаментальним цілям. Він фіксує динамічну поведінку системи [15].

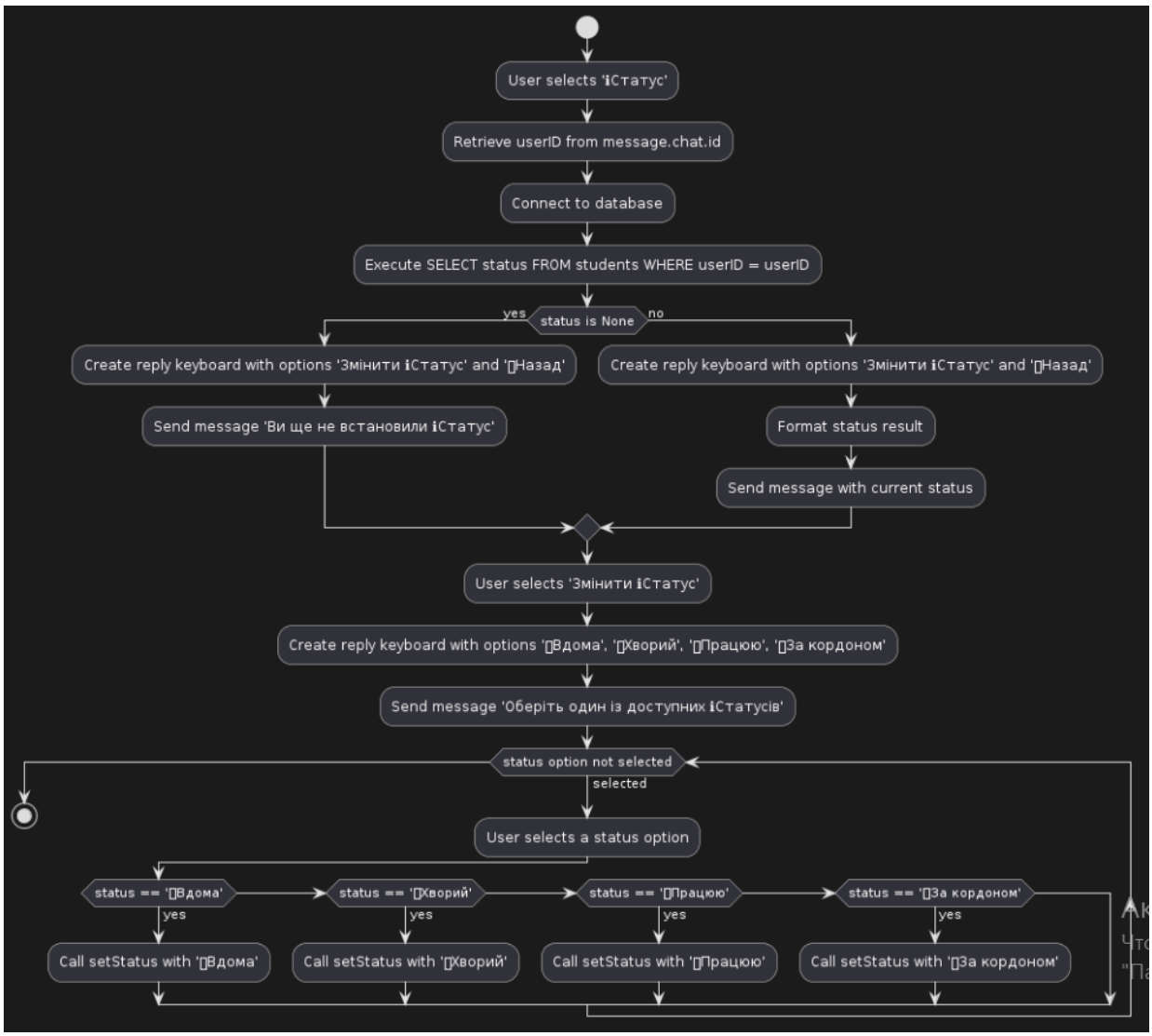

Рисунок 2.4 – Activity Diagram для зміни статусу користувача

Діаграма на рис. 2.5 відображає процес створення та відправки повідомлення з кнопками для соціальних мереж.

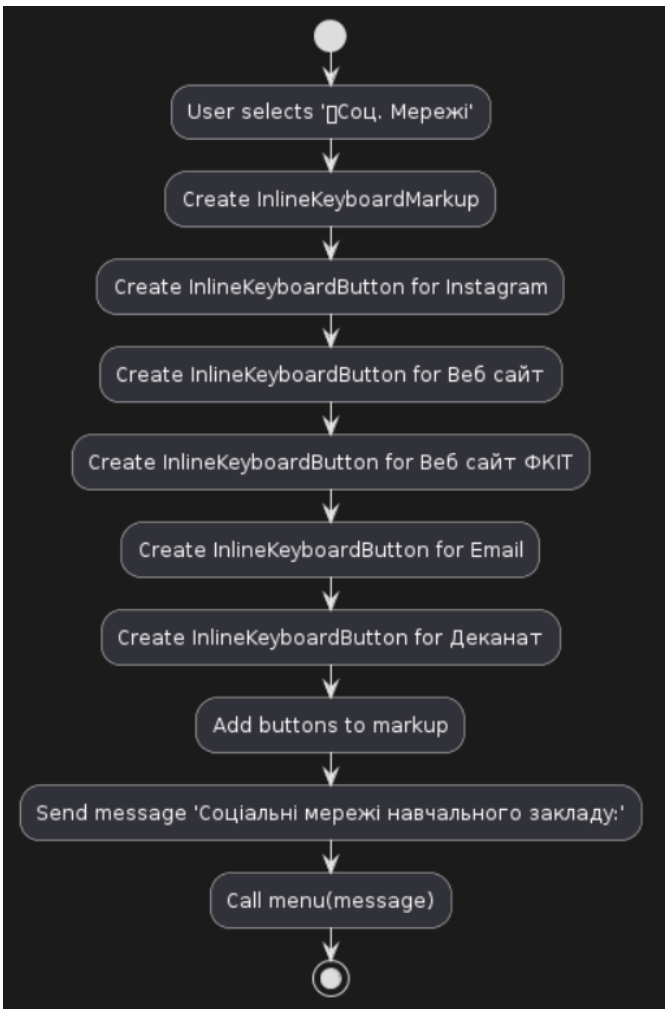

Рисунок 2.5 – Activity Diagram для перегляду соц. мереж деканату

Діаграма на рис. 2.6 відображає процес вибору курсу та надсилання відповідного розкладу, якщо файл знайдено.

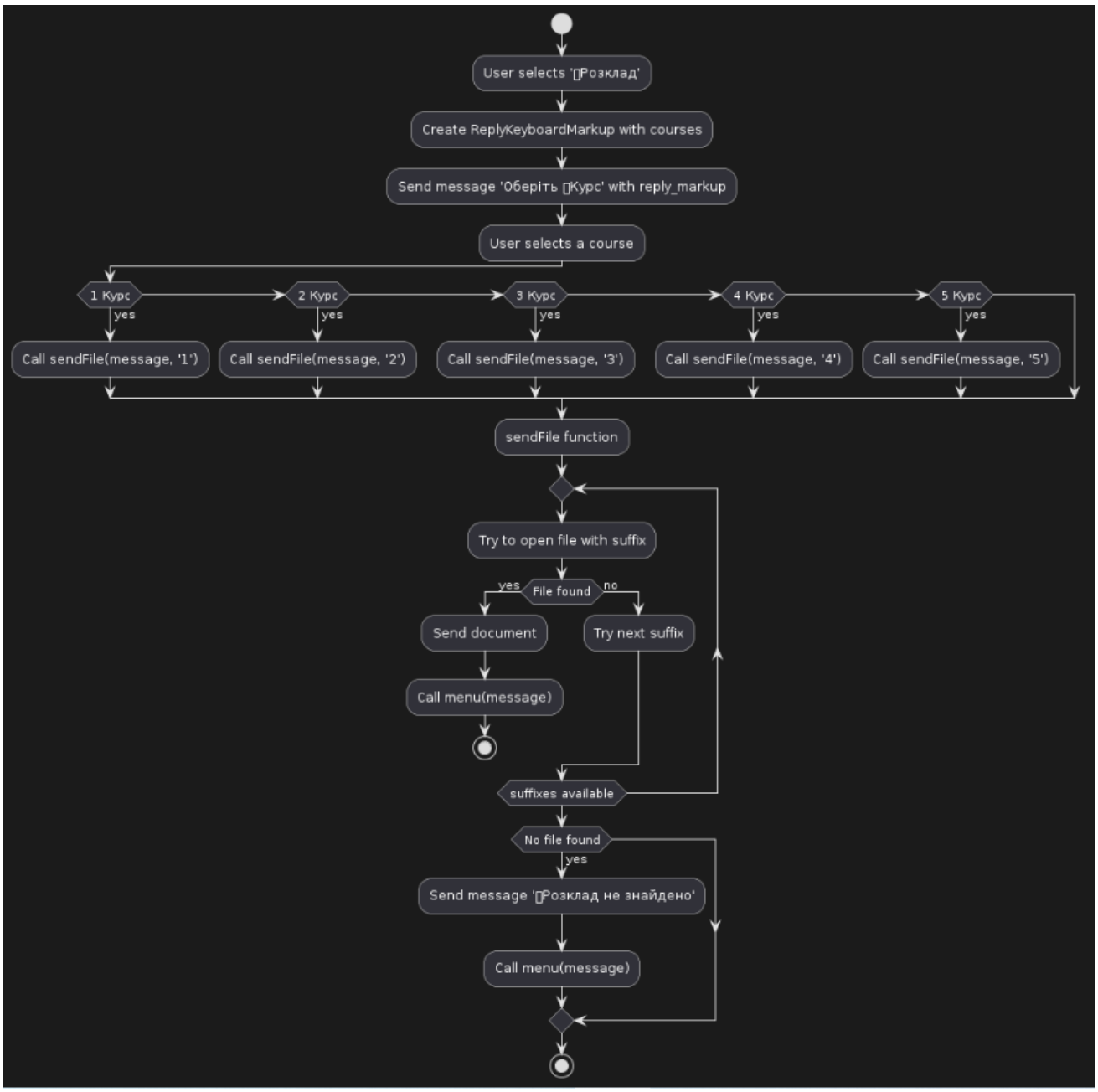

Рисунок 2.6 – Activity Diagram для перегляду розкладу

2.5 Висновки до розділу

В розділі розглянуто дизайн бази даних з таблицею "Students", яка містить всю необхідну інформацію про користувачів бота. UML-діаграми, такі як:

– Use Cases Diagram для студента;

– Use Cases Diagram для адміністратора.

Activity-діаграми:

– Activity Diagram для зміни статусу користувача;
– Activity Diagram для перегляду соц. мереж деканату;
– Activity Diagram для перегляду розкладу.

Також визначено основні етапи проєктування бази даних, а саме: концептуальне проєктування, логічне, та фізичне. До кожного з етапів є мета та низка послідовних процедур, яка допоможе з проєктуванням БД.

3 РОЗРОБКА ОСВІТНЬОГО ТЕЛЕГРАМ-БОТА

3.1 Отримання токену та налаштування бота

Для того, щоб створити бот, ми повинні отримати унікальний токен з офіційного бота @BotFather. Надалі ми його будемо використовувати для керування ботом та доступом до його API.

Також, щоб цей токен був згенерований потрібно вигадати username нашому боту. Було прийняте рішення вибрати «olaacademy_bot», так як цей username був не зайнятий та чудово підходив під наш проєкт. На рис 3.1 показано процес отримання токену.

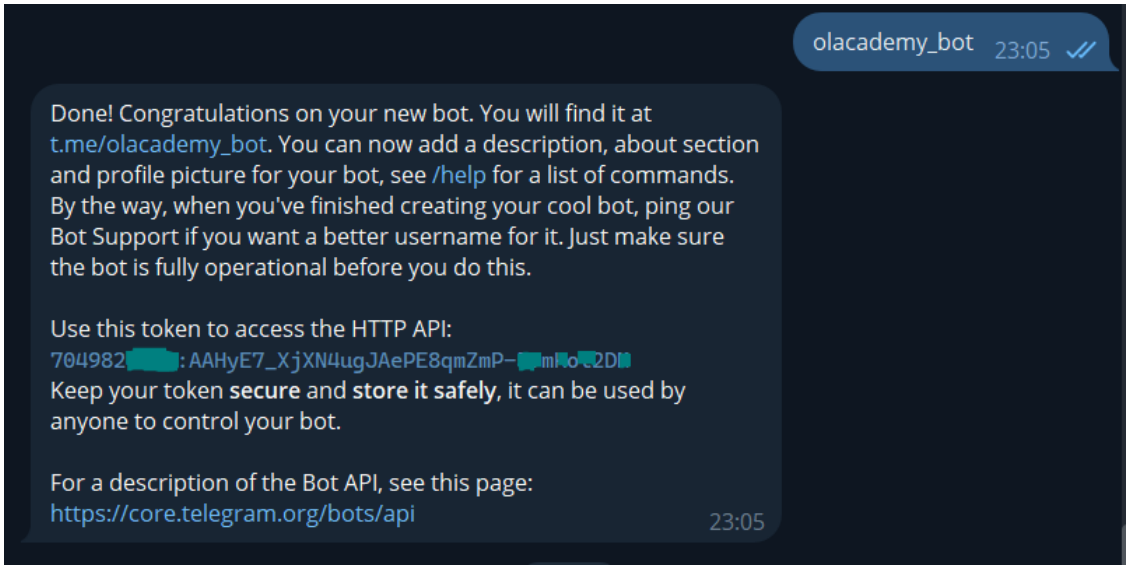

Рисунок 3.1 – Отримання унікального токену

Важливо, щоб в username, був префікс «bot» в кінці, без нього не можливо буде створити унікальний токен. Нижче на рис. 3.2 показано процес встановлення username бота.

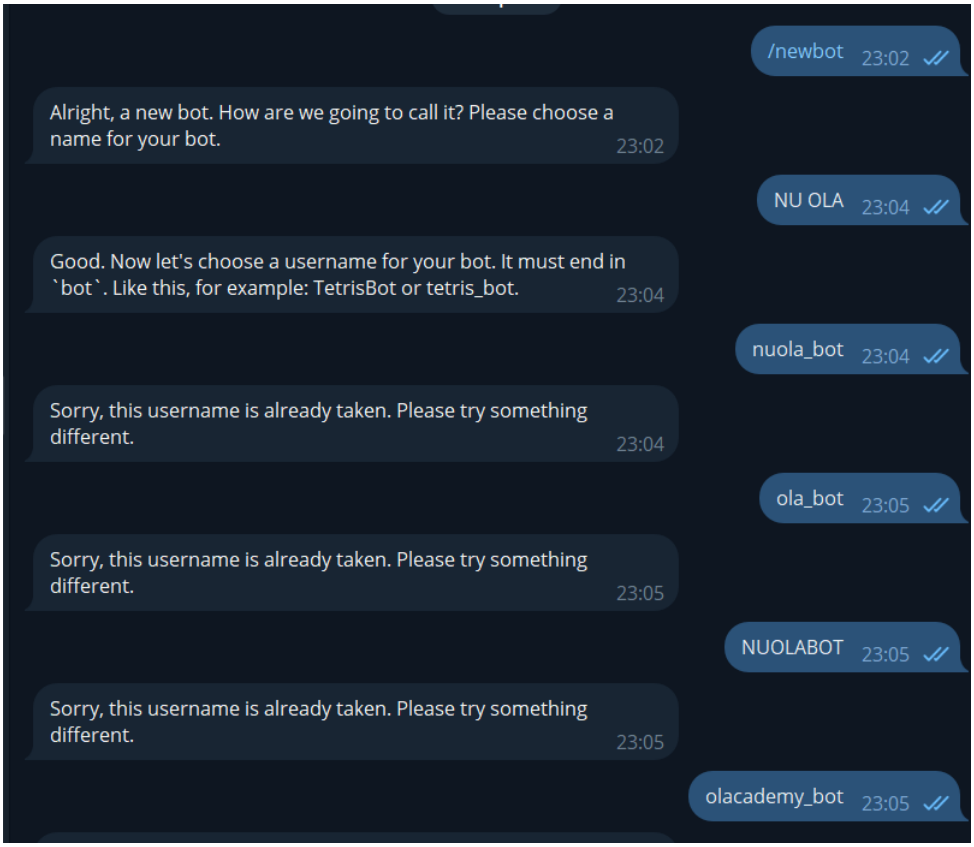

Рисунок 3.2 – Встановлення username боту

Наступним кроком є надання боту привабливого вигляду за допомогою фотографії профілю (рис. 3.3). Хоча це не обов'язково, але воно суттєво покращить візуальне сприйняття бота та зробить його більш привабливим для користувачів. В нашому випадку ідеальним вибором буде логотип факультету. Він чітко ідентифікує бота та дає користувачам зрозуміти, чого можна очікувати від взаємодії з ним.

Рисунок 3.3 – Встановлення фото бота

3.2 Функціонал бота для студента

Щоб забезпечити зручне та інтуїтивно зрозуміле керування ботом, було прийнято рішення використовувати кнопки. Ця можливість дозволяє користувачам швидко та легко отримувати доступ до основних функцій бота.

В даній системі на основі телеграм-бота реалізований функціонал для студента та адміністратора, нижче буде детально описано функціонал кожного з них.

Після реєстрації користувача на головному екрані з'являються чотири головні кнопки:

1. Моя інформація – цей розділ дає користувачам можливість переглянути та оновити інформацію про себе, яку вони вказали при реєстрації.

2. Статус – тут користувачі можуть встановити та змінити свій статус.

3. Соц. мережі – цей розділ надає користувачам доступ до офіційних сторінок факультету та університету в соціальних мережах. Також тут можна знайти номер

телефону деканату. Це зручний спосіб для користувачів бути в курсі всіх новин та подій, пов'язаних з їхнім навчальним закладом.

4. Розклад – у цьому розділі користувачі можуть переглянути розклад занять для свого курсу.

Це дозволяє їм завжди бути в курсі розкладу та планувати свій час. На рис. 3.4 ми можемо побачити головне меню бота.

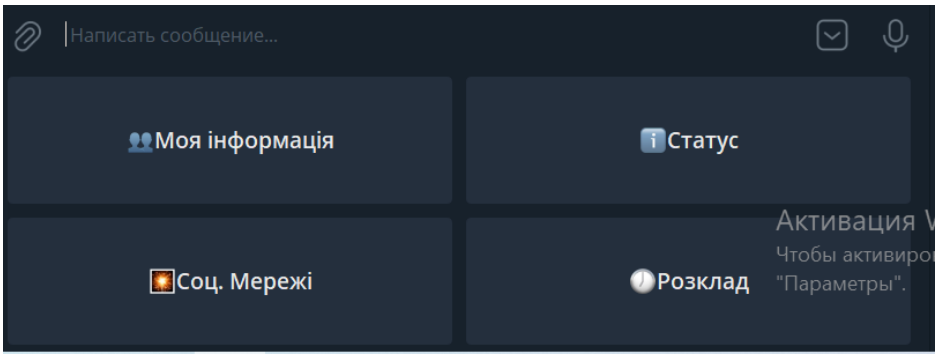

Рисунок 3.4 – Головне меню бота

Для того, щоб зареєструватись в боті, потрібно відправити команду /start та вказати свої дані.

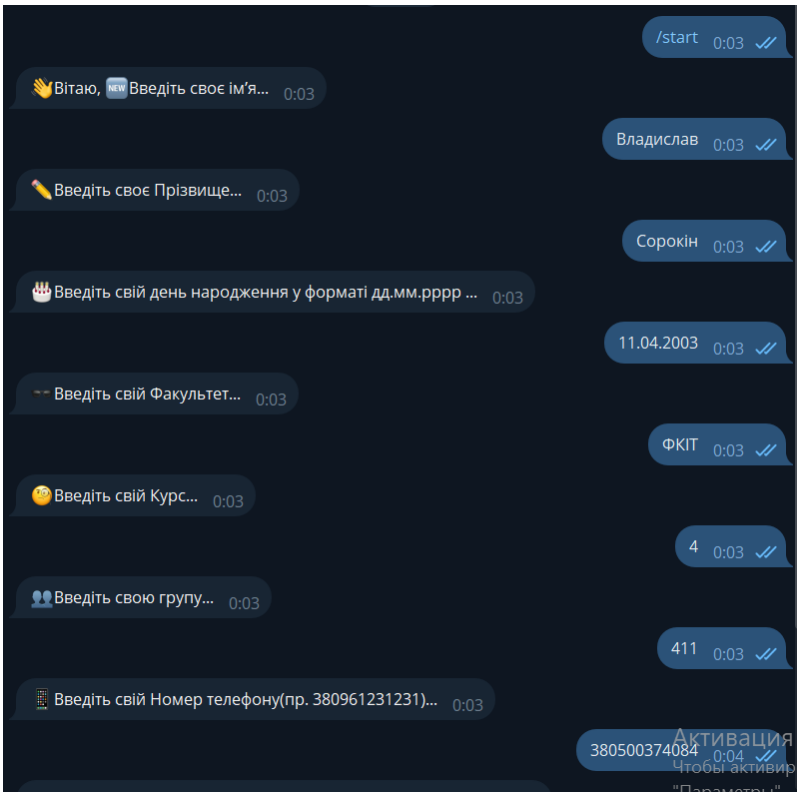

Рисунок 3.5 – Реєстрація користувача

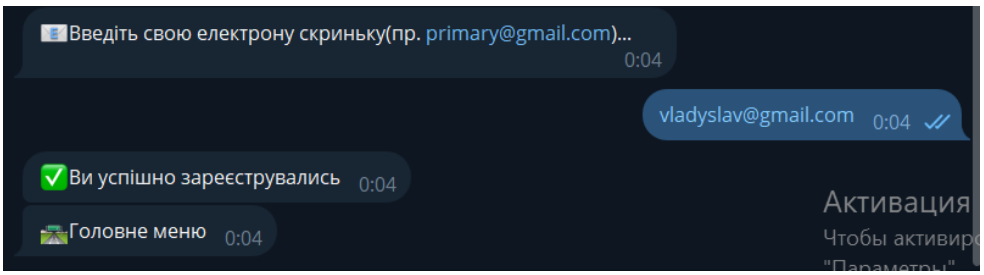

Рисунок 3.6 – Кінець реєстрації

Як ми бачимо, то після введення всіх даних, отримуємо повідомлення про успішну реєстрацію. Це означає, що користувач повністю зареєстрований та його дані збереглись в базі даних.

Розглянемо кожну кнопку та її функціонал окремо:

1) Моя інформація. Саме тут студент може переглянути всю інформацію, яка була вказана при реєстрації (рис. 3.7) та за необхідністю її змінити (рис. 3.6).

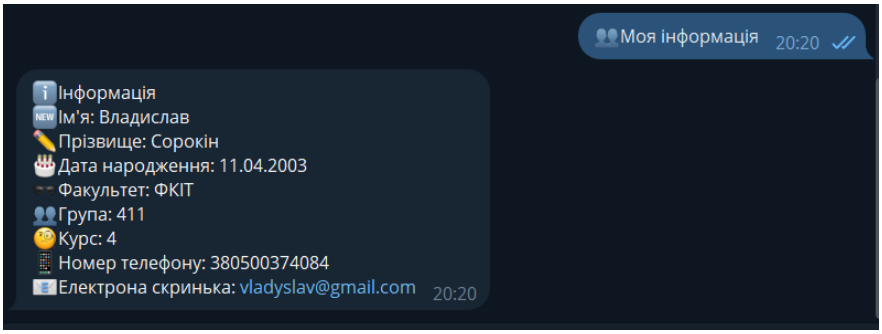

Рисунок 3.7 – Перегляд інформації

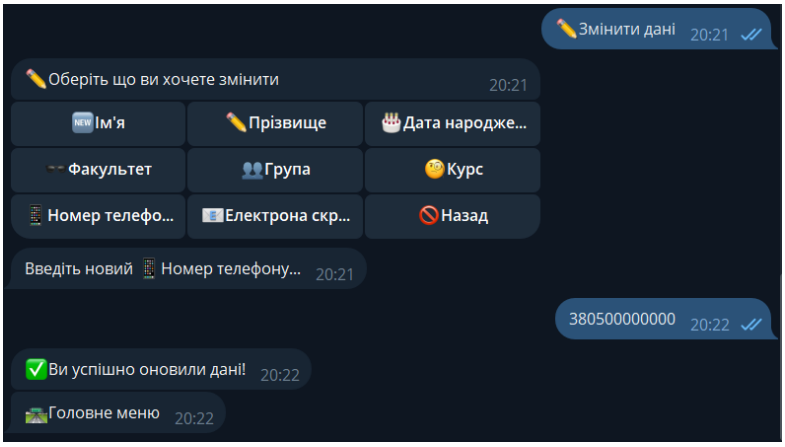

Рисунок 3.8 – Зміна інформації

2) Статус. Тут користувач встановлює свій статус та за необхідністю також може його змінити (рис. 3.9).

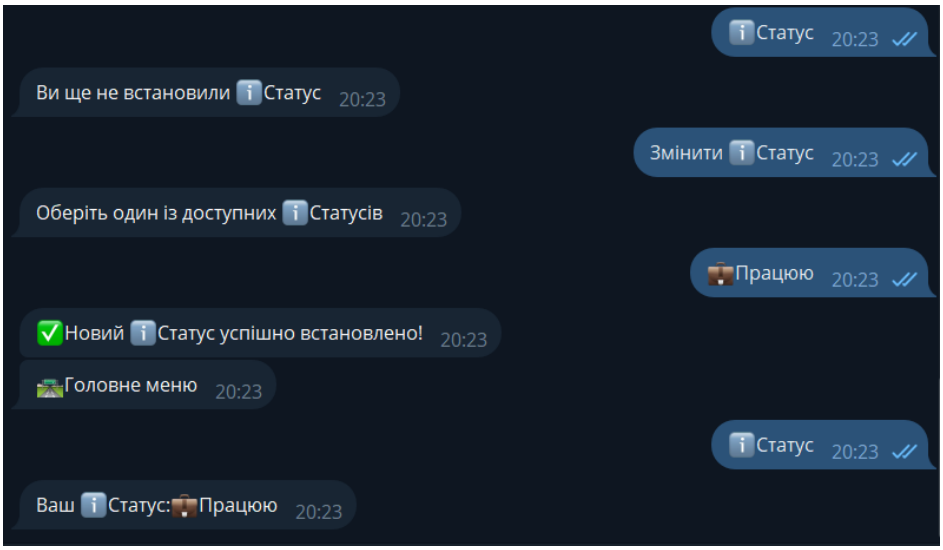

Рисунок 3.9 – Встановлення та зміна статусу

3) Соц. мережі. Ця кнопка дає змогу переглянути соціальні мережі факультету та контакти деканату (рис. 3.10).

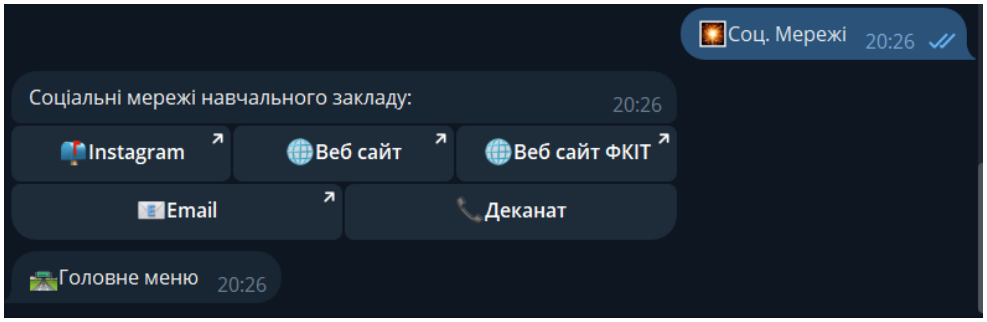

Рисунок 3.10 – Перегляд соц. мереж

4) Розклад. При натисканні на дану кнопку студент потрапляє в меню, де може вибрати курс в якому він хоче переглянути свій розклад (рис. 3.2.8). Після того, як курс обраний, студенту відправляється файл з розкладом, деталі зазначені на рис. 3.11

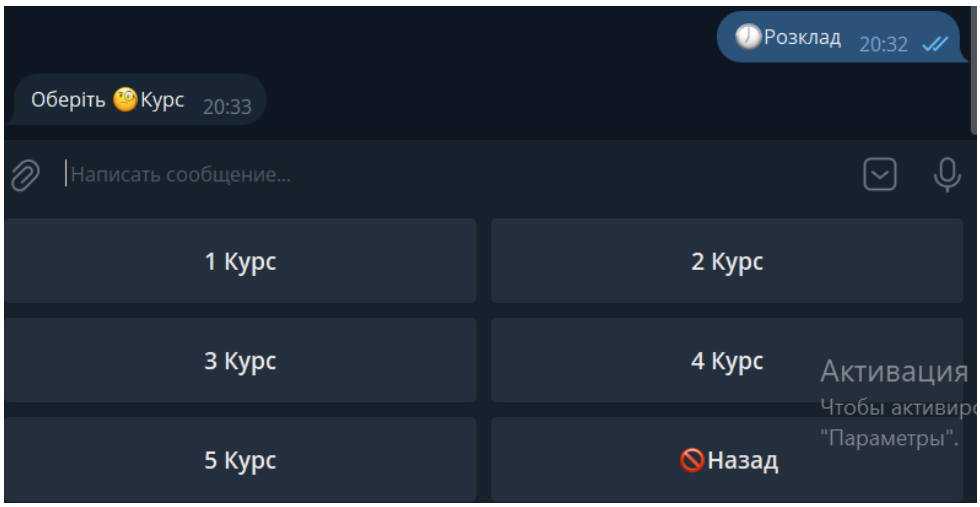

Рисунок 3.11 – Перегляд та вибір курсу

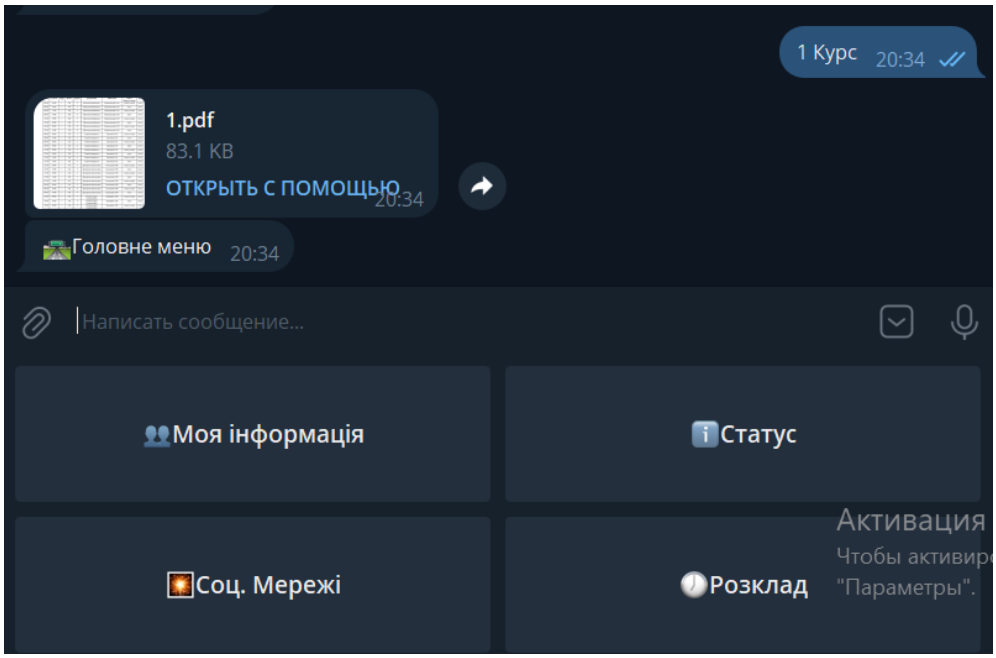

Рисунок 3.12 – Отримання розкладу

3.3 Функціонал бота для адміністратора

Цей розділ описує функціональні можливості бота, доступні адміністратору.

Адміністратор має розширений набір функцій, що дозволяє йому керувати системою, її користувачами та контентом:

1) Встановити розклад. Адміністратор в даному розділі може легко та зручно завантажити розклад для студентів. Для цього йому потрібно вибрати курс

на який він хоче завантажити розклад (рис. 3.13) і після цього відправити файл (рис. 3.14).

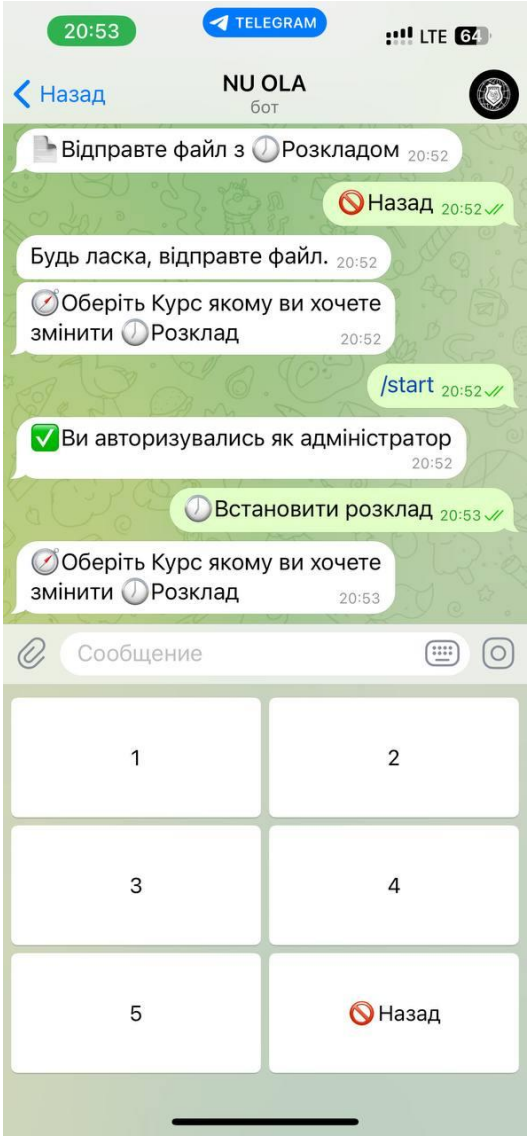

Рисунок 3.13 – Перегляд курсів для завантаження розкладу

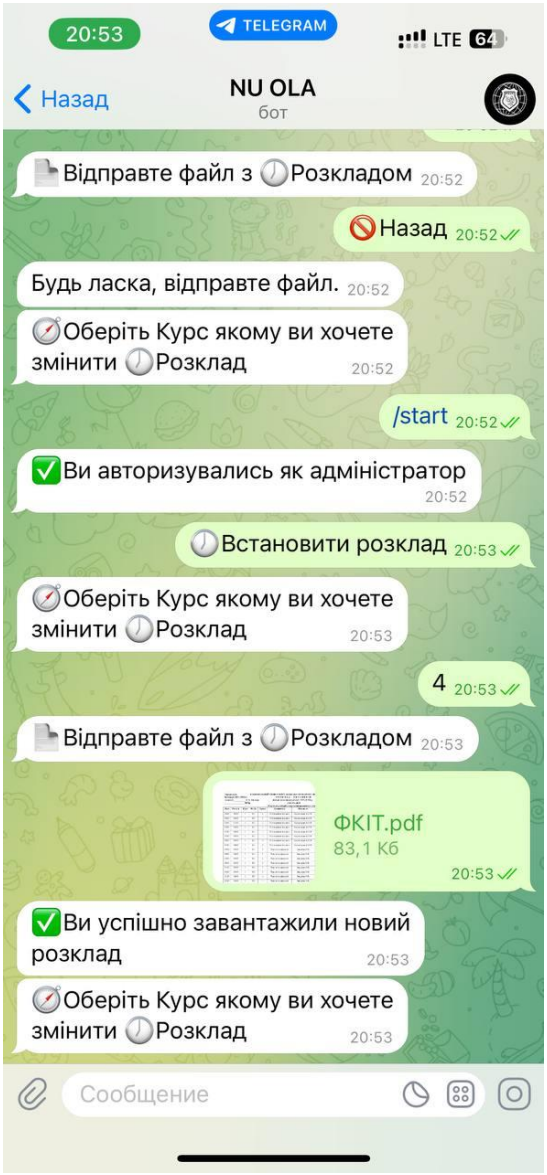

Рисунок 3.14 – Завантаження розкладу для вибраного курсу

Наступним етапом є детальний перегляд кнопки «Пошук студентів», яка в нас розміщена в головному меню бота. При натисканні на неї, система нам пропонує увести прізвище студента. Після введення даних, адміністратор бачить всю необхідну інформацію про студента, яка може облегшити навчальний процес. На рисунку 3.15 наведена реалізація даної функції.

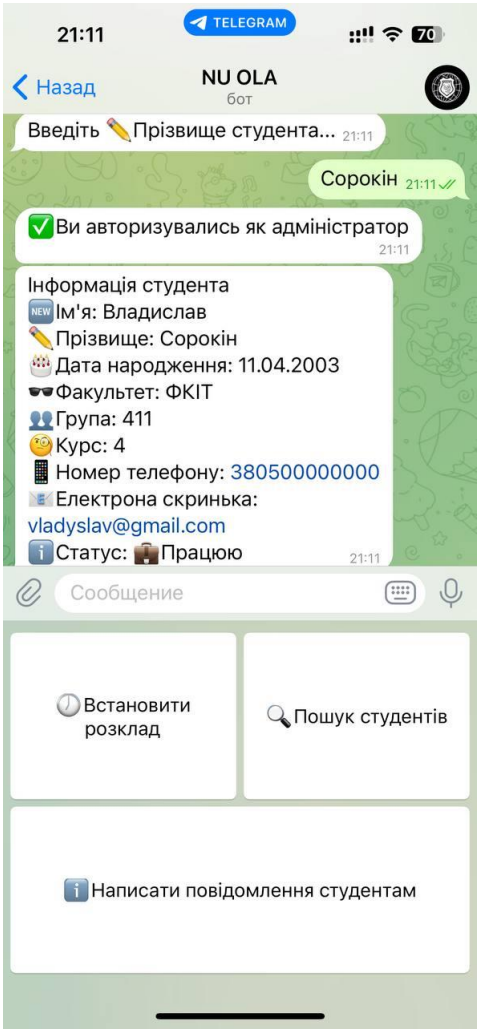

Рисунок 3.15 – Перегляд інформації про студента

Не менш важливою є функція «Написати повідомлення студентам». На мій погляд вона є корисною з кількох причин:

Ефективна комунікація: Ця функція дозволяє адміністратору швидко та легко надсилати повідомлення всім користувачам бота. Це може бути корисно для оголошення новин, нагадування про події, розповсюдження важливої інформації.

Персоналізація: Адміністратор може персоналізувати повідомлення, додаючи до них ім'я студента, групу або інші дані. Це робить повідомлення більш релевантними та зрозумілим для студентів.

Економія часу: Завдяки цій функції адміністратор може економити час, який би він витратив на надсилання повідомлень кожному студенту окремо або через старосту.

В даній функції є можливість відправити інформацію студентам як і в текстовому вигляді, так і у вигляді файлу.

На рисунку 3.16 показано процес відправки повідомлення студентам в текстовому вигляді і відповідно отримання повідомлення студентами (рис. 3.17).

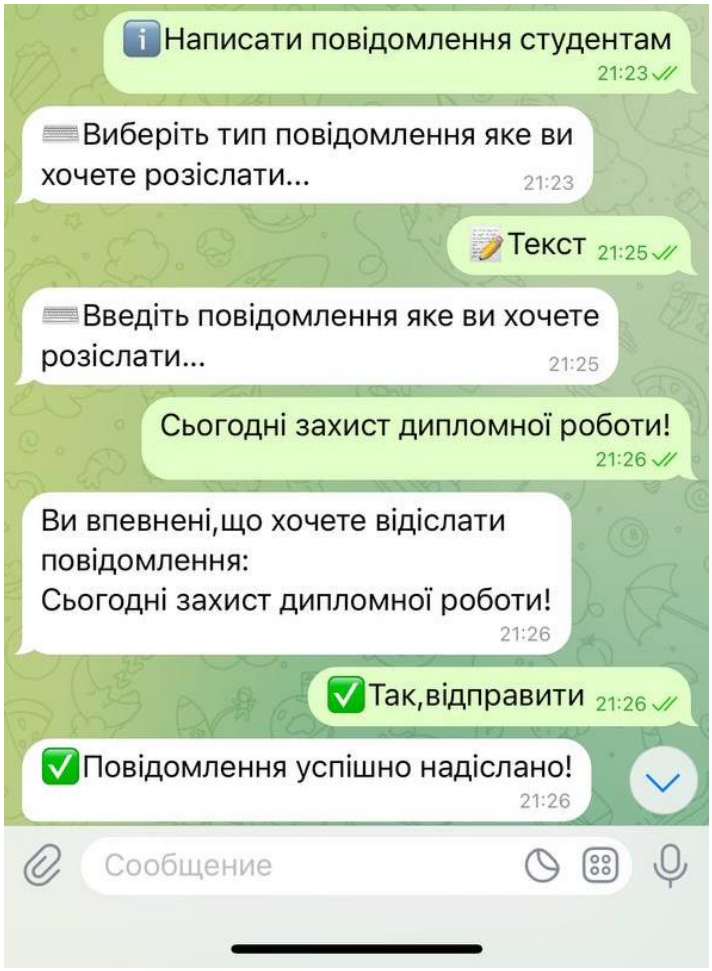

Рисунок 3.16 – Процес відправки текстового повідомлення

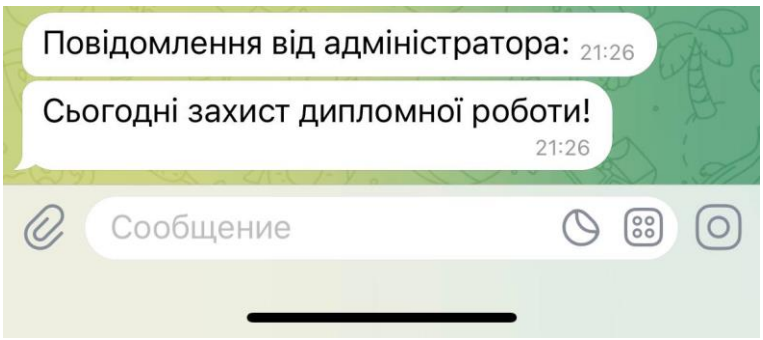

Рисунок 3.17 – Отримання повідомлення від адміністратора

Далі буде показана функція надсилання файлу студентам через бот (рис. 3.18) і отримання даного файлу студентами (рис. 3.19).

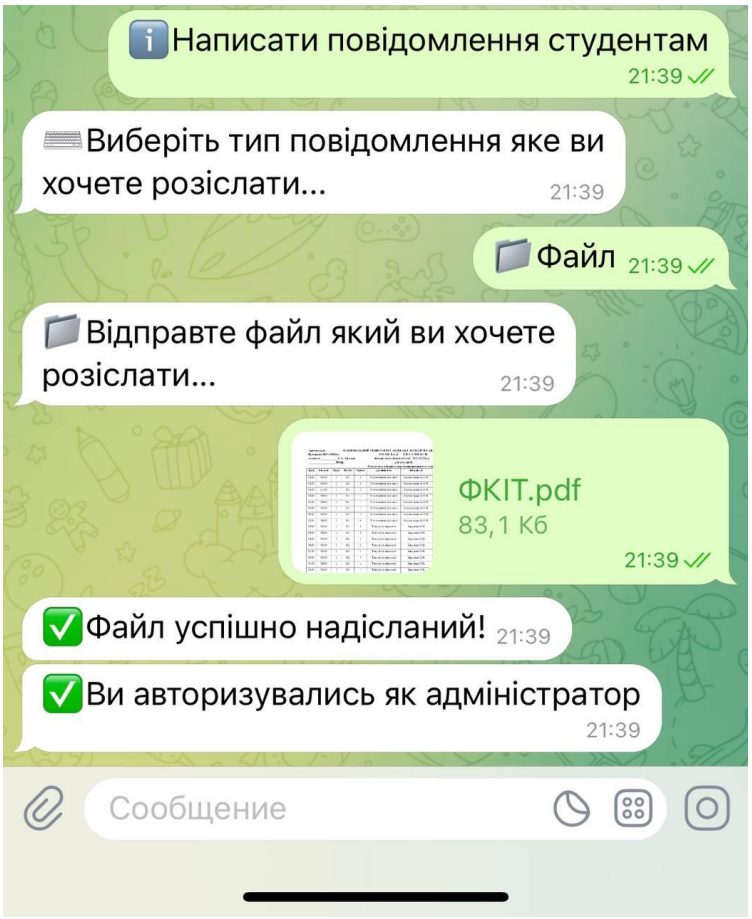

Рисунок 3.18 – Надсилання файлу студентам

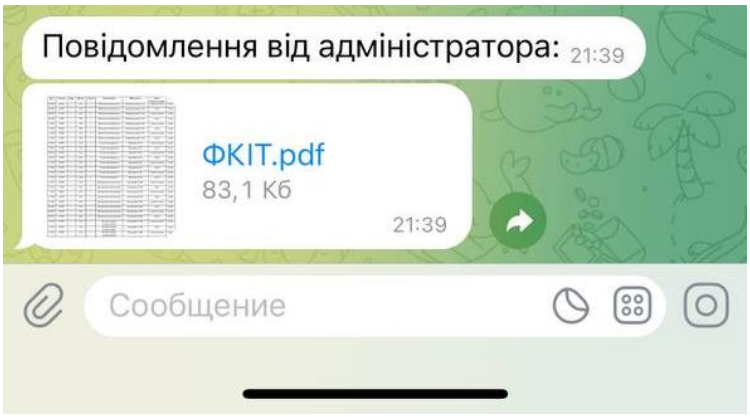

Рисунок 3.19 – Отримання файлу від адміністратора

3.5 Тестування бота

Функціональне тестування – один із видів тестування, спрямованого на перевірку відповідностей функціональних вимог ПЗ його реальним характеристикам. Основним завданням функціонального тестування є підтвердження того, що програмний продукт, який розробляється, володіє усім необхідним функціоналом [16].

Важливим етапом початку роботи з ботом є процес реєстрації. В табл. 3.1 наведено тест-кейс, завдяки якому ми можемо побачити чи успішно зареєструвався користувач в системі.

Таблиця 3.1 – Тест-кейси для функціонального тестування

Назва тест-кейсу	Опис тест-кейсу	Очікуваний результат
Перевірка введення імені, прізвища, дати народження, факультету, курсу, групи, номеру телефону, емейлу	Перевірити кожний пункт, на заборонені символи під час введення даних на кожному етапі	Всі функції працюють коректно. Під час введення неправильних даних, система пише помилку, якщо дані правильно уведено – наступний крок реєстрації
Перевірка завантаження розкладу	Натиснути на кожну кнопку курсу та переглянути чи є там файл з розкладом	Всі файли були завантажені успішно та відображаються коректно
Перевірка зміни особистої інформації	Перевірити зміну номеру телефону, натиснувши на клавішу «Змінити дані»	Зміна відбулась успішно та дані оновлено
Перевірка встановлення та зміни статусу	При натисканні на кнопку «Статус» – встановити свій статус. Потім натиснути на кнопку «Змінити статус» і встановити новий статус	Статус був успішно встановлений, а після чого успішно змінений
Перевірка відображення соціальних мерех	Натиснути на кнопку «Соц. Мережі» та переглянути всі додаткові кнопкі	Всі функції працюють коректно та виконують своє призначення.

3.6 Висновки до розділу

Було створено тест-кейси та проведено функціональне тестування. Всі тест-кейси пройдено успішно, виходячи з цього можна зробити висновок, що всі функції в телеграм-боті працюють коректно. Даною системою буде зручно та легко користуватися як і студентам, так і працівникам деканату.

ВИСНОВКИ

У роботі розроблено Telegram-бота, який надає такі функціональні можливості: надання інформації про розклад занять, перегляд завдань, оновлення статусу, взаємодія з соціальними мережами навчального закладу, а також інформування користувачів через інтеграцію з базою даних.

Метою роботи було створення Telegram-бота, який спрощує життя студентів, автоматизує рутинні завдання та надає швидкий доступ до важливої інформації. В результаті розробки було створено інформаційну систему, яка стала корисним інструментом для користувачів.

Проведено аналіз наявних Telegram-ботів з подібними функціональними можливостями, що дозволило визначити сильні та слабкі сторони. На основі проведеного аналізу було розроблено власний Telegram-бот, який має унікальні функції та зручний інтерфейс.

Для розробки бота було використано мову програмування Python, а також фреймворк pyTelegramBotAPI і бібліотеку SQLite3 для роботи з базою даних. Проведене тестування бота показало, що він працює коректно, виконує всі заплановані функції та відповідає очікуванням користувачів. Усі тест-кейси були успішними, що підтверджує високу якість реалізації програмного продукту.

В майбутньому планується додати нові функції, розширити аудиторію користувачів та інтегрувати бота з іншими системами, що дозволить зробити його ще більш корисним та зручним для використання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Python. URL: https://en.wikipedia.org/wiki/Python_(programming_language)
2. What is Visual Studio? URL: https://learn.microsoft.com/uk-ua/visualstudio/get-started/visual-studio-ide?view=vs-2022
3. Що таке функціональні вимоги: приклади, визначення, повний посібник/ URL: https://visuresolutions.com/uk/блог/функціональні-вимоги/
4. SQLite. URL: https://uk.wikipedia.org/wiki/SQLite
5. What Is SQLite? URL: https://www.sqlite.org/
6. SQLite Tutorial. URL: https://www.sqlitetutorial.net/
7. Telegram-боти на Python: огляд п’яти найкращих фреймворків/бібліотек. URL: https://drukarnia.com.ua/articles/telegram-boti-na-python-oglyad-p-yati-naikrashikh-freimvorkiv-bibliotek-L7UA7
8. Система керування базами даних. URL: https://uk.wikipedia.org/wiki/Система керування базами даних
9. Архітектура СКБД. URL: https://is.gd/ez9lP2
10. Основні етапи проєктування баз даних. URL: https://javarush.com/ua/quests/lectures/ua.questhibernate.level17.lecture01
11. SQLite Підручник з бази даних для початківців: навчайтеся на прикладах. URL: https://www.guru99.com/uk/sqlite-tutorial.html
12. UML. URL: https://en.wikipedia.org/wiki/Unified_Modeling_Language
13. UML Use Case Diagram Tutorial. URL: https://www.lucidchart.com/pages/uml-use-case-diagram
14. How to Draw UML Use Case Diagram. URL: https://www.uml-diagrams.org/use-case-diagrams-how-to.html
15. Моралес Дж. Дізнайтеся все про діаграму активності. UML [з методами]. URL: https://www.mindonmap.com/uk/blog/uml-activity-diagram/
16. Функціональне тестування. URL: https://qalight.ua/baza-znaniy/funktsionalne-testuvannya/

ДОДАТКИ

Додаток А. Програмний код

```
import telebot
from telebot import types
import time
from random import randint
import sqlite3
import re
import asyncio
import os.path
from pathlib import Path
group , phone , email , departament , surname , name , userID , course ,
birthday  = '' , '' , '' , '' , '' , '' , '' , '' , ''
bot = telebot.TeleBot('7049824274:AAHyE7_XjXN4ugJAePE8qmZmP-57mKol2DM')
isMessage = False
admID = [6900960499 , 6113448235 ]
def is_convertible_to_int(str):
    any(char.isdigit() for char in str)
def is_convertible_to_str(string):
    try:
        int(string)
        return True
    except ValueError:
        return False
def check_user_auth(user_id):
    conn = sqlite3.connect('database.sql')
    cur = conn.cursor()
    cur.execute('SELECT userID FROM students WHERE userID = ?', (user_id,))
    result = cur.fetchone()
    cur.close()
    conn.close()
    return result is not None
def validate_date(data):
        pattern = r'^\d{2}\.\d{2}\.\d{4}$'  # Паттерн для дати у форматі
"дд.мм.рррр"
        if re.match(pattern, data):
            return True
        else:
            return False

def admCabinet(message):
    markup = types.ReplyKeyboardMarkup(row_width=2)
    item1 = types.KeyboardButton('🕐Встановити розклад')
    item2 = types.KeyboardButton('🔍Пошук студентів')
    item3 = types.KeyboardButton('ℹ️Написати повідомлення студентам')
    markup.add(item1, item2 , item3)
```

```
    bot.send_message(message.chat.id , '✅Ви авторизувались як адміністратор',
reply_markup=markup)
@bot.message_handler(commands = ['start'])
def hello(message):
    global userID
    userID = message.chat.id
    conn = sqlite3.connect('database.sql')
    cur = conn.cursor()
    cur.execute('CREATE TABLE IF NOT EXISTS students (userID int(50) ,name
varchar(20) , surname varchar(20) ,birthday varchar(20), department varchar(50) ,
studentGroup varchar(50) , course int(2) ,  phoneNumber int(20) , email
varchar(50) , status varchar(50))')
    cur.close()
    conn.close()
    if message.chat.id in admID:
        admCabinet(message)
    elif check_user_auth(userID):
        menu(message)
    else:
        bot.send_message(message.chat.id , '👋Вітаю, 🆕Введіть своє ім’я...')
        bot.register_next_step_handler(message , nameFunc )
def nameFunc(message):
    global name
    if is_convertible_to_int(message.text):
        bot.send_message(message.chat.id , "🚫Помилка! Ім'я може вміщати тільки
букви! Спробуйте ще раз...")
        bot.register_next_step_handler(message , nameFunc )
    else:
        name = message.text
        bot.send_message(message.chat.id , '🖍️Введіть своє Прізвище...')
        bot.register_next_step_handler(message , surnameFunc)
def surnameFunc(message):
    global surname
    if is_convertible_to_int(message.text):
        bot.send_message(message.chat.id , '🚫Помилка! 🖍️Прізвище може вміщати
тільки букви! Спробуйте ще раз...')
        bot.register_next_step_handler(message , surnameFunc )
    else:
        surname = message.text
        bot.send_message(message.chat.id , '🎂Введіть свій день народження у
форматі дд.мм.рррр ...')
        bot.register_next_step_handler(message , birthdayFunc)
def birthdayFunc(message):
    global birthday
    if validate_date(message.text):
        birthday = message.text
        bot.send_message(message.chat.id , '👓Введіть свій Факультет...')
        bot.register_next_step_handler(message , departamentFunc)
    else:
```

```
        bot.send_message(message.chat.id , '🚫Помилка!🎂Дата народження повинна
бути у форматі дд.мм.рррр! Спробуйте ще раз...')
        bot.register_next_step_handler(message , birthdayFunc)
def departamentFunc(message):
    global departament
    departament = message.text
    bot.send_message(message.chat.id , '🧐Введіть свій Курс...')
    bot.register_next_step_handler(message , courseFunc)
def courseFunc(message):
    global course
    course = message.text
    bot.send_message(message.chat.id , '👥Введіть свою групу...')
    bot.register_next_step_handler(message , groupFunc)
def groupFunc(message):
    global group
    group = message.text
    bot.send_message(message.chat.id , '📱Введіть свій Номер телефону(пр.
380961231231)...')
    bot.register_next_step_handler(message , phoneFunc)
def phoneFunc(message):
    global phone
    if is_convertible_to_str(message.text) and len(message.text) == 12:
        phone = message.text
        bot.send_message(message.chat.id , '📧Введіть свою електрону
скриньку(пр. primary@gmail.com)...')
        bot.register_next_step_handler(message , emailFunc)
    else:
        bot.send_message(message.chat.id , '🚫Помилка! 📱Номер телефону може
вміщати тільки цифри (пр. 380961231231)! Спробуйте ще раз...')
        bot.register_next_step_handler(message , phoneFunc )
def emailFunc(message):
    global email
    if '@' in message.text:
        email = message.text
        insertIntoDB(message)
    else:
        bot.send_message(message.chat.id , "🚫Помилка! 📧Електрона скринька має
обов'язково вміщати @! Спробуйте ще раз...")
        bot.register_next_step_handler(message , emailFunc )
def insertIntoDB(message):
    global userID
    userID = message.chat.id
    global group , phone , email , departament , surname , name , course ,
birthday
    student_info = [userID , name , surname , birthday , departament , group ,
course , phone , email]
    conn = sqlite3.connect('database.sql')
    cur = conn.cursor()
```

```
    cur.execute('INSERT INTO students (userID , name, surname, birthday ,
department, studentGroup, course , phoneNumber, email) VALUES (?, ?, ?, ?, ?, ?,
?, ? , ?)', student_info)
    conn.commit()
    cur.close()
    conn.close()
    bot.send_message(message.chat.id , '✅Ви успішно зареєструвались')
    menu(message)
def menu(message):
    if message.chat.id in admID:
        admCabinet(message)
    else:
        markup = types.ReplyKeyboardMarkup(row_width=2)
        item1 = types.KeyboardButton('👥Моя інформація')
        item2 = types.KeyboardButton('ℹ️Статус')
        item3 = types.KeyboardButton('🎆Соц. Мережі')
        item4 = types.KeyboardButton('🕐Розклад')
        markup.add(item1, item2, item3, item4)
        bot.send_message(message.chat.id , '🎏Головне меню' ,
reply_markup=markup)
@bot.message_handler(func=lambda message: message.text == '👥Моя інформація')
def send_info(message):
    global userID
    userID = message.chat.id

    markup = types.ReplyKeyboardMarkup(row_width=2)
    item1 = types.KeyboardButton('🖍Змінити дані')
    item2 = types.KeyboardButton('🚫Назад')
    markup.add(item1 , item2)
    conn = sqlite3.connect('database.sql')
    cur = conn.cursor()
    cur.execute('SELECT * FROM students WHERE userID = ?', (userID,))
    result = cur.fetchone()
    cur.close()
    conn.close()

    name, surname, birthday, department, group, course , phone, email =
result[1:9]
    info_message = f"ℹ️Інформація\n🆕Ім'я: {name}\n🖍Прізвище:
{surname}\n🎂Дата народження: {birthday}\n🕶Факультет: {department}\n👥Група:
{group}\n🧐Курс: {course}\n📱Номер телефону: {phone}\n📧Електрона скринька:
{email}"
    bot.send_message(message.chat.id, info_message, parse_mode='HTML' ,
reply_markup=markup)
@bot.message_handler(func=lambda message: message.text == '🚫Назад')
def returnToMenu(message):
    menu(message)
@bot.message_handler(func=lambda message: message.text == '🖍Змінити дані')
def changeInfo(message):
    markup = types.InlineKeyboardMarkup()
```

```
    button1 = types.InlineKeyboardButton("🆕Ім'я", callback_data='name')
    button2 = types.InlineKeyboardButton("🖍Прізвище", callback_data='surname')
    button3 = types.InlineKeyboardButton("🎂Дата народження",
callback_data='birthday')
    button4 = types.InlineKeyboardButton("🕶Факультет",
callback_data='departament')
    button5 = types.InlineKeyboardButton("👥Група", callback_data='group')
    button6 = types.InlineKeyboardButton("🧐Курс", callback_data='course')
    button7 = types.InlineKeyboardButton("📱Номер телефону",
callback_data='phone')
    button8 = types.InlineKeyboardButton("📧Електрона скринька",
callback_data='email')
    button9 = types.InlineKeyboardButton("🚫Назад", callback_data='menu')
    markup.add(button1, button2 , button3 , button4 , button5 , button6 , button7
, button8 , button9 )
    bot.send_message(message.chat.id , '🖍Оберіть що ви хочете змінити' ,
reply_markup=markup)
@bot.callback_query_handler(func=lambda call: True)
def callback_query(call):
    markup = types.ReplyKeyboardMarkup(row_width=2)
    item1 = types.KeyboardButton('🚫Назад')
    markup.add(item1)
    if call.data == 'phoneDecanat':
        bot.send_message(call.message.chat.id , 'Номер телефону деканату:
<code>+380123123123</code>' , parse_mode='HTML' )
    if call.data == 'menu':
        menu(call.message)
    if call.data == 'name':
        bot.send_message(call.message.chat.id, "Введіть нове
🆕Ім'я...",reply_markup=markup)
    elif call.data == 'surname':
        bot.send_message(call.message.chat.id, "Введіть нове
🖍Прізвище...",reply_markup=markup)
    elif call.data == 'departament':
        bot.send_message(call.message.chat.id, "Введіть новий
🕶Факультет...",reply_markup=markup)
    elif call.data == 'group':
        bot.send_message(call.message.chat.id, "Введіть нову
👥групу...",reply_markup=markup)
    elif call.data == 'phone':
        bot.send_message(call.message.chat.id, "Введіть новий 📱Номер
телефону...",reply_markup=markup)
    elif call.data == 'email':
        bot.send_message(call.message.chat.id, "Введіть нову 📧електрону
скриньку...",reply_markup=markup)
    elif call.data == 'course':
        bot.send_message(call.message.chat.id, "Введіть новий
🧐Курс...",reply_markup=markup)
    elif call.data == 'birthday':
```

```
        bot.send_message(call.message.chat.id, "Введіть нову 🎂дату
народження...",reply_markup=markup)
    bot.register_next_step_handler(call.message, insertNewData,
call.message.chat.id , call)

def insertNewData(message, chat_id , call):
    newData = message.text
    conn = sqlite3.connect('database.sql')
    cur = conn.cursor()
    global userID
    userID = message.chat.id
    if call.data == 'name':
        if any(char.isdigit() for char in message.text) and message.text !=
'🚫Назад':
            bot.send_message(message.chat.id , "🚫Помилка! Ім'я може вміщати
тільки букви! Спробуйте ще раз...")
            bot.register_next_step_handler(message, insertNewData,
call.message.chat.id , call )
        elif message.text != '🚫Назад':
            cur.execute("UPDATE students SET name = ? WHERE userID = ?",
(newData, userID))
            bot.send_message(chat_id, "✅Ви успішно оновили дані!")
            menu(message)
    elif call.data == 'surname':
        if any(char.isdigit() for char in message.text) and message.text !=
'🚫Назад':
            bot.send_message(message.chat.id , "🚫Помилка! Прізвище може вміщати
тільки букви! Спробуйте ще раз...")
            bot.register_next_step_handler(message, insertNewData,
call.message.chat.id , call )
        elif message.text != '🚫Назад':
            cur.execute("UPDATE students SET surname = ? WHERE userID = ?",
(newData, userID))
            bot.send_message(chat_id, "✅Ви успішно оновили дані!")
            menu(message)
    elif call.data == 'departament':
        if message.text != '🚫Назад':
            cur.execute("UPDATE students SET department = ? WHERE userID = ?",
(newData, userID))
            bot.send_message(chat_id, "✅Ви успішно оновили дані!")
            menu(message)
    elif call.data == 'group':
        if message.text != '🚫Назад':
            cur.execute("UPDATE students SET studentGroup = ? WHERE userID = ?",
(newData, userID))
            bot.send_message(chat_id, "✅Ви успішно оновили дані!")
            menu(message)
    elif call.data == 'phone':
        if is_convertible_to_str(message.text) and message.text != '🚫Назад' :
            if len(message.text) == 12:
```

```
                cur.execute("UPDATE students SET phoneNumber = ? WHERE userID =
?", (newData, userID))
                bot.send_message(chat_id, "✅Ви успішно оновили дані!")
                menu(message)
            else:
                bot.send_message(message.chat.id , "🚫Помилка! Номер телефону
може вміщати тільки цифри та 12 знаків (пр. 380961231231)! Спробуйте ще раз...")
                bot.register_next_step_handler(message, insertNewData,
call.message.chat.id , call )
        elif message.text != '🚫Назад' :
            bot.send_message(message.chat.id , "🚫Помилка! Номер телефону може
вміщати тільки цифри та 12 знаків (пр. 380961231231)! Спробуйте ще раз...")
            bot.register_next_step_handler(message, insertNewData,
call.message.chat.id , call )
    elif call.data == 'email':
        if '@' in message.text:
            cur.execute("UPDATE students SET email = ? WHERE userID = ?",
(newData, userID))
            bot.send_message(chat_id, "✅Ви успішно оновили дані!")
            menu(message)
        elif message.text != '🚫Назад':
            bot.send_message(message.chat.id , "🚫Помилка! 📧Електрона скринька
має обов'язково вміщати @! Спробуйте ще раз...")
            bot.register_next_step_handler(message , insertNewData,
call.message.chat.id , call )
    elif call.data == 'course':
        cur.execute("UPDATE students SET course = ? WHERE userID = ?", (newData,
userID))
        bot.send_message(chat_id, "✅Ви успішно оновили дані!")
        menu(message)
    elif call.data == 'birthday':
        if validate_date(message.text):
            cur.execute("UPDATE students SET birthday = ? WHERE userID = ?",
(newData, userID))
            bot.send_message(chat_id, "✅Ви успішно оновили дані!")
            menu(message)
        else:
            bot.send_message(message.chat.id , '🚫Помилка!🎂Дата народження
повинна бути у форматі дд.мм.рррр! Спробуйте ще раз...')
            bot.register_next_step_handler(message ,insertNewData,
call.message.chat.id , call)

    conn.commit()
    cur.close()
    conn.close()
@bot.message_handler(func=lambda message: message.text == 'ℹ️Статус')
def status(message):
    global userID
    userID = message.chat.id
    conn = sqlite3.connect('database.sql')
```

```
    cur = conn.cursor()
    cur.execute('SELECT status FROM students WHERE userID = ?', (userID,))
    result = cur.fetchone()
    cur.close()
    conn.close()
    if result == (None,):
        markup = types.ReplyKeyboardMarkup(row_width=2)
        item1 = types.KeyboardButton('Змінити ℹ️Статус')
        item2 = types.KeyboardButton('🚫Назад')
        markup.add(item1 , item2)
        bot.send_message(message.chat.id , 'Ви ще не встановили ℹ️Статус' ,
reply_markup=markup)
    else:
        markup =  types.ReplyKeyboardMarkup(row_width=2)
        item1 = types.KeyboardButton('Змінити ℹ️Статус')
        item2 = types.KeyboardButton('🚫Назад')
        markup.add(item1 , item2)
        result = str(result)
        result = result.replace("('", "").replace("',)", "")
        bot.send_message(message.chat.id , f'Ваш ℹ️Статус:{result}'  ,
reply_markup=markup)
@bot.message_handler(func=lambda message: message.text == 'Змінити ℹ️Статус')
def changeStatus(message):
    markup =  types.ReplyKeyboardMarkup(row_width=2)
    item1 = types.KeyboardButton('🏠Вдома')
    item2 = types.KeyboardButton('😷Хворий')
    item3 = types.KeyboardButton('💼Працюю')
    item4 = types.KeyboardButton('🛬За кордоном')
    markup.add(item1 , item2 , item3 ,item4)
    bot.send_message(message.chat.id , 'Оберіть один із доступних ℹ️Статусів'  ,
reply_markup=markup)
    @bot.message_handler(func=lambda message: message.text == '🏠Вдома')
    def temp(message):setStatus(message , '🏠Вдома')
    @bot.message_handler(func=lambda message: message.text == '💼Працюю')
    def temp(message):setStatus(message , '💼Працюю')
    @bot.message_handler(func=lambda message: message.text == '😷Хворий')
    def temp(message):setStatus(message , '😷Хворий')
    @bot.message_handler(func=lambda message: message.text == '🛬За кордоном')
    def temp(message):setStatus(message , '🛬За кордоном')
def setStatus(message , statusStr):
    global userID
    userID = message.chat.id
    conn = sqlite3.connect('database.sql')
    cur = conn.cursor()
    cur.execute("UPDATE students SET status = ? WHERE userID = ?", (statusStr,
userID))
    conn.commit()
    cur.close()
    conn.close()
```

```
    bot.send_message(message.chat.id , '✅Новий ℹ️Статус успішно встановлено!')
    menu(message)
@bot.message_handler(func=lambda message: message.text == '🕐Розклад')
def selectCourse(message):
    markup =  types.ReplyKeyboardMarkup(row_width=2)
    item1 = types.KeyboardButton('1 Курс')
    item2 = types.KeyboardButton('2 Курс')
    item3 = types.KeyboardButton('3 Курс')
    item4 = types.KeyboardButton('4 Курс')
    item5 = types.KeyboardButton('5 Курс')
    item6 = types.KeyboardButton('🚫Назад')
    markup.add(item1 , item2 , item3 ,item4 , item5 , item6)
    bot.send_message(message.chat.id , 'Оберіть 🧐Курс' , reply_markup=markup)
    @bot.message_handler(func=lambda message: message.text == '1 Курс')
    def temp(message):
        sendFile(message , '1')
    @bot.message_handler(func=lambda message: message.text == '2 Курс')
    def temp(message):
        sendFile(message , '2')
    @bot.message_handler(func=lambda message: message.text == '3 Курс')
    def temp(message):
        sendFile(message , '3')
    @bot.message_handler(func=lambda message: message.text == '4 Курс')
    def temp(message):
        sendFile(message , '4')
    @bot.message_handler(func=lambda message: message.text == '5 Курс')
    def temp(message):
        sendFile(message , '5')
    def sendFile(message , num):
        suffixes = ["txt", "pdf", "docx" , "csv" , "xls" , "xlsx", "png" , "PNG"
, "jpeg" , "jpg", "JPEG" , "JPG" , "HEIF" , "heif" , "mp3" , "MP3" , "mp4" ,
"MP4"]
        file_found = False
        for suffix in suffixes :
            try:
                with open(f"{num}.{suffix}", 'rb') as file:
                    bot.send_document(message.chat.id, file)
                    file_found = True
                    menu(message)

            except FileNotFoundError:
                pass
        if not file_found:
            bot.send_message(message.chat.id , '🕐Розклад не знайдено')
            menu(message)

@bot.message_handler(func=lambda message: message.text == '🕐Встановити
розклад')
def setTable(message):
    if message.chat.id in admID:
```

```
        markup = types.ReplyKeyboardMarkup(row_width=2)
        item1 = types.KeyboardButton('1')
        item2 = types.KeyboardButton('2')
        item3 = types.KeyboardButton('3')
        item4 = types.KeyboardButton('4')
        item5 = types.KeyboardButton('5')
        item6 = types.KeyboardButton('🚫Назад')
        markup.add(item1,item2 , item3 , item4 , item5 , item6)
        bot.send_message(message.chat.id , '🧭Оберіть Курс якому ви хочете
змінити 🕐Розклад' , reply_markup=markup)
        @bot.message_handler(func=lambda message: message.text == '1')
        def setTableHandle(message):
            markup = types.ReplyKeyboardMarkup(row_width=2)
            item1 = types.KeyboardButton('🚫Назад')
            markup.add(item1)
            bot.send_message(message.chat.id , '📄Відправте файл з 🕐Розкладом'
, reply_markup=markup)
            bot.register_next_step_handler(message, handle_document1)
        def handle_document1(message):
                if message.document:
                    file_info = bot.get_file(message.document.file_id)
                    downloaded_file = bot.download_file(file_info.file_path)
                    file_extension = Path(file_info.file_path).suffix
                    filename = f"1{file_extension}"
                    with open(filename, 'wb') as new_file:
                        new_file.write(downloaded_file)
                    bot.send_message(message.chat.id , '✅Ви успішно завантажили
новий розклад')
                    setTable(message)
                else:
                    bot.send_message(message.chat.id, 'Будь ласка, відправте
файл.')
                    setTable(message)
        @bot.message_handler(func=lambda message: message.text == '2')
        def setTableHandle(message):
            markup = types.ReplyKeyboardMarkup(row_width=2)
            item1 = types.KeyboardButton('🚫Назад')
            markup.add(item1)
            bot.send_message(message.chat.id , '📄Відправте файл з 🕐Розкладом'
, reply_markup=markup)
            bot.register_next_step_handler(message, handle_document2)
        def handle_document2(message):
                if message.document:
                    file_info = bot.get_file(message.document.file_id)
                    downloaded_file = bot.download_file(file_info.file_path)
                    file_extension = Path(file_info.file_path).suffix
                    filename = f"2{file_extension}"
                    with open(filename, 'wb') as new_file:
                        new_file.write(downloaded_file)
```

```
                    bot.send_message(message.chat.id , '✅Ви успішно завантажили
новий розклад')
                    setTable(message)
                else:
                    bot.send_message(message.chat.id, 'Будь ласка, відправте
файл.')
                    setTable(message)
        @bot.message_handler(func=lambda message: message.text == '3')
        def setTableHandle(message):
            markup = types.ReplyKeyboardMarkup(row_width=2)
            item1 = types.KeyboardButton('🚫Назад')
            markup.add(item1)
            bot.send_message(message.chat.id , '📄Відправте файл з 🕐Розкладом'
, reply_markup=markup)
            bot.register_next_step_handler(message, handle_document3)
        def handle_document3(message):
                if message.document:
                    file_info = bot.get_file(message.document.file_id)
                    downloaded_file = bot.download_file(file_info.file_path)
                    file_extension = Path(file_info.file_path).suffix
                    filename = f"3{file_extension}"
                    with open(filename, 'wb') as new_file:
                        new_file.write(downloaded_file)
                    bot.send_message(message.chat.id , '✅Ви успішно завантажили
новий розклад')
                    setTable(message)
                else:
                    bot.send_message(message.chat.id, 'Будь ласка, відправте
файл.')
                    setTable(message)
        @bot.message_handler(func=lambda message: message.text == '4')
        def setTableHandle(message):
            markup = types.ReplyKeyboardMarkup(row_width=2)
            item1 = types.KeyboardButton('🚫Назад')
            markup.add(item1)
            bot.send_message(message.chat.id , '📄Відправте файл з 🕐Розкладом'
, reply_markup=markup)
            bot.register_next_step_handler(message, handle_document4)
        def handle_document4(message):
                if message.document:
                    file_info = bot.get_file(message.document.file_id)
                    downloaded_file = bot.download_file(file_info.file_path)
                    file_extension = Path(file_info.file_path).suffix
                    filename = f"4{file_extension}"
                    with open(filename, 'wb') as new_file:
                        new_file.write(downloaded_file)
                    bot.send_message(message.chat.id , '✅Ви успішно завантажили
новий розклад')
                    setTable(message)
                else:
```

```
                    bot.send_message(message.chat.id, 'Будь ласка, відправте
файл.')
                    setTable(message)
        @bot.message_handler(func=lambda message: message.text == '5')
        def setTableHandle(message):
            markup = types.ReplyKeyboardMarkup(row_width=2)
            item1 = types.KeyboardButton('🚫Назад')
            markup.add(item1)
            bot.send_message(message.chat.id , '📄Відправте файл з 🕐Розкладом'
, reply_markup=markup)
            bot.register_next_step_handler(message, handle_document5)
        def handle_document5(message):
                if message.document:
                    file_info = bot.get_file(message.document.file_id)
                    downloaded_file = bot.download_file(file_info.file_path)
                    file_extension = Path(file_info.file_path).suffix
                    filename = f"5{file_extension}"
                    with open(filename, 'wb') as new_file:
                        new_file.write(downloaded_file)
                    bot.send_message(message.chat.id , '✅Ви успішно завантажили
новий розклад')
                    setTable(message)
                else:
                    bot.send_message(message.chat.id, 'Будь ласка, відправте
файл.')
                    setTable(message)
    else:
        bot.send_message(message.chat.id , '🚫Вам відмовлено в доступі')
        menu(message)
@bot.message_handler(func=lambda message: message.text == '🔍Пошук студентів')
def searchStudents(message):
    if message.chat.id in admID:
        markup = types.ReplyKeyboardMarkup(row_width=2)
        item1 = types.KeyboardButton('🚫Назад')
        markup.add(item1)
        bot.send_message(message.chat.id , 'Введіть 🖍Прізвище студента...' ,
reply_markup=markup)
        bot.register_next_step_handler(message, searchStudentsHandle)
    else:
        bot.send_message(message.chat.id , '🚫Вам відмовлено в доступі')
        menu(message)
def searchStudentsHandle(message):
    if message.text != '🚫Назад':
            conn = sqlite3.connect('database.sql')
            cur = conn.cursor()
            cur.execute('SELECT name , surname , birthday , department ,
studentgroup , phoneNumber , email , status , course  FROM students WHERE surname
= ?', (message.text,))
            result = cur.fetchall()
            cur.close()
```

```
            conn.close()
            if not result:
                bot.send_message(message.chat.id , '🚫Помилка! Студента з таким прізвищем не знайдено! Спробуйте ще раз...')
                searchStudents(message)
            for row in result:
                name = row[0]
                surname = row[1]
                birthday = row[2]
                department = row[3]
                studentGroup = row[4]
                phoneNumber = row[5]
                email = row[6]
                status = row[7]
                course = row[8]
                info_message = f"Інформація студента\n🆕Ім'я: {name}\n🖍️Прізвище: {surname}\n🎂Дата народження: {birthday}\n🕶️Факультет: {department}\n👥Група: {studentGroup}\n🧐Курс: {course}\n🔢Номер телефону: {phoneNumber}\n📧Електрона скринька: {email}\n🚦Статус: {status}"
                bot.send_message(message.chat.id , info_message )
                menu(message)
@bot.message_handler(func=lambda message: message.text == '🎆Соц. Мережі')
def socials(message):
    markup = types.InlineKeyboardMarkup()
    button1 = types.InlineKeyboardButton(text="📫Instagram", url='https://www.instagram.com/fcit_nuoua?igsh=bHc2cmdxbHA1cDQ1')
    button2 = types.InlineKeyboardButton(text="🌐Веб сайт", url='https://onua.edu.ua/ua/')
    button3 = types.InlineKeyboardButton(text="🌐Веб сайт ФКІТ", url='http://moodle.onua.edu.ua')
    button4 = types.InlineKeyboardButton(text="📧Email", url='https://mail.google.com/mail/u/0/?view=cm&fs=1&to=vstup@onua.edu.ua')
    button5 = types.InlineKeyboardButton(text="📞Деканат", callback_data='phoneDecanat')
    markup.add(button1 , button2 , button3 , button4 , button5)
    bot.send_message(message.chat.id , 'Соціальні мережі навчального закладу:' , reply_markup=markup)
    menu(message)
@bot.message_handler(func=lambda message: message.text == '🚦Написати повідомлення студентам')
def whatTypeMessage(message):
    markup = types.ReplyKeyboardMarkup(row_width=2)
    item1 = types.KeyboardButton('📁Файл')
    item2 = types.KeyboardButton('📝Текст')
    item3 = types.KeyboardButton('🚫Назад')
    markup.add(item1 , item2 , item3)
    bot.send_message(message.chat.id , '⌨️Виберіть тип повідомлення яке ви хочете розіслати...' , reply_markup=markup)
@bot.message_handler(func=lambda message: message.text == '📝Текст')
def whatMessage(message):
```

```
    markup = types.ReplyKeyboardMarkup(row_width=2)
    item1 = types.KeyboardButton('🚫Назад')
    markup.add(item1)
    bot.send_message(message.chat.id , '⌨️Введіть повідомлення яке ви хочете
розіслати...' , reply_markup=markup)
    bot.register_next_step_handler(message , confirmMessage)
def confirmMessage(message):
    if message.text != '🚫Назад':
        messageData = message.text
        markup = types.ReplyKeyboardMarkup(row_width=2)
        item1 = types.KeyboardButton('🚫Назад')
        item2 = types.KeyboardButton('✅Так,відправити')
        markup.add(item1 , item2)
        bot.send_message(message.chat.id , f'Ви впевнені,що хочете відіслати
повідомлення:\n{messageData}' , parse_mode='HTML' , reply_markup=markup)
        @bot.message_handler(func=lambda message: message.text ==
'✅Так,відправити')
        def sendMessage(message):
            conn = sqlite3.connect('database.sql')
            cur = conn.cursor()
            cur.execute('SELECT userID  FROM students')
            result = cur.fetchall()
            cur.close()
            conn.close()
            result = [id[0] for id in result]
            for i in result:
                bot.send_message(i , 'Повідомлення від адміністратора:')
                bot.send_message(i , messageData)
            bot.send_message(message.chat.id , '✅Повідомлення успішно
надіслано!')
            menu(message)
    else:
        menu(message)
@bot.message_handler(func=lambda message: message.text == '📁Файл')
def sendFile(message):
    markup = types.ReplyKeyboardMarkup(row_width=2)
    item1 = types.KeyboardButton('🚫Назад')
    markup.add(item1)
    bot.send_message(message.chat.id , '📁Відправте файл який ви хочете
розіслати...' , reply_markup=markup)
    bot.register_next_step_handler(message, handle_document)

def handle_document(message):
    if message.document:
        file_info = bot.get_file(message.document.file_id)
        downloaded_file = bot.download_file(file_info.file_path)
        file_extension = Path(file_info.file_path).suffix
        filename = message.document.file_name
        with open(filename, 'wb') as new_file:
            new_file.write(downloaded_file)
```

```
            conn = sqlite3.connect('database.sql')
            cur = conn.cursor()
            cur.execute('SELECT userID  FROM students')
            result = cur.fetchall()
            cur.close()
            conn.close()
            result = [id[0] for id in result]
            for i in result:
                bot.send_message(i, 'Повідомлення від адміністратора:')
                bot.send_document(i , open(filename, 'rb'))
            bot.send_message(message.chat.id , '✅Файл успішно надісланий!')
            menu(message)
        elif message.audio:
            file_info = bot.get_file(message.audio.file_id)
            downloaded_file = bot.download_file(file_info.file_path)
            file_extension = Path(file_info.file_path).suffix
            filename = message.audio.file_name
            with open(filename, 'wb') as new_file:
                new_file.write(downloaded_file)
            conn = sqlite3.connect('database.sql')
            cur = conn.cursor()
            cur.execute('SELECT userID  FROM students')
            result = cur.fetchall()
            cur.close()
            conn.close()
            result = [id[0] for id in result]
            for i in result:
                bot.send_message(i, 'Повідомлення від адміністратора:')
                bot.send_audio(i , open(filename, 'rb'))
            bot.send_message(message.chat.id , '✅Файл успішно надісланий!')
            menu(message)
        else:
            bot.send_message(message.chat.id, 'Будь ласка, відправте файл.')
            sendFile(message)

bot.polling(none_stop=True)
```