Полтавський університет економіки і торгівлі

Навчально-науковий інститут денної освіти

Форма навчання денна

Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____Олена ОЛЬХОВСЬКА
(підпис)
«_____» ___202_ р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему

« РОЗРОБКА ЧАТ-БОТА ДЛЯ АВТОМАТИЗАЦІЇ СТУДЕНТСЬКОГО СЕРВІСУ »

зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня бакалавра

Виконавець роботи Хоменко Артем Андрійович
_____«_____»_____ 202_ р.
(підпис)

Науковий керівник зав. кафедри, к.ф.-м.н. Ольховська Олена Володимирівна
_____«_____»_____ 202_ р.
(підпис)

Рецензент

ПОЛТАВА 2026

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	Пояснення умовних позначень, скорочень, символів
Telegram Bot	Програмний засіб, який працює в месенджері Telegram та автоматично взаємодіє з користувачем за допомогою текстових повідомлень, команд і кнопок.
Telegram Bot API	Програмний інтерфейс, який надається Telegram для створення ботів та забезпечує можливість надсилання повідомлень, отримання запитів і взаємодії з користувачами.
API (Application Programming Interface)	Набір правил і методів, який дозволяє різним програмам обмінюватися даними та взаємодіяти між собою.
FSM (Finite State Machine)	Скінченний автомат, що використовується для організації послідовності дій користувача та керування станами під час роботи Telegram-бота.
JSON (JavaScript Object Notation)	Текстовий формат обміну даними, який використовується для передачі структурованої інформації між сервером і програмою.
HTTP (HyperText Transfer Protocol)	Протокол передачі даних у мережі Інтернет, який використовується для виконання запитів до API та отримання відповідей від серверів.
Python	Високорівнева мова програмування, яка використовується для реалізації логіки роботи Telegram-бота, обробки даних та взаємодії з API.
aiogram	Асинхронна бібліотека для мови

	програмування Python, призначена для створення Telegram-ботів на основі Telegram Bot API.
Асинхронне програмування	Підхід до розробки програмного забезпечення, який дозволяє виконувати декілька операцій одночасно без блокування основного потоку виконання програми.
Callback-запит	Спеціальний запит, який надсилається Telegram після натискання користувачем inline-кнопки та використовується для обробки дій користувача.
Inline-кнопка	Інтерактивний елемент інтерфейсу Telegram-бота, який дозволяє користувачеві виконувати певні дії без введення текстових команд.
Endpoint	Конкретна адреса API, до якої надсилається HTTP-запит для отримання або передачі даних.
URL (Uniform Resource Locator)	Унікальна адреса мережевого ресурсу, яка використовується для доступу до веб-сторінок та API-сервісів.
Requests	Бібліотека Python, яка використовується для виконання HTTP-запитів до зовнішніх сервісів та API
DevTools	Інструменти розробника браузера, які використовуються для аналізу мережевих запитів, дослідження API та налагодження веб-застосунків.

ВСТУП

У сучасних умовах розвитку інформаційних технологій важливого значення набуває автоматизація освітніх сервісів. Значна частина навчальної інформації сьогодні надається через електронні системи університетів, веб-сайти розкладу занять, електронні кабінети студентів та мобільні сервіси. Проте використання таких систем не завжди є достатньо зручним для користувачів, особливо при необхідності швидкого отримання інформації через мобільний пристрій.

Студенти щоденно користуються розкладом занять, перевіряють аудиторії, викладачів та час проведення пар. У більшості випадків для отримання такої інформації необхідно відкривати веб-сайт університету, вводити параметри пошуку та самостійно переглядати результати. Такий спосіб отримання інформації займає додатковий час та створює певні незручності під час використання мобільних пристроїв.

Одним із сучасних рішень даної проблеми є використання Telegram-чат-ботів. Telegram є одним із найпопулярніших месенджерів серед студентів, тому використання Telegram-бота дозволяє забезпечити швидкий доступ до навчальної інформації без необхідності використання браузера або окремого мобільного застосунку.

Актуальність теми полягає у необхідності створення швидкого та зручного інструменту доступу до інформації про розклад занять. Використання Telegram-бота дозволяє автоматизувати процес отримання розкладу, мінімізувати кількість ручних дій користувача та забезпечити швидкий доступ до інформації через смартфон.

У межах даної роботи було розроблено Telegram-чат-бот для автоматизації студентського сервісу Полтавського університету економіки

і торгівлі. Основним призначенням системи є отримання студентами актуального розкладу занять за групою або викладачем через Telegram.

Для реалізації Telegram-бота було використано мову програмування Python та бібліотеку aiogram 3. Для взаємодії із системою розкладу ПУЕТ використовуються HTTP-запити та API системи розкладу. Також у роботі застосовуються JSON-структури, асинхронне програмування та FSM-модель взаємодії користувача із системою.

Об’єктом дослідження є процес автоматизації студентського сервісу у закладах вищої освіти. Предметом дослідження є програмна реалізація Telegram-чат-бота для автоматизованого отримання розкладу занять через API системи розкладу ПУЕТ.

Метою роботи є розробка Telegram-чат-бота для автоматизації студентського сервісу, який забезпечує швидкий пошук розкладу занять за групою або викладачем через месенджер Telegram. У процесі виконання роботи було поставлено задачі дослідити принципи роботи Telegram-ботів, проаналізувати можливості Telegram Bot API, реалізувати взаємодію з API ПУЕТ, створити модульну структуру проєкту, реалізувати систему пошуку розкладу та забезпечити форматування повідомлень для Telegram.

Кваліфікаційна робота складається з чотирьох розділів. У першому розділі сформульовано постановку задачі, що полягає у створенні Telegram-чат-бота для автоматизації студентського сервісу та отримання розкладу занять через месенджер Telegram. У другому розділі розглянуто теоретичні основи роботи Telegram-чат-ботів, Telegram Bot API, асинхронного програмування, а також особливості роботи з API та JSON-даними. У третьому розділі здійснено аналіз системи розкладу ПУЕТ, API-запитів, JSON-структур та модульної структури програмного забезпечення. У четвертому розділі висвітлено процес розробки Telegram-бота, реалізацію взаємодії з API ПУЕТ, пошук розкладу за групою та викладачем, форматування повідомлень, систему збереження груп

користувачів і тестування програмного забезпечення. Обсяг пояснювальної записки: 49 стор., у т.ч. основна частина – 47 стор., джерела – 24 назв.

1. ПОСТАНОВКА ЗАДАЧІ

У сучасних умовах розвитку інформаційних технологій важливого значення набуває автоматизація освітніх сервісів. Значна частина навчальної інформації сьогодні надається через електронні системи університетів, веб-сайти розкладу занять, електронні кабінети студентів та мобільні сервіси. Проте доступ до такої інформації не завжди є достатньо швидким та зручним для користувачів.

Метою роботи є розробка Telegram-чат-бота для автоматизації студентського сервісу, який забезпечує швидкий пошук розкладу занять за групою або викладачем через месенджер Telegram.

Для досягнення поставленої мети необхідно вирішити такі задачі:

1. Проаналізувати предметну область студентського сервісу:
   • визначити, яку інформацію найчастіше потребують студенти;
   • обґрунтувати доцільність використання Telegram-бота;
   • визначити основні сценарії роботи користувача з ботом.
2. Обґрунтувати вибір технологій:
   • вибрати мову програмування;
   • обґрунтувати використання Python;
   • обґрунтувати використання бібліотеки aiogram 3;
   • визначити спосіб взаємодії з API ПУЕТ.
3. Спроєктувати структуру програмного забезпечення:
   • створити модульну структуру проєкту;
   • розділити код на обробники, сервіси, утиліти, клавіатури, стани та сховище;
   • визначити роль кожного файлу у проєкті.
4. Реалізувати Telegram-бота:
   • створити запуск бота у файлі main.py;

• підключити router-модулі;

• реалізувати команду /start;

• створити inline-кнопки для вибору типу пошуку та дати.

5. Реалізувати пошук розкладу:

• створити функції запиту до API ПУЕТ у файлі app/services/puet_api.py;

• реалізувати пошук груп;

• реалізувати пошук викладачів;

• реалізувати отримання розкладу за вибраною датою.

6. Реалізувати обробку введених даних:

• створити функцію normalize_text() для нормалізації назв груп;

• створити функцію split_tokens() для пошуку схожих варіантів;

• створити функцію parse_date() для ручного введення дати.

7. Реалізувати форматування результатів:

• створити функцію format_group_schedule();

• створити функцію format_teacher_schedule();

• забезпечити коректне відображення предмета, аудиторії, часу, викладача та груп.

8. Реалізувати збереження груп користувачів:

• створити функції save_user_group(), get_user_group(), delete_user_group();

• організувати збереження даних у файлі users_groups.json.

9. Провести тестування:

• перевірити пошук за групою;

• перевірити пошук за викладачем;

• перевірити ручне введення дати;

• перевірити збереження та видалення групи;

• перевірити обробку помилок API.

У процесі виконання роботи було використано мову програмування Python, бібліотеку aiogram 3, Telegram Bot API, бібліотеку requests, JSON-файли та асинхронне програмування. Результатом виконання роботи є Telegram-чат-бот, який автоматизує процес отримання розкладу занять та забезпечує швидкий доступ студентів до навчальної інформації через Telegram.

2. ТЕОРЕТИЧНА ЧАСТИНА

2.1 Теоретичні основи чат-ботів

Чат-бот — це програмний засіб, який забезпечує автоматичну взаємодію користувача з інформаційною системою через текстові повідомлення або інтерактивні елементи інтерфейсу.

Сьогодні чат-боти активно використовуються у:

- банківських системах;
- онлайн-магазинах;
- службах підтримки;
- інформаційних сервісах;
- освітніх платформах.

Особливого поширення чат-боти набули в освітній сфері. Навчальні заклади використовують їх для інформування студентів про розклад занять, зміни в навчальному процесі, проведення опитувань та надання довідкової інформації. Використання чат-ботів дозволяє зменшити навантаження на працівників деканатів та інформаційних служб, оскільки значна частина типових запитів обробляється автоматично.

Однією з основних переваг використання чат-ботів в освіті є доступність. Студент може отримати необхідну інформацію у будь-який час доби без необхідності звернення до співробітників університету. Крім того, чат-боти забезпечують швидке отримання відповідей та зручну взаємодію через звичний для користувача месенджер. Саме тому використання Telegram-ботів розглядається як перспективний напрям розвитку сучасних студентських сервісів.

У межах даної роботи Telegram-бот використовується для автоматизації студентського сервісу. Користувач взаємодіє з ботом через Telegram, а система автоматично виконує пошук розкладу занять та повертає результат у вигляді повідомлення.

Telegram-бот дозволяє:

- швидко отримувати інформацію;
- використовувати інтерактивні кнопки;
- реалізувати багатокрокову взаємодію;
- працювати через смартфон;
- автоматизувати типові дії користувача.

У розробленому проєкті бот підтримує:

- пошук за групою;
- пошук за викладачем;
- вибір дати;
- збереження груп користувачів;
- форматування результатів.

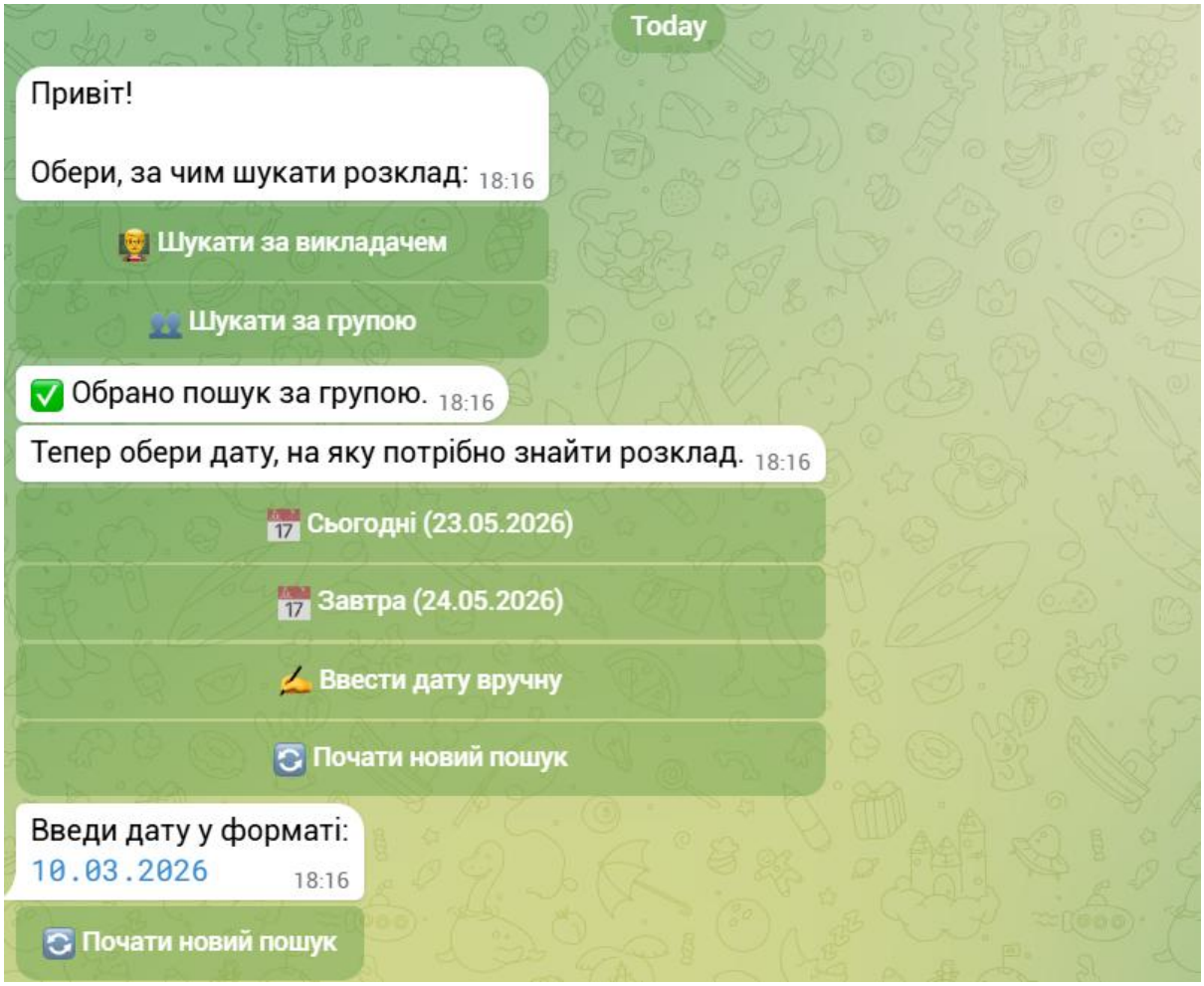


Рисунок 2.1 – Приклад роботи Telegram-бота

Використання Telegram-бота дозволяє значно спростити доступ студентів до навчальної інформації та зменшити кількість дій, необхідних для отримання розкладу занять.

2.2 Telegram Bot API та принципи взаємодії з користувачем

Telegram Bot API є програмним інтерфейсом, який дозволяє створювати Telegram-ботів та взаємодіяти з користувачами через сервери Telegram.[3]

Робота Telegram-бота базується на:

- отриманні повідомлень;
- обробці callback-запитів;

- надсиланні відповідей;
- використанні inline-кнопок.

У межах розробленого проєкту Telegram API використовується для:

- отримання повідомлень користувачів;
- надсилання результатів пошуку;
- створення inline-клавіатур;
- обробки callback-кнопок.

Для взаємодії з Telegram API у проєкті використовується бібліотека aiogram 3. Запуск Telegram-бота реалізовано у файлі - main.py та запуск Telegram-бота в самому коді: await dp.start_polling(bot)

Telegram-боти можуть працювати у двох основних режимах отримання повідомлень від користувачів: Polling та Webhook.[17]

Режим Polling передбачає періодичне звернення бота до серверів Telegram з метою перевірки наявності нових повідомлень. Якщо користувач надсилає повідомлення або натискає кнопку, Telegram зберігає відповідну інформацію, а бот під час чергового звернення отримує ці дані для подальшої обробки. Основною перевагою Polling є простота налаштування та відсутність необхідності використовувати окремий веб-сервер або зовнішній домен.

Webhook працює за іншим принципом. У цьому випадку Telegram самостійно надсилає інформацію про нові події на заздалегідь визначену адресу сервера. Такий підхід дозволяє швидше отримувати повідомлення та зменшує кількість зайвих запитів до серверів Telegram. Проте для використання Webhook необхідна наявність публічного сервера з налаштованим HTTPS-з’єднанням.

У межах даного проєкту було використано режим Polling, оскільки він є більш простим для розробки, тестування та локального запуску Telegram-бота. Використання Polling дозволило реалізувати необхідний

функціонал без додаткового налаштування серверної інфраструктури та забезпечити стабільну роботу програмного забезпечення під час розробки.

Polling-режим в боті постійно перевіряє наявність нових повідомлень. Однією з важливих можливостей Telegram Bot API є підтримка inline-кнопок.

Створення inline-кнопки:

```
InlineKeyboardButton(
  text="Пошук за групою",callback_data="group_search"
)
```

Inline-кнопки дозволяють реалізувати більш зручну взаємодію користувача з ботом та мінімізують кількість ручного введення.

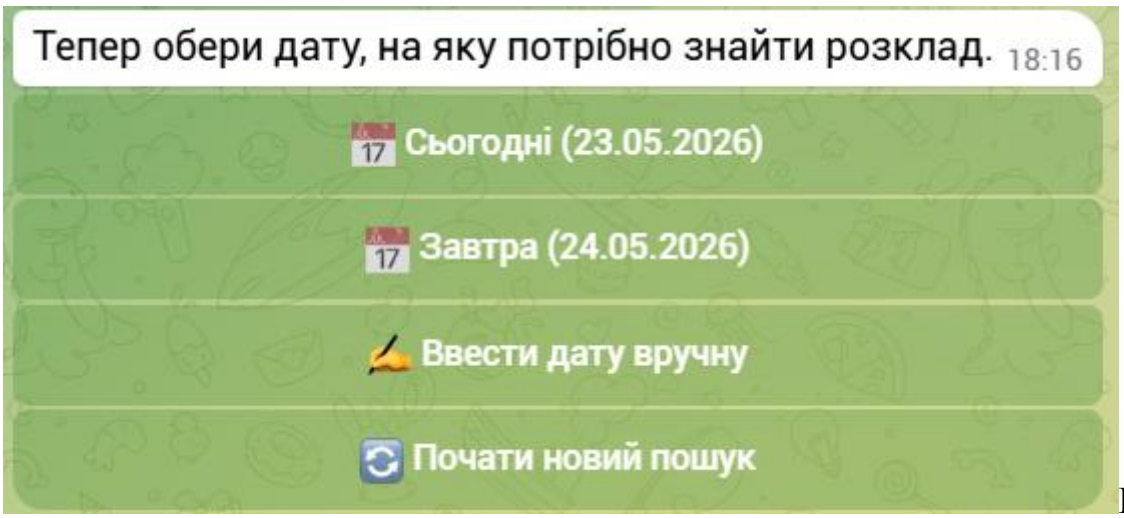


Рисунок 2.2 – Inline-кнопки Telegram-бота

Telegram Bot API також підтримує callback-запити, які використовуються для обробки натискань кнопок користувачем.

2.3 Обґрунтування вибору мови програмування Python

Для реалізації Telegram-чат-бота було обрано мову програмування Python.[5,6]

Основними причинами вибору є:

- простий синтаксис;
- підтримка асинхронного програмування;
- велика кількість бібліотек;
- зручність роботи з API;
- підтримка JSON;
- швидка розробка програмного забезпечення.

Python активно використовується для:

- автоматизації;
- створення Telegram-ботів;
- роботи з API;
- веб-розробки;
- обробки даних.

У межах розробленого проєкту Python використовується для:

- створення логіки Telegram-бота;
- виконання HTTP-запитів;
- обробки JSON-відповідей;
- форматування повідомлень;
- реалізації FSM-моделі.

У проєкті використовуються бібліотеки:

- aiogram;
- requests;
- asyncio;
- json;
- datetime;
- logging;

Бібліотека requests використовується для роботи з API ПУЕТ.

HTTP-запит до API:

```
response = requests.get(
    url,
    headers={"token": TOKEN}
)
```

Бібліотека json використовується для роботи з JSON-файлом: users_groups.json Бібліотека asyncio забезпечує асинхронну роботу Telegram-бота. Використання Python дозволило реалізувати модульну структуру проєкту та спростити підтримку коду.

2.4 Обґрунтування вибору бібліотеки aiogram

Для створення Telegram-ботів мовою Python існує декілька популярних бібліотек. Найбільш відомими серед них є python-telegram-bot, PyTelegramBotAPI та Telethon.

Бібліотека python-telegram-bot має широкий набір інструментів для роботи з Telegram Bot API та підтримує створення складних проєктів. Проте її архітектура суттєво відрізняється від aiogram, а деякі механізми реалізації багатокрокових сценаріїв потребують додаткового налаштування. Для даного проєкту важливим фактором була наявність зручної системи станів FSM, яка реалізована в aiogram на високому рівні.

PyTelegramBotAPI відзначається простотою використання та невисоким порогом входження для початківців. Однак дана бібліотека більше орієнтована на невеликі проєкти та менш активно використовує можливості асинхронного програмування. У розроблюваному Telegram-боті передбачено одночасну роботу з API ПУЕТ, обробку повідомлень користувачів та багатокрокові сценарії взаємодії, тому використання асинхронної архітектури є важливим.

Бібліотека Telethon призначена переважно для роботи з повним Telegram API та створення клієнтських застосунків. Її функціональність

значно ширша, ніж необхідно для реалізації звичайного Telegram-бота, що ускладнює розробку та збільшує обсяг коду.

Таким чином, саме aiogram найбільш повно відповідає вимогам розроблюваного проєкту завдяки підтримці асинхронного програмування, FSM-моделі, модульної архітектури та активному розвитку спільноти розробників. Для створення Telegram-бота у проєкті використовується бібліотека aiogram 3.[4]

Дана бібліотека підтримує:

- асинхронне програмування;
- FSM;
- router-систему;
- callback-запити;
- inline-клавіатури.

У межах проєкту aiogram 3 використовується для:

- запуску Telegram-бота;
- обробки повідомлень;
- обробки callback-кнопок;
- реалізації FSM;
- створення клавіатур.

Підключення router-модулів виконується у файлі main.py.

Підключення router-модулів:

```
dp.include_router(start_router)
dp.include_router(callbacks_router)
dp.include_router(group_router)
dp.include_router(teacher_router)
```

У наведеному коді підключаються окремі router-модулі Telegram-бота. Такий підхід дозволяє розділити логіку системи між різними файлами.

FSM-стани реалізовані у файлі: app/states/search_states.py
FSM використовується для:

- вибору типу пошуку;
- вибору дати;
- введення назви групи;
- введення прізвища викладача.

Використання aiogram 3 дозволило створити сучасну структуру Telegram-бота та забезпечити правильну послідовність взаємодії користувача із системою.

2.5 Асинхронне програмування в Telegram-ботах

Традиційний підхід до виконання програм називається синхронним програмуванням. У цьому випадку всі операції виконуються послідовно одна за одною. Поки одна операція не завершиться, наступна не може розпочати виконання. Такий підхід є простим для реалізації, проте при роботі з мережевими запитами або великою кількістю користувачів може призводити до затримок у роботі системи.

Асинхронне програмування дозволяє виконувати декілька операцій паралельно без блокування основного процесу роботи програми. Якщо система очікує відповідь від сервера або виконує іншу тривалу операцію, вона може продовжувати обробляти запити інших користувачів. Завдяки цьому значно підвищується швидкодія та ефективність використання ресурсів.[8]

Основна різниця між синхронним та асинхронним підходами полягає у способі виконання задач. Синхронне програмування виконує операції послідовно, тоді як асинхронне дозволяє виконувати декілька процесів одночасно під час очікування завершення інших операцій. Для Telegram-ботів це особливо важливо, оскільки система повинна

оперативно реагувати на повідомлення багатьох користувачів та одночасно виконувати запити до зовнішніх API.

Telegram-бот одночасно працює з великою кількістю повідомлень, callback-запитів та HTTP-запитів до API. Для забезпечення стабільної роботи системи використовується асинхронне програмування. Використовується бібліотека: asyncio, яка дає змогу використовувати асинхронні функції. Асинхронність реалізована через використання конструкцій: async def та await

Асинхронна функція запуску бота:

```
async def main():
    bot = Bot(token=BOT_TOKEN)
    dp = Dispatcher()

    await dp.start_polling(bot)
```

Асинхронне програмування дозволяє:

- одночасно обробляти декілька запитів;
- уникнути блокування програми;
- забезпечити швидку роботу Telegram-бота;
- коректно працювати з API.

У процесі роботи декілька користувачів можуть одночасно:

- виконувати пошук;
- надсилати повідомлення;
- отримувати результати;
- натискати callback-кнопки.

Без використання асинхронного програмування Telegram-бот міг би працювати повільніше або тимчасово блокувати обробку інших повідомлень. Використання асинхронного програмування дозволило забезпечити стабільну роботу Telegram-бота та швидку взаємодію користувачів із системою.

2.6 Робота з API та JSON-даними

Однією з важливих частин розробленого Telegram-бота є взаємодія з API системи розкладу ПУЕТ. API використовується для отримання актуальної інформації про групи, викладачів та розклад занять.

API — це програмний інтерфейс, який дозволяє одній програмі отримувати дані з іншої системи. У межах даної роботи Telegram-бот надсилає HTTP-запити до системи розкладу ПУЕТ, отримує відповідь у форматі JSON, обробляє її та формує повідомлення для користувача.[11]

JSON-формат є зручним способом передавання структурованих даних. Він дозволяє зберігати інформацію у вигляді об’єктів, списків, рядків і числових значень.[9]

У проєкті використовуються такі API-запити:

- отримання списку груп;
- отримання списку викладачів;
- отримання розкладу занять.

Для виконання HTTP-запитів використовується бібліотека requests.[7]

Приклад HTTP-запиту до API:

```
response = requests.get(
   url,
   headers={"token": TOKEN}
)
```

У наведеному фрагменті коду виконується GET-запит до API. Додатково передається token-заголовок, який необхідний для авторизації запиту в системі розкладу ПУЕТ. Після виконання запиту система перевіряє код відповіді сервера. Якщо відповідь успішна, дані перетворюються у JSON-формат.

Перетворення відповіді у JSON: data = response.json()

У Telegram-боті JSON використовується для:

- обробки відповіді API ПУЕТ;
- отримання назв груп;
- отримання списку викладачів;
- обробки розкладу занять;
- збереження груп користувачів.

Для збереження груп користувачів використовується файл: users_groups.json Робота з цим файлом реалізована у модулі: app/storage/users_storage.py У цьому модулі реалізовано функції збереження, отримання та видалення груп користувачів. Завдяки цьому користувач може один раз зберегти свою групу, а під час наступного використання бот запропонує використати її повторно.

Приклад збереження даних у JSON-файл:

```
with open(USERS_FILE, "w", encoding="utf-8") as f:
    json.dump(data, f, ensure_ascii=False, indent=4)
```

Параметр ensure_ascii=False дозволяє коректно зберігати українські символи у файлі. Параметр indent=4 робить JSON-файл більш зручним для читання.

При роботи з API важливо враховувати, що структура JSON-відповіді може відрізнятися залежно від типу запиту. Наприклад, інформація про викладача, аудиторію або групу може бути представлена у вигляді рядка, списку або словника. Тому у проєкті було реалізовано додаткову обробку таких даних у модулі форматування. Таким чином, використання API та JSON-даних дозволило реалізувати автоматичне отримання актуального розкладу занять та забезпечити збереження даних користувачів у Telegram-боті.

3. ІНФОРМАЦІЙНИЙ ОГЛЯД

3.1 Аналіз існуючих систем розкладу занять

У більшості закладів вищої освіти інформація про розклад занять надається через:

- веб-сайти;
- електронні кабінети;
- мобільні застосунки;
- Google-сервіси;
- Telegram-боти.

Основним джерелом інформації про розклад у Полтавському університеті економіки і торгівлі є веб-сайт системи розкладу: https://schedule.puet.edu.ua/

Веб-сайт дозволяє:

- виконувати пошук за групою;
- виконувати пошук за викладачем;
- переглядати інформацію про заняття;
- отримувати актуальний розклад.

Проте використання веб-сайту має ряд недоліків:

- необхідність постійного відкриття браузера;
- ручне введення параметрів пошуку;
- незручність використання через смартфон;
- велика кількість дій для отримання інформації;
- відсутність інтерактивної взаємодії.

Для перегляду розкладу користувачу необхідно:

1. Відкрити сайт.

2. Перейти до сторінки розкладу.
3. Вибрати тип пошуку.
4. Ввести параметри.
5. Обрати дату.
6. Переглянути результати.

Також було проаналізовано Telegram-боти інших університетів. Більшість із них підтримують лише базовий функціонал:

- перегляд розкладу;
- вибір групи;
- прості кнопки навігації.

У багатьох випадках відсутні:

- FSM-модель;
- система збереження груп;
- гнучкий пошук;
- обробка схожих назв;
- підтримка ручного введення дати.

У результаті аналізу було встановлено, що Telegram-бот є більш зручним рішенням для автоматизації студентського сервісу, оскільки дозволяє швидше отримувати інформацію через мобільний месенджер.

3.2 Порівняння Telegram-бота та веб-сайту

Під час аналізу було проведено порівняння між Telegram-ботом та традиційним веб-сайтом системи розкладу.

Telegram-бот має ряд переваг:

- швидший доступ до інформації;
- підтримка inline-кнопок;
- робота через мобільний месенджер;

- інтерактивна взаємодія;
- збереження груп користувачів;
- мінімальна кількість дій для пошуку.

На відміну від веб-сайту, Telegram-бот дозволяє отримати інформацію без відкриття браузера.

Користувачеві достатньо:

1. Відкрити Telegram.
2. Обрати тип пошуку.
3. Вибрати дату.
4. Отримати результат.

У межах проєкту реалізовано систему збереження груп користувачів, що дозволяє повторно використовувати раніше введені параметри. Також Telegram-бот підтримує callback-кнопки, які значно спрощують взаємодію користувача із системою.

Основними перевагами Telegram-бота є:

- мобільність;
- швидкість;
- інтерактивність;
- автоматизація;
- простота використання.

3.3 Аналіз API системи розкладу ПУЕТ

Для отримання актуальної інформації про розклад занять Telegram-бот використовує API системи розкладу ПУЕТ. Під час аналізу системи розкладу було досліджено HTTP-запити, які використовуються веб-сайтом університету.

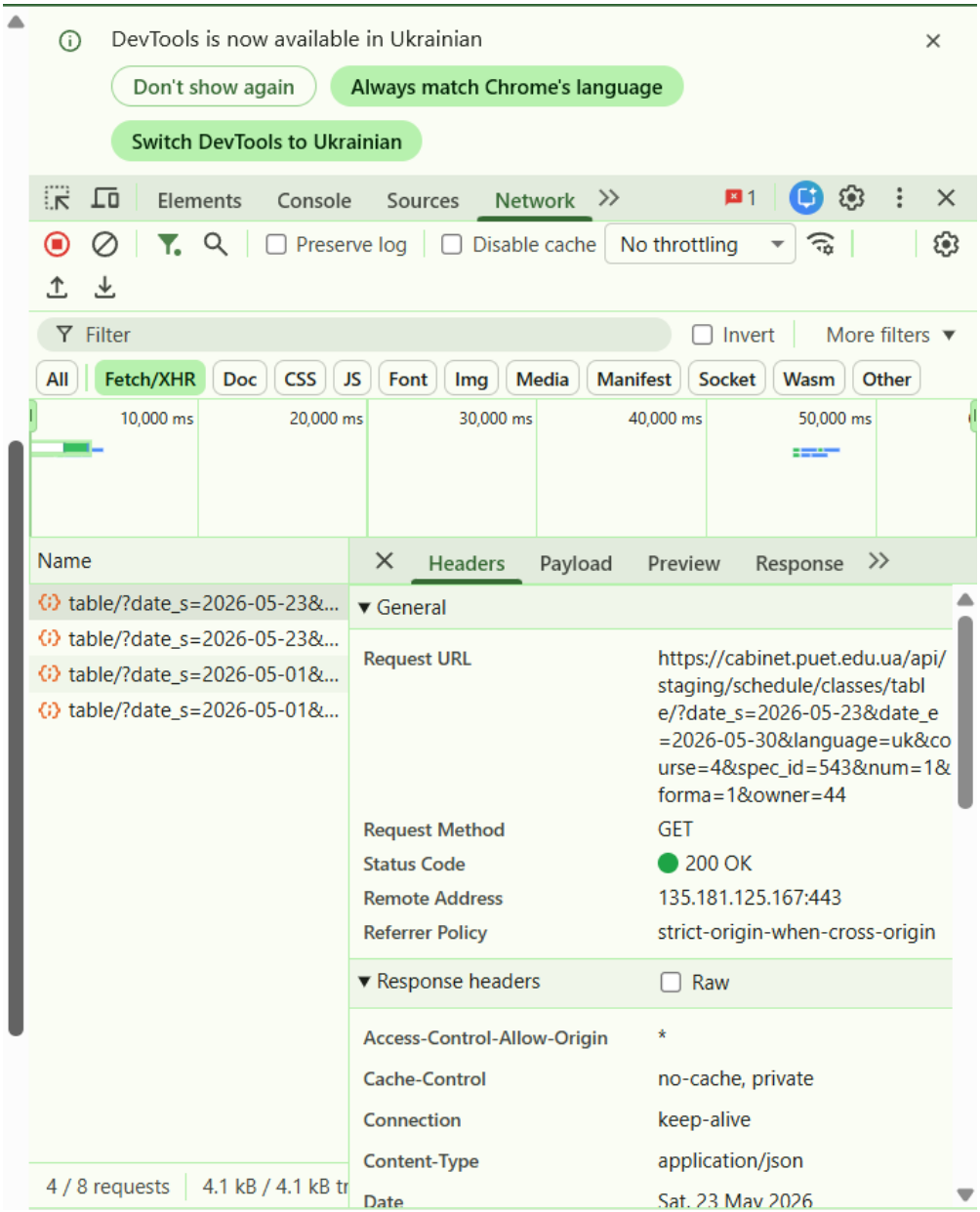


Рисунок 3.1 – Аналіз API через DevTools[20]

Для аналізу API використовувались:

- DevTools;
- вкладка Network;
- Fetch/XHR-запити.

У результаті аналізу були визначені основні API endpoint:

API для отримання списку груп:

https://cabinet.puet.edu.ua/api/v1/schedule/groups

API для отримання списку викладачів:

https://cabinet.puet.edu.ua/api/v1/schedule/teachers

API для отримання розкладу:

https://cabinet.puet.edu.ua/api/staging/schedule/classes/table/

Під час тестування було встановлено, що деякі API-запити потребують token-заголовок для авторизації. В коді використовується token-заголовок, який дозволяє отримати доступ до API системи розкладу. Після отримання відповіді система перевіряє статус HTTP-запиту: if response.status_code == 200 та перетворює JSON-відповідь: data = response.json() Використання API дозволило автоматично отримувати актуальні дані без необхідності ручного оновлення інформації у Telegram-боті. Важливою перевагою використання API є можливість отримання актуальних даних без необхідності аналізу HTML-коду веб-сторінок. У випадку використання традиційного веб-парсингу будь-яка зміна структури сайту може призвести до порушення роботи програмного забезпечення. Використання API дозволяє отримувати структуровані дані у стандартизованому форматі, що значно спрощує процес їх подальшої обробки.

У процесі роботи Telegram-бота використовується клієнт-серверна архітектура. Користувач надсилає запит через месенджер Telegram, після чого бот формує HTTP-запит до API системи розкладу ПУЕТ. Сервер обробляє запит та повертає відповідь у форматі JSON. Після цього Telegram-бот аналізує отримані дані, виконує необхідне форматування та надсилає результат користувачу у вигляді текстового повідомлення. Загальна схема роботи системи має такий вигляд: користувач → Telegram-бот → API системи розкладу ПУЕТ → Telegram-бот → користувач.

Проаналізувавши API було встановлено, що система підтримує окремі кінцеві точки (endpoint) для отримання списку груп, викладачів та розкладу занять. Такий підхід дозволяє зменшити обсяг передаваних

даних та підвищити швидкість роботи системи. Отримані результати підтвердили можливість використання API як основного джерела інформації для реалізації Telegram-бота автоматизації студентського сервісу.

3.4 Аналіз JSON-структури відповіді API

Після отримання відповіді від API Telegram-бот повинен правильно обробити JSON-структуру та сформувати повідомлення для користувача. JSON-відповідь містить інформацію про:

- дату;
- список занять;
- аудиторії;
- викладачів;
- групи;
- типи занять;
- час проведення пар.

Для роботи з JSON-структурою у проєкті використовується модуль: app/utils/formatters.py

Основними задачами модуля є:

- аналіз JSON-відповіді;
- витягування необхідних даних;
- форматування повідомлень;
- створення тексту для Telegram.

В ході аналізу JSON-відповідей було встановлено, що структура даних має вкладений характер. Це означає, що окремі об'єкти можуть містити всередині себе інші об'єкти або масиви даних. Наприклад,

інформація про заняття може містити назву дисципліни, тип заняття, аудиторію, викладача, групу та часові параметри проведення пари.[10]
У процесі обробки JSON система отримує:

- назву предмета;
- викладача;
- аудиторію;
- групу;
- час проведення пари.

Особливістю роботи з API ПУЕТ є те, що окремі поля можуть повертатися у різних форматах залежно від типу запиту або версії відповіді сервера. Зокрема, інформація про викладачів може бути представлена через поля teacher, teachers, prep або preps. У зв'язку з цим у процесі розробки було реалізовано додаткові механізми перевірки структури даних та визначення типу отриманого об'єкта.

Під час тестування було виявлено проблему, при якій Telegram-бот у деяких випадках відображав повідомлення: “Не вказано”. Причиною помилки була різна структура JSON-відповідей API. Для забезпечення коректної роботи Telegram-бота було створено універсальні функції обробки JSON-структур, які дозволяють працювати з рядками, словниками та списками. Такий підхід значно підвищує стійкість програмного забезпечення до змін структури API та забезпечує правильне відображення інформації для користувачів. Аналіз JSON-відповідей також дозволив визначити оптимальний набір даних, необхідний для формування розкладу занять. До нього належать назва дисципліни, час проведення заняття, аудиторія, викладач та навчальна група. Саме ці дані використовуються Telegram-ботом для формування зрозумілого та зручного повідомлення для кінцевого користувача. Реалізація окремого модуля форматування дозволила забезпечити коректне відображення інформації у Telegram-боті незалежно від структури JSON-відповіді.

3.5 Проєктування структури програмного забезпечення

Під час розробки Telegram-бота було прийнято рішення використовувати модульну структуру проєкту.[19]

Такий підхід дозволяє:

- розділити логіку системи;
- уникнути дублювання коду;
- спростити підтримку проєкту;
- забезпечити можливість подальшого розширення.

Основний файл main.py відповідає за:

- запуск Telegram-бота;
- підключення router-модулів;
- запуск polling.

Папка handlers містить:

- обробку повідомлень;
- callback-запити;
- команду /start;
- пошук груп;
- пошук викладачів.

Папка services використовується для роботи з API ПУЕТ.

У папці utils знаходяться допоміжні функції:

- форматування повідомлень;
- нормалізація тексту;
- обробка дат.

Окрему роль у структурі програмного забезпечення відіграє механізм керування станами користувача.[18] Для реалізації послідовної

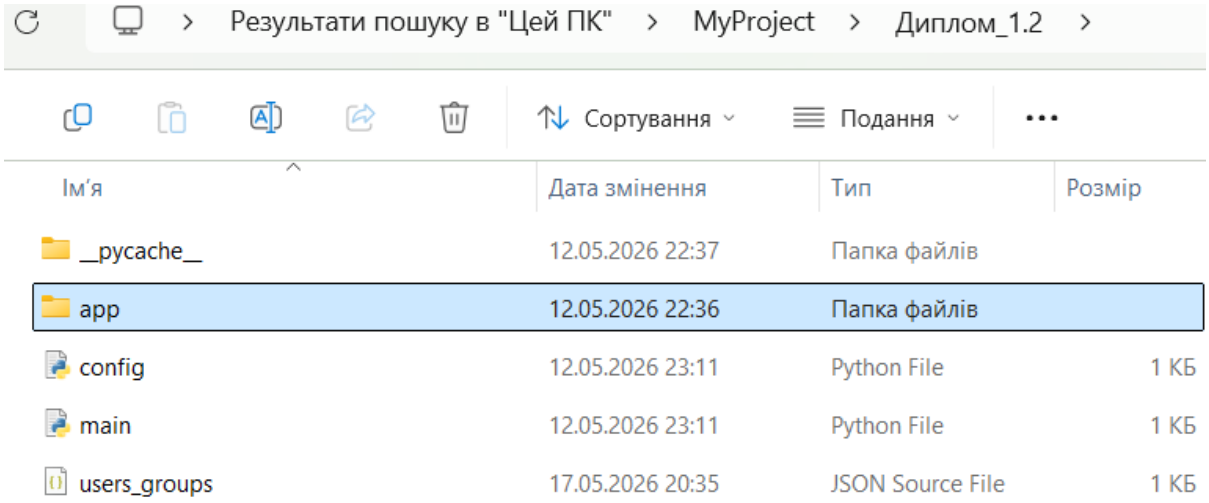
взаємодії між користувачем та Telegram-ботом використовується скінченний автомат станів FSM (Finite State Machine).[16]

Використання FSM дозволяє розділити процес роботи користувача на окремі етапи та забезпечити правильну послідовність виконання дій. Після запуску бота користувач переходить до стану вибору способу пошуку розкладу. Далі система очікує вибір дати, після чого виконується введення назви групи або прізвища викладача. На кожному етапі Telegram-бот контролює коректність введених даних та визначає подальший сценарій роботи.

FSM-стани реалізовані у папці: app/states/

Система збереження груп користувачів реалізована у папці: app/storage/

Структура проєкту має вигляд:



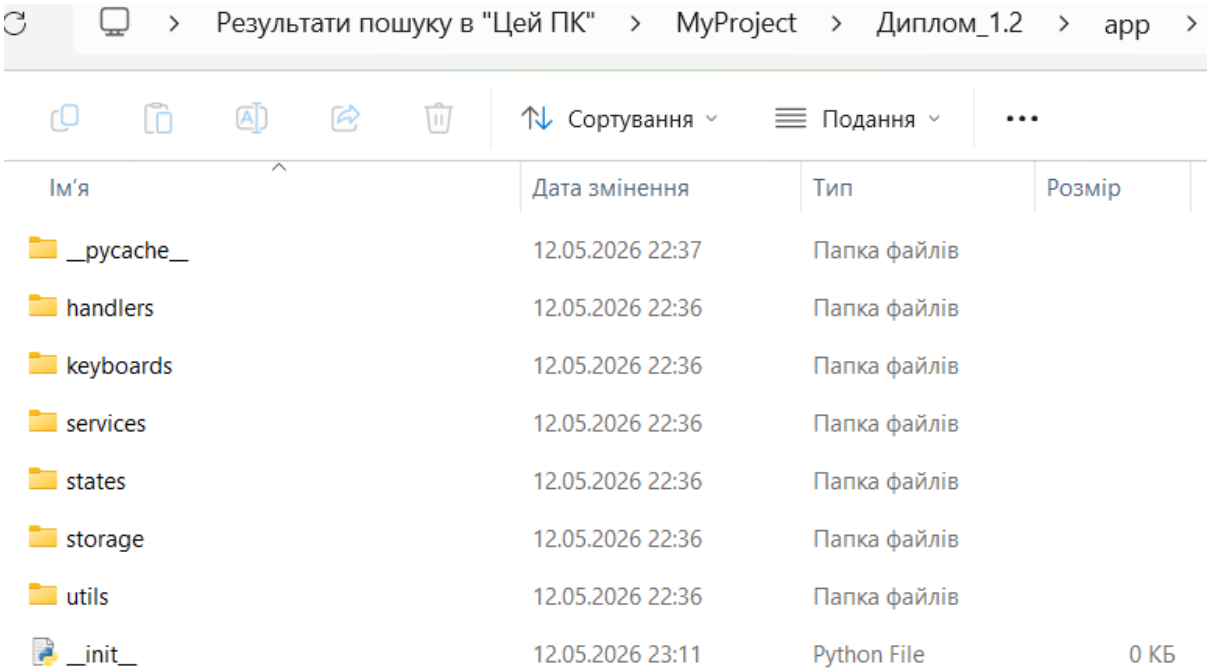


Рисунок 3.2 – Структура проєкту Telegram-бота

Застосування модульної структури проєкту спільно з FSM-моделлю дозволило створити масштабовану архітектуру програмного забезпечення, яка відповідає сучасним вимогам до розробки інформаційних систем та забезпечує зручність подальшої підтримки програмного продукту.

4. ПРАКТИЧНА ЧАСТИНА

4.1 Реалізація структури Telegram-бота та запуск системи

Під час розробки чат-бота для автоматизації студентського сервісу ПУЕТ було використано модульний підхід до організації програмного коду. Основною причиною такого рішення стала необхідність розподілу функціоналу між окремими компонентами системи. Оскільки Telegram-бот виконує декілька різних задач, зокрема обробку повідомлень користувачів, взаємодію з API університету, форматування розкладу та збереження даних, розміщення всього коду в одному файлі значно ускладнило б його підтримку та подальший розвиток.

Для обробки повідомлень користувачів використовується папка handlers. У ній реалізовано обробники команд, callback-запитів та текстових повідомлень. Саме модулі цієї папки відповідають за реакцію бота на натискання кнопок «Пошук за групою», «Пошук за викладачем», вибір дати та інші дії користувача.

Для взаємодії з API системи розкладу ПУЕТ використовується папка services. У модулі puet_api.py реалізовано функції отримання списку груп, списку викладачів та розкладу занять. Окремо використовується модуль search_service.py, який виконує пошук груп та викладачів за введеними користувачем даними.

Папка utils містить допоміжні функції, які використовуються в різних частинах проєкту. Зокрема, функція normalize_text() виконує нормалізацію тексту для коректного пошуку груп та викладачів, а модуль formatters.py відповідає за формування зручного для користувача вигляду розкладу занять.

Для створення інтерфейсу користувача використовується папка keyboards. У ній реалізовано генерацію inline-кнопок для вибору режиму пошуку, дати перегляду розкладу, вибору групи зі списку схожих результатів та повторного запуску пошуку.

Життєвий цикл запуску Telegram-бота починається з виконання файлу main.py. Після запуску створюється об'єкт Bot, який забезпечує взаємодію з Telegram Bot API. Далі створюється об'єкт Dispatcher, що відповідає за маршрутизацію повідомлень між обробниками. Наступним етапом є підключення Router-модулів, у яких зареєстровано всі обробники команд та повідомлень. Після завершення ініціалізації запускається режим Polling, у межах якого бот періодично перевіряє наявність нових повідомлень на серверах Telegram. Після запуску Polling система переходить у режим очікування повідомлень та готова до обробки запитів користувачів.

Код запуску Telegram-бота(файл: main.py):

```
import asyncio

from aiogram import Bot, Dispatcher

from config import BOT_TOKEN
from app.handlers.start import router as start_router
from app.handlers.callbacks import router as callbacks_router
from app.handlers.group import router as group_router
from app.handlers.teacher import router as teacher_router


async def main():
    bot = Bot(token=BOT_TOKEN)
    dp = Dispatcher()
```

```
    dp.include_router(start_router)
    dp.include_router(callbacks_router)
    dp.include_router(group_router)
    dp.include_router(teacher_router)

    await dp.start_polling(bot)


if __name__ == "__main__":
    asyncio.run(main())
```

У наведеному коді створюється об’єкт Bot, який використовується для взаємодії з Telegram API через токен бота. Об’єкт Dispatcher відповідає за маршрутизацію повідомлень користувача та callback-запитів. Для розділення логіки системи використовуються router-модулі:

- start.py;
- callbacks.py;
- group.py;
- teacher.py.

Підключення router-модулів виконується через:
dp.include_router()
Такий підхід дозволяє:

- уникнути дублювання коду;
- спростити підтримку системи;
- реалізувати окремі логічні модулі;
- забезпечити можливість подальшого розширення проєкту.

Для реалізації багатокрокової взаємодії використовується папка states. У ній описані стани FSM, які дозволяють визначати поточний етап роботи користувача з ботом. Завдяки цьому система розуміє, чи очікується

введення дати, назви групи або прізвища викладача. FSM-стани реалізовані у файлі: app/states/search_states.py

FSM використовується для:

- вибору типу пошуку;
- вибору дати;
- введення назви групи;
- введення викладача.

У процесі розробки також використовувалось асинхронне програмування. Асинхронність реалізована через використання:

- async def;
- await;
- бібліотеки asyncio.

Використання асинхронного програмування дозволило забезпечити стабільну роботу Telegram-бота при одночасній обробці декількох запитів користувачів.

4.2 Реалізація взаємодії з API ПУЕТ

Для отримання актуальної інформації про розклад занять Telegram-бот взаємодіє з API системи розкладу ПУЕТ. У процесі роботи використовуються окремі URL-адреси для отримання списку груп, списку викладачів та безпосередньо розкладу занять. Базова URL-адреса визначається в модулі puet_api.py, після чого до неї додається шлях до необхідного ресурсу API. Основний модуль роботи з API реалізовано у файлі: app/services/puet_api.py

У модулі реалізовано функції:

- get_groups();
- get_teachers();

- get_schedule_by_group();
- get_schedule_by_teacher().

Для виконання HTTP-запитів використовується бібліотека - requests

Для доступу до API використовується token, який передається в заголовках HTTP-запиту. Передача токена дозволяє серверу перевірити коректність запиту та надати доступ до необхідних даних. Без використання токена сервер повертає помилку авторизації, тому його наявність є обов'язковою умовою роботи Telegram-бота. Після отримання відповіді Telegram-бот перевіряє статус HTTP-запиту: if response.status_code == 200: та перетворює JSON-відповідь: data = response.json()[13,14]

У процесі взаємодії із зовнішнім API можуть виникати різні помилки. Для забезпечення стабільної роботи Telegram-бота реалізовано механізми обробки винятків. Якщо сервер недоступний, користувач отримує повідомлення про тимчасову помилку. Якщо API повертає порожню відповідь або група не знайдена, бот повідомляє про відсутність результатів пошуку. Такий підхід дозволяє уникнути аварійного завершення роботи програми та забезпечує коректну взаємодію користувача із системою навіть у випадку виникнення непередбачених ситуацій. Наприклад, під час тестування було виявлено проблему: 401Unauthorized. Причиною помилки була відсутність token-заголовка у HTTP-запитах. Після додавання заголовка проблема була усунена.

Отримані JSON-відповіді містять інформацію про:

- групи;
- викладачів;
- пари;
- аудиторії;
- типи занять;

- час проведення.

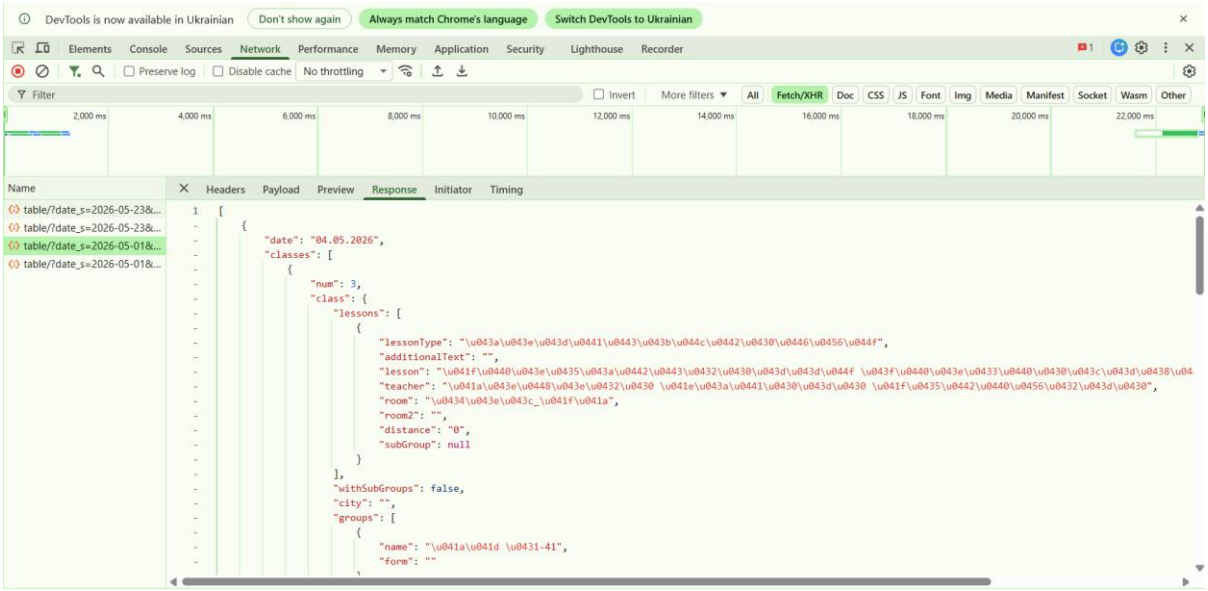


Рисунок 4.1 – JSON-відповідь API ПУЕТ

Використання API дозволило автоматично отримувати актуальний розклад без необхідності ручного оновлення інформації.

4.3 Реалізація пошуку розкладу та обробки даних

Однією з основних функцій Telegram-бота є пошук розкладу за групою або викладачем. Логіка пошуку груп реалізована у файлі: app/handlers/group.py Логіка пошуку викладачів реалізована у файлі: app/handlers/teacher.py Для покращення пошуку було реалізовано функцію: normalize_text() Функція знаходиться у файлі: app/utils/helpers.py

Нормалізація тексту:

```
def normalize_text(text: str) -> str:
    text = text.lower().strip()
```

```
    text = re.sub(r"[^a-za-яіїєґ0-9]+", " ", text)
    return " ".join(text.split())
```

Функція виконує:

- переведення тексту у нижній регістр;
- видалення спеціальних символів;
- нормалізацію пробілів.

Для пошуку схожих назв використовується функція: split_tokens()

Функція дозволяє:

- знаходити часткові співпадіння;
- виконувати пошук за частиною назви;
- обробляти різні варіанти написання.

Після пошуку Telegram-бот автоматично створює inline-кнопки зі списком знайдених варіантів. Після натискання на одну із запропонованих кнопок бот визначає обрану групу та виконує запит на отримання розкладу занять. Такий підхід дозволяє зменшити кількість помилок під час введення назв груп та підвищує зручність використання системи.

Аналогічний механізм використовується під час пошуку розкладу за викладачем. Спочатку виконується пошук викладача у списку, отриманому з API, після чого визначається його ідентифікатор та формується запит для отримання розкладу занять на обрану дату.

Telegram-бот також підтримує ручне введення дати. Для роботи з датами була створена функція: parse_date()

Функція використовується для:

- перевірки правильності формату;
- перевірки коректності дати;
- перетворення дати у формат API.

Для вибору дати використовуються inline-кнопки Telegram.

Створення кнопки вибору дати:

```
InlineKeyboardButton(
    text="Сьогодні",
    callback_data="today"
)
```

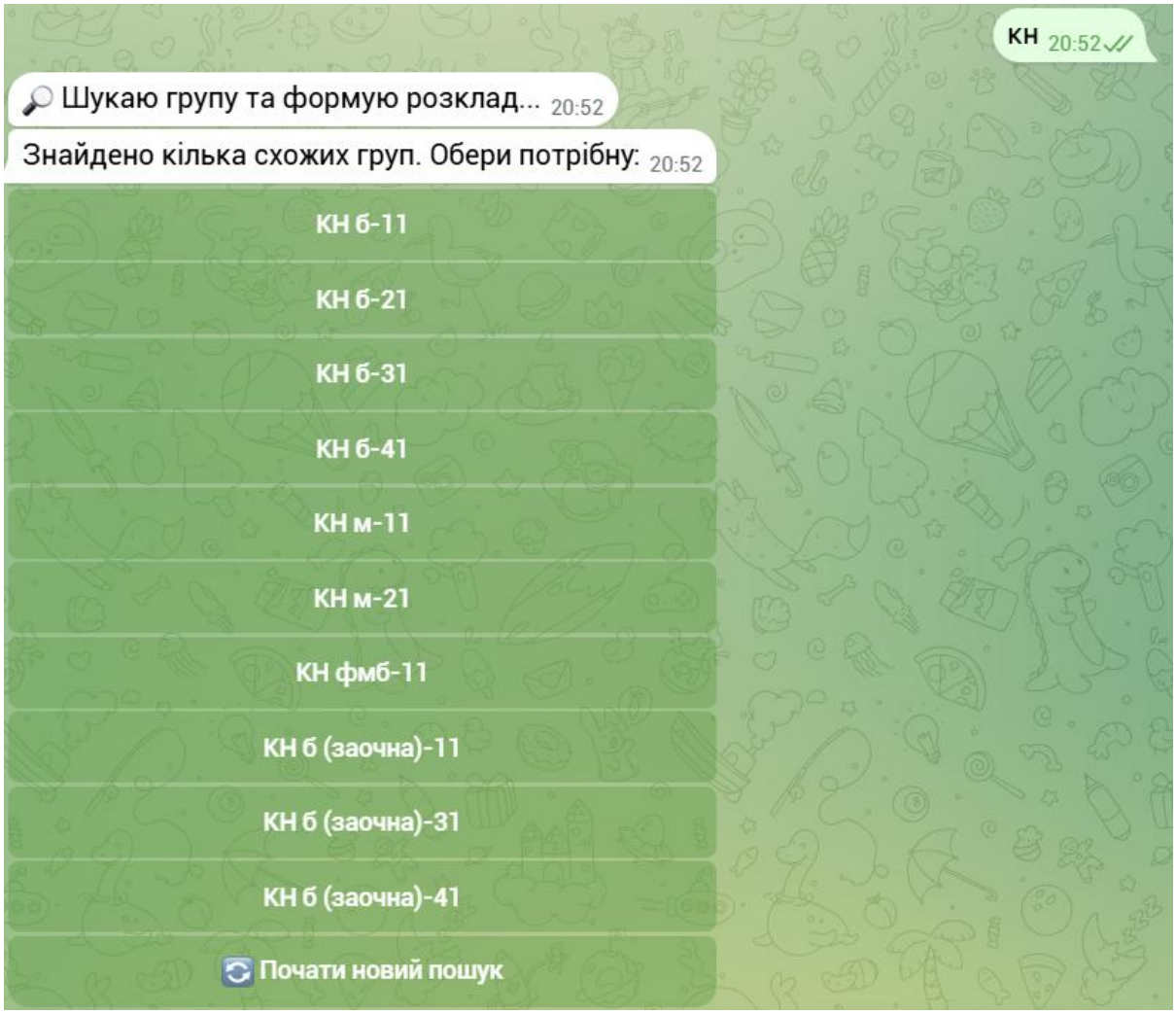


Рисунок 4.2 – Пошук групи у Telegram-боті

Telegram-бот підтримує:

- перегляд розкладу на сьогодні;
- перегляд розкладу на завтра;
- ручне введення дати.

Після отримання даних система автоматично формує API-запит та отримує розклад за вибраною датою.

4.4 Реалізація форматування повідомлень та збереження даних

Для коректного відображення інформації було створено окремий модуль форматування повідомлень. Основний файл форматування: app/utils/formatters.py

У модулі реалізовані функції:

- format_group_schedule();
- format_teacher_schedule().

Формування повідомлення: text += f"{pair_num}. {lesson_name}\n"

У процесі форматування Telegram-бот автоматично додає:

- номер пари;
- назву предмета;
- аудиторію;
- час проведення;
- викладача;
- групу.

Під час розробки Telegram-бота було враховано обмеження платформи Telegram щодо максимальної довжини повідомлень. У випадку, коли користувач виконує пошук розкладу на декілька днів або отримує великий обсяг інформації, сформоване повідомлення може перевищувати допустимий розмір одного повідомлення Telegram.

Для вирішення цієї проблеми в проєкті реалізовано функцію send_long_message(). Основним завданням даної функції є автоматичний поділ великого тексту на декілька окремих повідомлень із подальшим

послідовним надсиланням користувачу. Завдяки цьому Telegram-бот може коректно відображати розклад будь-якого обсягу без втрати інформації та без виникнення помилок під час надсилання повідомлень.

Перед відправкою функція аналізує довжину сформованого тексту та визначає місця для розбиття повідомлення. Після цього кожна частина надсилається окремо, зберігаючи логічну структуру та послідовність відображення інформації. Використання такого підходу дозволяє забезпечити стабільну роботу Telegram-бота навіть у випадку отримання великої кількості даних від API системи розкладу ПУЕТ.

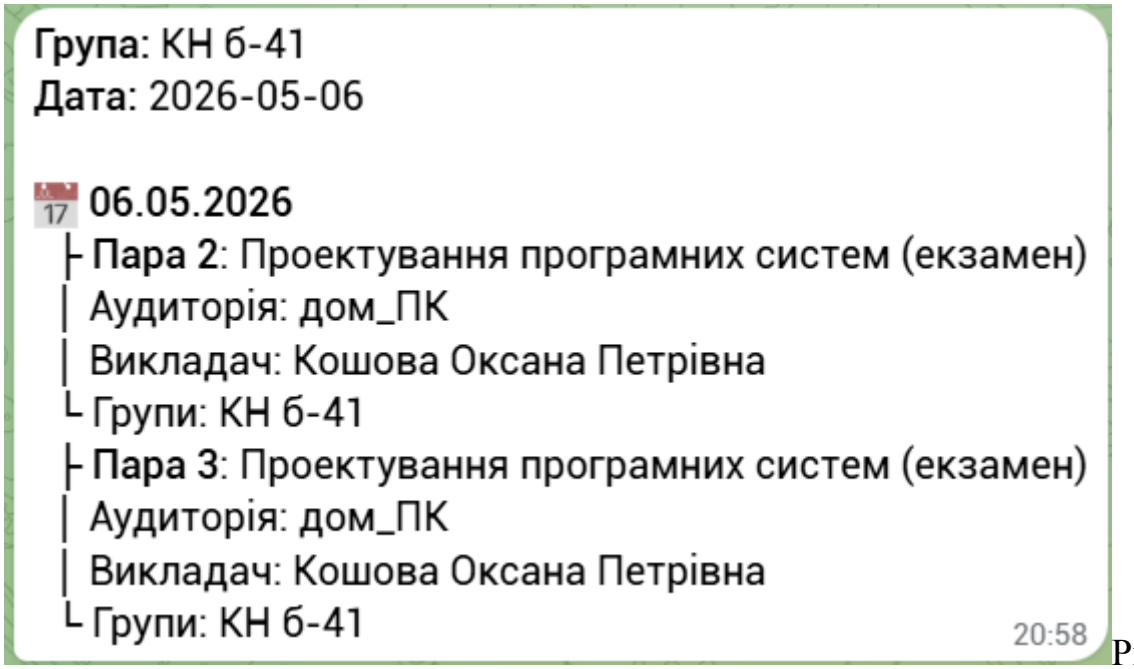


Рисунок 4.3 – Приклад сформованого розкладу

Для спрощення роботи користувачів було реалізовано систему збереження груп. Інформація про групи користувачів зберігається у файлі: users_groups.json Робота із JSON-файлом реалізована у модулі: app/storage/users_storage.py

У модулі реалізовано функції:

- save_user_group();

- get_user_group();
- delete_user_group().

Збереження даних у JSON-файл: json.dump(data, f, ensure_ascii=False, indent=4). Після першого успішного пошуку Telegram-бот пропонує користувачу зберегти групу для подальшого використання.

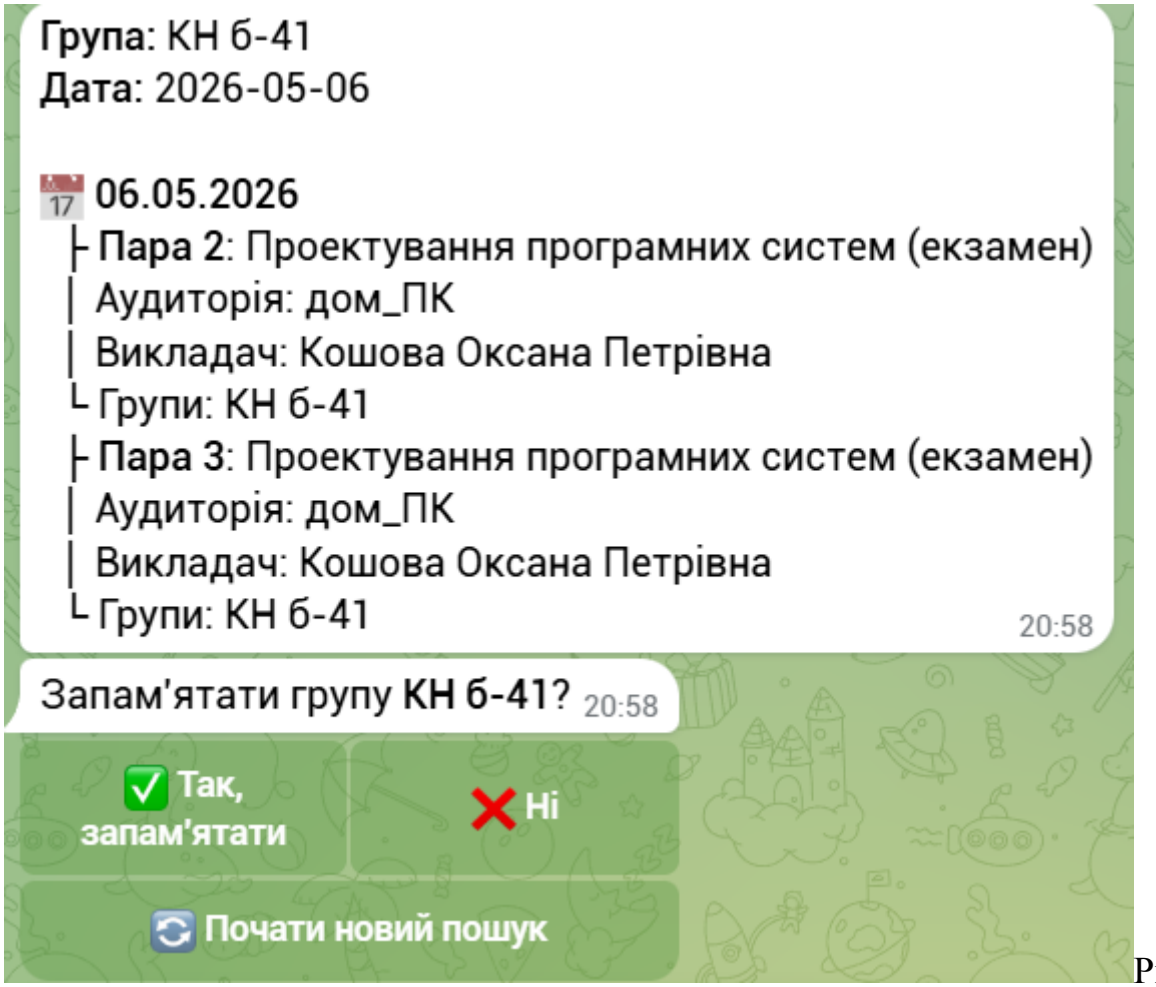


Рисунок 4.4 – Збереження групи користувача

Реалізація системи збереження даних дозволила значно спростити повторне використання Telegram-бота.

ВИСНОВКИ

Результатом виконання бакалаврської роботи було розроблено Telegram-чат-бот для автоматизації студентського сервісу Полтавського університету економіки і торгівлі. У процесі роботи було досліджено принципи роботи Telegram-ботів, особливості Telegram Bot API, асинхронного програмування та взаємодії із API системи розкладу ПУЕТ.

Під час виконання роботи було проаналізовано існуючі способи отримання розкладу занять та визначено переваги використання Telegram-бота для автоматизації студентського сервісу. У межах проєкту було обрано мову програмування Python та бібліотеку aiogram 3, що дозволило реалізувати асинхронну роботу системи та модульну структуру проєкту.

У практичній частині було реалізовано Telegram-бота, який підтримує пошук розкладу за групою та викладачем, вибір дати перегляду, ручне введення дати, форматування повідомлень та систему збереження груп користувачів. Для взаємодії з API ПУЕТ було створено окремий модуль роботи з HTTP-запитами та JSON-відповідями.

Протягом процесу розробки було реалізовано FSM-модель взаємодії користувача із системою, inline-кнопки, callback-запити та систему форматування повідомлень для Telegram. Також було проведено тестування Telegram-бота та усунено проблеми, пов’язані з обробкою API-запитів і JSON-структур. Результатом виконання роботи стало створення Telegram-чат-бота, який забезпечує швидкий та зручний доступ студентів до інформації про розклад занять через месенджер Telegram.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ольховська О.В., Черненко О.О. Методичні рекомендації до виконання кваліфікаційної роботи для студентів спеціальності 122 Комп'ютерні науки освітня програма «Комп'ютерні науки» ступеня бакалавра / О.В. Ольховська, О.О. Черненко. – Полтава : ПУЕТ, 2025. – 58 с.
2. Що таке Telegram-боти та як вони працюють [Електронний ресурс]. – Режим доступу: https://core.telegram.org/bots
3. Документація Telegram Bot API для створення та керування ботами [Електронний ресурс]. – Режим доступу: https://core.telegram.org/bots/api
4. Документація бібліотеки aiogram для розробки Telegram-ботів мовою Python [Електронний ресурс]. – Режим доступу: https://docs.aiogram.dev
5. Документація мови програмування Python [Електронний ресурс]. – Режим доступу: https://docs.python.org/3
6. Офіційний сайт мови програмування Python [Електронний ресурс]. – Режим доступу: https://www.python.org
7. Робота з HTTP-запитами у бібліотеці Requests [Електронний ресурс]. – Режим доступу: https://requests.readthedocs.io
8. Асинхронне програмування в Python за допомогою asyncio [Електронний ресурс]. – Режим доступу: https://docs.python.org/3/library/asyncio.html

9. Що таке JSON та як використовується формат обміну даними [Електронний ресурс]. – Режим доступу: https://www.json.org/json-uk.html
10. Вступ до роботи з JSON у веб-застосунках [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Scripting/JSON
11. Що таке API та як працюють програмні інтерфейси [Електронний ресурс]. – Режим доступу: https://aws.amazon.com/what-is/api
12. Принципи побудови REST API та робота з веб-сервісами [Електронний ресурс]. – Режим доступу: https://restfulapi.net
13. HTTP-протокол та принципи обміну даними в мережі [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTTP
14. Методи HTTP-запитів та їх використання у веб-сервісах [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods
15. Проєкт aiogram на GitHub та приклади використання бібліотеки [Електронний ресурс]. – Режим доступу: https://github.com/aiogram/aiogram
16. Що таке скінченний автомат станів FSM та де він використовується [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Finite-state_machine
17. Порівняння режимів Polling та Webhook у роботі веб-сервісів [Електронний ресурс]. – Режим доступу: https://hostman.com/tutorials/difference-between-polling-and-webhooks
18. Основи архітектури програмного забезпечення [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Архітектура_програмного_забезпечення

19. Розробка програмного забезпечення та етапи створення програмних систем [Електронний ресурс]. – Режим доступу: https://studfile.net/preview/5118185
20. Інструменти розробника Chrome DevTools для аналізу мережевих запитів [Електронний ресурс]. – Режим доступу: https://developer.chrome.com/docs/devtools/
21. Система розкладу занять Полтавського університету економіки і торгівлі [Електронний ресурс]. – Режим доступу: https://schedule.puet.edu.ua/
22. API системи розкладу ПУЕТ для отримання списку груп [Електронний ресурс]. – Режим доступу: https://cabinet.puet.edu.ua/api/v1/schedule/groups
23. API системи розкладу ПУЕТ для отримання списку викладачів [Електронний ресурс]. – Режим доступу: https://cabinet.puet.edu.ua/api/v1/schedule/teachers
24. API системи розкладу ПУЕТ для отримання розкладу занять [Електронний ресурс]. – Режим доступу: https://cabinet.puet.edu.ua/api/staging/schedule/classes/table/

ДОДАТОК А. КОД ПРОГРАМИ

Файл config:

```
import logging

BOT_TOKEN =
"8588189139:AAFbI7UTrwYosJijbPiBNVKcx7KYrQowy6Y"
PUET_TOKEN = "4ydwkwfkjj02yzgwsd9a"

GROUPS_URL = "https://cabinet.puet.edu.ua/api/v1/schedule/groups"
TEACHERS_URL = "https://cabinet.puet.edu.ua/api/v1/schedule/teachers"
SCHEDULE_URL = "https://cabinet.puet.edu.ua/api/staging/schedule/classes/table/"

HEADERS = {"token": PUET_TOKEN}
USERS_FILE = "users_groups.json"

logging.basicConfig(level=logging.INFO)
```

Файл main:

```
import asyncio

from aiogram import Bot, Dispatcher

from config import BOT_TOKEN
from app.handlers.start import router as start_router
from app.handlers.callbacks import router as callbacks_router
from app.handlers.group import router as group_router
from app.handlers.teacher import router as teacher_router


async def main():
    bot = Bot(token=BOT_TOKEN)
    dp = Dispatcher()
```

```
    dp.include_router(start_router)
    dp.include_router(callbacks_router)
    dp.include_router(group_router)
    dp.include_router(teacher_router)

    await dp.start_polling(bot)


if __name__ == "__main__":
    asyncio.run(main())
```

Файл callbacks:

```
import html
import logging

from aiogram import Router, F
from aiogram.types import CallbackQuery
from aiogram.fsm.context import FSMContext

from app.states.search_states import SearchState
from app.keyboards.inline import build_restart_keyboard,
build_save_group_keyboard    from app.storage.users_storage import (
    get_user_group,
    delete_user_group,
    get_temp_group,
    delete_temp_group,
    save_user_group,
    save_temp_group,
)
from app.services.puet_api import get_groups, get_schedule_by_group
from app.services.search_service import get_group_by_name
from app.utils.formatters import format_group_schedule
from app.utils.messages import send_long_message
from app.handlers.start import ask_for_next_input

router = Router()
```

```
async def send_group_schedule_and_offer_save(message, group_obj: dict,
target_date: str):
    group_name = str(group_obj.get("name", "Невідома група"))
    schedule = get_schedule_by_group(group_obj, target_date)
    formatted_text = format_group_schedule(schedule, group_name, target_date)

    await send_long_message(message, formatted_text)

    await message.answer(
        f"Запам'ятати групу <b>{html.escape(group_name)}</b>?",
        parse_mode="HTML",
        reply_markup=build_save_group_keyboard()
    )


@router.callback_query(F.data.startswith("date:"))
async def select_date_handler(callback: CallbackQuery, state: FSMContext):
    target_date = callback.data.split("date:", 1)[1]
    await state.update_data(target_date=target_date)

    await callback.message.answer(
        f"□ Обрана дата: <b>{html.escape(target_date)}</b>",
        parse_mode="HTML"
    )
    await ask_for_next_input(callback.message, state)
    await callback.answer()


@router.callback_query(F.data == "date_manual")
async def date_manual_handler(callback: CallbackQuery, state: FSMContext):
    await state.set_state(SearchState.waiting_for_date)
    await callback.message.answer(
        "Введи дату у форматі:\n<code>10.03.2026</code>",
        parse_mode="HTML",
        reply_markup=build_restart_keyboard()
    )
    await callback.answer()
```

```
@router.callback_query(F.data == "use_saved_group")
async def use_saved_group_handler(callback: CallbackQuery, state:
FSMContext):
    saved_group = get_user_group(callback.from_user.id)

    if not saved_group:
        await callback.message.answer("Збережену групу не знайдено.")
        await callback.answer()
        return

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

    if not target_date:
        await callback.message.answer("Спочатку обери дату.",
reply_markup=build_restart_keyboard())
        await callback.answer()
        return

    try:
        schedule = get_schedule_by_group(saved_group, target_date)
        formatted_text = format_group_schedule(
            schedule,
            saved_group.get("name", "Невідома група"),
            target_date
        )
        await send_long_message(callback.message, formatted_text)
        await callback.message.answer("Що далі?",
reply_markup=build_restart_keyboard())
    except Exception as e:
        await callback.message.answer(
            f"□□ Помилка:\n<code>{html.escape(str(e))}</code>",
            parse_mode="HTML",
            reply_markup=build_restart_keyboard()
        )
```

```
    await callback.answer()


@router.callback_query(F.data == "enter_new_group")
async def enter_new_group_handler(callback: CallbackQuery, state:
FSMContext):
    await state.set_state(SearchState.waiting_for_group)
    await callback.message.answer("Введи назву групи текстом.",
reply_markup=build_restart_keyboard())
    await callback.answer()


@router.callback_query(F.data == "delete_saved_group")
async def delete_saved_group_handler(callback: CallbackQuery):
    delete_user_group(callback.from_user.id)
    await callback.message.answer(
        "Збережену групу видалено. Тепер введи іншу групу.",
        reply_markup=build_restart_keyboard()
    )
    await callback.answer()


@router.callback_query(F.data == "save_group")
async def save_group_handler(callback: CallbackQuery):
    temp_group = get_temp_group(callback.from_user.id)

    if not temp_group:
        await callback.message.answer(
            "Не вдалося знайти групу для збереження.",
            reply_markup=build_restart_keyboard()
        )
        await callback.answer()
        return

    save_user_group(callback.from_user.id, temp_group)
    delete_temp_group(callback.from_user.id)

    await callback.message.answer(
```

```
        f"□ Групу <b>{html.escape(temp_group.get('name', 'Невідома
група'))}</b> збережено.",
        parse_mode="HTML",
        reply_markup=build_restart_keyboard()
    )
    await callback.answer()


@router.callback_query(F.data == "dont_save_group")
async def dont_save_group_handler(callback: CallbackQuery):
    delete_temp_group(callback.from_user.id)
    await callback.message.answer("Добре, групу не зберігаю.",
reply_markup=build_restart_keyboard())
    await callback.answer()


@router.callback_query(F.data.startswith("select_group:"))
async def select_group_handler(callback: CallbackQuery, state: FSMContext):
    try:
        group_name = callback.data.split("select_group:", 1)[1]
        groups = get_groups()
        group_obj = get_group_by_name(group_name, groups)

        if not group_obj:
            await callback.message.answer(
                "□ Не вдалося знайти вибрану групу.",
                reply_markup=build_restart_keyboard()
            )
            await callback.answer()
            return

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

        if not target_date:
            await callback.message.answer("Спочатку обери дату.",
reply_markup=build_restart_keyboard())
            await callback.answer()
```

```
            return

        save_temp_group(callback.from_user.id, group_obj)
        await send_group_schedule_and_offer_save(callback.message, group_obj,
target_date)
        await callback.answer()

    except Exception as e:
        logging.exception("Помилка вибору групи")
        await callback.message.answer(
            f"□□ Сталася помилка:\n<code>{html.escape(str(e))}</code>",
            parse_mode="HTML",
            reply_markup=build_restart_keyboard()
        )
        await callback.answer()
```

Файл group:

```
import html
import logging
import requests
from aiogram import Router, F
from aiogram.types import Message
from aiogram.fsm.context import FSMContext

from app.states.search_states import SearchState
from app.keyboards.inline import build_groups_keyboard,
build_restart_keyboard
from app.storage.users_storage import save_temp_group
from app.services.puet_api import get_groups
from app.services.search_service import find_exact_group, find_similar_groups
from app.handlers.callbacks import send_group_schedule_and_offer_save

router = Router()


@router.message(SearchState.waiting_for_group, F.text)
async def group_handler(message: Message, state: FSMContext):
```

```
search_text = message.text.strip()

    if len(search_text) < 2:
        await message.answer("Введи хоча б 2 символи назви групи.",
reply_markup=build_restart_keyboard())
        return

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

    if not target_date:
        await message.answer("Спочатку обери дату через /start.",
reply_markup=build_restart_keyboard())
        return

    await message.answer("□ Шукаю групу та формую розклад...")

    try:
        groups = get_groups()

        exact_group = find_exact_group(search_text, groups)
        if exact_group:
            save_temp_group(message.from_user.id, exact_group)
            await send_group_schedule_and_offer_save(message, exact_group,
target_date)
            return

        similar_groups = find_similar_groups(search_text, groups)

        if not similar_groups:
            await message.answer("□ Групу не знайдено.",
reply_markup=build_restart_keyboard())
            return

        if len(similar_groups) == 1:
            save_temp_group(message.from_user.id, similar_groups[0])
            await send_group_schedule_and_offer_save(message,
similar_groups[0], target_date)
```

```
            return

        keyboard = build_groups_keyboard(similar_groups)
        await message.answer(
            "Знайдено кілька схожих груп. Обери потрібну:",
            reply_markup=keyboard
        )

    except requests.exceptions.RequestException as e:
        logging.exception("Помилка API")
        await message.answer(
            f"□□ Помилка під час запиту до
API:\n<code>{html.escape(str(e))}</code>",
            parse_mode="HTML",
            reply_markup=build_restart_keyboard()
        )

    except Exception as e:
        logging.exception("Невідома помилка")
        await message.answer(
            f"□□ Сталася помилка:\n<code>{html.escape(str(e))}</code>",
            parse_mode="HTML",
            reply_markup=build_restart_keyboard()
        )
```

Файл start:

```
import html

from aiogram import Router, F
from aiogram.filters import CommandStart
from aiogram.types import Message, CallbackQuery
from aiogram.fsm.context import FSMContext

from app.states.search_states import SearchState
from app.keyboards.inline import (
```

```
    build_search_type_keyboard,
    build_date_keyboard,
    build_saved_group_keyboard,
    build_restart_keyboard,
)
from app.storage.users_storage import get_user_group, delete_temp_group

router = Router()


async def ask_for_search_type(message: Message, state: FSMContext):
    await state.set_state(SearchState.waiting_for_search_type)
    await message.answer(
        "Привіт!\n\nОбери, за чим шукати розклад:",
        reply_markup=build_search_type_keyboard()
    )


async def ask_for_date(message: Message, state: FSMContext):
    await state.set_state(SearchState.waiting_for_date)
    await message.answer(
        "Тепер обери дату, на яку потрібно знайти розклад.",
        reply_markup=build_date_keyboard()
    )


async def ask_for_next_input(message: Message, state: FSMContext):
    data = await state.get_data()
    target_date = data.get("target_date")
    search_type = data.get("search_type")

    if search_type == "group":
        saved_group = get_user_group(message.from_user.id)

        if saved_group:
            group_name = saved_group.get("name", "Моя група")
            await message.answer(
                f"Дата вибрана: <b>{html.escape(target_date)}</b>\n\n"
```

```
            f"У тебе вже збережена група:
<b>{html.escape(group_name)}</b>\n\n"
            f"Обери дію:",
            parse_mode="HTML",
            reply_markup=build_saved_group_keyboard(group_name)
        )
    else:
        await message.answer(
            f"Дата вибрана: <b>{html.escape(target_date)}</b>\n\n"
            f"Тепер надішли <b>назву групи</b>, наприклад:\n"
            f"<code>КН б-41</code>",
            parse_mode="HTML",
            reply_markup=build_restart_keyboard()
        )

    await state.set_state(SearchState.waiting_for_group)
    return

if search_type == "teacher":
    await message.answer(
        f"Дата вибрана: <b>{html.escape(target_date)}</b>\n\n"
        f"Тепер надішли <b>ПІБ викладача</b> або його частину.",
        parse_mode="HTML",
        reply_markup=build_restart_keyboard()
    )
    await state.set_state(SearchState.waiting_for_teacher)


@router.message(CommandStart())
async def start_handler(message: Message, state: FSMContext):
    await state.clear()
    await ask_for_search_type(message, state)


@router.callback_query(F.data == "restart_search")
async def restart_search_handler(callback: CallbackQuery, state: FSMContext):
    await state.clear()
    delete_temp_group(callback.from_user.id)
```

```
    await ask_for_search_type(callback.message, state)
    await callback.answer()


@router.callback_query(F.data.startswith("search_type:"))
async def select_search_type_handler(callback: CallbackQuery, state:
FSMContext):
    search_type = callback.data.split("search_type:", 1)[1]
    await state.update_data(search_type=search_type)

    if search_type == "teacher":
        await callback.message.answer("□ Обрано пошук за викладачем.")
    else:
        await callback.message.answer("□ Обрано пошук за групою.")

    await ask_for_date(callback.message, state)
    await callback.answer()
```

Файл teacher:

```
import html
import logging
import requests

from aiogram import Router, F
from aiogram.types import Message
from aiogram.fsm.context import FSMContext

from app.states.search_states import SearchState
from app.keyboards.inline import build_restart_keyboard
from app.utils.helpers import parse_date
from app.services.puet_api import get_teachers, get_schedule_by_teacher
from app.services.search_service import find_teacher
from app.utils.formatters import format_teacher_schedule
from app.utils.messages import send_long_message
from app.handlers.start import ask_for_next_input

router = Router()
```

```
async def send_teacher_schedule(message: Message, teacher_name: str,
teacher_id: int,

target_date: str):
    schedule = get_schedule_by_teacher(teacher_id, target_date)
    formatted_text = format_teacher_schedule(schedule, teacher_name,
target_date)
    await send_long_message(message, formatted_text)
    await message.answer("Що далі?", reply_markup=build_restart_keyboard())


@router.message(SearchState.waiting_for_date, F.text)
async def manual_date_input_handler(message: Message, state: FSMContext):
    target_date = parse_date(message.text)

    if not target_date:
        await message.answer(
            "□ Неправильний формат дати. Введи так:
<code>10.03.2026</code>",
            parse_mode="HTML",
            reply_markup=build_restart_keyboard()
        )
        return

    await state.update_data(target_date=target_date)
    await ask_for_next_input(message, state)


@router.message(SearchState.waiting_for_teacher, F.text)
async def teacher_handler(message: Message, state: FSMContext):
    search_name = message.text.strip()

    if len(search_name) < 2:
        await message.answer("Введи хоча б 2 символи для пошуку
викладача.", reply_markup=build_restart_keyboard())
        return
```

```
    data = await state.get_data()
    target_date = data.get("target_date")

    if not target_date:
        await message.answer("Спочатку обери дату через /start.",
reply_markup=build_restart_keyboard())
        return

    await message.answer("□ Шукаю викладача та формую розклад...")

    try:
        teachers = get_teachers()
        teacher_id, teacher_name = find_teacher(search_name, teachers)

        if not teacher_id:
            await message.answer("□ Викладача не знайдено.",
reply_markup=build_restart_keyboard())
            return

        await send_teacher_schedule(message, teacher_name, teacher_id,
target_date)

    except requests.exceptions.RequestException as e:
        logging.exception("Помилка запиту до API")
        await message.answer(
            f"□□ Помилка під час запиту до
API:\n<code>{html.escape(str(e))}</code>",
            parse_mode="HTML",
            reply_markup=build_restart_keyboard()
        )

    except Exception as e:
        logging.exception("Невідома помилка")
        await message.answer(
            f"□□ Сталася помилка:\n<code>{html.escape(str(e))}</code>",
            parse_mode="HTML",
            reply_markup=build_restart_keyboard()
        )
```

Файл inline:

```
from datetime import datetime, timedelta

from aiogram.types import InlineKeyboardMarkup, InlineKeyboardButton

def build_search_type_keyboard() -> InlineKeyboardMarkup:
    return InlineKeyboardMarkup(
        inline_keyboard=[
            [InlineKeyboardButton(text="□□ Шукати за викладачем",
callback_data="search_type:teacher")],
            [InlineKeyboardButton(text="□ Шукати за групою",
callback_data="search_type:group")],
        ]
    )


def build_groups_keyboard(groups_list: list) -> InlineKeyboardMarkup:
    keyboard = []

    for group in groups_list:
        group_name = str(group.get("name", "Невідома група"))
        keyboard.append([
            InlineKeyboardButton(
                text=group_name,
                callback_data=f"select_group:{group_name}"
            )
        ])

    keyboard.append([
        InlineKeyboardButton(text="□ Почати новий пошук",
callback_data="restart_search")
    ])

    return InlineKeyboardMarkup(inline_keyboard=keyboard)
```

```
def build_date_keyboard() -> InlineKeyboardMarkup:
    today_obj = datetime.today()
    tomorrow_obj = today_obj + timedelta(days=1)

    today_display = today_obj.strftime("%d.%m.%Y")
    tomorrow_display = tomorrow_obj.strftime("%d.%m.%Y")

    today_value = today_obj.strftime("%Y-%m-%d")
    tomorrow_value = tomorrow_obj.strftime("%Y-%m-%d")

    return InlineKeyboardMarkup(
        inline_keyboard=[
            [InlineKeyboardButton(
                text=f"□ Сьогодні ({today_display})",
                callback_data=f"date:{today_value}"
            )],
            [InlineKeyboardButton(
                text=f"□ Завтра ({tomorrow_display})",
                callback_data=f"date:{tomorrow_value}"
            )],
            [InlineKeyboardButton(text="□□ Ввести дату вручну",
callback_data="date_manual")],
            [InlineKeyboardButton(text="□ Почати новий пошук",
callback_data="restart_search")],
        ]
    )


def build_saved_group_keyboard(group_name: str) -> InlineKeyboardMarkup:
    return InlineKeyboardMarkup(
        inline_keyboard=[
            [InlineKeyboardButton(text=f"□ {group_name}",
callback_data="use_saved_group")],
            [InlineKeyboardButton(text="□□ Ввести іншу групу",
callback_data="enter_new_group")],
            [InlineKeyboardButton(text="□ Забути мою групу",
callback_data="delete_saved_group")],
            [InlineKeyboardButton(text="□ Почати новий пошук",
```

```
callback_data="restart_search")],
        ]
    )


def build_save_group_keyboard() -> InlineKeyboardMarkup:
    return InlineKeyboardMarkup(
        inline_keyboard=[
            [
                InlineKeyboardButton(text="□ Так, запам'ятати",
callback_data="save_group"),
                InlineKeyboardButton(text="□ Ні",
callback_data="dont_save_group"),
            ],
            [InlineKeyboardButton(text="□ Почати новий пошук",
callback_data="restart_search")]
        ]
    )


def build_restart_keyboard() -> InlineKeyboardMarkup:
    return InlineKeyboardMarkup(
        inline_keyboard=[
            [InlineKeyboardButton(text="□ Почати новий пошук",
callback_data="restart_search")]
        ]
    )
```

Файл puet_api:

```
import requests

from config import GROUPS_URL, TEACHERS_URL, SCHEDULE_URL,
HEADERS


def get_groups() -> list:
    response = requests.get(GROUPS_URL, headers=HEADERS, timeout=20)
```

```
    response.raise_for_status()
    data = response.json()
    return data if isinstance(data, list) else []


def get_teachers() -> list:
    response = requests.get(TEACHERS_URL, headers=HEADERS,
timeout=20)
    response.raise_for_status()
    data = response.json()
    return data if isinstance(data, list) else []


def get_schedule_by_group(group_obj: dict, target_date: str) -> list:
    params = {
        "date_s": target_date,
        "date_e": target_date,
        "language": "uk",
        "course": group_obj.get("course"),
        "spec_id": group_obj.get("spec_id"),
        "num": group_obj.get("num"),
        "forma": group_obj.get("forma"),
        "owner": group_obj.get("owner"),
    }

    response = requests.get(SCHEDULE_URL, headers=HEADERS,
params=params, timeout=20)
    response.raise_for_status()
    data = response.json()
    return data if isinstance(data, list) else []


def get_schedule_by_teacher(teacher_id: int, target_date: str) -> list:
    params = {
        "date_s": target_date,
        "date_e": target_date,
        "language": "uk",
        "teacher": teacher_id,
```

```
    }

    response = requests.get(SCHEDULE_URL, headers=HEADERS,
params=params, timeout=20)
    response.raise_for_status()
    data = response.json()
    return data if isinstance(data, list) else []
```

Файл search_service:

```
from typing import Optional

from app.utils.helpers import normalize_text, split_tokens


def find_exact_group(search_text: str, groups: list) -> Optional[dict]:
    search_text = search_text.strip()
    search_norm = normalize_text(search_text)

    for group in groups:
        name = str(group.get("name", "")).strip()
        if name.lower() == search_text.lower():
            return group

    for group in groups:
        name = str(group.get("name", "")).strip()
        if normalize_text(name) == search_norm:
            return group

    return None


def find_similar_groups(search_text: str, groups: list, limit: int = 10) -> list:
    search_text = search_text.strip()
    search_norm = normalize_text(search_text)
```

```
    search_tokens = set(split_tokens(search_text))

    found = []

    for group in groups:
        name = str(group.get("name", "")).strip()
        name_lower = name.lower()
        name_norm = normalize_text(name)
        name_tokens = set(split_tokens(name))

        matched = (
            search_text.lower() in name_lower
            or (search_norm and search_norm in name_norm)
        )

        if not matched:
            continue

        if "жов" in name_tokens and "жов" not in search_tokens:
            continue

        score = 0

        if name_lower.startswith(search_text.lower()):
            score += 100

        if search_norm and name_norm.startswith(search_norm):
            score += 80

        score += len(search_tokens & name_tokens) * 10
        score -= len(name_tokens)

        found.append((score, group))

    found.sort(key=lambda item: (-item[0], item[1].get("name", "")))
    return [item[1] for item in found[:limit]]


def get_group_by_name(group_name: str, groups: list) -> Optional[dict]:
```

```
    for group in groups:
        if str(group.get("name", "")).strip().lower() == group_name.strip().lower():
            return group
    return None


def find_teacher(search_name: str, teachers: list) -> tuple[Optional[int],
Optional[str]]:
    search_name = search_name.strip().lower()

    for teacher in teachers:
        teacher_name = str(teacher.get("name", "")).strip()
        if search_name in teacher_name.lower():
            return teacher.get("id_prep"), teacher_name

    return None, None
```

Файл search_states:

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


class SearchState(StatesGroup):
    waiting_for_search_type = State()
    waiting_for_date = State()
    waiting_for_group = State()
    waiting_for_teacher = State()
```

Файл users_storage:

```
import json
import os
from typing import Optional

from config import USERS_FILE


def load_users_data() -> dict:
```

```
    if not os.path.exists(USERS_FILE):
        return {}

    try:
        with open(USERS_FILE, "r", encoding="utf-8") as f:
            return json.load(f)
    except (json.JSONDecodeError, OSError):
        return {}


def save_users_data(data: dict):
    with open(USERS_FILE, "w", encoding="utf-8") as f:
        json.dump(data, f, ensure_ascii=False, indent=2)


def save_user_group(user_id: int, group_obj: dict):
    data = load_users_data()
    data[str(user_id)] = {
        "name": group_obj.get("name"),
        "course": group_obj.get("course"),
        "spec_id": group_obj.get("spec_id"),
        "num": group_obj.get("num"),
        "forma": group_obj.get("forma"),
        "owner": group_obj.get("owner"),
    }
    save_users_data(data)


def get_user_group(user_id: int) -> Optional[dict]:
    data = load_users_data()
    return data.get(str(user_id))


def delete_user_group(user_id: int):
    data = load_users_data()
    if str(user_id) in data:
        del data[str(user_id)]
        save_users_data(data)
```

```
def save_temp_group(user_id: int, group_obj: dict):
    data = load_users_data()
    data[f"temp_{user_id}"] = {
        "name": group_obj.get("name"),
        "course": group_obj.get("course"),
        "spec_id": group_obj.get("spec_id"),
        "num": group_obj.get("num"),
        "forma": group_obj.get("forma"),
        "owner": group_obj.get("owner"),
    }
    save_users_data(data)


def get_temp_group(user_id: int) -> Optional[dict]:
    data = load_users_data()
    return data.get(f"temp_{user_id}")


def delete_temp_group(user_id: int):
    data = load_users_data()
    temp_key = f"temp_{user_id}"
    if temp_key in data:
        del data[temp_key]
        save_users_data(data)
```

Файл formatters:

```
import html
from typing import Any

def extract_groups(groups_data: Any) -> str:
    if isinstance(groups_data, dict):
        result = []
        for group in groups_data.values():
            if isinstance(group, dict):
                result.append(group.get("name", "Без назви"))
        return ", ".join(result) if result else "Немає груп"

    if isinstance(groups_data, list):
```

```
        result = []
        for group in groups_data:
            if isinstance(group, dict):
                result.append(group.get("name", "Без назви"))
            elif isinstance(group, str):
                result.append(group)
        return ", ".join(result) if result else "Немає груп"

    return "Немає груп"


def extract_teachers(class_info: dict) -> str:
    if not isinstance(class_info, dict):
        return "Не вказано"

    result = []

    def add_teacher(value):
        if not value:
            return

        if isinstance(value, str):
            value = value.strip()
            if value:
                result.append(value)
            return

        if isinstance(value, dict):
            name = (
                value.get("name")
                or value.get("fio")
                or value.get("full_name")
                or value.get("teacherName")
                or value.get("prepName")
            )
            if name:
                result.append(str(name))
                return
```

```
            for item in value.values():
                add_teacher(item)
            return

        if isinstance(value, list):
            for item in value:
                add_teacher(item)

    for key in ["teachers", "teacher", "preps", "prep", "prepName",
"teacherName"]:
        add_teacher(class_info.get(key))

    unique_result = list(dict.fromkeys(result))
    return ", ".join(unique_result) if unique_result else "Не вказано"


def format_group_schedule(schedule: list, group_name: str, target_date: str) ->
str:
    if not schedule:
        return (
            f"<b>Група:</b> {html.escape(group_name)}\n"
            f"<b>Дата:</b> {html.escape(target_date)}\n\n"
            f"Розклад не знайдено."
        )

    lines = [
        f"<b>Група:</b> {html.escape(group_name)}",
        f"<b>Дата:</b> {html.escape(target_date)}",
        ""
    ]

    found_classes = False

    for day in schedule:
        date = str(day.get("date", target_date))
        classes = day.get("classes", [])
```

```
        if not classes:
            continue

        found_classes = True
        lines.append(f"□ <b>{html.escape(date)}</b>")

        for cls in classes:
            pair_num = cls.get("num", "?")
            class_info = cls.get("class", {})
            lessons = class_info.get("lessons", [])
            groups = extract_groups(class_info.get("groups"))
            teachers = extract_teachers(class_info)

            if not lessons:
                lines.append(f"  └ <b>Пара {pair_num}</b>: дані відсутні")
                continue

            for lesson in lessons:
                lesson_name = str(lesson.get("lesson", "Невідома дисципліна"))
                lesson_type = str(lesson.get("lessonType", "Без типу"))
                room = str(lesson.get("room", "Не вказано"))
                lesson_teachers = extract_teachers(lesson)
                if lesson_teachers == "Не вказано":
                    lesson_teachers = teachers

                time_start = lesson.get("time_start") or lesson.get("timeStart")
                time_end = lesson.get("time_end") or lesson.get("timeEnd")

                if time_start and time_end:
                    pair_header = f"  ├ <b>Пара {pair_num}</b> [{time_start} -
{time_end}]"
                else:
                    pair_header = f"  ├ <b>Пара {pair_num}</b>"

                lines.append(f"{pair_header}: {html.escape(lesson_name)}
({html.escape(lesson_type)})")
                lines.append(f"  │ Аудиторія: {html.escape(room)}")
                lines.append(f"  │ Викладач: {html.escape(lesson_teachers)}")
```

```
            lines.append(f"  ┗ Групи: {html.escape(groups)}")

        lines.append("")

    if not found_classes:
        return (
            f"<b>Група:</b> {html.escape(group_name)}\n"
            f"<b>Дата:</b> {html.escape(target_date)}\n\n"
            f"На цю дату занять не знайдено."
        )

    return "\n".join(lines)


def format_teacher_schedule(schedule: list, teacher_name: str, target_date: str) -
> str:
    if not schedule:
        return (
            f"<b>Викладач:</b> {html.escape(teacher_name)}\n"
            f"<b>Дата:</b> {html.escape(target_date)}\n\n"
            f"Розклад не знайдено."
        )

    lines = [
        f"<b>Викладач:</b> {html.escape(teacher_name)}",
        f"<b>Дата:</b> {html.escape(target_date)}",
        ""
    ]

    has_classes = False

    for day in schedule:
        date = str(day.get("date", target_date))
        classes = day.get("classes", [])

        if not classes:
            continue
```

```
        has_classes = True
        lines.append(f"□ <b>{html.escape(date)}</b>")

        for cls in classes:
            pair_num = cls.get("num", "?")
            class_info = cls.get("class", {})
            groups = extract_groups(class_info.get("groups"))
            lessons = class_info.get("lessons", [])

            if not lessons:
                lines.append(f"  └ Пара {pair_num}: дані відсутні")
                continue

            for lesson in lessons:
                lesson_name = str(lesson.get("lesson", "Невідома дисципліна"))
                lesson_type = str(lesson.get("lessonType", "Без типу"))
                room = str(lesson.get("room", "Не вказано"))

                time_start = lesson.get("time_start") or lesson.get("timeStart")
                time_end = lesson.get("time_end") or lesson.get("timeEnd")

                if time_start and time_end:
                    pair_header = f"  ├ <b>Пара {pair_num}</b> [{time_start} -
{time_end}]"
                else:
                    pair_header = f"  ├ <b>Пара {pair_num}</b>"

                lines.append(f"{pair_header}: {html.escape(lesson_name)}
({html.escape(lesson_type)})")
                lines.append(f"  │ Аудиторія: {html.escape(room)}")
                lines.append(f"  └ Групи: {html.escape(groups)}")

        lines.append("")

    if not has_classes:
        return (
            f"<b>Викладач:</b> {html.escape(teacher_name)}\n"
            f"<b>Дата:</b> {html.escape(target_date)}\n\n"
```

```
            f"На цю дату занять не знайдено."
        )

    return "\n".join(lines)
```

Файл helpers:

```
import re
from datetime import datetime
from typing import Optional


def normalize_text(text: str) -> str:
    text = text.lower().strip()
    text = re.sub(r"\s+", " ", text)
    text = re.sub(r"[^a-za-яіїєґ0-9]+", "", text, flags=re.IGNORECASE)
    return text


def split_tokens(text: str) -> list[str]:
    text = text.lower()
    return re.findall(r"[a-za-яіїєґ0-9]+", text, flags=re.IGNORECASE)


def parse_date(date_text: str) -> Optional[str]:
    try:
        dt = datetime.strptime(date_text.strip(), "%d.%m.%Y")
        return dt.strftime("%Y-%m-%d")
    except ValueError:
        return None
```

Файл messages:

```
from aiogram.types import Message


async def send_long_message(message: Message, text: str, chunk_size: int =
4000):
```

```
for i in range(0, len(text), chunk_size):
    await message.answer(text[i:i + chunk_size], parse_mode="HTML")
```