МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)
« » травня 2024

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-наукової програми «Інформатика»
на тему: «Інформаційна технологія інтенсифікації навчального процесу за допомогою telegram-бота»
здобувача групи ІН.м-21н Кузьмука Данили Андрійовича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Данила КУЗЬМУК
_____
(підпис)

Керівник
старший викладач кафедри
комп'ютерних наук,
к.т.н., доцент
Борис КУЗІКОВ
_____
(підпис)

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук, освітньо- наукової програми «Інформатика»
здобувача групи ІН.м-21н Кузьмука Данили Андрійовича

1. Тема роботи: «Інформаційна технологія інтенсифікації навчального процесу за допомогою telegram-бота»
затверджую наказом по СумДУ від *«8» березня 2024 року № 0234-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 21 травня 2024 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми предметної області та постановка завдань дослідження. 2) Огляд технологій розробки Telegram-ботів. 3) Розробка механізмів інтенсифікації навчального процесу через Telegram-бот. 4) Вибір технологій реалізації та методології розробки. 5) Розробка бота та веб-серверу. 6) Аналіз результатів.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проєкту (роботи), із значенням розділів проєкту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання « » р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області та постановка завдань дослідження*		
2	*Огляд технологій розробки Telegram-ботів*		
3	*Розробка механізмів інтенсифікації навчального процесу через Telegram-бот*		
4	*Вибір технологій реалізації та методології розробки*		
5	*Розробка бота та веб-серверу*		
6	*Аналіз результатів.*		
7	*Оформлення пояснювальної записки до кваліфікаційної роботи.*		

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 68 стор., 27 рис., 1 додаток, 34 джерела.

Обґрунтування актуальності теми роботи – тема кваліфікаційної роботи є актуальною, оскільки присвячена інтенсифікації навчального процесу через застосування сучасних інформаційних технологій, зокрема інтеграції месенджера Telegram у навчальні платформи, що дозволяє вдосконалити механізми взаємодії та зворотного зв'язку між учасниками.

Об’єкт дослідження – механізми інтенсифікації навчання через забезпечення взаємодії, відстеження прогресу та оперативного інформування учасників освітнього процесу.

Мета роботи – розроблення моделі інтенсифікації навчального процесу за допомогою оптимізації комунікації між викладачами та студентами з використанням технології Telegram-bot.

Методи дослідження – аналітичний огляд існуючих рішень, порівняльний аналіз, експериментальна розробка.

Результати – розроблено модель інтенсифікації навчального процесу, інтегровану з Telegram-bot, за рахунок ведення поточного стану навчання та оперативного інформування користувачів системи. Модель сприяє підвищенню ефективності взаємодії і забезпеченню безперервного зв'язку між учасниками навчального процесу. Інформаційну технологію реалізовано як веб-сервер з модулем Telegram-bot з використанням технології NodeJS та фреймворку Telegraf.

ІНТЕНСИФІКАЦІЯ НАВЧАННЯ, ВЕБ СЕРВЕР, NODEJS, TELEGRAF, NESTJS, TELEGRAM-BOT.

ЗМІСТ

ВСТУП 5

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 7

1.1 Дослідження актуальності проблеми 7

1.2 Роль та потенціал використання месенджерів як інструменту підвищення ефективності навчальних платформ 8

1.3 Огляд існуючих інструментів інтенсифікації навчального процесу 10

1.4 Технічні виклики при інтеграції Telegram-ботів в освітні платформи 11

1.5 Постановка задачі 12

2 АНАЛІТИЧНА ЧАСТИНА 13

2.1 Визначення вимог до інформаційної технології 13

2.2 Аналіз інструментів розробки Telegram-ботів 14

2.3 Структурно-функціональне моделювання 24

2.4 Моделювання варіантів використання 26

2.5 Моделювання бази даних 32

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ 35

3.1 Ініціалізація та налаштування проєкту на платформі NestJS 35

3.2 Програмна реалізація telegram-боту 39

3.3 Програмна реалізація веб-серверу 46

3.4 Огляд створеної інформаційної технології 48

ВИСНОВОК 53

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 54

ДОДАТОК А 58

ВСТУП

Актуальність даної роботи полягає у необхідності підвищення ефективності та інтенсивності навчального процесу за допомогою інноваційних інформаційних технологій. Сучасні освітні системи часто зіштовхуються з проблемами відстеження прогресу навчання та оперативного зв'язку між учасниками. Хоча деякі рішення такі як Telegram Bot to Open edX® platform [1] та наукові розробки закордонних авторів таких як О. Е. Моліна [2] та М. Халіл [3] частково вирішують їх, але не забезпечують повної інтеграції з навчальною платформою бо позиціонуються як додаток до існуючої системи. Тому необхідно розробити нову модель інтенсифікації навчального процесу з веденням поточного стану навчання та оперативного інформування учасників навчального процесу, яка вирішить цю задачу використовуючи технологію telegram-bot.

Об'єкт дослідження - механізми інтенсифікації навчання через забезпечення взаємодії, відстеження прогресу та оперативного інформування учасників освітнього процесу.

Предмет дослідження - модель організації навчального процесу із ведення поточного стану навчання та оперативного інформування учасників навчального платформи із використанням telegram-bot.

Суперечність, що вирішується у роботі. Існуючі моделі не забезпечують повної інтеграції із навчальними платформами, не мають ефективного механізму зворотного зв’язку та інформування учасників освітнього процесу. Має бути вирішена практична задача розроблення нової, більш комплексної за попередні, моделі з можливістю відстеження прогресу та ефективними механізмами комунікації користувачів.

Гіпотеза - якщо використати технологію telegram-bot та виконати повну інтеграцію із навчальною платформою, то можна інтенсифікувати навчальний

процес через точне відстеження прогресу та оперативну реакцію на освітні потреби учасників.

Новизна даної роботи полягає в тому, що на відміну від існуючих розробок таких як Telegram Bot to Open edX® platform або робіт авторів О. Е. Моліна, М. Халіл та ін., модель що пропонується має глибоку інтеграції із навчальною платформою, що дозволяє інтенсифікувати навчальний процес за допомогою відстеження поточного стану навчання та оперативного інформування користувачів системи.

Структура роботи. Дана кваліфікаційна робота магістра складається зі вступу, інформаційно-аналітичного огляду для аналізу використання Telegram-ботів у навчальному процесі, аналітичної частини з розробкою функціональних вимог та оглядом технологій, практичної реалізації інформаційної технології, висновків, що підсумовують результати дослідження, списку використаних джерел та додатків.

Впровадження: Одним із ключових етапів розробки нової моделі було активне тестування інтегрованої системи на навчальній платформі MIX. Це дозволило не лише перевірити практичну придатність та ефективність запропонованої моделі, але й зібрати необхідні дані для її подальшого удосконалення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Дослідження актуальності проблеми

У сучасному освітньому ландшафті, де швидкість і точність інформаційного обміну відіграють вирішальну роль, розробка телеграм-бота для забезпечення комунікації між студентами та викладачами стає надзвичайно актуальною. Це стає ще важливішим в умовах, коли навчальні заклади змушені адаптуватися до гнучких і змішаних форматів навчання, що включають дистанційне та очне навчання.

Період пандемії COVID-19 [4] і повномасштабне вторгнення в Україні ще більше підкреслили необхідність розвитку цифрових засобів комунікації. Обмеження, пов'язані з карантином, та періоди невизначеності змусили багато навчальних закладів швидко перейти на онлайн-формати навчання, зробивши ефективний обмін інформацією критично важливим.

У цьому контексті, використання месенджерів як Telegram, а також інших інтерактивних платформ як Slido та Mentimeter [5,6], стало не просто зручністю, а необхідністю, відкриваючи нові шляхи для оптимізації освітнього процесу. Подібні інструменти дозволяють проводити інтерактивні опитування та голосування, які можуть використовуватись під час онлайн-лекцій або вебінарів для залучення студентів і підвищення їх активності. Ці платформи надають викладачам миттєвий зворотній зв'язок від студентів, що допомагає адаптувати навчальний процес до потреб аудиторії, покращуючи розуміння матеріалу та забезпечуючи більш ефективне навчання.

Однак, ключовою перевагою телеграм-бота є його можливість автоматизації розсилки повідомлень. Навчальні заклади часто стикаються з необхідністю оперативно інформувати велику кількість студентів про зміни в розкладах, проведення заходів, терміни здачі робіт та інші організаційні питання. Телеграм-бот може автоматично розсилати такі повідомлення, що

забезпечує своєчасне отримання актуальної інформації кожним учасником навчального процесу.

Інша важлива функція телеграм-бота — забезпечення швидкого зворотного зв'язку. Бот може слугувати зручним інструментом для збору запитів від студентів, що може включати відповіді на анкети, оцінки лекцій, а також запитання і пропозиції до викладачів. Це підвищує залученість та активність студентів що допомагає навчальним закладам швидко реагувати на потреби та вимоги учнівського складу.

Телеграм-бот також може ефективно організовувати дискусійні групи, автоматично створюючи та управляючи групами чату для різних академічних груп або проєктних команд, що сприяє спілкуванню і співпраці між студентами.

Таким чином, впровадження телеграм-бота як засобу інтенсифікації навчання та оперативного інформування учасників навчального платформи в освітньому процесі стає стратегічним кроком для університетів, що прагнуть підвищити ефективність та адаптивність, реагуючи на динамічні зміни у світі освіти.

1.2 Роль та потенціал використання месенджерів як інструменту підвищення ефективності навчальних платформ

Telegram в Україні відіграє важливу роль в обміні даними, особливо під час війни, забезпечуючи швидкий доступ до інформації та комунікацій. За результатами дослідження компанії InMind [7], Telegram продемонстрував значне зростання використання серед українців (Рисунок 1.1) як для особистого спілкування, так і для отримання актуальних новин.

Рисунок 1.1 Користування соціальними мережами для спілкування та отримання новин

Наукові дослідження [2] підтверджують, що інтеграція месенджерів зокрема Telegram у навчальні курси сприяє підвищенню рівня студентської залученості за допомогою інтерактивних та спільних активностей. Такі інструменти як опитування, групові обговорення та спільні проєкти вносять значний внесок у динаміку навчання, стимулюючи інтенсивне сприйняття матеріалу та підвищуючи мотивацію студентів. Месенджери також допомагають формувати спільноту, надаючи платформи для спільної роботи та обговорень, що є критично важливим в умовах дистанційного навчання. Вони підтримують безперервне навчання, дозволяючи студентам доступати до матеріалів у будь-який час і з будь-якого місця, що особливо важливо для зайнятих студентів або тих, хто поєднує навчання з роботою. Усе це допомагає викладачам адаптувати навчальний процес до індивідуальних потреб учнів, роблячи навчання більш ефективним.

Крім того, Telegram полегшує управління освітніми ресурсами, автоматизуючи збір і аналіз даних про використання матеріалів і студентську активність, допомагаючи оптимізувати навчальні плани і розподіл ресурсів. Він надає навчальним закладам гнучкий інструмент для швидкої адаптації до змін у технологіях і освітніх методиках, дозволяючи легко впроваджувати інновації та забезпечувати студентам доступ до сучасних освітніх ресурсів. Використання Telegram, таким чином, не тільки сприяє підвищенню залученості і мотивації студентів, але і дозволяє університетам ефективніше

управляти освітніми ресурсами та адаптуватися до постійно змінюваного освітнього ландшафту.

1.3 Огляд існуючих інструментів інтенсифікації навчального процесу

Окрім месенджерів, у сучасному освітньому просторі активно використовуються інші цифрові інструменти, які сприяють підвищенню ефективності навчальних платформ. Інтерактивні відео, наприклад, на платформі Edpuzzle [8], дозволяють викладачам вставляти запитання та завдання прямо в відеоконтент, що допомагає збільшити залученість студентів та покращити засвоєння матеріалу. Дослідження вказують на значне зростання уваги та активності студентів під час перегляду інтерактивних відео порівняно з традиційними лекційними відео.

Ще одним ефективним інструментом є електронні книги з адаптивним навчанням, які використовують платформи типу Smart Sparrow чи Knewton [9]. Ці платформи дозволяють адаптувати навчальний матеріал до індивідуальних потреб кожного студента на основі аналізу їх відповідей та навчального прогресу в реальному часі. Такий підхід дозволяє студентам працювати з матеріалом, який найкраще відповідає їх поточному рівню знань та здібностям, забезпечуючи більш ефективне та цільове навчання.

Не можна ігнорувати й вплив гейміфікації на навчальний процес. Використання ігрових елементів, як на платформі Kahoot! [10], сприяє збільшенню мотивації студентів та їх залученню через змагальні та інтерактивні аспекти навчання. За результатами досліджень, впровадження гейміфікації в освіту значно покращує засвоєння матеріалу та підтримує високий рівень активності студентів в навчальному процесі.

Таким чином, інтеграція цих новітніх цифрових інструментів в навчальні платформи відкриває широкі можливості для оптимізації та

персоналізації навчального процесу, сприяючи підвищенню ефективності освіти та адаптації до індивідуальних потреб та здібностей кожного студента.

1.4 Технічні виклики при інтеграції Telegram-ботів в освітні платформи

Однією з першочергових проблем є сумісність API Telegram з іншими освітніми системами. Веб-сервіси і API кожної платформи можуть мати унікальні вимоги та специфікації, які потребують детального аналізу та адаптації з боку розробників. В статті Jose Belda-Medina та ін. [11] наголошується, що розробка міжплатформенної взаємодії вимагає не тільки глибокого розуміння веб-сервісів, але й способів їх інтеграції для забезпечення стабільного обміну даними між системами, що часто включає вирішення проблем сумісності та виконання тестувань.

Безпека даних є ще одним критичним аспектом. Використання месенджерів у освіті створює додаткові виклики для забезпечення конфіденційності та захисту інформації. Дослідження J. J. Merelo [12] підкреслює важливість захисту особистих даних і забезпечення конфіденційності в освітньому середовищі. Це особливо актуально при інтеграції технологій, які взаємодіють з великими обсягами особистої інформації. Також необхідно створити чіткі протоколи відповіді на інциденти, що включають витік даних або їх компрометацію, забезпечуючи швидке відновлення та мінімізацію збитків.

Масштабованість є вирішальною для забезпечення ефективності та стабільності освітніх технологій. У великих освітніх установах, де тисячі користувачів можуть одночасно взаємодіяти з системою, здатність швидко обробляти великий об'єм запитів без зниження продуктивності є обов'язковою. Jeya Amantha Kumar [13] у своїй публікації вказує на необхідність ретельного планування архітектури системи, включаючи вибір потужних серверів, оптимізацію баз даних та використання розподілених обчислень особливо в

контексті великих навчальних установ, де чатботи повинні ефективно обробляти велику кількість запитів.

Таким чином, ретельне вирішення цих технічних викликів є важливим для успішної інтеграції Telegram-ботів в освітні платформи. Ефективне вирішення цих питань забезпечить не тільки високу функціональність та безпеку систем, але й сприятиме покращенню загальної надійності та користувацької прийнятності цих інноваційних освітніх технологій.

1.5 Постановка задачі

Метою даної кваліфікаційної роботи є розробка та впровадження інформаційної технології інтенсифікації навчального процесу за допомогою оптимізації комунікації між викладачами та студентами, автоматизації процесів створення навчальних чатів по кожному предмету, а також надсилання актуальних повідомлень про оновлення розкладу, додавання завдань та виставлення оцінок.

Серед завдань виділимо наступні:

1. Дослідження потреб учасників системи в контексті використання месенджерів для навчальних цілей.
2. Розробка архітектури Telegram-бота, що включає модулі для створення чатів, управління навчальними матеріалами, інформування про оновлення.
3. Реалізація інтерфейсу для зв'язку бота з базою даних навчальної платформи, забезпечуючи синхронізацію інформації.
4. Впровадження механізму автоматичного надсилання повідомлень.

2 АНАЛІТИЧНА ЧАСТИНА

2.1 Визначення вимог до інформаційної технології

Для успішної інтеграції телеграм-бота з навчальною платформою університету необхідно визначити ключові технічні та функціональні вимоги. Це дозволить забезпечити високу ефективність взаємодії між користувачами та системою, а також максимальну користь від використання інтегрованих рішень.

Технічні вимоги

- Масштабованість: система повинна бути здатна масштабуватися для обслуговування зростаючої кількості користувачів і збільшення обсягу даних без втрати продуктивності.
- Безпека: важливо забезпечити захист персональних даних студентів та викладачів, а також гарантувати безпеку обміну інформацією. Бот повинен використовувати сучасні методи шифрування та аутентифікації.
- Інтеграція API: бот має підтримувати інтеграцію через API для зв'язку з навчальною платформою, що дозволить витягувати та відправляти дані безпосередньо до/з системи.
- Вартість: система повинна мати конкурентоспроможну ціну з урахуванням всіх необхідних функцій та послуг. Очікується, що вартість впровадження та обслуговування системи буде оптимізована для забезпечення високої віддачі інвестицій.

Функціональні вимоги

- Автоматизація комунікацій: бот має забезпечувати автоматизацію розсилки повідомлень, оголошень, нагадувань про терміни здачі робіт, зміни у розкладі та інші важливі події.

- Інтерактивність: бот повинен підтримувати інтерактивність, дозволяючи студентам та викладачам відповідати на повідомлення, задавати питання та отримувати відповіді в режимі реального часу.
- Підтримка групових взаємодій: бот повинен підтримувати групові взаємодії, такі як організація групових чатів, координація спільних проєктів і ініціювання дискусій, що забезпечує платформу для співпраці та підвищує залученість студентів.

Ці вимоги встановлюють основу для ефективної та безпечної роботи телеграм-бота, що інтегрований з навчальною платформою, і відповідають сучасним стандартам цифрової освіти.

2.2 Аналіз інструментів розробки Telegram-ботів

Для розробки Telegram-бота існує багато технологій, серед яких Python, Java, PHP, і Node.js. Вибір конкретного інструменту для реалізації залежить від специфічних вимог проєкту, наявності розробників, що володіють відповідними навичками, та інтеграційних можливостей з іншими системами. Розглянемо кілька популярних мов для розробки Telegram-ботів, аналізуючи їх переваги та недоліки [14].

Python [15] – часто використовується завдяки простоті синтаксису та великій кількості бібліотек. Python добре підходить для швидкої розробки прототипів.

Переваги:

- Простота та читабельність: python відомий своїм чистим і легким для читання синтаксисом, що спрощує процес навчання і розробки.
- Велика стандартна бібліотека і екосистема: численні бібліотеки, такі як python-telegram-bot, полегшують створення ботів з широким спектром функціоналу.

- Спільнота: велика та активна спільнота розробників надає значну підтримку та багато ресурсів для вирішення проблем.

Недоліки:

- Швидкість виконання: як інтерпретована мова, Python може бути повільнішим за компільовані мови.
- Управління пам'яттю: автоматичне управління пам'яттю в Python може призвести до використання більшої кількості ресурсів порівняно з більш низькорівневими мовами.

Java [16] – володіє високою продуктивністю та широкими можливостями для масштабування. Це мова з сильною типізацією, що може бути перевагою у великих проєктах.

Переваги:

- Масштабованість і продуктивність: Java має високу продуктивність і підтримує масштабованість, що робить її відмінним вибором для великих корпоративних систем.
- Переносимість: програми на Java можуть працювати на будь-якій машині, що підтримує Java Virtual Machine (JVM), забезпечуючи високу переносимість.
- Міцність: строга типізація і об'єктно-орієнтоване програмування сприяють створенню надійних і легко підтримуваних програм.

Недоліки:

- Складність: Java може здаватися занадто складною для простих проєктів через вимогу до написання більшої кількості коду і жорсткої структури.

PHP [17] – традиційно використовується для розробки веб-сайтів, але також може бути застосований для створення простих ботів.

Переваги:

- Швидкий старт: легко почати розробку, особливо для розробників, знайомих з веб-розробкою.

- Добре інтегрується з вебом: вбудована підтримка HTTP протоколу та HTML.

Недоліки:

- Не найкраща підтримка асинхронності: PHP традиційно синхронний, що може ускладнити розробку реалізацій, що потребують великої кількості паралельних операцій.
- Безпека: PHP може бути вразливим, якщо не використовувати сучасні практики безпеки.

Node.js [18] – кросплатформенне середовище виконання JavaScript на стороні сервера. Завдяки асинхронній, орієнтованій на події архітектурі, вона ефективно підходить для створення додатків, які функціонують у реальному часі, таких як веб-сервери та API. Екосистема Node.js, що включає в себе набір бібліотек та інструментів, доступних через менеджер пакетів, що значно розширює можливості платформи, забезпечуючи швидке розгортання функціональних рішень та інновацій.

Переваги:

- Асинхронність з неблокуючим вводом/виводом: Node.js ідеально підходить для розробки програм, що потребують високої продуктивності та обробки великої кількості запитів.
- Велика екосистема: npm, менеджер пакетів Node.js, містить величезну кількість доступних пакетів і модулів.
- Використання JavaScript: можливість використовувати той же язик програмування на клієнтській та серверній стороні може спростити розробку і зменшити кількість помилок.

Недоліки:

- Управління пам'яттю: хоча Node.js ефективно використовує пам'ять, робота з великими об'ємами бінарних даних може бути складною.
- Залежність від зовнішніх бібліотек: часто залежність від зовнішніх модулів може призвести до проблем із безпекою або стабільністю.

Node.js було обрано як основний інструмент для розробки Telegram-бота з кількох причин:

- універсальність мови;
- простоту в використанні;
- орієнтованість на створення API;
- багатий функціонал;
- велику кількість відкритих бібліотек та готових рішень;
- підтримка асинхронності;

Розробка Telegram ботів на Node.js може бути спрощена за допомогою спеціалізованих фреймворків. Ось декілька популярних фреймворків, які часто використовуються розробниками:

Telegraf [19] є одним з найпопулярніших фреймворків для розробки Telegram ботів у Node.js. Він пропонує багатий набір функцій, які спрощують обробку повідомлень, команд та інших типів даних, що надходять з Telegram.

- Переваги: легка інтеграція, підтримка middleware, широкі можливості для розширення функціоналу, велика спільнота.
- Сценарії використання: ідеально підходить для складних ботів, які вимагають обробки різноманітних взаємодій з користувачем і інтеграції з зовнішніми API.

node-telegram-bot-api [20] – є ще одним популярним вибором серед розробників Node.js для створення Telegram ботів. Він пропонує простий і зрозумілий API для взаємодії з Telegram Bot API.

- Переваги: простота використання, хороша документація, активна підтримка з боку спільноти.
- Сценарії використання: підходить для розробників, які шукають легкий у використанні інструмент для швидкої розробки простих або середньої складності ботів.

Для виконання поставленої задачі був обраний фреймворк Telegraf з кількох причин:

- розширені можливості для обробки повідомлень;
- інтеграцію з клавіатурою Telegram;
- підтримку inline-режиму;
- Можливість використання middleware дозволяє легко інтегрувати додаткову логіку обробки повідомлень, що спрощує розробку складних ботів;

Враховуючи ці переваги, Node.js разом з фреймворком Telegraf є оптимальним вибором для створення Telegram-бота, орієнтованого на високу продуктивність, ефективну роботу з веб-технологіями та потреби інтеграції з іншими системами. Це дозволяє реалізувати вимоги до інтерфейсу взаємодії, а також забезпечити високий рівень адаптивності та налаштування під конкретні потреби користувачів та інфраструктури університету.

Для керування залежностями в нашому проєкті потрібен надійний менеджер пакетів. На даний момент разом з середовищем NodeJS можна використовувати npm, yarn або pnpm [21], кожен з цих інструментів має свої унікальні переваги та особливості.

Пакетний менеджер npm що постачається з Node.js за замовчуванням. Він встановлює залежності в ієрархічному порядку, кожен пакет розміщується у власній папці node_modules. Ця структура забезпечує ізоляцію залежностей та мінімізує конфлікти версій, хоча й вимагає додаткового дискового простору.

Yarn, розроблений Facebook, є альтернативним менеджером пакетів, який пропонує швидке завантаження завдяки паралельному завантаженню модулів. Він використовує плоску модель встановлення, що допомагає уникнути дублювання пакетів, але може спричинити ризик появи конфліктів. Yarn також дозволяє встановлювати пакети офлайн.

Останній, pnpm — це економічніший варіант, який використовує загальне сховище пакетів та жорсткі посилання для зменшення використання дискового простору. Його модель ієрархічної встановлення залежностей схожа на npm, що може призводити до дублювання, але із значно меншим обсягом використовуваного дискового простору.

З огляду на лідируючі позиції npm як стандартного пакетного менеджера для Node.js, його велику популярність та широке визнання серед розробників, було вирішено використати саме його для реалізації інформаційної системи.

Це рішення гарантує стабільне управління залежностями та відмінно інтегрується з нашим проєктом на базі Node.js, при цьому не вимагаючи додаткових дій для встановлення чи налаштування програмного забезпечення.

Для створення веб-серверу в нашому проєкті було розглянуто кілька відомих фреймворків і технологій, кожна з яких має свої унікальні специфікації, що робить їх придатними для різних типів проєктів.

Express [22] є одним із найпопулярніших фреймворків для Node.js, відомий своєю швидкістю та мінімалізмом. Він надає потужні інструменти для маршрутизації та проміжного програмного забезпечення, що дозволяє розробникам легко створювати веб-сервери та API.

Переваги:

- Швидкість розробки: Express спрощує налаштування веб-сервера, що зменшує час розробки.
- Гнучкість: Завдяки великій кількості проміжного програмного забезпечення та спільноти, Express дуже гнучкий у використанні.
- Сильна підтримка спільноти: Велика кількість доступних ресурсів і модулів спрощує розв'язання проблем.
- Недоліки:
- Масштабованість: У деяких випадках, Express може не забезпечити достатній рівень абстракції або функціональності для великих та складних додатків.

Koa [23] — це фреймворк нового покоління, створений розробниками Express, який використовує сучасні функції JavaScript, такі як асинхронні функції (async/await), що дозволяє краще контролювати обробку запитів і відповідей.

Переваги:

- Сучасний синтаксис: Koa дозволяє використовувати новітні функції JavaScript для спрощення і покращення якості коду.
- Мінімалізм: Koa не має вбудованих мідлварів, що надає розробникам повний контроль над стеком технологій.

Недоліки:

- Крива навчання: для ефективного використання Koa потрібне добре розуміння асинхронного програмування в JavaScript.

NestJS є прогресивним Node.js фреймворком, який побудований на базі Express і опціонально може використовувати Fastify. Це забезпечує NestJS усіма перевагами Express, такими як швидкість і гнучкість, але виправляє деякі з його недоліків, наприклад, недостатню структурованість та масштабованість при роботі з великими додатками.

NestJS [24] використовує декларативний підхід до маршрутизації і зберігає архітектурну консистентність завдяки використанню декораторів і модулів. Водночас, інтеграція TypeScript забезпечує строгу типізацію та поліпшує якість коду, що робить NestJS ідеальним для розробки масштабованих та складних серверних додатків.

Переваги:

- Інтеграція TypeScript: NestJS ідеально підходить для тих, хто віддає перевагу строгої типізації і сучасним підходам до програмування.
- Архітектурна консистентність: використання декораторів і модулів сприяє створенню організованого та легко підтримуваного коду.
- Міцна екосистема: підтримка різних інструментів і бібліотек, таких як TypeORM і GraphQL, розширює можливості розробки.

Недоліки:

- Вища складність: NestJS може бути надмірно складним для дрібних або простих проєктів через багатий набір функціональностей та абстракцій.

З урахуванням нашої потреби в створенні масштабованого і легко підтримуваного веб-сервера, NestJS було обрано як основний фреймворк для розробки. Використання TypeScript та розширені можливості архітектурного планування забезпечують стабільність та легкість у впровадженні нового функціоналу. Окрім того, для NestJS є наявна спеціалізованої бібліотеки nestjs-telegraf для інтеграції з Telegraf. Ця бібліотека спрощує роботу з Telegram API, надаючи готові до використання абстракції та методи, що ідеально вписуються в модульну структуру NestJS. Використання цієї бібліотеки дозволяє максимально використати можливості обох технологій, оптимізувати процес розробки та забезпечити високу продуктивність веб-сервера.

При створенні веб-сервера ключовим аспектом є вибір бази даних, яка найкраще відповідає потребам інформаційної системи. Були розглянуті популярні бази даних, серед яких SQLite, MySQL, PostgreSQL, MongoDB та MariaDB.

SQLite [25] — це легка вбудована база даних, яка є ідеальною для розробки та тестування, оскільки не потребує окремого серверного програмного забезпечення. Вона використовується у мобільних застосунках, вбудованих системах та невеликих додатках, де основними перевагами є мінімальні вимоги до конфігурації системи та легкість використання.

MySQL [26] — це найпоширеніша реляційна система управління базами даних, яка використовується у багатьох комерційних та відкритих проєктах. Дана СУБД відома своєю високою продуктивністю, масштабованістю та гнучкістю у підтримці різноманітних типів додатків, включаючи веб-сайти високої навантаженості та бізнес-додатки.

PostgreSQL [27] — це об'єктно-реляційна система управління базами даних, яка надає розширену підтримку SQL стандартів та інші передові функції, такі як складні запити, зовнішні ключі, тригери, види, транзакції, підтримка MVCC та велика кількість розширень. Цю СУБД часто вибирають для розробки великих систем із складними даними та високими вимогами до надійності.

MongoDB [28] — це документоорієнтована NoSQL база даних, яка надає високу продуктивність, доступність та легку масштабованість. MongoDB використовує формат JSON-подібних документів з динамічними схемами, зробивши її ідеальною для розробки застосунків, що потребують швидкого розгортання та гнучкості в управлінні структурами даних.

MariaDB [29] — це форк, створений оригінальними розробниками MySQL після її придбання компанією Oracle. MariaDB розроблена з акцентом на відкритість, інновації та високу продуктивність. Вона сумісна з MySQL, що дозволяє користувачам легко мігрувати без змін у програмах. MariaDB продовжує розвиватися з новими функціями, такими як підтримка швидкісних кешувань, оптимізації для масштабованих систем і розширення для зберігання даних.

На основі ретельного аналізу та врахування технічних вимог, MariaDB було обрано як оптимальну базу даних для розробки. Цей вибір ґрунтується на потребі в масштабованості, високій продуктивності та легкості інтеграції з іншими компонентами системи.

Для забезпечення ефективної взаємодії з даними і плавному розвитку системи критично важливим є вибір бібліотеки для роботи з базами даних. У середовищі Node.js існують такі відомі бібліотеки для роботи з базами даних, як, Sequelize, TypeORM та Knex, кожна з яких пропонує різні інструменти та можливості для оптимізації взаємодії з різними типами баз даних.

Sequelize [30] — це об'єктно-реляційний відображувач (ORM), який підтримує всі популярні бази даних. Він надає багатий набір функцій для

управління реляційними базами даних, включаючи транзакції, асоціації, завантаження за запитом та багато іншого, що робить його дуже потужним інструментом для складних проєктів.

TypeORM [31] — це ORM, який підходить для використання з TypeScript. Він підтримує багато функцій реляційних баз даних, таких як відносини, наслідування, реплікація, індекси та інше. TypeORM легко інтегрується з NestJS, що робить його популярним вибором у розробників, які використовують цей фреймворк.

Knex [32] — це SQL-будівельник для Node.js, який підтримує роботу з більшістю популярних СУБД. Він дозволяє легко створювати та виконувати SQL-запити через ланцюжковий інтерфейс. Він також підтримує міграції, що дозволяє систематично управляти змінами у базі даних, що є ключовим для підтримки і розвитку великих проєктів.

На основі наших технічних вимог та потреб у гнучкості управління схемами баз даних, для нашого проєкту було обрано Knex. Ця бібліотека дозволяє нам гнучко керувати SQL-запитами і забезпечує підтримку міграцій

Для інтеграції Knex у середовище NestJS існує спеціалізована бібліотека nestjs-knex, яка спрощує конфігурацію та використання Knex у проєктах на базі NestJS. Для взаємодії з базою даних MariaDB, необхідно встановити додатковий драйвер. Хоча MariaDB є форком MySQL, для оптимальної роботи з цією базою даних у середовищі Node.js рекомендується використовувати драйвер mysql2 [33]. Даний драйвер пропонує поліпшену продуктивність і додаткові можливості, які оптимізовані для роботи з сучасними веб-додатками.

Для ефективного управління конфігураціями та змінними середовища в проєктах на базі NestJS, використовується модуль @nestjs/config [34]. Цей модуль дозволяє легко інтегрувати та використовувати змінні середовища з файлів .env для різних стадій розробки, тестування та виробництва, забезпечуючи безпечне та ефективне управління конфігурацією.

Отже, для реалізації інтенсифікації навчального процесу за допомогою telegram-бота було обрано:

- платформу Node.js як основу для веб серверу та телеграм-боту;
- фреймворк NestJS для створення серверної частини застосунка;
- бібліотеку Telegraf разом з nest-telegraf для створення Telegram-бота;
- систему контролю баз даних MarinaDB;
- бібліотеку Knex разом з драйвером mysql2 як SQL-будівельник для роботи з базою даних;
- бібліотеку @nestjs/config для управління конфігураціями в NestJS;

2.3 Структурно-функціональне моделювання

IDEF0 є стандартизованим підходом до візуального документування процесів за допомогою блоків, що відображають процеси, і стрілок, які показують входи (інформація або ресурси, необхідні для виконання функції), виходи (результати діяльності), механізми (інструменти виконання функції) та керування (засоби, що впливають на процес).

Основний процес, розглянутий у моделі — "Робота телеграм-бота" (Рисунок 2.1). У якості входів використовуються повідомлення та зворотний зв'язок від учасників системи, а також оновлення з навчальної платформи. До механізмів належать учасники системи, сервер обробки та база даних. Процес керується через правила використання та інструкції для учасників системи. Виходами є інформування для учасників системи та зворотній зв'язок від них.

У результаті декомпозиції основного процесу були виділені 3 підпроцеси (Рисунок 2.2).

Рисунок 2.1 Контекстна діаграма процесу «Робота телеграм боту» в нотації IDEF0

Підпроцес обробки повідомлень користувачів включає прийом та аналіз вхідних звернень від учасників системи. В якості входів для цього процесу виступають відгуки і запити користувачів, які надходять через інтерфейс телеграм-бота. Механізм обробки базується на сервері, який використовує алгоритми аналізу тексту для класифікації звернень і визначення подальших дій. Контроль за процесом здійснюється через набір правил і інструкцій, які допомагають визначити необхідність подальшої взаємодії з користувачем. Виходами цього підпроцесу є відповідь на повідомлення з навчальної системи.

Інтеграція з навчальною платформою забезпечує синхронізацію даних між телеграм-ботом і навчальною платформою. Основна мета полягає в автоматизації передачі інформації про оновлення та нові матеріали, доступні на платформі. Вхідні дані включають повідомлення для учасників системи та оновлення на навчальній платформі, які обробляються сервером обробки даних. Керування процесом здійснюється через ряд інструкцій і протоколів, що гарантують коректність та безпеку передачі даних. Виходи цього підпроцесу включають інформація про оновлення на платформі та

персоніфіковані повідомлення для користувачів, що стимулює їх активність і сприяє ефективному навчанню.

Підпроцес розсилки повідомлень орієнтований на масову комунікацію з учасниками системи. Вхід для цього процесу – це інформація, отримана від навчальної платформи, персоніфіковані повідомлення та відповіді. Механізми включають сервер обробки та учасників системи. Керування здійснюється за допомогою правил використання, які визначають час і обсяг розсилок залежно від актуальності інформації. Виходом є доставка повідомлень до кінцевих користувачів, що підтримує інформованість учасників і забезпечує зворотній зв’язок.

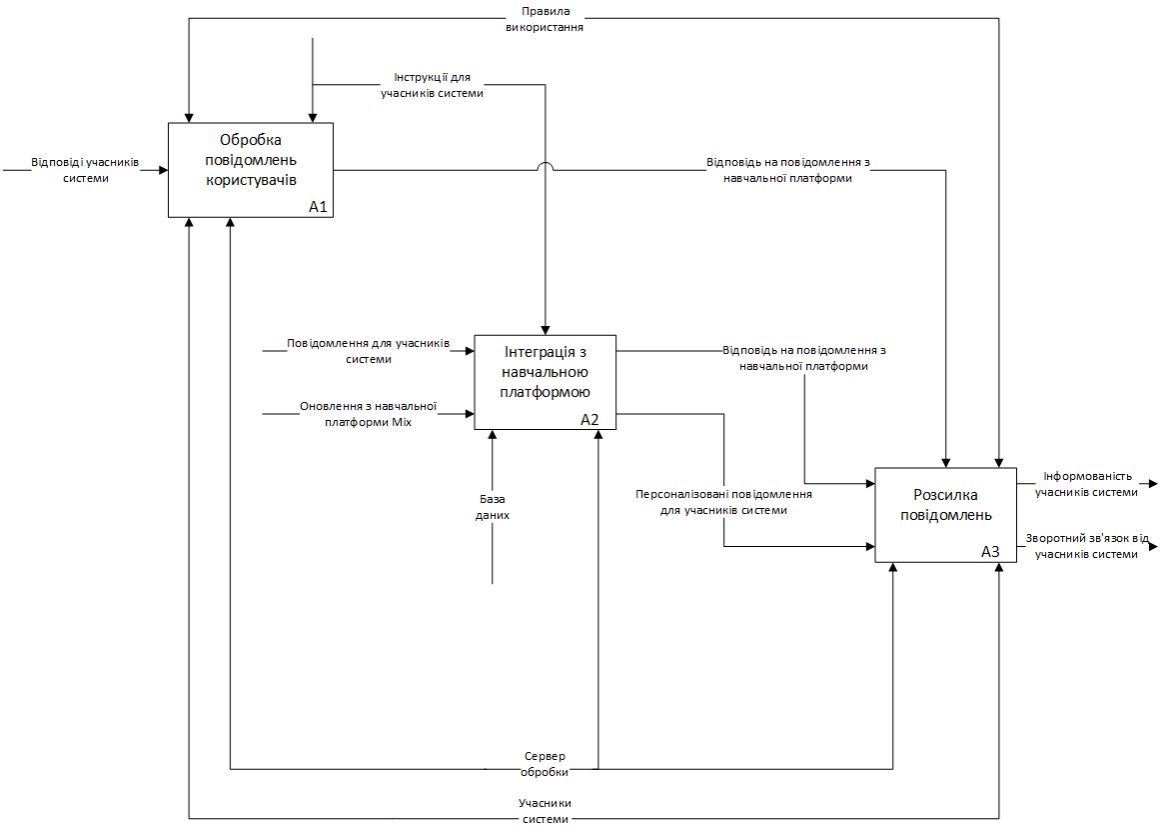

Рисунок 2.2 Діаграма декомпозиції процесу «Робота віртуального тренажера» в нотації IDEF0

2.4 Моделювання варіантів використання

Задля опису функціональності та поведінки системи була побудована Use case діаграма. Це дозволяє визначити дійових осіб (акторів), їх взаємодію

з системою та очікувану функціональність, поняття в глосарії предметної області, які стосуються детального опису функціональності системи (тобто прецедентів).

Модель варіантів використання складається з опису акторів системи (Таблиця 2.1), опису всіх варіантів використання системи (Таблиця 2.2) та діаграми варіантів використання (Рисунок 2.3).

Таблиця 2.1 Опис усіх варіантів використання системи

Назва актора	**Опис**
Актори-користувачі системи	
Приватний користувач(ПК)	Користувач який може зареєструватися в телеграм боті та отримувати або відповідати на повідомлення від АСУ університету.
Користувач групи(КГ)	Функціонал відповідний до приватного користувача, проте, також має додаткові можливості для роботи з групами: реєстрація групи в навчальній платформі, отримання класових повідомлень, запрошення учасників класу до групи.
Зовнішні актори системи	
АСУ університету (АСУ)	Центр взаємодії учасників системи дозволяє реєструвати користувачів, відправляти і отримувати відповіді на повідомлення.
Навчальна платформа університету (НПУ)	Навчальна платформа, що дозволяє додавати до класів телеграм групи та відправляти до них повідомлення.

Таблиця 2.2 Опис усіх варіантів використання системи

Назва ВВ	Сценарій
UC01 Генерація персонального посилання	ПК та АСУ зацікавлені у створенні персонального посилання для реєстрації. Х.1 Передумови ПК знаходиться в персональному кабінеті. Х.2 Процес ПК натискає на кнопку «Підключити телеграм», АСУ генерує персональне посилання на бот на надає його ПК Х.3 Альтернативні потоки (обробка помилок) При збою зв’язку виникає відповідне повідомлення у кабінеті. Х.4 Післяумови Після отримання персонального повідомлення ПК потрапляє в телеграм бот.
UC02 Реєстрація в системі	ПК та АСУ зацікавлені у реєстрації користувача. Х.1 Передумови ПК в телеграм боті із введеною командую отриманою з персонального посилання. Х.2 Процес ПК виконує команду, після чого данні через телеграм-бот передаються в АСУ для реєстрації користувача. Х.3 Альтернативні потоки (обробка помилок) При збою зв’язку виникає відповідне повідомлення у телеграм боті. Х.4 Післяумови

Назва ВВ	Сценарій
	Після реєстрації користувача в АСУ, ПК отримує повідомлення про успішну реєстрацію.
UC03 Інформування користувача	ПК та АСУ зацікавлені інформуванні користувача про оновлення в системі. Х.1 Передумови АСУ має підготовлене повідомлення для ПК. Х.2 Процес АСУ надсилає повідомлення для ПК через телеграм-бот. Х.3 Альтернативні потоки (обробка помилок) При збою зв'язку виникає відповідне повідомлення у системі АСУ. Х.4 Післяумови ПК отримує повідомлення від АСУ
UC04 Відповідь на повідомлення	ПК та АСУ зацікавлені в отриманні відповіді на повідомлення. Х.1 Передумови ПК обирає повідомлення на яке хоче надати відповідь, та підготовлений текст. Х.2 Процес ПК надсилає відповідь через телеграм-бот. Х.3 Альтернативні потоки (обробка помилок) При збою зв'язку виникає відповідне повідомлення у телеграм боті. Х.4 Післяумови АСУ отримує відповідь на повідомлення від ПК.

Назва ВВ	Сценарій
UC05 Додавання бота в групу	КГ зацікавлений в отриманні ідентифікатору групи в телеграмі. Х.1 Передумови КГ знаходиться в телеграм групі з доданим ботом. Х.2 Процес КГ виконує команду для отримання ідентифікатора групи. Х.3 Альтернативні потоки (обробка помилок) При збою зв'язку виникає відповідне повідомлення від телеграм боті. Х.4 Післяумови КГ отримую ідентифікатор групи.
UC06 Реєстрація групи в навчальній системі	КГ та НПУ зацікавлені в реєстрації телеграм групи для класу платформи. Х.1 Передумови КГ знаходиться в профілі навчальної платформи. Х.2 Процес КГ вводить ідентифікатор групи та натискає на кнопку реєстрації. Х.3 Альтернативні потоки (обробка помилок) При збою зв'язку виникає відповідне повідомлення у системі навчальної платформи. Х.4 Післяумови НПУ реєструє телеграм групу для класу.

Назва ВВ	Сценарій
UC07 Інформування класу	КГ та НПУ зацікавлені інформуванні класу про оновлення даних в навчальні платформі. Х.1 Передумови НПУ має підготовлене повідомлення для КГ. Х.2 Процес НПУ надсилає повідомлення для КГ через телеграм-бот. Х.3 Альтернативні потоки (обробка помилок) При збою зв'язку виникає відповідне повідомлення у системі НПУ. Х.4 Післяумови КГ отримує повідомлення від НПУ.
UC08 Запрошення членів класу в групу	КГ та ПК зацікавлені запрошенні членів класу навчальної платформи в телеграм групу. Х.1 Передумови КГ знаходиться в телеграм групі з доданим ботом. Х.2 Процес КГ для запрошення користувачів. Х.3 Альтернативні потоки (обробка помилок) При збою зв'язку виникає відповідне повідомлення від телеграм боті. Х.4 Післяумови ПК отримує запрошення у групу класу.

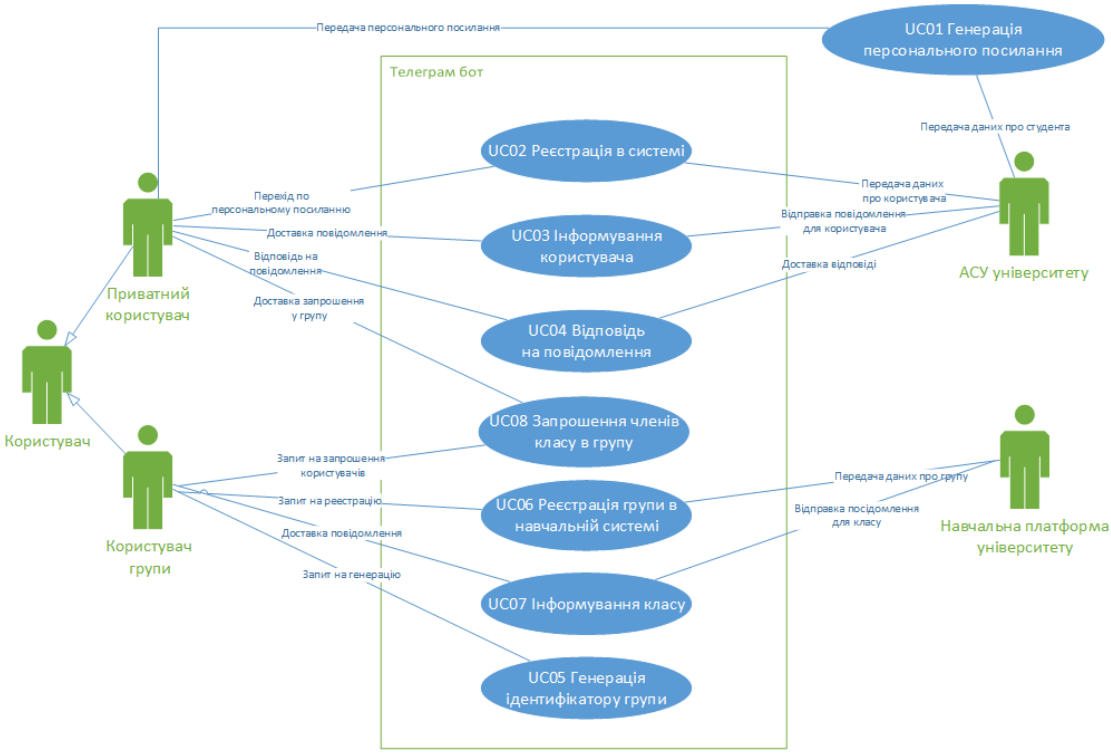

Рисунок 2.3 Діаграма варіантів використання

Маючи на руках усі ці структурно-функціональні моделі ми можем чітко оцінити які взаємодії відбуваються в системі, і що саме нам необхідно імплементувати.

2.5 Моделювання бази даних

У сучасному процесі розробки програмного забезпечення важливим аспектом є ефективне моделювання бази даних, що забезпечує структуроване зберігання та оптимізований доступ до даних. Для візуалізації структури бази даних і її взаємозв'язків використовується ERD (Entity-Relationship Diagram). ERD допомагає розробникам і аналітикам зрозуміти логічну структуру бази даних, представляючи сутності (таблиці), їх атрибути і зв'язки між ними.

Структура майбутньої бази даних складається з трьох основних сутностей: Subscribers, Messages, та Sources (Рисунок 2.4). Кожна з них має свої унікальні атрибути і зв'язки, які відображають логіку взаємодії між ними.

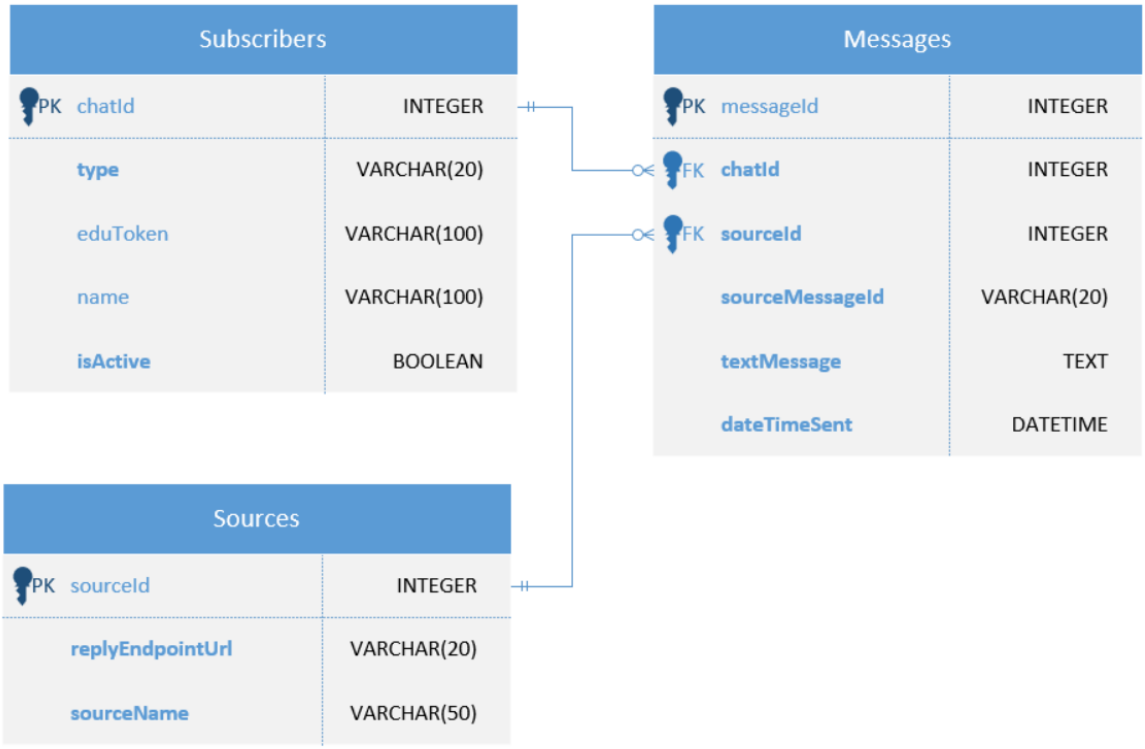

Рисунок 2.4 Структура бази даних інформаційної системи в нотації crow's foot

Дані таблиці та їх атрибути були спроєктовані таким чином, щоб відповідати визначеним раніше вимогам інформаційної системи. Аналізуючи опис кожного поля сутностей (

Таблиця 2.3 – 2.5), можемо не тільки ідентифікувати структуру даних, але й визначити ключові параметри та типи даних, які використовуються в кожній таблиці.

Таблиця 2.3 Атрибути сутності Subscribers

Поле	Зміст	Тип	Ключ	Обмеження
chatId	Ідентифікатор чату абонента	Integer	PK	NN, UQ
type	Тип абонента (приватний/груповий)	Varchar(20)	-	NN
eduToken	Ідентифікатор абонента в навчальній платформі	Varchar(100)	-	UQ
name	Назва/ім’я абонента	Varchar(100)	-	

Поле	Зміст	Тип	Ключ	Обмеження
isActive	Показник активності абонента	Boolean	-	NN

Таблиця 2.4 Атрибути сутності Sources

Поле	Зміст	Тип	Ключ	Обмеження
sourceId	Ідентифікатор джерела	Integer	PK	NN, UQ
replyEndpointUrl	Посилання на API для відправки відповіді	Varchar(20)	-	NN, UQ
sourceName	Назва джерела	Varchar(50)	-	NN, UQ

Таблица 2.5 Атрибути сутності Messages

Поле	Зміст	Тип	Ключ	Обмеження
messageId	Ідентифікатор повідомлення	Integer	PK	NN, UQ
chatId	Ідентифікатор чату абонента	Integer	FK	NN
sourceId	Ідентифікатор джерела	Integer	FK	NN
sourceMessageId	Ідентифікатор повідомлення з джерела	Varchar(20)	-	NN
textMessage	Текст повідомлення	Text	-	NN
dateTimeSent	Дата відправки повідомлення	DateTime	-	NN

Даний аналіз є необхідним для розуміння механізмів взаємодії між сутностями в контексті загальної архітектури системи.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Ініціалізація та налаштування проєкту на платформі NestJS

Для створення нового проєкту необхідно скористатися командою nest new <project-name> що є частиною @nestjs/cli. Після цього потрібно обрати пакетний менеджер та дочекатися створення базової структури проєкту. На даному етапі ми отримуємо готовий nest проєкт який можна налаштовувати відповідно до функціональних та технічних вимог.

На верхньому рівні структури проєкту складається з файлів налаштування фреймворку та двох директорій: src та migrations (Рисунок 3.1).

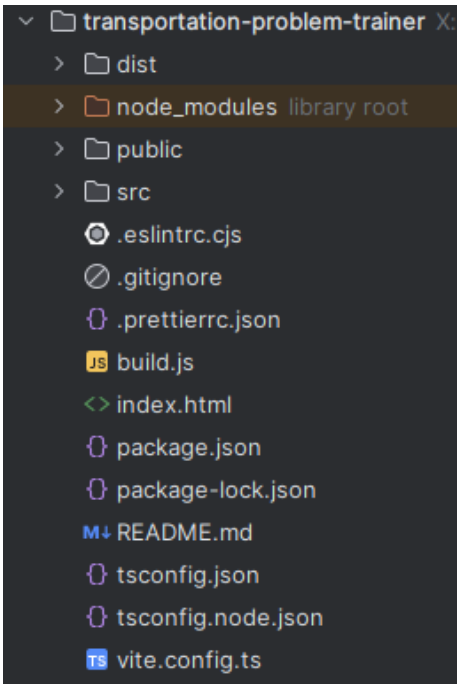

Рисунок 3.1 Структура проєкту серверу телеграм боту

У файлі package.json зазначено ключову інформацію про пакет, таку як назва (name), приватність (private), версія (version), інформацію про автора (author), опис (description), ліцензія (license) команди керування проєктом (scripts), а також списки runtime-залежностей (dependencies) та залежностей для розробки (devDependencies) необхідних для функціонування системи.

Команди start:dev та start:debug запускають локальний веб-сервер, який працює в режимі «гарачого» оновлення. Даний режим відслідковує зміни внесені у код під час роботи серверу, та оновлює модулі в реальному часі без повного перезапуску серверу. Це прискорює процес розробки адже дозволяє швидко побачити як зміни у коді впливають на систему. Дебагер автоматично підключається при використанні команди start:debug, та допомагає покроково перевіряти процеси що перебігають у коді.

Виклик build запускає процес збирання проєкту, після чого у директорії dist вибудовується nodeJS додаток що дозволяє запустити його на будь-якій платформі що підтримує дану технологію без необхідності встановлювати nest.

Команда lint використовується для лінтингу — статичного аналізу коду відповідно до встановлених правил з метою виявлення потенційних помилок.

Команда format застосовується для корекції форматування коду у проєкті, забезпечуючи таким чином єдність стилю написання.

Файл .prettierrc.json визначає налаштування для форматування коду в рамках проєкту (Рисунок 3.2). Ці налаштування включають використання одинарних лапок (singleQuote: true) для рядкових літералів та додавання коми після останнього елементу в переліках і об'єктах (trailingComma: "all"). Такі параметри були обрані для забезпечення консистентності та поліпшення читабельності коду при його перегляді у редакторі коду. Ці зміни сприяють уніфікації стилістики коду та зменшенню можливості виникнення помилок під час ручних змін чи додавань нового коду.

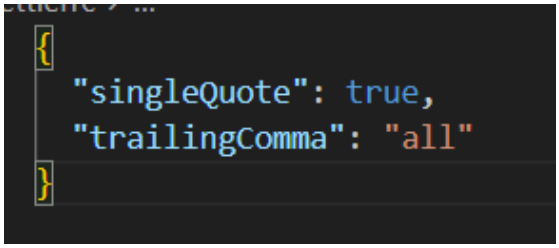

Рисунок 3.2 Налаштування форматування коду

Конфігурація правил для ESLint розташована у файлі .eslintrc.js (Рисунок 3.3).

```
module.exports = {
  parser: '@typescript-eslint/parser',
  parserOptions: {
    project: 'tsconfig.json',
    tsconfigRootDir: __dirname,
    sourceType: 'module',
  },
  plugins: ['@typescript-eslint/eslint-plugin'],
  extends: [
    'plugin:@typescript-eslint/recommended',
    'plugin:prettier/recommended',
  ],
  ignorePatterns: ['.eslintrc.js'],
  rules: {
    '@typescript-eslint/interface-name-prefix': 'off',
    '@typescript-eslint/explicit-function-return-type': 'off',
    '@typescript-eslint/explicit-module-boundary-types': 'off',
    '@typescript-eslint/no-explicit-any': 'off',
  },
};
```

Рисунок 3.3 Налаштування для ESLint

Конфігурація використовує парсер @typescript-eslint/parser з налаштуваннями в tsconfig.json і визначає модулі як основний тип коду. Завдяки інтеграції з плагіном @typescript-eslint/eslint-plugin та Prettier забезпечується єдність стилю кодування. Вимкнення правил, таких як вимога до префіксів у іменах інтерфейсів («interface-name-prefix»), обов'язкове вказання типу повернення у функціях («explicit-function-return-type»), декларація типів на межах модулів («explicit-module-boundary-types»), та дозвіл на використання типу any («no-explicit-any»), спрямоване на зниження формальностей та сприяння більшій гнучкості в написанні та інтеграції коду. Ці зміни у правилах допомагають забезпечити оптимальний баланс між строгістю та практичністю, що є ключовим для ефективної розробки в динамічних проєктних середовищах.

У .env файлі зберігається конфіденційна інформація, така як параметри підключення до бази даних, та токен телеграм-боту, у форматі пар "ключ=значення".

Файл knexfile.js використовується для налаштування Knex.js, інструмента для взаємодії з базами даних. Цей файл містить всі необхідні параметри для з'єднання з базою даних, такі як тип клієнта, дані для підключення (хост, порт, назва бази даних, користувач та пароль), а також налаштування пулу з'єднань, які визначають мінімальну та максимальну кількість активних з'єднань (Рисунок 3.4). Усі дані отримуються із змінних оточення.

```
import * as dotenv from 'dotenv';

dotenv.config();

import { getDatabaseConfig } from './src/config/database.config';

const config = getDatabaseConfig();

const getKnexConfig = () => {
  return {
    client: config.client,
    connection: {
      host: config.host,
      port: config.port,
      database: config.schema,
      user: config.username,
      password: config.password,
    },
    pool: {
      min: 2,
      max: 10,
    },
  };
};

export default getKnexConfig();
```

Рисунок 3.4 Налаштування підключення до СУБД

Файл .gitignore, який постачається разом з фреймворком, містить перелік файлів та директорій, які слід ігнорувати системою контролю версій Git.

Файлах tsconfig.json та tsconfig.build.json описують конфігурацію транскомпілятора TypeScript та поставляються разом з базовим проєктом NestJS.

3.2 Програмна реалізація telegram-боту

Перш за все, для створення бота необхідно зареєструвати його в системі Telegram та отримати API токен. Для цього потрібно у додатку Telegram у пошуку знайти спеціалізованого бота @BotFather та запустити його (Рисунок 3.5). @BotFather дозволяє створювати та управляти ботами в Telegram і саме через нього ми отримаємо токен для нашого бота.

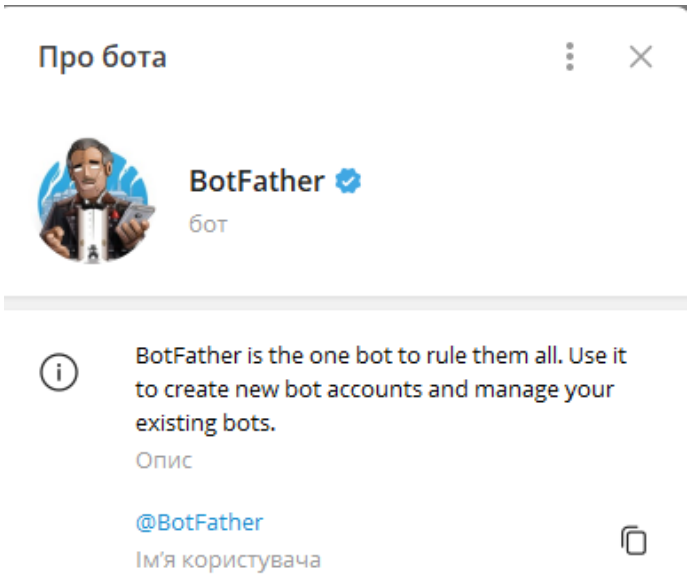

Рисунок 3.5 Telegram-бот @BotFather

Для реєстрації нового бота потрібно послідовно ввести дві команди /start (початок роботи із @BotFather) та /newbot (запит на створення нового бота). Останнім кроком вводимо унікальну для нього назву та доменне ім'я. І якщо реєстрація бота пройшла успішно отримуємо повідомлення де вказано його API токен (Рисунок 3.6).

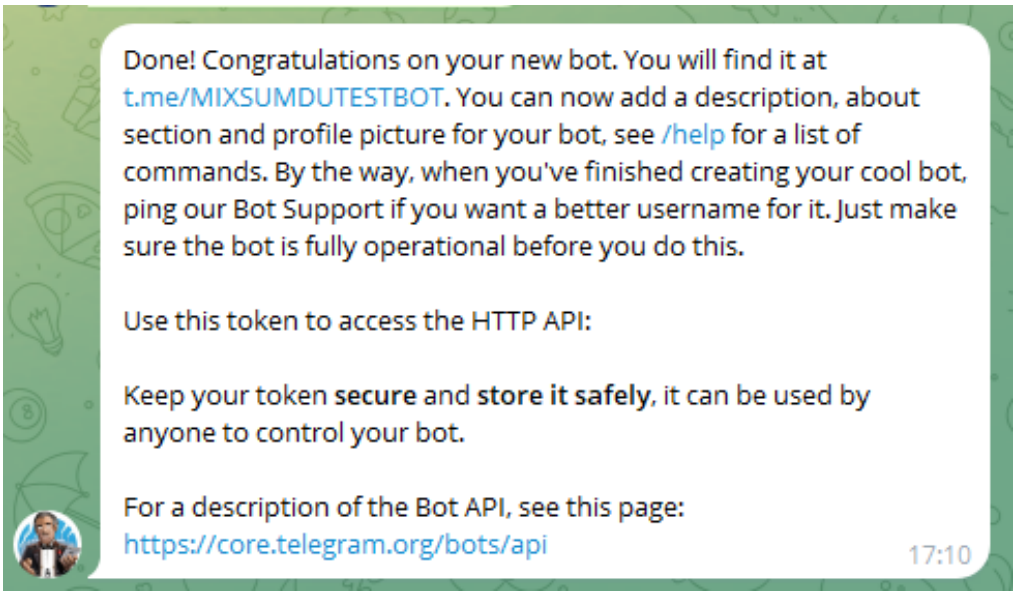

Рисунок 3.6 Повідомлення про успішну реєстрацію боту

На даний момент бот створений та може використовуватись, проте на даному етапі його потрібного додатково налаштувати відповідно до вимог системи. Для цього виконуємо команду /mybots та обираємо щойно створеного бота зі списку. Після цього натискаємо на кнопку «Bot Settings» та потрапляємо в меню налаштування бота (Рисунок 3.7).

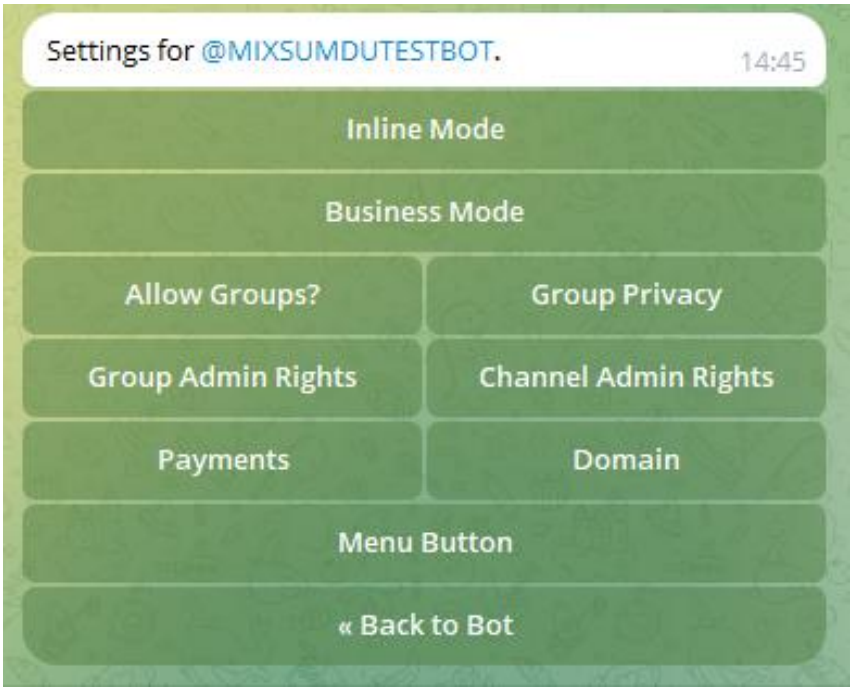

Рисунок 3.7 Налаштування телеграм боту

По-перше вимкнемо Group Privacy через відповідний пункт меню. Таким чином бот буде отримувати повідомлення у групах і адміністратору не доведеться робити це вручну.

По друге через пункт Group Admin Rights, розширимо можливості нашого бота для роботи у групах (Рисунок 3.8). Для цього увімкнемо такі права:

- Change group title, photo etc;
- Restrict, ban or unban members;
- Invite new users;
- Pin messages;
- Manage chat;

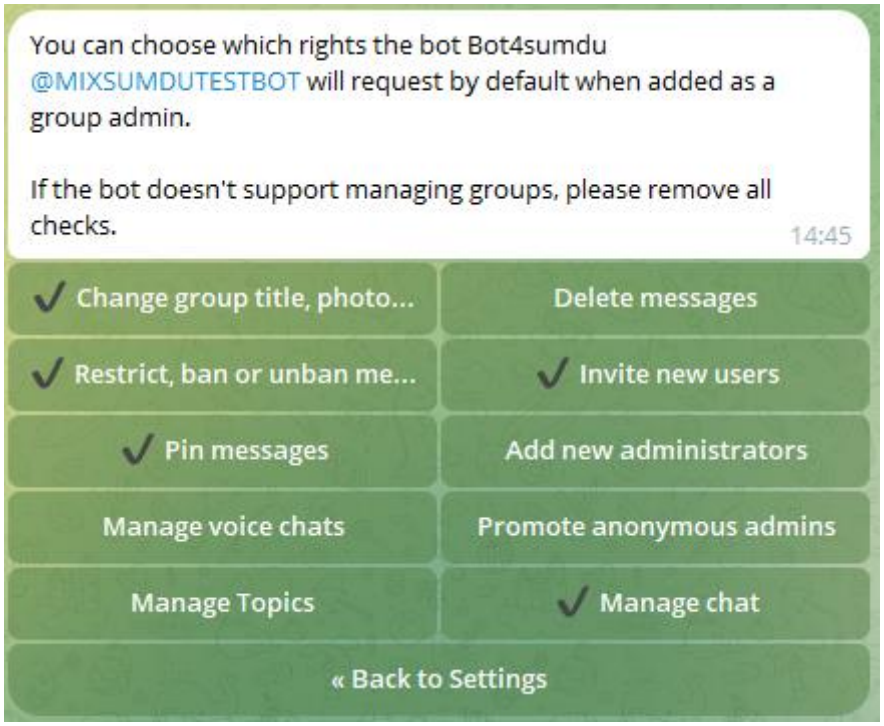

Рисунок 3.8 Налаштування прав адміністратора боту

Дані права дозволять забезпечити відповідність вимог нашої інформаційної системи.

Для обробки вхідних повідомлень та команд від користувачів у Telegram був розроблений контролер TelegramController (Рисунок 3.9). Даний обробник

інтегрований з основною системою навчальної платформи, забезпечуючи гнучкість взаємодії та персоналізацію обслуговування користувачів.

```
@Update()
export class TelegramController {
  constructor( ···
  ) {}

  @Start()
  async handleStart(@Ctx() ctx: Context): Promise<void> { ···
  }

  @Command('chatid')
  async handleChatId(@Ctx() ctx: Context): Promise<void> { ···
  }

  @On('message')
  async handleMessage(@Ctx() ctx: SessionContext): Promise<void> { ···
  }

  @On('callback_query')
  onCallbackQuery(@Ctx() ctx: SessionContext) { ···
  }
}
```

Рисунок 3.9 Методи контролера Telegram

Метод handleStart, огорнутий декоратором @Start(), є першим точкою входу в бота для нових користувачів. Коли користувач вперше відправляє команду /start, метод активується і використовує subscribersService для ініціювання процесу реєстрації користувача в системі. Цей процес включає збір необхідної інформації про користувача та її відправки у систему навчальної платформи.

Метод handleChatId, огорнутий декоратором @Command('chatid'), викликається командою /chatid, що надсилається користувачем. Відповідь на цю команду передбачає залучення subscribersService для генерації унікального ідентифікатора чату, який може бути використаний для асоціації користувача з конкретним класом або групою на навчальній платформі. Цей ідентифікатор служить ключем для зберігання та відстеження інформації про взаємодію користувача в рамках класу.

Метод onCallbackQuery, визначений через декоратор @On('callback_query'), призначений для обробки взаємодій користувачів з inline кнопками або іншими інтерактивними компонентами у боті. Коли користувач використовує такий компонент і надсилає команду 'reply', метод активізує запит на відповідь і налаштовує стан сесії для очікування відповіді від користувача.

Останній з основних методів, handleMessage, визначений через декоратор @On('message'), призначений для обробки всіх типів вхідних повідомлень, які можуть включати запити від користувачів, відповіді на запитання або ж просто інформаційні повідомлення. Використовуючи MessagesService, цей метод аналізує зміст повідомлень, визначає наявні запити або команди та відправляє відповідні автоматизовані або напівавтоматизовані відповіді. Це забезпечує динамічне та ефективне спілкування, яке адаптується до контексту взаємодії з кожним користувачем.

Кожен із згаданих методів використовує контекст Telegram, доступний через змінну ctx, що дозволяє детально маніпулювати інформацією про користувача, чат, та історію повідомлень. Це дає можливість боту надавати відповіді та взаємодії, які враховують індивідуальні потреби та вподобання користувача, підвищуючи залученість і задоволення від використання навчальної платформи.

Сервіси SubscribersService та MessagesService відділяють логіку роботи з даними від задач контролера, діючи як посередник між ним та репозиторієм даних.

Сервіс SubscribersService забезпечує обробку даних абонентів, ефективно управляючи реєстрацією, оновленням та видаленням інформації про користувачів у контексті комунікації через месенджер (Рисунок 3.10).

Метод addSubscriber відповідає за реєстрацію нових абонентів. Якщо користувач вже існує в системі, він повідомляє про це через виняток BadRequestException, запобігаючи повторній реєстрації. Цей метод також

адаптується до типу чату, що є критично важливим для персоналізації повідомлень.

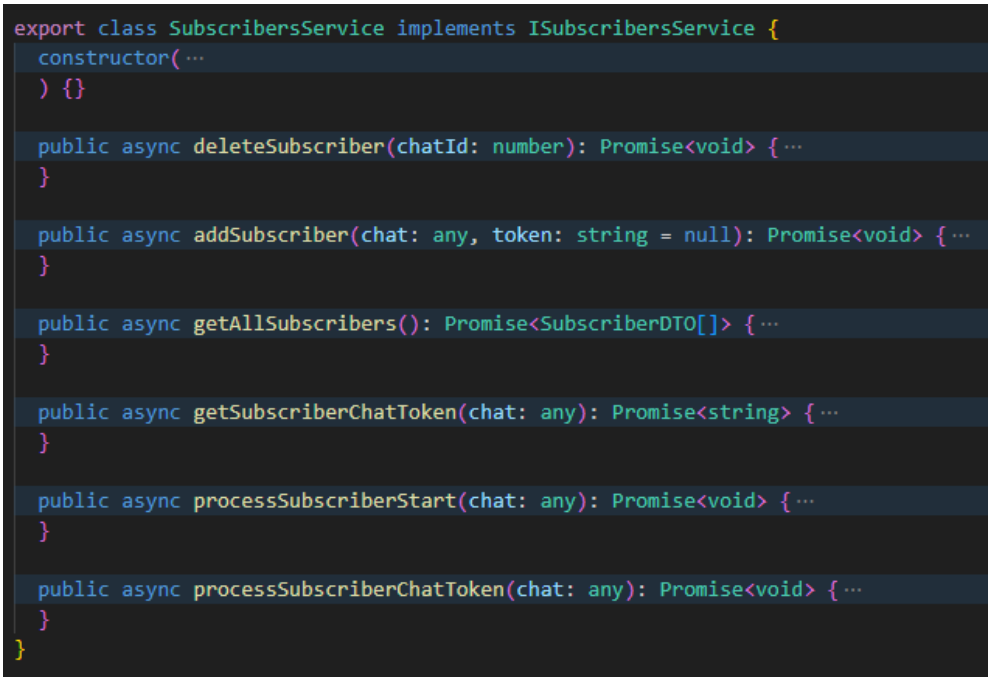

Рисунок 3.10 Методи сервісу Subscribers

Метод processSubscriberStart обробляє ініціацію взаємодії користувачів з ботом. Цей процес включає не тільки реєстрацію користувача але й надсилання вітального повідомлення або повідомлення про помилку, якщо реєстрація не можлива.

Метод deleteSubscriber використовується для видалення даних про абонента з бази, що є необхідним для забезпечення конфіденційності та відповідності до правил ведення даних користувачів.

Метод getAllSubscribers дозволяє адміністраторам бота отримувати загальний перегляд всіх абонентів.

Метод getSubscriberChatToken генерує унікальні ідентифікатори для групових чатів. Ці токени можуть бути використані для ідентифікації та верифікації груп у межах навчальних платформ, дозволяючи адмініструвати доступ до освітніх матеріалів і спілкування.

Сервіс MessagesService використовуються для ефективного оброблення вхідних повідомлень, розподілу їх серед абонентів та подальшої обробки.

Метод possessIncomingMessage даного сервісу слугує для аналізу та обробки повідомлень, які надходять від користувачів бота. Цей процес включає ідентифікацію відповідей на інформацію з навчальної платформи. Якщо визначено, що текст є реакцією на раніше отримане повідомлення, він автоматично пересилається на платформу для подальшої обробки.

Окремим модулем роботи телеграм бота є сервіс NotificationsService, взаємодіє безпосередньо з Telegram через бібліотеку Telegraf. Цей сервіс має прямий доступ до бота, дозволяючи відправляти повідомлення та пересилати існуючі повідомлення між користувачами (Рисунок 3.11).

```
export class NotificationsService implements INotificationsService {
  constructor( ···
  ) {}

  public async forwardMessage( ···
  }

  public async sendTextMessage(chatId: number, text: string): Promise<void> { ···
  }

  public async sendTextMessageWithReply( ···
  }
}
```

Рисунок 3.11 Методи сервісу Notifications

Сервіс NotificationsService використовує методи forwardMessage і sendTextMessage для управління комунікацією.

Метод forwardMessage, забезпечує пересилання повідомлень від одного користувача до іншого. Ця функція реалізується через безпосередній виклик bot.telegram.forwardMessage, що демонструє пряме використання API Telegram для взаємодії з користувачами.

Останні два методи, sendTextMessage та sendTextMessageWithReply, використовуються для відправлення текстових повідомлень користувачам. Ці методи також активно взаємодіють з Telegram через API бота, для доступу до чату користувача. У випадку виникнення помилки, наприклад, якщо користувач заблокував бота або вийшов з групи, сервіс відбувається видалення даних про цього абонента, забезпечуючи актуальність бази даних

абонентів. Метод sendTextMessageWithReply додатково додає до повідомлення inline-кнопку для відповіді.

3.3 Програмна реалізація веб-серверу

Для реалізації веб-серверу, який управляє обробкою запитів користувачів, був створений контролер MessageController. Цей контролер інтегрований з основною платформою комунікацій та забезпечує ефективне управління повідомленнями між користувачами та системою (Рисунок 3.12).

```
@Controller(`api/message`)
export class MessageController {
  constructor(
    @InjectBot() private bot: Telegraf<TelegrafContext>,
    @Inject(SUBSCRIBERS_SERVICE_TOKEN)
    private readonly subscribersService: ISubscribersService,
    @Inject(MESSAGES_SERVICE_TOKEN)
    private readonly MessagesService: IMessagesService,
  ) {}

  @Post(`/sendPrivateMessage/:userId`)
  @HttpCode(200)
  public async sendMessageToUsers(···
  }

  @Post(`/sendGroupMessage/:classToken`)
  @HttpCode(200)
  public async sendMessageToGroup(···
  }
}
```

Рисунок 3.12 Методи контролера Message

Перший метод, sendMessageToUsers, обробляє відправлення приватних повідомлень конкретним користувачам. Метод активується через POST-запит і використовує userId для ідентифікації отримувача та sendMessageForm для передачі тексту повідомлення та інформації, чи може повідомлення примати відповіді.

Другий метод, sendMessageToGroup, керує відправленням групових повідомлень від класу через POST-запит. Він використовує classToken для ідентифікації групи та sendMessageForm для тексту повідомлення.

Контролер взаємодіє з MessagesService, який відповідає за безпосередню обробку повідомлень. MessagesService використовує метод sendMessageToSubscriber для надсилання текстових повідомлень до підписника. Якщо підписник не знайдений або інші помилки, система генерує BadRequestException, запобігаючи подальшій неправильній обробці.

Інтеграція з NotificationsService дозволяє цьому сервісу управляти повідомленнями, які потребують прямого взаємодії з користувачем через Telegram. Це забезпечує, що повідомлення доставляються ефективно і вчасно, враховуючи потреби і контекст комунікації.

Для інтеграції веб-сервера з навчальною платформою та іншими сервісами університету використовується API клієнт EduAPIClient, який дозволяє відправляти дані на платформу (Рисунок 3.13). Цей клієнт реалізований з використанням `HttpService` з бібліотеки `@nestjs/axios`, що надає засоби для здійснення HTTP-запитів.

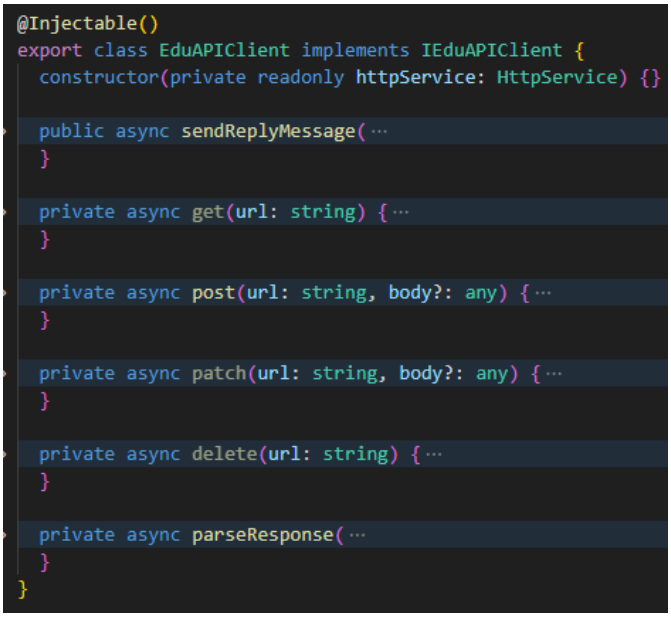

Рисунок 3.13 Методи клієнту навчальної платформи

Основні методи клієнта включають `get`, `post`, `patch`, і `delete`, які дозволяють здійснювати відповідні HTTP-запити до навчальної платформи.

Метод, `sendReplyMessage`, дозволяє відправляти відповіді на повідомлення, які були отримані від користувачів. Цей метод використовує

`post` для відправки тексту повідомлення на платформу, що ідентифікується через унікальний ідентифікатор повідомлення. Функціональність цього методу критично важлива для забезпечення зворотного зв'язку від користувачів системи.

Всі запити формуються шляхом додавання конкретного URL до базової адреси і включають обробку відповідей через метод `parseResponse`, який витягує дані з відповідей Axios та перетворює їх у відповідний формат для подальшого використання у системі.

3.4 Огляд створеної інформаційної технології

Ініціація роботи з ботом вимагає, щоб користувач перейшов по спеціальному посиланню, яке надається через навчальну платформу або АСУ (Рисунок 3.14).

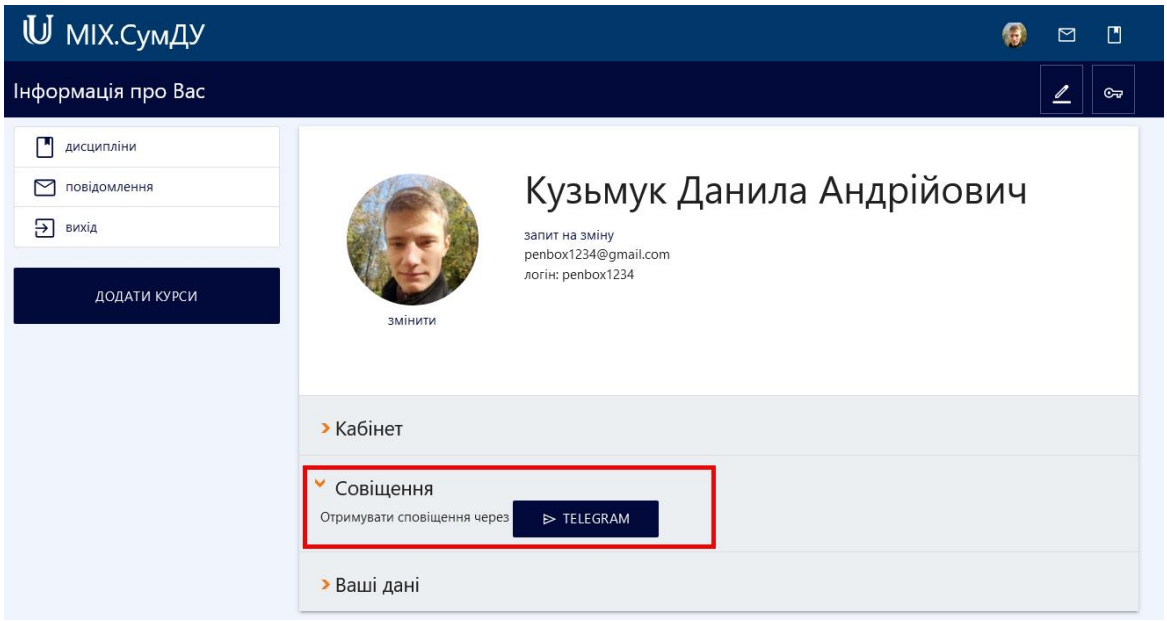

Рисунок 3.14 Секція прев'язки телеграм-боту в профілі користувача навчальної платформи

Це посилання автоматично запускає бота з командою /start, яка інтегрована з унікальним ідентифікатором користувача (Рисунок 3.15), тим самим забезпечуючи персоналізовану взаємодію.

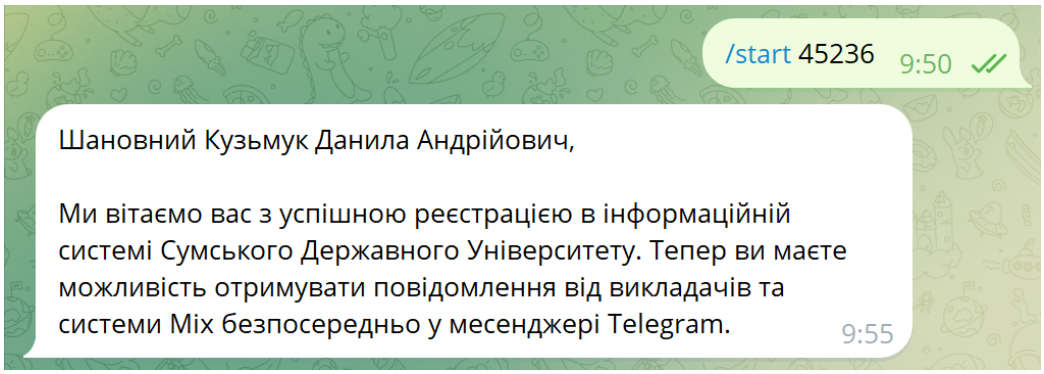

Рисунок 3.15 Авторизація через персональне посилання

Якщо користувач відкриє бота в ручну, без вказання коректного ідентифікатора користувача, він отримає відповідне повідомлення із інструкцією (Рисунок 3.16).

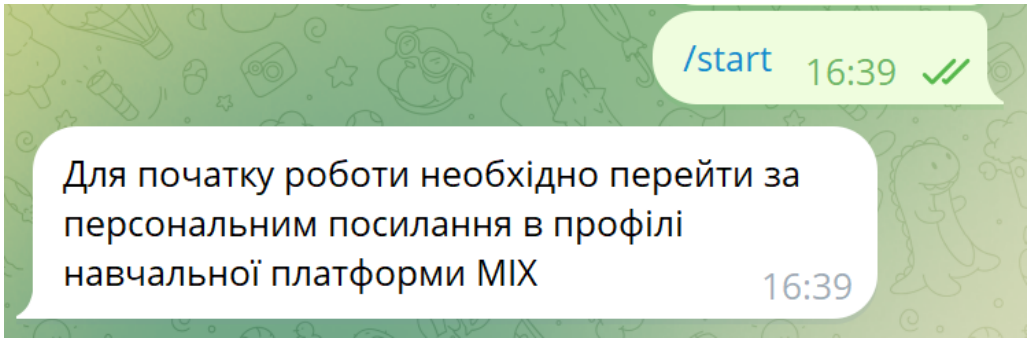

Рисунок 3.16 Повідомлення при авторизації без ідентифікатору користувача

Коли учасник освітнього процесу відправляє повідомлення через навчальну платформу, воно автоматично пересилається до телеграм-бота з вказівкою автора та тексту повідомлення (Рисунок 3.17).

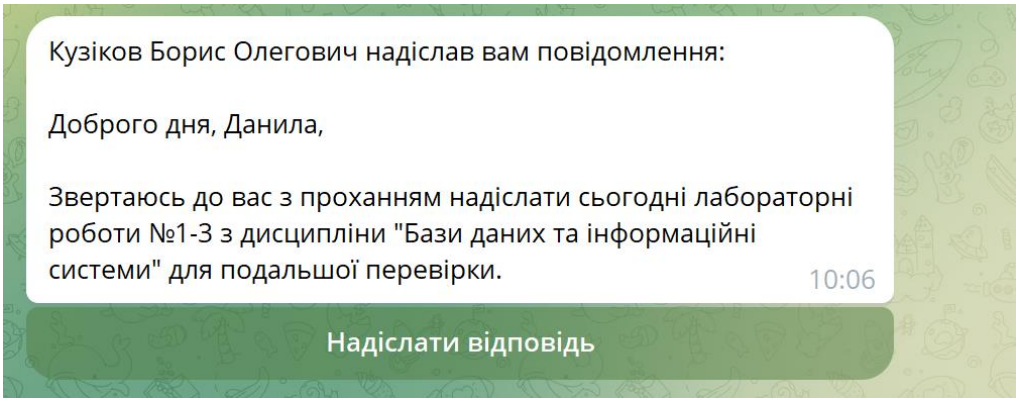

Рисунок 3.17 Персональне повідомлення

У випадку, коли повідомлення підтримує можливість відповіді, користувачі можуть легко надіслати відповідь, натиснувши на кнопку "Надіслати відповідь" і введення відповідного тексту (Рисунок 3.18).

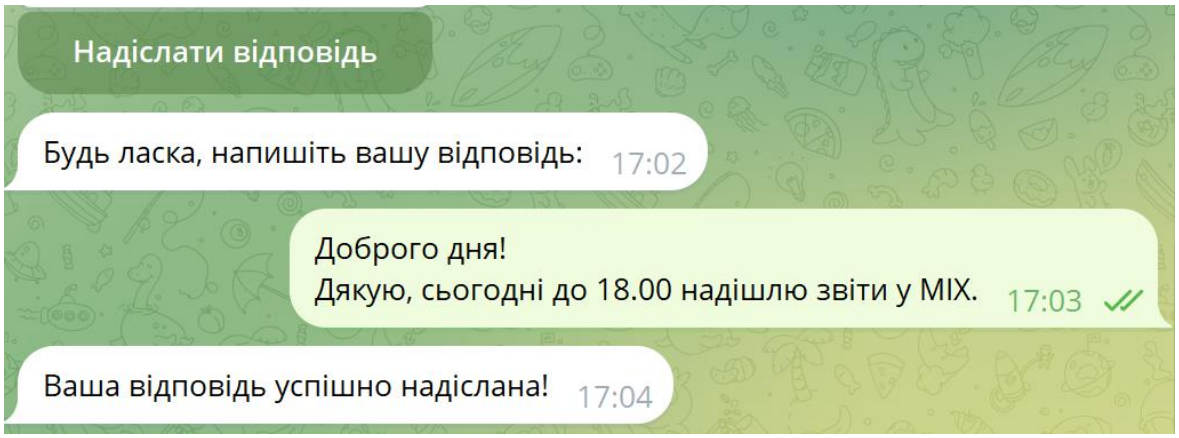

Рисунок 3.18 Відповідь на персональне повідомлення

Для інтеграції телеграм-групи з класом на навчальній платформі потрібно спочатку додати бота в групу, використати команду /chatid для отримання ідентифікатора (Рисунок 3.19).

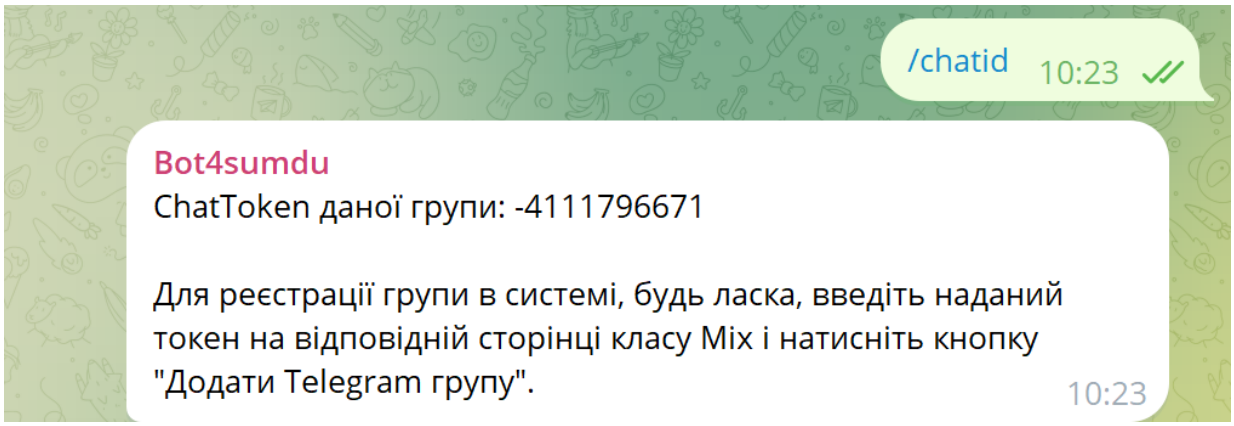

Рисунок 3.19 Генерація ідентифікатора групи

Цей унікальний ідентифікатор необхідно вказати в конфігураційних параметрах класу (Рисунок 3.20), щоб налагодити взаємодію між конкретною телеграм- групою користувачів та дисципліною.

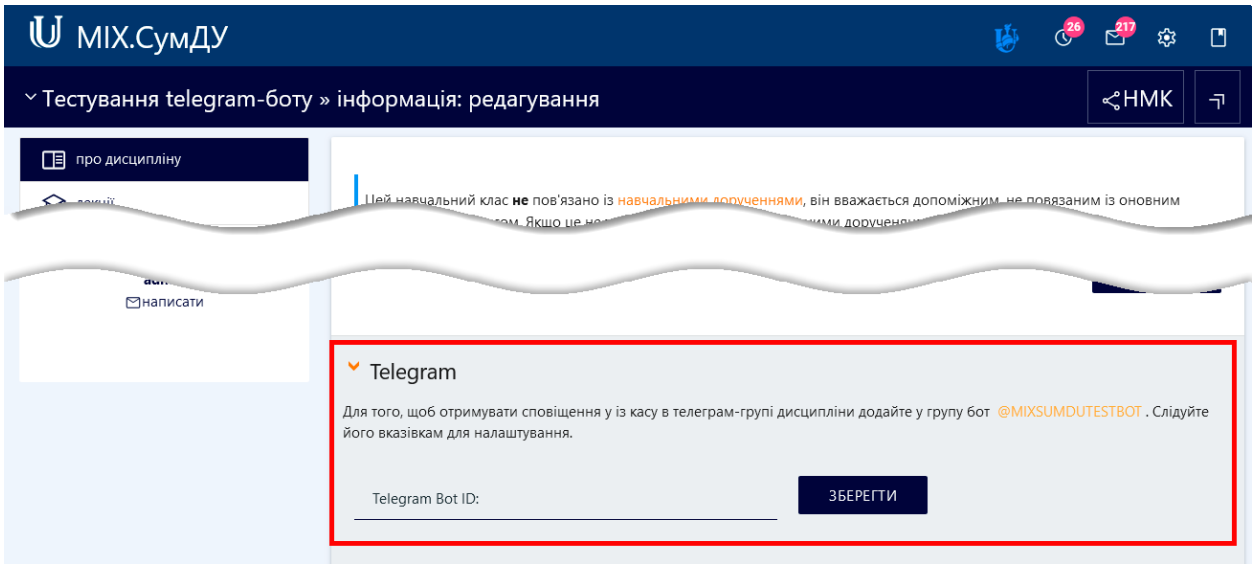

Рисунок 3.20 Секція реєстрації телеграм-групи у налаштуваннях класу навчальної платформи

Після успішної реєстрації буде надіслане відповідне повідомлення (Рисунок 3.21).

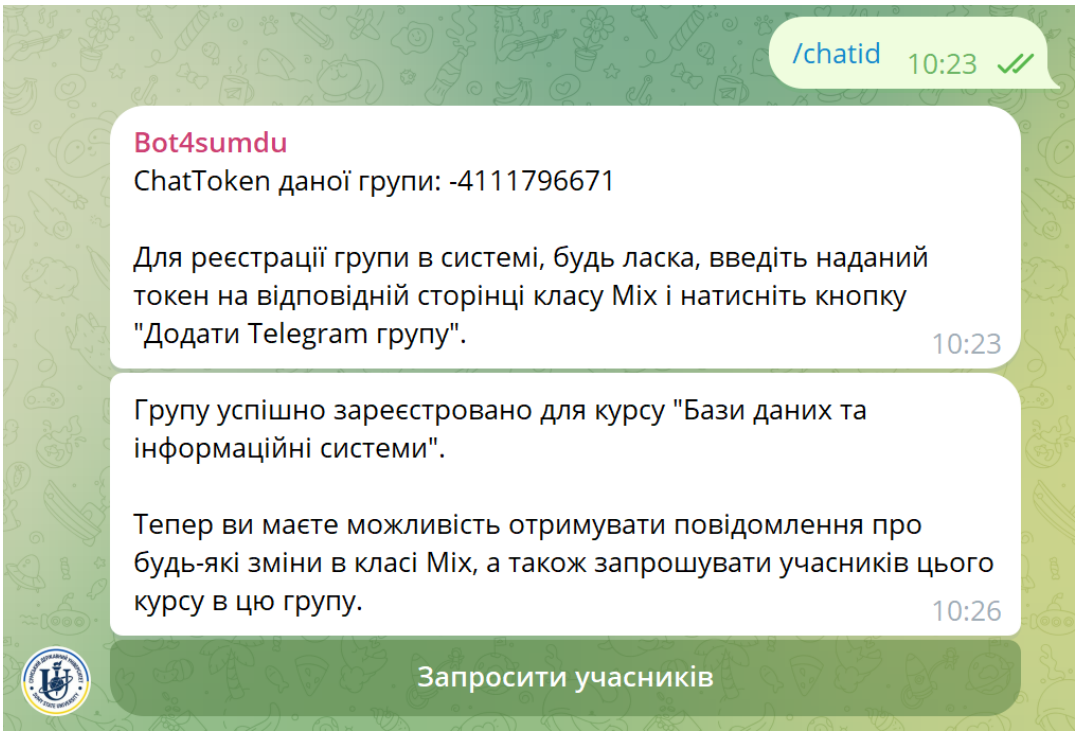

Рисунок 3.21 Реєстрація групи в навчальній платформі

Після зв'язування групи з класом можливе автоматичне запрошення учасників класу у телеграм-групу за допомогою кнопки "Запросити

учасників". Запрошення містить посилання, яке надсилається кожному учаснику у привітному повідомленні (Рисунок 3.22).

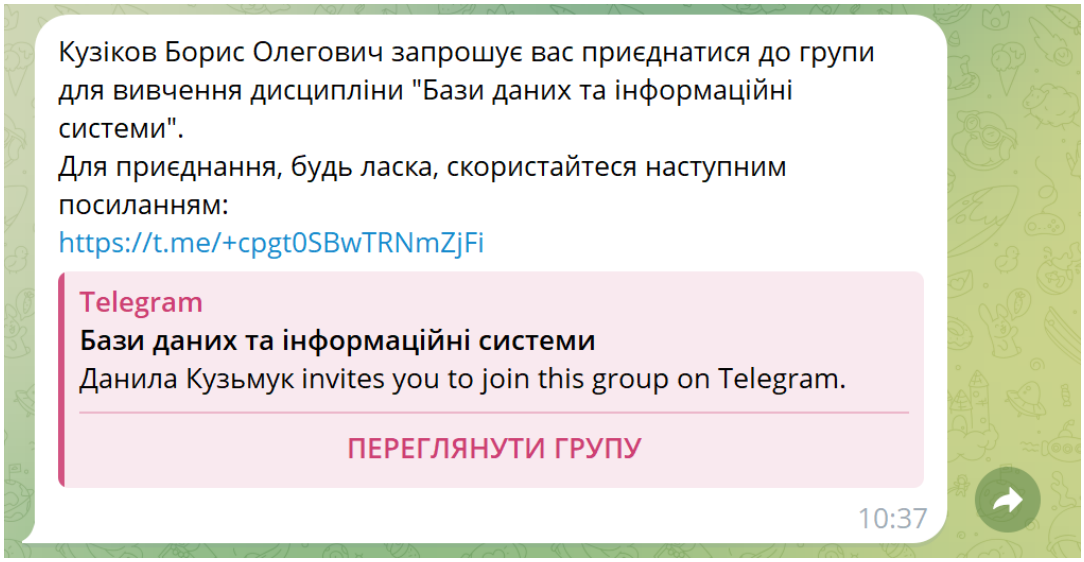

Рисунок 3.22 Персональне запрошення в групу класу

Також в телеграм-групу можуть надходити автоматичні повідомлення (Рисунок 3.23) від навчальної платформи, що підвищує оперативність комунікації та інформування учасників про важливі події та зміни в навчальному процесі.

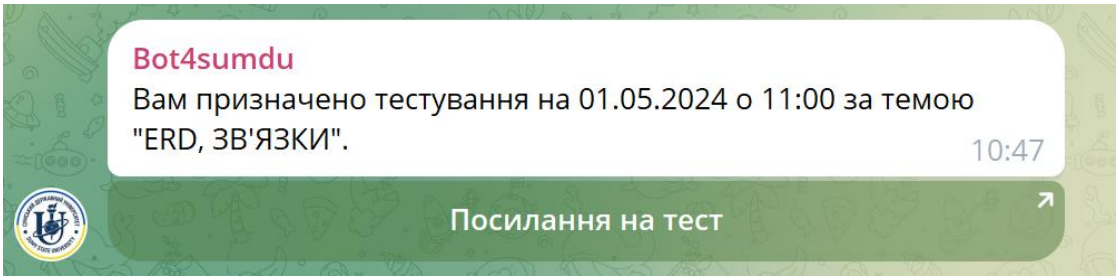

Рисунок 3.23 Автоматизоване повідомлення в групу класу

ВИСНОВОК

У процесі виконання кваліфікаційної роботи магістра було розроблено інформаційну технологію, що інтегрує Telegram-бот для значного підвищення ефективності та інтенсифікації навчального процесу. Детальний аналіз існуючих систем виявив основні прогалини в їх здатності швидко адаптуватися до змінних умов навчання та забезпечити надійний зворотний зв'язок між учасниками. Відповідно до цього аналізу було визначено важливі напрями оптимізації.

Розроблена модель не лише забезпечує ефективний зв'язок між викладачами та студентами, але й дозволяє оперативно стежити за станом навчального процесу, значно підвищуючи його ефективність. Інтерфейс користувача бота було розроблено з акцентом на інтуїтивність і доступність, що робить навчальний процес зручним.

Реалізація проєкту підтвердила гіпотезу про те, що використання технології Telegram-бота в комбінації з глибокою інтеграцією з навчальною платформою може істотно збільшити інтенсивність та якість освітнього процесу. Особливо важливим є впровадження системи миттєвого зворотного зв'язку, яка дозволяє студентам і викладачам вчасно реагувати на виклики навчального процесу.

Загалом, запровадження даної інформаційної технології представляє собою значний крок уперед у розвитку методів оптимізації навчання, надаючи більше можливостей для персоналізації освітнього процесу та покращення взаємодії між учасниками. Цей проєкт може слугувати як основа для подальших досліджень і розвитку передових освітніх технологій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Telegram Bot as an Interface to Open edX platform [Electronic resource]. URL: https://raccoongang.com/blog/telegram-bot-interface-open-edx/.
2. Molina O.E. The Effects of WhatsApp and Telegram on Student Engagement: An Analysis from the Mixed-Methods Approach // Educ Res Int. Hindawi Limited, 2022. Vol. 2022.
3. Khalil M., Rambech M. Eduino: A Telegram Learning-Based Platform and Chatbot in Higher Education // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer, Cham, 2022. Vol. 13329 LNCS. P. 188–204.
4. Clark R.M., Kaw A.K., Braga Gomes R. Adaptive learning: Helpful to the flipped classroom in the online environment of COVID? // Computer Applications in Engineering Education. John Wiley & Sons, Ltd, 2022. Vol. 30, № 2. P. 517–531.
5. Slido - Audience Interaction Made Easy [Electronic resource]. URL: https://www.slido.com/.
6. Interactive presentation software - Mentimeter [Electronic resource]. URL: https://www.mentimeter.com/.
7. Українці значною мірою покладаються на Telegram канали для отримання новин під час війни - Internews in Ukraine [Electronic resource]. URL: https://internews.in.ua/uk/news/ukraintsi-znachnoiu-miroiu-pokladaiutsia-na-telegram-kanaly-dlia-otrymannia-novyn-pid-chas-viyny/.
8. Ware E. Edpuzzle // J Med Libr Assoc. Medical Library Association, 2021. Vol. 109, № 2. P. 349.
9. Bagheri M.M. Intelligent and Adaptive Tutoring Systems: How to Integrate Learners. 2015. Vol. 7, № 2.
10. Wang A.I., Tahir R. The effect of using Kahoot! for learning – A literature review // Comput Educ. Pergamon, 2020. Vol. 149. P. 103818.

11. Belda-Medina J., Kokošková V. Integrating chatbots in education: insights from the Chatbot-Human Interaction Satisfaction Model (CHISM) // International Journal of Educational Technology in Higher Education. Springer Science and Business Media Deutschland GmbH, 2023. Vol. 20, № 1. P. 1–20.
12. Merelo J.J. et al. Chatbots and messaging platforms in the classroom: an analysis from the teacher's perspective // Educ Inf Technol (Dordr). Springer, 2022. Vol. 29, № 2. P. 1903–1938.
13. Kumar J.A. Educational chatbots for project-based learning: investigating learning outcomes for a team-based design course // International Journal of Educational Technology in Higher Education. Springer Science and Business Media Deutschland GmbH, 2021. Vol. 18, № 1. P. 1–28.
14. Nur T. et al. Telegram Bot Usage as An Instructional Design Method // International Journal of Accounting, Finance and Business. 2022. Vol. 7, № 42. P. 13–18.
15. Pros and Cons of Python [Electronic resource]. URL: https://serokell.io/blog/python-pros-and-cons.
16. Pros and Cons of Java Development in 2023 [Electronic resource]. URL: https://www.netguru.com/blog/java-pros-and-cons.
17. Advantages and Disadvantages of PHP Programming Language | EPAM Anywhere [Electronic resource]. URL: https://anywhere.epam.com/en/blog/pros-and-cons-of-php.
18. Pros and Cons of Node.js Web App Development [Electronic resource]. URL: https://www.altexsoft.com/blog/the-good-and-the-bad-of-node-js-web-app-development/ .
19. telegraf.js - v4.16.3 [Electronic resource]. URL: https://telegraf.js.org/ .
20. yagop/node-telegram-bot-api: Telegram Bot API for NodeJS [Electronic resource]. URL: https://github.com/yagop/node-telegram-bot-api/tree/master .

21. Comparing NPM, YARN, and PNPM Package Managers: Which One is Right for Your Distributed Project to handle High Loads? | by Roman Glushach | Medium [Electronic resource]. URL: https://romanglushach.medium.com/comparing-npm-yarn-and-pnpm-package-managers-which-one-is-right-for-your-distributed-project-to-4d7de2f0db8e .
22. Express web framework (Node.js/JavaScript) - Learn web development | MDN [Electronic resource]. URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs .
23. What is Koa Javascript (Koa Js)? | node js web development | Blog [Electronic resource]. URL: https://cubettech.com/resources/blog/introduction-to-koa-javascript-koa-js-pros-and-cons/.
24. A Quick Introduction to NestJS | Mitrais Blog [Electronic resource]. URL: https://www.mitrais.com/news-updates/a-quick-introduction-to-nestjs/ .
25. SQLite Home Page [Electronic resource]. URL: https://www.sqlite.org/ .
26. MySQL [Electronic resource]. URL: https://www.mysql.com/.
27. PostgreSQL: The world’s most advanced open source database [Electronic resource]. URL: https://www.postgresql.org/ .
28. MongoDB: The Developer Data Platform | MongoDB [Electronic resource]. URL: https://www.mongodb.com/ .
29. MariaDB Foundation - MariaDB.org [Electronic resource]. URL: https://mariadb.org/ .
30. Sequelize | Feature-rich ORM for modern TypeScript & JavaScript [Electronic resource]. URL: https://sequelize.org/ .
31. TypeORM - Amazing ORM for TypeScript and JavaScript (ES7, ES6, ES5). Supports MySQL, PostgreSQL, MariaDB, SQLite, MS SQL Server, Oracle, WebSQL databases. Works in NodeJS, Browser, Ionic, Cordova and Electron platforms. [Electronic resource]. URL: https://typeorm.io/ .

32. SQL Query Builder for Javascript | Knex.js [Electronic resource]. URL: https://knexjs.org/ .
33. MySQL2 | Quickstart [Electronic resource]. URL: https://sidorares.github.io/node-mysql2/docs .
34. Configuration | NestJS - A progressive Node.js framework [Electronic resource]. URL: https://docs.nestjs.com/techniques/configuration.

ДОДАТОК А

Файл telegram.controller.ts:

```
import { Inject } from '@nestjs/common';
import { Command, Ctx, On, Start, Update } from 'nestjs-telegraf';
import { Context } from 'telegraf';

import {
  IMessagesService,
  MESSAGES_SERVICE_TOKEN,
} from '../messages/messages.service.interface';
import {
  ISubscribersService,
  SUBSCRIBERS_SERVICE_TOKEN,
} from '../subscribers/subscribers.service.interface';
import { SessionContext } from 'src/models/telegraf-session-context';

@Update()
export class TelegramController {
  constructor(
    @Inject(SUBSCRIBERS_SERVICE_TOKEN)
    private readonly subscribersService: ISubscribersService,
    @Inject(MESSAGES_SERVICE_TOKEN)
    private readonly MessagesService: IMessagesService,
  ) {}

  @Start()
  async handleStart(@Ctx() ctx: Context): Promise<void> {
    await this.subscribersService.processSubscriberStart(
      ctx.chat,
      ctx['startPayload'],
    );
  }

  @Command('chatid')
  async handleChatId(@Ctx() ctx: Context): Promise<void> {
    await this.subscribersService.processSubscriberChatToken(ctx.chat);
  }

  @On('message')
```

```
  async handleMessage(@Ctx() ctx: SessionContext): Promise<void> {
    await this.MessagesService.possessIncomingMessage(ctx.message, ctx.session);
  }

  @On('callback_query')
  onCallbackQuery(@Ctx() ctx: SessionContext) {
    if ('data' in ctx.callbackQuery) {
      const callbackQuery = ctx.callbackQuery;
      const data = callbackQuery.data;
      const messageId = callbackQuery.message?.message_id;
      if (data === 'reply' && messageId) {
        ctx.reply('Будь ласка, напишіть вашу відповідь:');
        ctx.session.messageId = messageId;
        ctx.session.awaitingReply = true;
      }
    }
  }
}
```

Файл message.controller.ts:

```
import {
  Body,
  Controller,
  Get,
  HttpCode,
  Inject,
  Param,
  Post,
} from '@nestjs/common';
import { SendMessageForm } from 'src/models/forms/send-message.form';
import { SendPrivateMessageForm } from 'src/models/forms/send-private-message.form';
import {
  IMessagesService,
  MESSAGES_SERVICE_TOKEN,
} from '../messages/messages.service.interface';
import {
  ISubscribersService,
  SUBSCRIBERS_SERVICE_TOKEN,
} from '../subscribers/subscribers.service.interface';
```

```
@Controller(`api/message`)
export class MessageController {
  constructor(
    @Inject(SUBSCRIBERS_SERVICE_TOKEN)
    private readonly subscribersService: ISubscribersService,
    @Inject(MESSAGES_SERVICE_TOKEN)
    private readonly MessagesService: IMessagesService,
  ) {}

  @Post(`/sendPrivateMessage/:userId`)
  @HttpCode(200)
  public async sendMessageToUsers(
    @Body() sendMessageForm: SendPrivateMessageForm,
    @Param('userId') userId,
  ): Promise<any> {
    return await this.MessagesService.sendMessageToSubscriber(
      userId.toString(),
      sendMessageForm.textMessage,
      sendMessageForm.canBeReply,
    );
  }

  @Post(`/sendGroupMessage/:classToken`)
  @HttpCode(200)
  public async sendMessageToGroup(
    @Body() sendMessageForm: SendMessageForm,
    @Param('classToken') classToken,
  ): Promise<any> {
    return await this.MessagesService.sendMessageToSubscriber(
      classToken,
      sendMessageForm.textMessage,
      false,
    );
  }
}
```

Файл subscribers.service.ts:

```
import { ISubscribersService } from './subscribers.service.interface';
import { BadRequestException, Inject } from '@nestjs/common';
```

```
import {
  SUBSCRIBERS_REPOSITORY_TOKEN,
  ISubscribersRepository,
} from './subscribers.repository.interface';
import { SubscriberDTO } from 'src/models/dto/subscriber.dto';
import { v4 as uuid } from 'uuid';
import {
  NOTIFICATIONS_SERVICE_TOKEN,
  INotificationsService,
} from '../notifications/notifications.service.interface';

export class SubscribersService implements ISubscribersService {
  constructor(
    @Inject(SUBSCRIBERS_REPOSITORY_TOKEN)
    private readonly subscribersRepository: ISubscribersRepository,
    @Inject(NOTIFICATIONS_SERVICE_TOKEN)
    private readonly notificationsService: INotificationsService,
  ) {}

  public async deleteSubscriber(chatId: number): Promise<void> {
    await this.subscribersRepository.deleteSubscriber(chatId);
  }

  public async addSubscriber(chat: any, token: string = null): Promise<void> {
    const { id: chatId, type: chatType } = chat;
    const subscriber =
      await this.subscribersRepository.getSubscriberByChatId(chatId);
    if (subscriber) {
      throw new BadRequestException('Ви вже підписані!');
    }
    let name = '';
    if (chatType === 'private') {
      name = `${chat.first_name} ${chat.last_name}`;
    }
    if (chatType === 'group' || chatType === 'supergroup') {
      name = chat.title;
    }
    await this.subscribersRepository.createSubscriber({
      chatId,
      name,
```

```
    type: chatType,
    eduToken: token,
    isActive: true,
  });
}

public async getSubscriberChatToken(chat: any): Promise<string> {
  const { id: chatId, type: chatType } = chat;
  if (chatType === 'private') {
    return null;
  }

  const subscriber =
    await this.subscribersRepository.getSubscriberByChatId(chatId);

  if (subscriber && subscriber.eduToken) {
    return subscriber.eduToken;
  }

  const token = uuid();

  if (!subscriber) {
    await this.addSubscriber(chat, token);
  } else {
    await this.subscribersRepository.updateSubscriber(subscriber.chatId, {
      eduToken: token,
    });
  }

  return token;
}

public async processSubscriberStart(
  chat: any,
  eduToken: string,
): Promise<void> {
  if (!eduToken) {
    await this.notificationsService.sendTextMessage(
      chat.id,
```

```
      'Для початку роботи необхідно перейти за персональним посилання в профілі навчальної платформи MIX',
    );
    return;
   }

   try {
    await this.addSubscriber(chat, eduToken);
    await this.notificationsService.sendTextMessage(
     chat.id,
     'Ви підписані на глобальні повідомлення!',
    );
   } catch (e) {
    if (e instanceof BadRequestException) {
     await this.notificationsService.sendTextMessage(chat.id, e.message);
    }
   }
  }

  public async processSubscriberChatToken(chat: any): Promise<void> {
   const token = await this.getSubscriberChatToken(chat);
   if (!token) {
    await this.notificationsService.sendTextMessage(
     chat.id,
     'Неможливо отримати ChatToken для даного чату!',
    );
   } else {
    await this.notificationsService.sendTextMessage(
     chat.id,
     `Ваш Chat Token: ${token}`,
    );
   }
  }

  public async getSubscriberByToken(token: string): Promise<SubscriberDTO> {
   return this.subscribersRepository.getSubscriberByToken(token);
  }
}
```

Файл messages.service.ts:

```
import { BadRequestException, Inject } from '@nestjs/common';

import { IMessagesService } from './messages.service.interface';
import {
  MESSAGES_REPOSITORY_TOKEN,
  IMessagesRepository,
} from './messages.repository.interface';
import {
  INotificationsService,
  NOTIFICATIONS_SERVICE_TOKEN,
} from '../notifications/notifications.service.interface';
import {
  SUBSCRIBERS_SERVICE_TOKEN,
  ISubscribersService,
} from '../subscribers/subscribers.service.interface';
import { Message } from 'telegraf/typings/core/types/typegram';
import { SessionData } from 'src/models/telegraf-session-context';
import {
  EDU_CLIENT_TOKEN,
  IEduAPIClient,
} from 'src/clients/edu/edu-client.interface';

export class MessagesService implements IMessagesService {
  constructor(
    @Inject(MESSAGES_REPOSITORY_TOKEN)
    private readonly messagesRepository: IMessagesRepository,
    @Inject(NOTIFICATIONS_SERVICE_TOKEN)
    private readonly notificationsService: INotificationsService,
    @Inject(SUBSCRIBERS_SERVICE_TOKEN)
    private readonly subscribersService: ISubscribersService,
    @Inject(EDU_CLIENT_TOKEN)
    private readonly eduApiClient: IEduAPIClient,
  ) {}

  public async possessIncomingMessage(
    message: Message,
    session: SessionData,
  ): Promise<void> {
    if (session?.awaitingReply) {
      const replyUrl = await this.messagesRepository.getReplyURLByMessageId(
```

```
        session.messageId,
      );
      await this.eduApiClient.sendReplyMessage(
        replyUrl,
        session.messageId,
        message['text'],
      );
    }
  }

  public async sendMessageToSubscriber(
    token: string,
    textMessage: string,
    canBeReply: boolean,
  ): Promise<void> {
    const subscriber =
      await this.subscribersService.getSubscriberByToken(token);

    if (!subscriber) {
      throw new BadRequestException(`Subscriber with token ${token} not found`);
    }

    if (!canBeReply) {
      await this.notificationsService.sendTextMessageWithReply(
        subscriber.chatId,
        textMessage,
      );
    } else {
      await this.notificationsService.sendTextMessage(
        subscriber.chatId,
        textMessage,
      );
    }
  }
}
```

Файл notifications.service.ts:

```
import { Inject, forwardRef } from '@nestjs/common';
import { InjectBot } from 'nestjs-telegraf';
import { Telegraf, Context as TelegrafContext } from 'telegraf';
```

```
import {
  ISubscribersService,
  SUBSCRIBERS_SERVICE_TOKEN,
} from '../subscribers/subscribers.service.interface';
import { INotificationsService } from './notifications.service.interface';

export class NotificationsService implements INotificationsService {
  constructor(
    @InjectBot() private bot: Telegraf<TelegrafContext>,
    @Inject(forwardRef(() => SUBSCRIBERS_SERVICE_TOKEN))
    private readonly subscribersService: ISubscribersService,
  ) {}

  public async forwardMessage(
    chatId: number,
    fromChatId: number,
    messageId: number,
  ): Promise<void> {
    try {
      await this.bot.telegram.forwardMessage(chatId, fromChatId, messageId);
    } catch (e) {
      if (e.response.error_code === 403) {
        await this.subscribersService.deleteSubscriber(chatId);
      }
    }
  }

  public async sendTextMessage(chatId: number, text: string): Promise<void> {
    try {
      await this.bot.telegram.sendMessage(chatId, text);
    } catch (e) {
      if (e.response.error_code === 403) {
        await this.subscribersService.deleteSubscriber(chatId);
      }
    }
  }

  public async sendTextMessageWithReply(
    chatId: number,
    text: string,
```

```
  ): Promise<void> {
    try {
      await this.bot.telegram.sendMessage(chatId, text, {
        reply_markup: {
          inline_keyboard: [
            [
              {
                text: 'Надіслати відповідь',
                callback_data: 'reply',
              },
            ],
          ],
        },
      });
    } catch (e) {
      if (e.response.error_code === 403) {
        await this.subscribersService.deleteSubscriber(chatId);
      }
    }
  }
}
```

Файл edu-client.ts:

```
import { HttpService } from '@nestjs/axios';
import { Injectable } from '@nestjs/common';
import { AxiosResponse } from 'axios';
import { Observable, firstValueFrom } from 'rxjs';
import { IEduAPIClient } from './edu-client.interface';

@Injectable()
export class EduAPIClient implements IEduAPIClient {
  constructor(private readonly httpService: HttpService) {}

  public async sendReplyMessage(
    replyUrl: string,
    remoteMessageId: number,
    textMessage: string,
  ): Promise<void> {
    await this.post(`${replyUrl}/${remoteMessageId}`, { textMessage });
  }
```

```
  private async get(url: string) {
    const response = await this.httpService.get(url);
    return await this.parseResponse(response);
  }

  private async post(url: string, body?: any) {
    const response = await this.httpService.post(url, body);
    return await this.parseResponse(response);
  }

  private async patch(url: string, body?: any) {
    const response = await this.httpService.patch(url, body);
    return await this.parseResponse(response);
  }

  private async delete(url: string) {
    const response = await this.httpService.delete(url);
    return await this.parseResponse(response);
  }

  private async parseResponse(
    response: Observable<AxiosResponse<any, any>>,
  ): Promise<any> {
    const res = await firstValueFrom(response);

    return res.data as any;
  }
}
```