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

Кафедра інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«ЗАСТОСУВАННЯ ШТУЧНОГО ІНТЕЛЕКТУ У ІНФОРМАЦІЙНІЙ СИСТЕМІ ДЛЯ ЗДОБУВАЧІВ ЗВО НА ОСНОВІ ТЕЛЕГРАМ-БОТА»

Рівень «магістр»
Галузь знань 12 «Інформаційні технології»,
спеціальність 122 «Комп'ютерні науки»
Виконав здобувач 5 курсу
Сорокін Владислав Сергійович

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

Рецензент к.пед.н., доцент
Лобода Юлія Геннадіївна

Одеса – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ”
Факультет кібербезпеки та інформаційних технологій
Кафедра інформаційних технологій
Галузь знань 12 Інформаційні технології
Спеціальність 122 “Комп’ютерні науки”
Назва освітньої програми “Комп’ютерні науки”

ЗАТВЕРДЖУЮ
Завідувач кафедри інформаційних технологій
доцент Наталія Логінова_____
“____”_____ 2025 року

ЗАВДАННЯ
на кваліфікаційну (магістерську) роботу
здобувачу Сорокіну Владиславу Сергійовичу

1. Тема роботи: “Застосування штучного інтелекту в інформаційній системі для здобувачів ЗВО на основі телеграм-бота”
керівник роботи: к.т.н, доцент Манаков Сергій Юрійович
затверджені наказом НУ “ОЮА” від “10” жовтня 2025 року № 3183-06
2. Строк подання здобувачем роботи “25” листопада 2025 року
3. Дата видачі завдання “02” червня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Вибір теми, вивчення літературних джерел та складання плану роботи	15.09.2025	
2	Підготовка першого розділу роботи та подання його керівнику	01.10.2025	
3	Підготовка другого розділу роботи та подання його керівнику	14.10.2025	
4	Підготовка третього розділу роботи та подання його керівнику	07.11.2025	
5	Підготовка вступу та висновків	10.11.2025	
6	Доопрацювання роботи з урахуванням зауважень керівника	24.11.2025	
7	Подача електронного варіанту роботи на кафедру	25.11.2025	

Здобувач _____Владислав СОРОКІН_____
(підпис) (ім’я та прізвище)
Керівник роботи _____Сергій МАНАКОВ_____
(підпис) (ім’я та прізвище)

АНОТАЦІЯ

Сорокін В. С. "Застосування штучного інтелекту у інформаційній системі для здобувачів ЗВО на основі телеграм-бота". – Кваліфікаційна робота магістра за спеціальністю 122 «Комп'ютерні науки», освітньою програмою «Комп'ютерні науки».

Кваліфікаційна робота викладена на 58 сторінках загального тексту, містить 3 розділа, 20 рисунків, 2 додатки, 15 використаних джерел.

Метою роботи є розробка та впровадження розподіленої інформаційної системи для здобувачів вищої освіти на основі Telegram-бота, розширеної інтеграцією Telegram Mini Apps та аналітичним модулем штучного інтелекту.

Об'єктом дослідження є процеси автоматизації інформаційно-аналітичного забезпечення здобувачів та адміністративного персоналу ЗВО.

Предмет дослідження – методи та архітектурні засоби інтеграції веб-додатків та LLM у Telegram-боти для створення масштабованих та функціонально насичених рішень.

У кваліфікаційній роботі розроблено повноцінну розподілену інформаційну систему, що базується на трирівневій архітектурі (Bot Python, Backend NestJS та Frontend Next.js). Розширений функціонал реалізовано через Telegram Mini Apps, які забезпечують Персональний електронний кабінет здобувача з можливістю редагування профілю, керування фінансовим статусом та структурованої подачі звернень. Також створено адміністративну панель для керування даними, контентом та зверненнями.

За результатами роботи зроблено висновки щодо виконаних та поставлених задач, підтверджено ефективність застосування LLM для автоматизованого аналізу звернень.

Ключові слова: розподілена система, Telegram Mini App, LLM, Штучний інтелект, NestJS, Next.js, Python, REST API.

ABSTRACT

Sorokin V. S. "Application of artificial intelligence in the information system for applicants for higher education based on a Telegram bot". – Master's qualification work in specialty 122 "Computer Science", educational program "Computer Science".

The qualification work is presented on 58 pages of total text, contains 3 chapters, 20 figures, 2 appendix, and 15 used sources.

The goal of the work is the development and implementation of a distributed information system for higher education applicants based on a Telegram bot, extended by the integration of Telegram Mini Apps and an analytical module of Artificial Intelligence.

The object of the study is the processes of automation of information and analytical support for applicants and administrative staff of Higher Educational Institutions (HEIs).

The subject of the study is methods and architectural tools for integrating web applications and LLMs into Telegram bots to create scalable and functionally rich solutions.

The qualification work developed a fully distributed information system based on a three-tier architecture (Bot Python, Backend NestJS, and Frontend Next.js). The extended functionality is implemented through Telegram Mini Apps, which provide a Personal Electronic Cabinet for the applicant with the ability to edit the profile, manage financial status, and structured submission of appeals. An administrative panel for managing data, content, and appeals was also created.

Based on the results of the work, conclusions regarding the executed and set tasks were developed, and the effectiveness of using LLM for automated analysis of appeals was confirmed.

Keywords: distributed system, Telegram Mini App, LLM, Artificial intelligence, NestJS, Next.js, Python, REST API.

ЗМІСТ

ВСТУП

Сучасний освітній простір вищих навчальних закладів (ЗВО) вимагає інноваційних підходів до організації комунікації та навчального процесу. В умовах глобальних викликів, зокрема спричинених воєнними обставинами, значна частина здобувачів освіти вимушено перебуває за межами країни. Це створює критичні системні виклики для інституційної стійкості, безперервності освітнього процесу та забезпечення ефективної адміністративної підтримки [1].

Традиційні канали комунікації виявляються недостатніми для підтримки зв'язку та оперативної організації роботи в умовах територіальної роз'єднаності [2]. Це **актуалізує** необхідність розробки надійних, доступних та інтегрованих інформаційних систем, здатних забезпечити високий рівень взаємодії між студентським контингентом та адміністративними підрозділами (зокрема деканатом).

Метою роботи є розробка та впровадження розподіленої інформаційної системи для здобувачів вищої освіти на основі Telegram-бота, розширеної інтеграцією Telegram Mini Apps та аналітичним модулем штучного інтелекту.

Об’єктом дослідження є процеси автоматизації інформаційно-аналітичного забезпечення здобувачів та адміністративного персоналу ЗВО.

Предмет дослідження – методи та архітектурні засоби інтеграції веб-додатків та LLM у Telegram-боти для створення масштабованих та функціонально насичених рішень.

Для досягнення поставленої мети потрібно виконати наступні **завдання**:

1. Провести аналіз сучасних архітектурних рішень та комунікаційних платформ для обґрунтування вибору оптимального технологічного стеку та підходу до реалізації розподіленої системи.
2. Розробити архітектуру системи, що базується на трьох незалежних компонентах (Python Bot, Node.js API, TMA), та забезпечити чіткий розподіл відповідальності між ними.

3. Реалізувати розподілену архітектуру з гібридним розгортанням (Vercel для фронтенду, Render для бекенду) для мінімізації часу простою та спрощення впровадження оновлень (CI/CD).
4. Створити та впровадити розширений функціонал на базі Telegram Mini Apps для удосконалення підходу до взаємодії адміністратора та здобувачів з системою через графічний інтерфейс.
5. Інтегрувати аналітичний модуль Штучного Інтелекту (LLM) та розробити алгоритм для автоматичного формування аналітичних звітів на основі неструктурованих звернень.
6. Узагальнити отримані результати, сформувати висновки щодо виконаних завдань та надати економіко-організаційне обґрунтування практичного значення проєкту.

Наукова новизна отриманих результатів полягає у наступному:

1. Запропоновано та реалізовано гібридну мікросервісну архітектуру інформаційної системи ЗВО, що поєднує поліглотну бекенд-інфраструктуру (Python Bot та Node.js API) з інноваційним клієнтським інтерфейсом Telegram Mini App (TMA) для оптимізації адміністративних процесів.
2. Удосконалено підхід до інформаційно-аналітичного забезпечення шляхом інтеграції моделі LLM (Large Language Model) для автоматизованого аналізу неструктурованих звернень здобувачів та формування структурованих аналітичних звітів на запит деканату.

У межах даного дослідження пропонується комплексне архітектурне рішення, яке інтегрує високофункціональний Telegram-бот як основний користувацький інтерфейс. Вибір Telegram-бота обумовлений його широкою поширеністю, крос-платформністю та зручністю для оперативного обміну даними.

Розроблена система пропонує розширений спектр функцій, що виходять за рамки простого інформування, включаючи:

– систему персоналізованого статусу присутності;

- механізми збору, структуризації та аналізу даних (зокрема, потреби у формуванні звітів для деканату);
- забезпечення двонаправленого оперативного зв'язку (повідомлення від працівників та звернення студентів).

Практична цінність роботи полягає у наданні деканатам ЗВО готового, економічно ефективного та масштабованого інструменту для цифровізації рутинних процесів, що безпосередньо сприяє підвищенню якості адміністративного супроводу студентів, незалежно від їхнього фізичного місцезнаходження.

Робота апробована на науковій конференції [10].

Розділ 1. АНАЛІЗ АРХІТЕКТУРНИХ РІШЕНЬ ТА ОБҐРУНТУВАННЯ ВИБОРУ

1.1 Обґрунтування вибору платформи для комунікації

Аналіз сучасних платформ швидкого обміну повідомленнями (месенджерів) засвідчив, що Telegram є найбільш доцільним та ефективним рішенням для розробки сучасної інформаційної системи ЗВО, особливо в умовах територіальної розпорошеності студентського контингенту та необхідності забезпечення віддаленого доступу до навчальних даних. Прийняття рішення ґрунтувалося на порівнянні ключових технічних та користувацьких характеристик.

По-перше, висока крос-платформна доступність Telegram є критичною перевагою. Платформа є нативною для більшості операційних систем (iOS, Android, Windows, macOS, Web), що гарантує, що система не вимагатиме від студентів встановлення додаткового спеціалізованого програмного забезпечення. Цей фактор значно спрощує процес впровадження та мінімізує поріг входу для цільової аудиторії. Це контрастує з корпоративними інструментами, які часто вимагають специфічних клієнтських додатків.

По-друге, ключовим технічним аргументом став потужний інструментарій розробки. Наявність досконалого Bot API та сучасної технології Telegram Mini Apps (TMA) дозволила вийти за рамки обмежень, властивих традиційним чат-ботам (наприклад, використання лише інлайн-клавіатур). Впровадження Mini Apps дало змогу створити складні, високоінтерактивні інтерфейси, які за функціональністю не поступаються повноцінним веб-додаткам. Саме ця можливість реалізувати Персональний електронний кабінет з розширеними формами введення даних, таблицями та графічними статусами (наприклад, статус оплати), стала вирішальною перевагою Telegram у порівнянні з іншими месенджерами, покращуючи користувацький досвід [11].

По-третє, оперативність та надійність Telegram є незамінними для освітньої системи. Платформа забезпечує миттєве надсилання системних повідомлень та сповіщень, що є критично важливим для своєчасного інформування здобувачів про зміни у розкладі занять, наближення термінів оплати або оновлення навчальних матеріалів. Використання шифрування та надійної інфраструктури, орієнтованої на швидку доставку повідомлень, підтверджує, що Telegram є оптимальною основою для побудови надійної комунікаційної системи [12].

1.2 Вибір архітектурного підходу: Мікросервіси та TMA

Для забезпечення високої масштабованості, надійності, незалежності розгортання та можливості технологічного супроводу компонентів системи, було обрано мікросервісну архітектуру. Це рішення є стратегічно важливим відходом від монолітного підходу, оскільки дозволяє розділити функціональні обов'язки на вузькоспеціалізовані, незалежні сервіси. Така декомпозиція забезпечує можливість гнучкого оновлення окремих частин без впливу на інші, що критично важливо для підтримки безперервної роботи в умовах освітнього процесу. Компоненти системи розгорнуті на двох хмарних платформах: Render (для бекенд-сервісів) та Vercel (для фронтенд-додатків).

Архітектура системи складається з трьох ключових, незалежно розгорнутих елементів:

Основний бот-сервіс: Цей компонент, реалізований на мові програмування Python, слугує безпосереднім інтерфейсом взаємодії з Telegram Bot API. Його ключові функціональні обов'язки включають прийом та обробку Webhook-ів, обробку базових текстових команд та, що найважливіше, ініціалізацію Telegram Mini App (TMA). Бот виступає як захисний шлюз, передаючи користувачеві спеціально сформовану безпечну URL-адресу, необхідну для запуску веб-додатка.

API-сервіс для фронтенду (Node.js/NestJS): Цей компонент є сервером RESTful API, реалізованим за допомогою фреймворку NestJS, що забезпечує високу швидкість та структурованість розробки на базі Node.js. API-сервіс

відповідає за обробку всіх складних запитів, що надходять від клієнтської частини (TMA). Його ключова роль – це реалізація бізнес-логіки, включаючи валідацію даних, взаємодію з базою даних, а також ініціацію роботи модуля Штучного Інтелекту для формування аналітичних звітів. Використання NestJS дозволило забезпечити надійний механізм валідації initData для захисту ендпоінтів від несанкціонованого доступу.

Telegram Mini App (TMA): Цей елемент є сучасним, клієнтським веб-інтерфейсом, розробленим на базі фреймворку Next.js і розміщеним на платформі Vercel. TMA викликається безпосередньо з чату Telegram і використовується для виконання всіх складних операцій, які вимагають графічного введення даних (форми, перегляд таблиць, активація формування звітів ШІ). TMA не має прямого доступу до бази даних, а здійснює виключно захищений обмін даними з Node.js API-сервісом.

Такий розподіл функціоналу забезпечує високу відмовостійкість системи, оскільки потенційні проблеми з відображенням клієнтського інтерфейсу (TMA) не вплинуть на роботу основного бота чи API-сервісу, і навпаки.

Компонент	Технологічна роль	Призначення
Python Service (Render)	Telegram Bot Logic	Обробка Webhook, логіка, взаємодія з БД.
Node.js Service (Render)	RESTful API	Прошарок для обміну даними між Mini App та Python-логікою.
Mini App (Vercel)	Користувацький інтерфейс (Фронтенд)	Надання складного графічного інтерфейсу для адміністратора/студента.

Рисунок 1.1 – Ключові елементи архітектури

1.3 Обґрунтування вибору платформ безперервної інтеграції та розгортання (CI/CD)

Для забезпечення ефективного та швидкого циклу розробки і доставки (DevOps) було обрано хмарні платформи Render та Vercel. Їхнє спільне використання відображає сучасний підхід до розгортання розподілених систем.

Render для Бекенду

Render обрано для розміщення постійно працюючих сервісів (Python та Node.js), оскільки:

- Він підтримує розгортання різних мов програмування (показує гнучкість мікросервісів).
- Він забезпечує надійну роботу бекенд-сервісів, які повинні бути завжди доступні для прийому Webhook-ів від Telegram.

Vercel для Фронтенду (Mini App)

Vercel обрано для хостингу Telegram Mini App (ol-academy-frontend) з наступних причин:

- Спеціалізація на швидкому розгортанні фронтенд-додатків.
- Автоматична оптимізація та CDN, що гарантує високу швидкість завантаження Mini App для кінцевого користувача.
- Ефективна інтеграція з Git (GitHub/GitLab), що прискорює процес оновлення та CI/CD.

1.4 Вибір системи управління базами даних (СУБД)

Для першої ітерації системи, що була реалізована з фокусом на швидкому створенні мінімально життєздатного продукту (MVP), була обрана вбудована СУБД SQLite3. Такий вибір був обґрунтований його ключовими перевагами на початковому етапі розробки, зокрема, простотою впровадження, оскільки база даних зберігається як єдиний файл на локальному сховищі і не вимагає розгортання

окремого серверу. Це значно спростило архітектуру, мінімізувавши накладні витрати на конфігурацію та прискоривши розробку. Крім того, при невеликих обсягах даних і локальному зберіганні, SQLite3 демонструє високу швидкість операцій читання та запису.

Однак, на магістерському рівні є критично важливим ідентифікувати обмеження цього рішення в контексті промислової експлуатації та подальшого масштабування. В умовах обраного хмарного хостингу (Render/Vercel) файл бази даних SQLite3 зберігається в локальному сховищі контейнера бекенд-сервісу. Це створює високий ризик потенційної втрати даних при автоматичному перезапуску або оновленні контейнера хмарним провайдером, оскільки локально збережений файл БД не є стійким. Також, SQLite3 не підтримує багатопотоковий доступ і не може ефективно працювати у кластерному середовищі, що суттєво обмежує можливість горизонтального масштабування бекенд-сервісу.

Перспектива міграції: З огляду на необхідність переведення системи у промислову експлуатацію та забезпечення надійності фінансових (Оплата) та академічних даних, у подальших ітераціях системи пропонується обов'язкова міграція на хмарну СУБД (наприклад, PostgreSQL або аналогічні). Перехід до керованої СУБД забезпечить стійкість зберігання даних, відмовостійкість та можливість горизонтального масштабування, що відповідає вимогам до сучасної інформаційної системи ЗВО [13].

1.5 Висновки до розділу 1

Проведений детальний аналіз сучасних інформаційних технологій та архітектурних підходів дозволив обґрунтувати вибір оптимального технологічного стеку для розробки інформаційної системи ЗВО на базі Telegram-бота.

Обґрунтування платформи: Вибір Telegram як основної комунікаційної платформи підтверджено його високою доступністю, крос-платформністю та оперативністю зв'язку, що є критично важливим в умовах територіальної розпорошеності студентів. Інтеграція Telegram Mini App (TMA) дозволила

подолати обмеження традиційного текстового інтерфейсу бота, надаючи адміністраторам та здобувачам повноцінний, графічний інтерфейс для складних операцій, таких як керування зверненнями та формування звітів.

Архітектурне рішення: Обрана мікросервісна архітектура (поділ на Python-сервіс, Node.js API та TMA) забезпечила необхідну гнучкість та незалежність розгортання компонентів. Цей підхід дозволяє чітко відокремити логіку обробки Webhook-ів від логіки API для фронтенду, підвищуючи надійність і спрощуючи подальшу модернізацію кожного модуля.

CI/CD та Хостинг: Застосування хмарних платформ Render (для постійно працюючих бекенд-сервісів) та Vercel (для швидкого розгортання фронтенду Mini App) демонструє використання сучасних DevOps-практик. Це забезпечує безперервну інтеграцію та доставку (CI/CD), що є необхідною умовою для підтримки актуальності та функціональності системи.

База даних та перспективи: Вибір SQLite3 на початковому етапі розробки забезпечив швидкість впровадження. Проте, зважаючи на обмеження, пов'язані з хмарним хостингом, ідентифіковано необхідність подальшої міграції на хмарну СУБД для забезпечення стійкості зберігання даних у промисловому використанні.

Таким чином, розроблена архітектура є сучасною, гнучкою та повністю відповідає поставленим завданням щодо оптимізації адміністративних процесів у ЗВО.

Розділ 2. АРХІТЕКТУРА ТА ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

2.1 Архітектурна схема системи та взаємодія компонентів

Розроблена інформаційна система функціонує на основі розподіленої мікросервісної архітектури, що дозволяє досягти високої гнучкості, незалежного масштабування та чіткого розподілу відповідальності між програмними модулями. Такий архітектурний підхід, на відміну від традиційного монолітного, забезпечує високу відмовостійкість і можливість незалежного оновлення та супроводу кожного сервісу, що є критично важливим для хмарних програмних рішень. Система реалізована у вигляді трьох незалежних, слабозв'язаних компонентів, які взаємодіють між собою за допомогою асинхронних Webhook-ів та синхронних RESTful API-запитів.

В основі архітектури лежить принцип сегрегації відповідальності (Separation of Concerns). Комунікаційний рівень забезпечує Основний бот-сервіс на Python. Цей компонент виступає як первинний інтерфейс взаємодії з платформою Telegram, відповідаючи за обробку вхідних Webhook-ів, виконання простих текстових команд і, найголовніше, ініціалізацію клієнтської частини. Бізнес-логіка та обробка складних даних повністю покладені на RESTful API-сервіс, реалізований на Node.js із використанням NestJS. Цей сервіс функціонує як API , надаючи захищений інтерфейс для обробки усіх запитів, пов'язаних із даними (редагування профілю, керування зверненнями, активація модуля ШІ), а також керуючи взаємодією з базою даних. Фінальний компонент, Клієнтська частина (Telegram Mini App, TMA), є сучасним веб-інтерфейсом на Next.js/React. Він використовується виключно як рівень подання даних та забезпечення розширеного користувацького досвіду (UX), відокремлюючи логіку відображення від логіки обробки даних.

Взаємодія між цими трьома сервісами є багаторівневою. Коли користувач ініціює дію, яка вимагає графічного інтерфейсу, Python Bot надсилає команду для

відкриття TMA. Після запуску Mini App всі подальші складні операції (наприклад, надсилання форми зворотного зв'язку) здійснюються за допомогою синхронних HTTP-запитів безпосередньо до Node.js API-сервісу. Ключовим аспектом безпеки є механізм валідації. Кожен запит від Mini App містить спеціальні ініціаційні дані (initData), надані Telegram. API-сервіс виконує криптографічну валідацію цього хешу, щоб гарантувати, що запит є легітимним і походить від справжнього користувача Telegram, а не від зовнішнього зловмисника, тим самим забезпечуючи надійний захист бекенд-ендпоінтів від несанкціонованого доступу [14].

Лише після успішної валідації API-сервіс здійснює необхідну бізнес-логіку та взаємодію з базою даних, завершуючи цикл операції.

Рисунок 2.1 – Три ключові незалежно розгорнуті сервіси

2.1.1 Структурна схема системи

Взаємодія компонентів системи базується на принципі асинхронної обробки подій для Core Telegram-бота та синхронних HTTP-запитів для Mini App.

Такий гібридний підхід є типовим для розподілених систем, що інтегруються із зовнішніми платформами.

Процес комунікації ініціюється з асинхронної взаємодії з Telegram Bot API. Всі оновлення від користувачів (текстові команди, натискання кнопок, команди на запуск Mini App) передаються від Telegram на Python-сервіс, розгорнутий на Render, через механізм Webhook. Цей асинхронний зв'язок гарантує, що основний бот завжди готовий приймати нові події, незалежно від тривалості їхньої подальшої обробки, забезпечуючи високу відмовостійкість первинного каналу зв'язку. Після

отримання команди на запуск, Python-сервіс ініціює виклик Mini App, передаючи користувачеві безпечний URL-адресу. Telegram завантажує клієнтський фронтенд, розміщений на Vercel, безпосередньо у вікно чату.

Після успішного завантаження починається ключовий етап – синхронна взаємодія Mini App <-> Node.js API. Фронтенд Mini App виконує стандартні RESTful HTTP-запити до Node.js API (розгорнутого на Render) для обміну даними. Ця комунікація є синхронною, оскільки Mini App вимагає негайної відповіді від бекенду для оновлення свого інтерфейсу (наприклад, відображення списку звернень або підтвердження збереження оновленого профілю користувача).

Крім того, в архітектурі передбачений внутрішній канал взаємодії Node.js API <-> Python Bot. Цей канал є необхідним для дотримання принципу інкапсуляції даних та контролю доступу до БД. Оскільки Python Bot є єдиним компонентом, що має прямий доступ до бази даних SQLite3 (через обмеження хмарного хостингу), Node.js API не здійснює прямих запитів до БД. Натомість, він надсилає внутрішні команди або запити до Python-сервісу для доступу до даних або для ініціації складних процесів, таких як формування аналітичного звіту, який вимагає роботи з даними з БД. Цей патерн забезпечує додатковий рівень безпеки та дозволяє Python-сервісу контролювати всі операції з базою даних, підвищуючи надійність системи в цілому [15].

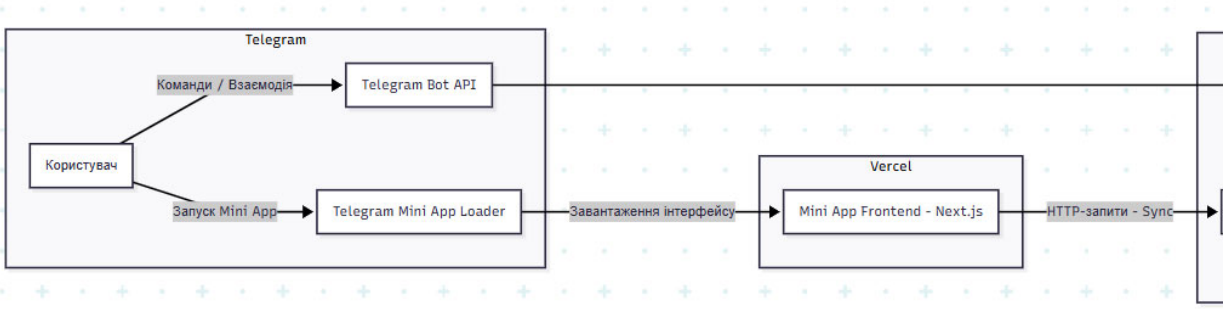


Рисунок 2.2 – Взаємодія компонентів системи

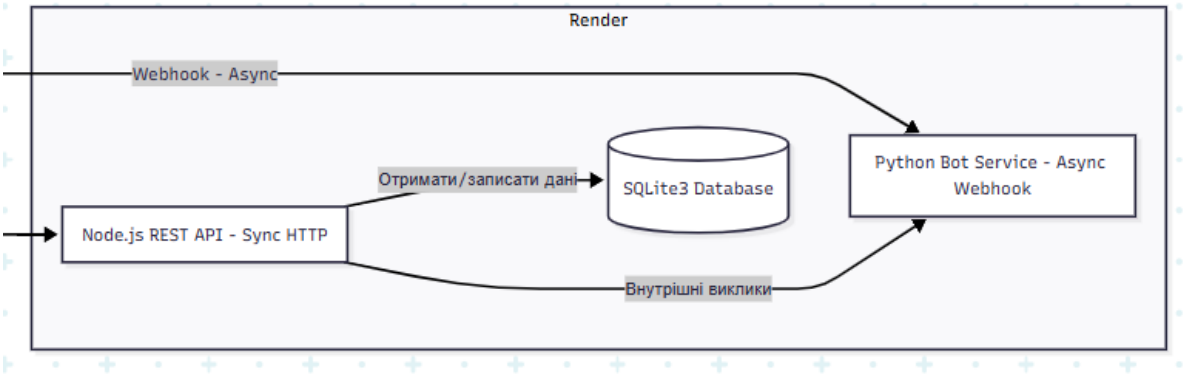


Рисунок 2.3 – Продовження рис.2.2

2.1.2 Розподіл функціональних обов'язків

Python Service (ol-academy-python): Виконує роль "ядра" бота. Його основна функція – обробка Webhook-ів, розпізнавання простих команд /start, /help та керування станом сесії користувача [8]. Також цей сервіс є єдиним легальним шлюзом до бази даних SQLite3, що гарантує цілісність даних, оскільки інші сервіси звертаються до нього через внутрішні механізми [4].

Node.js API Service (ol-academy-backend): Цей сервіс є проміжним шаром, спеціально розробленим для обслуговування Mini App. Він обробляє запити від фронтенду (наприклад, аутентифікація користувача Mini App, запити на отримання структурованих даних для відображення в таблицях) та трансформує їх у відповідні запити до Python-сервісу або бази даних. Використання Node.js тут обґрунтоване його ефективністю в обробці великої кількості паралельних I/O-операцій, типових для веб-сервісів.

Telegram Mini App (TMA) (ol-academy-frontend): Клієнтська частина, розгорнута на Vercel, відповідає за складний графічний інтерфейс користувача. Вона використовується адміністратором для:

- Перегляду таблиць звернень.
- Введення розгорнутої відповіді на звернення.
- Ініціації складного процесу формування звіту, який вимагає значних обчислень на бекенді.

2.2 Клієнтської частина: Telegram Mini App (TMA)

Клієнтська частина системи, реалізована у вигляді Telegram Mini App (TMA) під назвою ol-academy-frontend, є критично важливим компонентом для забезпечення адміністраторам та студентам зручного графічного інтерфейсу (GUI) для виконання складних операцій, які неможливо ефективно реалізувати через класичні текстові команди Telegram. TMA є, по суті, веб-додатком, який запускається безпосередньо у вікні месенджера Telegram. Його хостинг на платформі Vercel забезпечує високу швидкість завантаження та стабільність роботи, що є важливим для UX [5].

Основні функції TMA для адміністративного персоналу (Деканат):

1. Керування зверненнями: Надання структурованого інтерфейсу для перегляду, фільтрації та пошуку звернень студентів. Це значно ефективніше, ніж відображення даних у вигляді простих текстових списків.
2. Надання відповідей: Можливість введення розгорнутих відповідей на звернення за допомогою текстового поля з підтвердженням.
3. Генерація звітів: Активація складних бекенд-процесів для формування звітів (наприклад, звіт про кількість звернень за період, звіт про найпоширеніші теми скарг), що ініціюється натисканням на кнопку "Сформувати звіт".
4. Візуалізація даних: Використання веб-технологій дозволяє відображати дані у вигляді таблиць.

Інтеграція Mini App у загальну архітектуру має кілька ключових технічних аспектів:

- Telegram Web Apps SDK: Фронтенд Mini App використовує офіційний JavaScript Telegram Web Apps SDK. Цей SDK забезпечує двосторонній зв'язок між веб-додатком і клієнтом Telegram. Він використовується для:
    - Отримання даних користувача: Передача ідентифікатора користувача та даних ініціації від Telegram до Mini App для аутентифікації.

    - Зміна інтерфейсу: Керування зовнішнім виглядом вікна (зміна кольору заголовка, активація/дезактивація кнопки закриття тощо).
- Розгортання на Vercel: Проект ol-academy-frontend розгорнутий на Vercel. Ця платформа забезпечує автоматичне розгортання при кожному коміті в репозиторій Git, використовуючи принципи незмінної інфраструктури (Immutable Infrastructure). Vercel оптимізований для розповсюдження статичних ресурсів через глобальну мережу CDN (Content Delivery Network), гарантуючи мінімальну затримку при завантаженні Mini App.
- Взаємодія з API (Frontend ↔ Node.js): Mini App не спілкується безпосередньо з Python-ботом. Всі запити на отримання або зміну даних (наприклад, POST /api/submit-response або GET /api/get-tickets) направляються до Node.js API (ol-academy-backend), розміщеного на Render. Цей поділ забезпечує безпеку та чітке розділення відповідальності між фронтендом та бекендом.

2.3 Бекенд-інфраструктура: Node.js API та Python Bot

Програмна логіка системи реалізована за принципом розподілених сервісів, кожен з яких оптимізований під свої унікальні задачі та розгорнутий на платформі Render для забезпечення постійної доступності та масштабованості. Цей підхід є фундаментальною основою мікросервісної архітектури, обраної для проєкту.

Реалізація бекенду включає два незалежні сервіси: Node.js API та Python Bot, які були обрані для використання сильних сторін різних технологій у гібридному середовищі.

API Service (Node.js/NestJS): Центральним елементом обробки даних є сервіс, реалізований на Node.js, який функціонує як API Gateway, забезпечуючи захищений інтерфейс для обробки усіх складних запитів, що надходять від клієнтської частини (Telegram Mini App). Вибір платформи Node.js для цього шару обумовлений його асинхронною, неблокуючою моделлю вводу/виводу, що є ідеальною для ефективної обробки великої кількості паралельних HTTP-запитів від Mini App, гарантуючи низьку затримку відповідей та високу пропускну здатність.

Цей сервіс відповідає за виконання основної бізнес-логіки, включаючи обробку даних форм, керування обліковими записами, а також єдиною точкою входу для ініціації роботи аналітичного модуля Штучного Інтелекту. Найважливіша функція Node.js API – це аутентифікація Mini App. Кожен запит, що надходить від клієнта, повинен бути валідований шляхом криптографічної перевірки хешу, переданого від Telegram при ініціації Mini App, що є критично важливим для захисту API-ендпоінтів від несанкціонованого доступу та гарантує, що лише легітимні користувачі можуть взаємодіяти із системою.

Core Bot Service (Python) та доступ до БД: На противагу API-сервісу, Основний бот-сервіс реалізований на Python і виконує функцію "ядра" системи. Його роль – це комунікаційний інтерфейс з Telegram API, що обробляє Webhook-и та прості команди. Однак, унікальність його ролі полягає у тому, що він є єдиним легальним шлюзом до бази даних SQLite3. Оскільки у поточному MVP-рішенні використовується файлова база даних, яка є вразливою до паралельного доступу, Node.js API не здійснює прямих запитів до БД. Замість цього, він використовує внутрішній канал взаємодії для надсилання команд Python-сервісу. Це забезпечує принцип інкапсуляції даних: Python-сервіс виконує необхідні транзакції (наприклад, UPDATE, SELECT) і повертає оброблені дані назад Node.js API, який потім форматує їх для фронтенду. Таким чином, Python-сервіс виступає як контрольний шар доступу до даних, підвищуючи надійність усієї системи.

Розгортання та CI/CD: Уся бекенд-інфраструктура розгорнута на платформі Render, яка була обрана для забезпечення постійного хостингу (Always On). Це є критичною вимогою для Telegram-бота, який повинен безперервно очікувати Webhook-и, а також для API-сервісу, який повинен миттєво відповідати на запити від Mini App. Render підтримує поліглотну архітектуру, дозволяючи легко розгортати та керувати сервісами, написаними різними мовами (Python та Node.js) в рамках одного проєкту, підтверджуючи гнучкість обраного архітектурного рішення.

Платформа також забезпечує повністю автоматизований цикл безперервної інтеграції та доставки (CI/CD), інтегруючись із репозиторіями Git. При кожному

коміті до основної гілки Render автоматично ініціює збірку та розгортання відповідного сервісу, що значно прискорює впровадження оновлень та підвищує загальну швидкість розробки. Таким чином, бекенд-інфраструктура побудована за принципом технологічної доцільності та надійного розподілу функцій, що є основою масштабованості та стійкості системи.

Програмна логіка системи реалізована за принципом розподілених сервісів, кожен з яких оптимізований під свої унікальні задачі та розгорнутий на платформі Render для забезпечення постійної доступності та масштабованості. Цей підхід є основою мікросервісної архітектури, обраної для проєкту [9].

2.3.1 Node.js API як шлюз авторизації та агрегатор даних

Сервіс, реалізований на Node.js з використанням фреймворку Express.js, виконує роль проміжного шару (API Gateway) між Telegram Mini App (TMA) та основним ядром логіки (Python Bot).

Ключові завдання Node.js API:

1. Аутентифікація Mini App (TMA): Це найважливіша функція. Кожен запит, що надходить від клієнта TMA, повинен бути валідований. Node.js API використовує дані, передані від Telegram при ініціації Mini App (наприклад, hash або initData), для перевірки автентичності користувача. Це гарантує, що лише авторизовані адміністратори, ініційовані в Telegram, можуть отримати доступ до API-ендпоінтів для керування зверненнями.
    - Приклад ендпоінту: POST /api/auth/login – приймає дані ініціалізації TMA, валідує їх і повертає авторизаційний токен для подальших запитів.
2. Агрегація та Трансформація Даних: Node.js API слугує агрегатором, який обробляє складні запити від фронтенду і розбиває їх на простіші, внутрішні запити до Python-сервісу.

- Наприклад, коли адміністратор запитує список звернень, API надсилає запит до Python-сервісу, отримує сирі дані про звернення, форматує їх у необхідний для Mini App формат (JSON) і повертає на фронтенд. Цей сервіс відповідає за представлення даних.

3. Обробка I/O-інтенсивних операцій: Node.js є ідеальним вибором для цього шару, оскільки його асинхронна, неблокуюча модель I/O ефективно обробляє велику кількість паралельних HTTP-запитів від TMA, забезпечуючи низьку затримку відповідей.

2.3.2 Розгортання та CI/CD на платформі Render

Для забезпечення надійної та постійної роботи бекенд-сервісів (Python Bot та Node.js API) була обрана хмарна платформа Render.

Обґрунтування вибору Render:

- Постійний Хостинг (Always On): На відміну від деяких безсерверних рішень, Render надає можливість розгортати постійні веб-сервіси, які завжди активні. Це критично важливо для Telegram-бота, який очікує Webhook-и, та для API, яке повинно миттєво відповідати на запити від Mini App.
- Автоматизований CI/CD: Render інтегрується з репозиторієм Git (наприклад, GitHub/GitLab). При кожному коміті до гілки main або master платформа автоматично ініціює збірку та розгортання відповідного сервісу (Python або Node.js). Це забезпечує безперервну інтеграцію та доставку (CI/CD).
- Підтримка поліглотних архітектур: Render ідеально підходить для мікросервісів, оскільки дозволяє легко розгортати та керувати сервісами, написаними різними мовами (в даному випадку Python і Node.js) в рамках одного проєкту [6].
- Зберігання стану (SQLite3): Хоча використання SQLite3 на Render має певні нюанси, Render забезпечує стабільне файлове сховище, необхідне для цієї бази даних. Розміщення SQLite3 на тому ж хості, де працює Python-сервіс, мінімізує затримки доступу до даних [7].

Таким чином, Render забезпечує надійне, економічне та автоматизоване середовище для життєвого циклу бекенд-компонентів системи.

2.4 Висновки до розділу 2

Програмна реалізація системи на основі Telegram-бота була успішно здійснена за допомогою мікросервісної архітектури, що дозволило інтегрувати сильні сторони різних технологій та платформ.

1. Розподіл функціоналу: Було досягнуто чіткого розподілу між Python-сервісом (ядро бота, керування станом, доступ до БД), Node.js API (шлюз для аутентифікації та агрегації даних Mini App) та Telegram Mini App (TMA) (графічний інтерфейс користувача, розміщений на Vercel).
2. Деплоймент: Використання хмарних платформ Render для бекенд-сервісів та Vercel для фронтенду TMA забезпечило високу швидкість розгортання та надійність функціонування, підтверджуючи ефективність сучасних практик CI/CD.
3. Комплексний функціонал: Вдалося реалізувати складний функціонал, як-от формування звіту, за допомогою скоординованої взаємодії всіх сервісів. TMA ініціює запит, Node.js передає його, а Python виконує аналіз даних з SQLite3 і доставляє фінальний результат користувачеві.
4. Перспективи: Подальша робота передбачає міграцію на повноцінну хмарну СУБД для забезпечення стійкості даних та готовності системи до промислового масштабування.

Розділ 3. РОЗРОБКА ТА ВПРОВАДЖЕННЯ РОЗШИРЕНОГО ФУНКЦІОНАЛУ НА ОСНОВІ TELEGRAM MINI APPS

3.1 Обґрунтування розширення функціоналу системи

Програмний продукт, розроблений у попередніх розділах, забезпечує базову інтерактивну взаємодію між ЗВО та здобувачами освіти (ЗВО) за допомогою текстових команд і інлайн-клавіатури Telegram. Однак, із розвитком системи та зростанням вимог до функціональності, виникла потреба у впровадженні складніших інтерфейсів, які виходять за рамки можливостей стандартного бота.

Основною метою розширення функціоналу стало створення повноцінного особистого кабінету здобувача вищої освіти без необхідності використання окремих веб-додатків чи мобільних програм.

Традиційні інтерфейси Telegram-ботів мають суттєві обмеження щодо складності подання даних, графічного оформлення та інтерактивності. Наприклад, такі задачі, як заповнення розширених форм: "Профіль користувача" або "Зворотній зв'язок", відображення динамічних графічних елементів (наприклад, статусу оплати) або робота з багатосторінковими документами (як "Розклад занять"), є неоптимальними або неможливими для реалізації через стандартні механізми бота.

Технологія Telegram Mini Apps (Web Apps) вирішує ці проблеми, дозволяючи вбудовувати повноцінні веб-інтерфейси (побудовані на HTML, CSS, JavaScript) безпосередньо у вікно чату.

Основні переваги, які забезпечило впровадження Mini Apps у даний проект:

- Розширена інтерактивність та UX: Міні-додатки дозволили створити звичний та інтуїтивно зрозумілий графічний інтерфейс користувача, який значно підвищує зручність роботи з системою (наприклад, можливість редагувати профільні дані на одній сторінці "Профіль користувача".

- Складна візуалізація даних: З'явилася можливість візуалізувати складні дані, такі як статус оплати, список сповіщень чи завантаження файлів розкладу, що було б важко реалізувати за допомогою текстових повідомлень.
- Єдине середовище: Користувачеві не потрібно переходити до зовнішніх ресурсів. Увесь функціонал особистого кабінету інтегровано безпосередньо в Telegram, що спрощує авторизацію та використання.
- Динамічне оновлення: Mini Apps забезпечують можливість швидкого динамічного оновлення даних без перезавантаження всього інтерфейсу бота, що є критично важливим для відображення актуального Статусу оплати чи Списку сповіщень.

Таким чином, інтеграція Mini Apps стала необхідним кроком для трансформації базового Telegram-бота в повноцінну інформаційну систему для здобувачів ЗВО, що відповідає сучасним вимогам до зручності та функціональності.

3.2 Архітектура інтеграції Telegram Mini Apps та структура програмного комплексу

Розширення функціоналу системи за допомогою Mini Apps вимагало переходу до розподіленої архітектури, що дозволило ефективно розділити логіку бота від логіки веб-додатків. Реалізована архітектура є трирівневою і складається з трьох основних модулів: Core Bot, Backend API та Frontend Mini App.

Проект ol-academy розділений на три незалежні компоненти, як показано на структурі проекту (рис. 3.1):

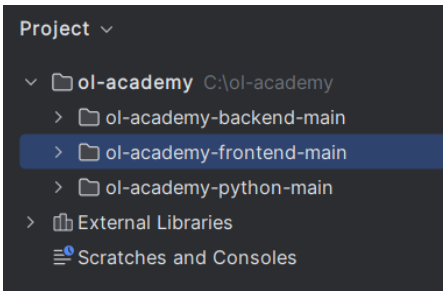


Рисунок 3.1 – Структура проєкту

1. ol-academy-python-main: містить основну логіку Telegram-бота, написаного мовою Python. Цей модуль відповідає за ініціалізацію бота, обробку текстових команд, а також за запуск Mini Apps через спеціальні кнопки або команди. Також тут знаходиться модуль взаємодії з БД, який може використовуватися як ботом, так і Backend API.
2. ol-academy-backend-main: серверний компонент, реалізований на фреймворку NestJS. Цей рівень забезпечує захищений REST API-інтерфейс, який використовується Mini Apps для отримання та оновлення даних. Його структура базується на модульності (user, feedback, notification, schedule), що відповідає принципам чистої архітектури.
3. ol-academy-frontend-main: клієнтський веб-додаток, реалізований на базі Next.js, що використовує TypeScript (TSX). Цей модуль містить візуальну частину Mini Apps (profile, fees, notifications, schedule, feedback), які відображаються у вікні Telegram.

3.2.2 Схема взаємодії компонентів

Взаємодія між трьома компонентами системи відбувається за такою послідовністю (рис. 3.2):

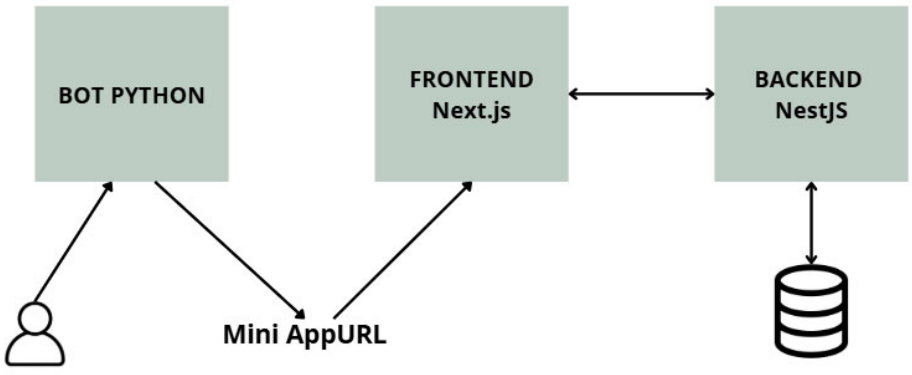


Рисунок 3.2 – Взаємодія між трьома компонентами системи

1. Ініціалізація Mini App: Користувач у Telegram натискає кнопку, яка містить URL-адресу Mini App. Бот відправляє URL-адресу клієнту.
2. Запуск Mini App: Клієнт Telegram відкриває веб-інтерфейс Mini App, який розміщений на сервері. Mini App отримує ініціалізаційні дані для безпечної авторизації користувача.
3. Взаємодія Mini App ↔ Backend API: Клієнтський додаток здійснює асинхронні запити до Backend API. Ці запити захищені з використанням даних, отриманих від Telegram, що гарантує, що запити надходять від легітимних користувачів.
4. Backend API ↔ База Даних: Backend API обробляє запити, викликає відповідні сервіси, які взаємодіють з базою даних для отримання або запису даних.

3.2.3 Технологічний стек Mini Apps

Для забезпечення сучасної та масштабованої розробки Mini Apps був обраний наступний технологічний стек:

- Фронтенд: Next.js / TypeScript / React (для модульності).
- Бекенд: NestJS / TypeScript (для високої продуктивності та структурованості API).
- Мова програмування для Бота: Python.

Таке розділення дозволило застосувати принципи SOLID та забезпечило високу модульність (кожен функціональний блок, такий як profile, feedback, fees, має власні компоненти та логіку запитів), що значно спрощує подальший розвиток та підтримку системи.

3.3 Реалізація функціоналу Mini App для здобувача вищої освіти

Реалізація функціоналу для здобувача вищої освіти є ключовою частиною розширеної системи, що перетворює базовий Telegram-бот на повноцінний

персональний електронний кабінет. Усі наступні модулі реалізовані як окремі компоненти (компоненти React у Next.js) в межах клієнтської частини ol-academy-frontend-main. Це забезпечило високу модульність, швидке завантаження та можливість незалежного оновлення інтерфейсу.

3.3.1 Модуль «Профіль користувача»

Цей модуль є центральним елементом і реалізований як інтерактивна форма, яка дозволяє користувачеві самостійно переглядати та оновлювати персональні дані (рис. 3.3).

- Функціональність: Відображення основних даних (Ім'я, Прізвище, Дата народження, Факультет, Група, Курс, Контакти) та можливість їх редагування.
- Реалізація Mini App: Використання Mini App дозволило реалізувати інтерактивні поля введення та механізм валідації даних, що неможливо зробити за допомогою звичайних повідомлень бота. При натисканні на кнопку редагування дані оновлюються через Backend API (NestJS) і зберігаються в БД.

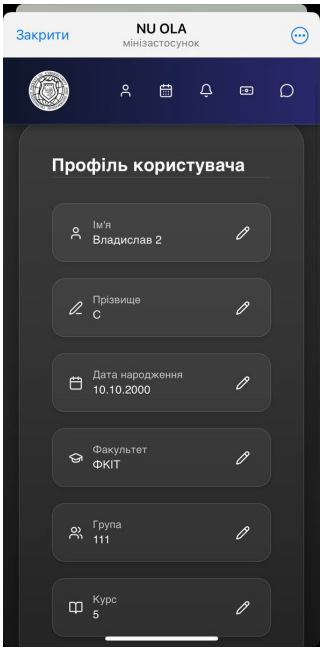


Рисунок 3.3 – Профіль користувача

3.3.2 Модуль «Розклад занять»

Модуль забезпечує доступ до актуального розкладу занять, вирішуючи проблему відображення складного табличного контенту у мобільному інтерфейсі.

- Функціональність: Надання користувачеві можливості вибрати курс (1-5) та завантажити відповідний файл розкладу у форматі PDF.
- Реалізація Mini App: Інтерфейс надає структурований вибір курсу та кнопку завантаження. Це мінімізує кількість кроків для користувача та забезпечує прямий доступ до файлового сховища, тоді як бот мав би спочатку відправити запит, а потім сам файл у чат.
-

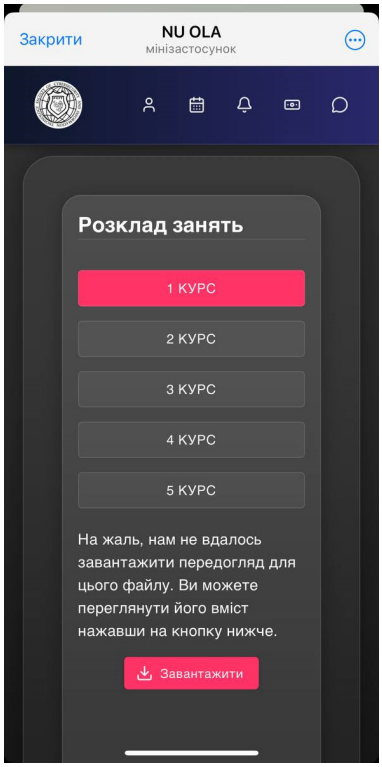


Рисунок 3.4 – Розклад занять

3.3.3 Модуль «Оплата навчання»

Цей модуль відображає фінансовий статус студента та надає можливість самостійного відстеження платежів.

- Функціональність: Відображення поточного статусу оплати ("Оплачено" / "Не оплачено") та можливість самостійної зміни цього статусу користувачем після фактичного підтвердження платежу (рис. 3.5, рис. 3.6).
- Реалізація Mini App: Динамічні елементи інтерфейсу (зелені/червоні маркери статусу, інтерактивний перемикач) забезпечують чітку візуалізацію фінансового стану. Це спрощує процес звірки платежів як для здобувача, так і для адміністрації ЗВО.

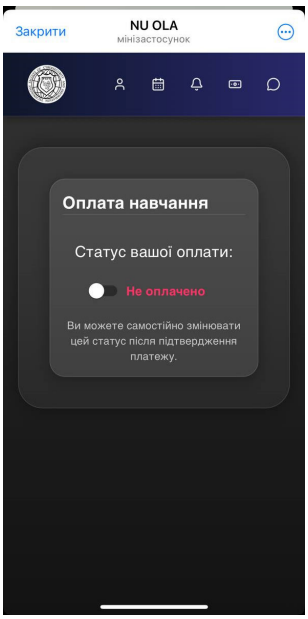


Рисунок 3.5 – Оплата навчання

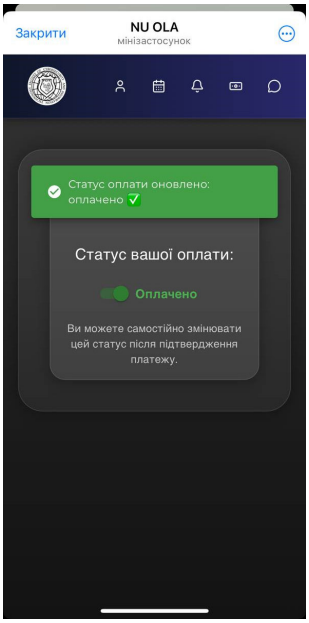


Рисунок 3.6 – Статус оплати навчання

3.3.4 Модуль «Зворотній зв'язок»

Модуль слугує уніфікованою точкою входу для офіційних звернень та запитань до адміністрації ЗВО.

- Функціональність: Надання можливості створити нове звернення (вказати тему та детальний опис) та переглянути історію попередніх звернень (див. скріншоти "Зворотній зв'язок").
- Реалізація Mini App: Міні-додаток надає багаторядкові поля та структуровану форму для подання запитів, що гарантує отримання

адміністрацією всієї необхідної інформації. Історія звернень, відображена нижче, покращує досвід користувача та дозволяє відстежувати комунікацію.

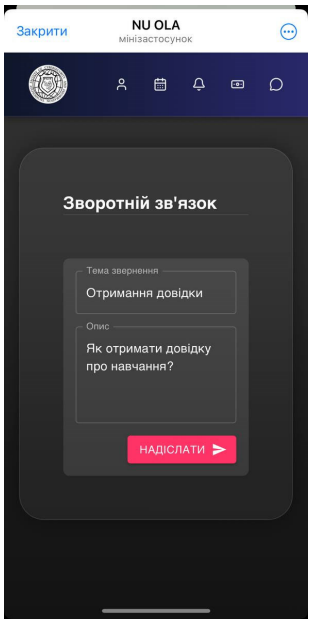


Рисунок 3.7 – Зворотній зв’язок

3.4 Реалізація функціоналу Mini App для адміністратора

Mini App для адміністратора (Модератор або Викладач) є ключовим елементом системи, що дозволяє централізовано керувати даними здобувачів, контентом та обробляти їхні звернення. Інтерфейс адміністратора реалізований як захищений розділ (/admin у структурі ol-academy-frontend-main), доступ до якого надається лише авторизованим користувачам з відповідними правами.

3.4.1 Модуль «Список студентів» та управління даними

Цей модуль є центром управління базою даних здобувачів вищої освіти.

- Призначення: Централізований пошук, фільтрація та перегляд детальних даних студентів.
- Функціональність:

- Розширений пошук: Можливість пошуку та фільтрації за кількома критеріями: Ім'я, Група, Курс, Email та Статус.
- Перегляд карток: Відображення карток студентів з ключовою інформацією, включаючи статус оплати ("Не оплачено"), що дозволяє швидко виявляти проблемних користувачів.

- Технічна реалізація: Використовується захищений API-маршрут (Backend) для отримання списку студентів із застосуванням пагінації та фільтрації на рівні запиту до БД. Редагування даних вимагає найвищого рівня авторизації (перевірка ролі адміністратора).

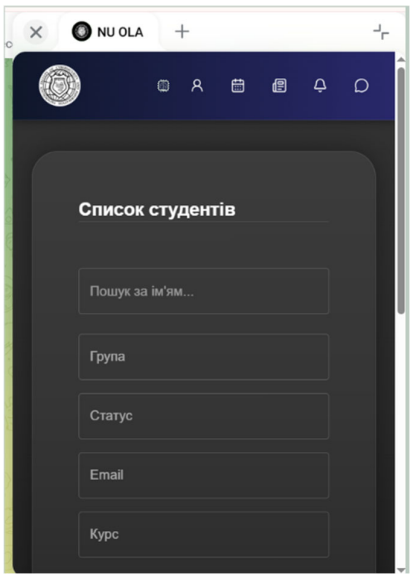


Рисунок 3.8 – Список студентів

3.4.2 Модуль «Керування зверненнями студентів»

Модуль забезпечує ефективну та структуровану обробку запитів, які надходять через модуль «Зворотній зв'язок» від студентів.

- Призначення: Ведення обліку, відстеження статусу та надання офіційних відповідей на звернення.

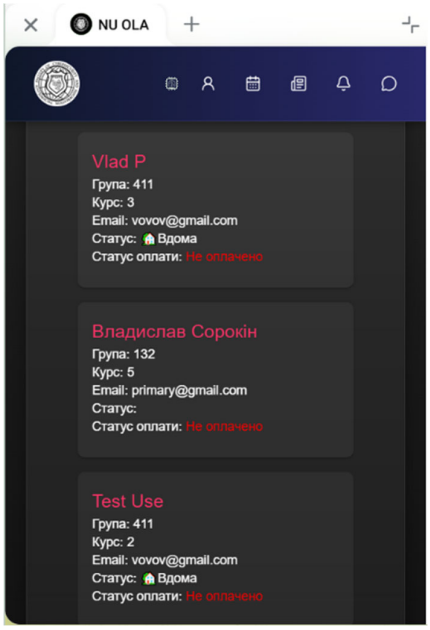


Рисунок 3.9 – Перегляд зареєстрованих користувачів

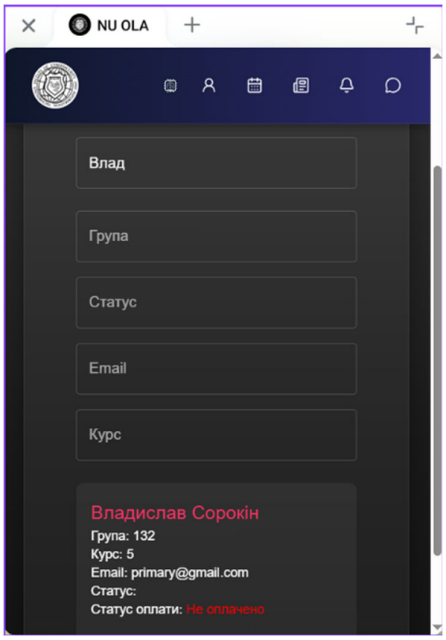


Рисунок 3.10 – Сортування користувачів

- Функціональність:
  - Таблиця звернень: Відображення списку звернень із зазначенням Імені студента, Теми, Дати та Статусу відповіді ("надано" / очікує).

- Відповідь: Можливість відкрити модальне вікно, що містить деталі звернення, і сформувати офіційну відповідь.

- Технічна реалізація: Взаємодія з таблицею feedback. Відповідь адміністратора фіксується у БД, а також, відправляється назад студенту як приватне сповіщення через Telegram-бота.

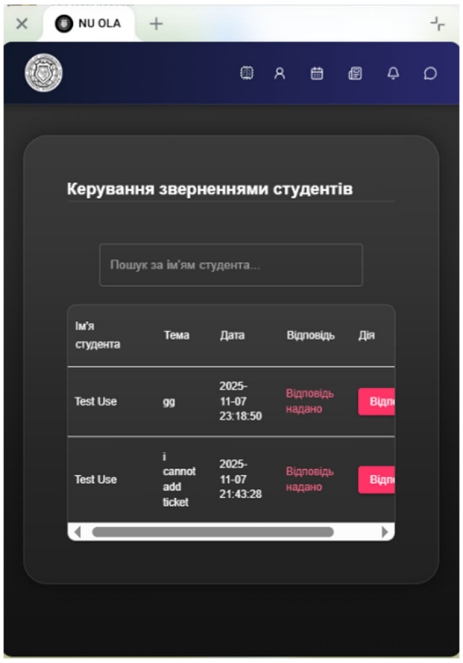


Рисунок 3.11 – Керування зверненнями студентів

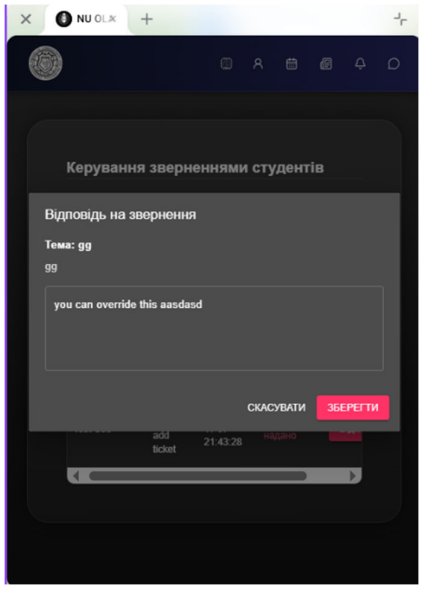


Рисунок 3.12 – Модальне вікно відповіді на звернення

3.4.3 Модуль «Керування контентом»

Цей модуль дозволяє адміністратору підтримувати актуальність інформаційного наповнення системи (новини та розклад).

- Додати новину: Реалізовано форму для створення новинного повідомлення. Додавання Заголовка, Дати, Тексту новини та можливість Завантажити зображення (що є значною перевагою Mini App) дозволяє створювати візуально привабливий контент, який буде відображатися у модулі «Сповіщення» у студентів.
- Редагування розкладу: Адміністратор може вибрати курс і переглянути поточний розклад. Ключова функція – Завантажити новий файл розкладу і Зберегти розклад, замінюючи попередній файл у сховищі. Це забезпечує оперативне оновлення навчальної інформації.

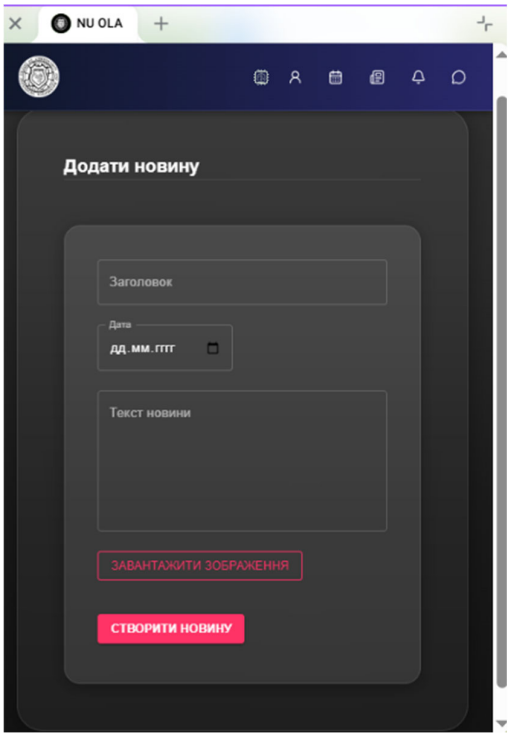


Рисунок 3.13 – Додавання новин

Рисунок 3.14 – Перегляд існуючих новин

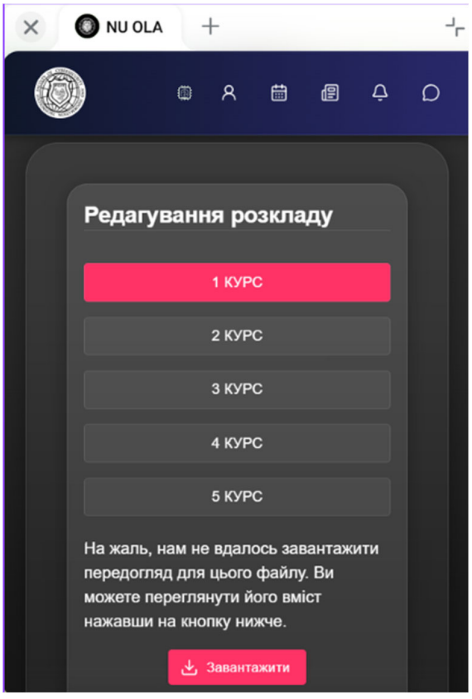


Рисунок 3.15 – Редагування розкладу

3.4.4 Інтеграція штучного інтелекту для аналітичної звітності

Для підвищення ефективності адміністративного контролю та прискорення реакції на проблеми здобувачів, до системи інтегровано модуль аналітичної звітності, який використовує методи машинного навчання. Цей модуль дозволяє перетворити великий обсяг неструктурованого тексту (звернення студентів) на коротку, структуровану та інформативну вижимку.

- Призначення: Автоматизація збору та аналізу даних зі звернень (скарг, побажань, технічних питань), що надходять через модуль «Зворотний зв'язок», для швидкого виявлення системних проблем або тенденцій.
- Інтерфейс: Адміністратор отримує звіт, натиснувши кнопку «СФОРМУВАТИ ЗВІТ». Звіт відображається безпосередньо у Mini App та містить Загальну статистику (кількість звернень, відсоток відповідей) та Деталізацію звернень з автоматично сформованою Суттю проблеми.

Процес формування аналітичного звіту відбувається за наступним алгоритмом:

1. Ініціація запиту: Адміністратор натискає кнопку «СФОРМУВАТИ ЗВІТ» у Mini App.
2. Збір даних: Backend (NestJS) виконує запит до БД, вивантажуючи всі текстові дані (опис проблеми) зі звернень студентів за обраний період.
3. Обробка LLM: Зібраний масив тексту передається в LLM. Модель виконує такі задачі:
    - Резюмування (Summarization): Аналіз тексту кожного звернення та створення короткого, інформативного висновку (Суть).
    - Класифікація: Віднесення звернень до однієї з попередньо визначених категорій (наприклад, "Технічна", "Фінансова", "Загальна").
4. Формування структури: Backend отримує оброблені дані від LLM, додає до них метадані (Ім'я студента, Статус) та формує фінальну JSON-структуру звіту.

5. Візуалізація: Mini App відображає отриману структуровану інформацію у зручному для читання форматі.

Таким чином, інтеграція ШІ дозволяє не лише реєструвати проблеми, а й миттєво виявляти пріоритетні питання, значно прискорюючи управлінські рішення.

Рисунок 3.16 – Використання ШІ для формування звіту

3.5 Висновки до розділу 3

У третьому розділі було успішно реалізовано ключову мету магістерської роботи: трансформацію базового Telegram-бота у повноцінну розподілену інформаційну систему для здобувачів вищої освіти шляхом інтеграції технології Telegram Mini Apps (Web Apps).

Ключові результати, досягнуті в цьому розділі:

1. Обґрунтовано доцільність переходу до Mini Apps, що дозволило подолати обмеження стандартних інтерфейсів Telegram та забезпечило можливість створення складних графічних елементів (форми, таблиці, графічні статуси).
2. Розроблено та описано трирівневу архітектуру системи, що складається з Bot Python (точка входу), Backend NestJS (REST API-сервер для обробки бізнес-логіки) та Frontend Next.js (клієнтський веб-додаток). Це забезпечило високу модульність, масштабованість та безпеку взаємодії.
3. Реалізовано розширений функціонал для здобувача ЗВО, створивши повноцінний Персональний електронний кабінет. Ключовими модулями стали: «Профіль користувача» (для інтерактивного редагування даних), «Оплата навчання» (для візуального відстеження фінансового статусу) та «Зворотний зв'язок» (для структурованої подачі запитів).
4. Створено захищену адміністративну панель, яка дозволяє адміністраторам ефективно керувати даними студентів («Список студентів»), контентом («Додати новину», «Редагування розкладу») та забезпечувати оперативну обробку звернень («Керування зверненнями студентів»).
5. Впроваджено інноваційний аналітичний модуль на основі Великої мовної моделі (LLM). Цей модуль автоматично генерує структуровану вижимку (резюме) зі звернень студентів, що значно прискорює процес ідентифікації системних проблем та прийняття управлінських рішень.

Таким чином, розроблений та впроваджений розширений функціонал повністю відповідає сучасним вимогам до інтерактивних освітніх інформаційних систем.

ВИСНОВКИ

У кваліфікаційній роботі було успішно вирішено актуальну науково-прикладну задачу оптимізації комунікаційних та адміністративних процесів між деканатом та здобувачами вищої освіти в умовах територіальної розпорошеності, спричиненої сучасними викликами. Для цього була розроблена та програмно реалізована інноваційна інформаційна система на базі архітектури Telegram-бота та Mini App.

Розроблена комплексна інформаційна система для здобувачів ЗВО, яка значно розширює функціонал класичного Telegram-бота за рахунок інтеграції графічного інтерфейсу та бекенд-аналізу.

Розробка базувалася на мікросервісній архітектурі, що включає три незалежні компоненти: основний бот-сервіс на Python, RESTful API для фронтенду на Node.js, та клієнтську частину – Telegram Mini App (TMA).

Досягнуто чіткого розподілу відповідальності та підвищено гнучкість системи, забезпечуючи її готовність до подальшого масштабування.

Вперше в межах даного дослідження запропоновано та реалізовано розподілену архітектуру, де фронтенд TMA розгорнутий на Vercel (що забезпечує швидкість CDN), а бекенд-логіка (Python/Node.js) розміщена на Render. Таке розділення є ефективнішим за монолітні рішення, оскільки мінімізує час простою (downtime) та спрощує впровадження оновлень (CI/CD).

Удосконалено підхід до взаємодії адміністратора з системою за допомогою Telegram Mini App (TMA). На відміну від відомих текстових інтерфейсів, TMA дозволяє оперативно керувати зверненнями та формувати структуровані звіти через інтуїтивно зрозумілий графічний інтерфейс.

Створено алгоритм для автоматичного формування аналітичних звітів на запит деканату. Це забезпечує швидку і структуровану вижимку інформації з бази даних SQLite3, перетворюючи сирі дані звернень у готовий аналітичний матеріал для прийняття адміністративних рішень.

Практичне значення отриманих результатів полягає у наданні ЗВО готового, економічно ефективного та надійного інструменту для цифровізації ключових адміністративних процесів. Система забезпечує ефективну комунікацію, зниження адміністративного навантаження та контроль якості.

Для подальшого розвитку та комерційного масштабування системи рекомендуються наступні напрямки досліджень:

- перехід від файлової бази даних SQLite3 до керованої хмарної реляційної СУБД (наприклад, PostgreSQL) для забезпечення стійкості даних, відмовостійкості та горизонтального масштабування;
- впровадження механізмів автоматичної класифікації та маршрутизації звернень студентів на основі машинного навчання, а також вдосконалення алгоритмів генерації звітів для включення прогнозного аналізу (наприклад, прогнозування пікового навантаження на підтримку);
- розробка складніших модулів Mini App для персонального кабінету студента (перегляд оцінок, оплата послуг, формування індивідуального графіку), що вимагатиме подальшого розвитку Node.js API.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Про вищу освіту : Закон України від 01.07.2014 р. № 1556-VII. Поточна редакція від 16.01.2020. URL: https://zakon.rada.gov.ua/laws/show/1556-18.

2. ДСТУ 3008-2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. [Чинний від 2017-07-01]. Київ, 2016. 31 с. URL: http://web.znu.edu.ua/NIS//2017/3659_3008-2015.pdf.

3. ДСТУ ISO 5807:2016. Обробляння інформації. Символи та угоди щодо документації стосовно даних, програм та системних блок-схем... (ISO 5807:1985, IDT). (Використовується для схем в Розділі 2). URL: https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=67202

4. Telegram Bots Platform. *Telegram Bot API Documentation*. URL: https://core.telegram.org/bots/api.

5. Vercel Documentation. *Deployment of Frontend Applications*. URL: https://vercel.com/docs.

6. Render Documentation. *Deploying Web Services (Python/Node.js)*. URL: https://render.com/docs.

7. SQLite. *The Official Website*. URL: https://sqlite.org.

8. Python Software Foundation. *Python documentation*. URL: https://docs.python.org/.

9. Node.js Documentation. *About Node.js*. URL: https://nodejs.org/en/docs.

10. Сорокін В.С., Манаков С.Ю. Розробка системи оперативного інформування студентів та адміністрації засобами месенджера Telegram. Інформаційні технології і автоматизація – 2025 / Матеріали XVIII міжнародної науково-практичної конференції. Одеса, 30-31 жовтня 2025 р. - Одеса, Видавництво ОНТУ, 2025 р. С. 469-471. URL: https://ontu.edu.ua/download/konfi/2025/Collection-of-abstracts-of-the-conference-ITIA-2025.pdf

11. Бова В. Як обрати платформу для створення чат-бота: Telegram чи Viber? *Run-IT*. 2023. URL: https://www.run-it.com.ua/iak-obraty-platformu-dlia-stvorennia-chat-bota-telegram-chy-viber/

12. Is Telegram truly safe? *DOU.ua*. 2023. URL: https://dou.ua/lenta/articles/is-telegram-truly-safe/

13. Програмування баз даних на SQLite. *Foxminded*. 2024. URL: https://foxminded.ua/prohramuvannia-baz-danykh-na-sqlite/

14. Telegram Research Guide. *KR.Labs*. 2024. URL: https://research.kr-labs.com.ua/telegram-research-guide/

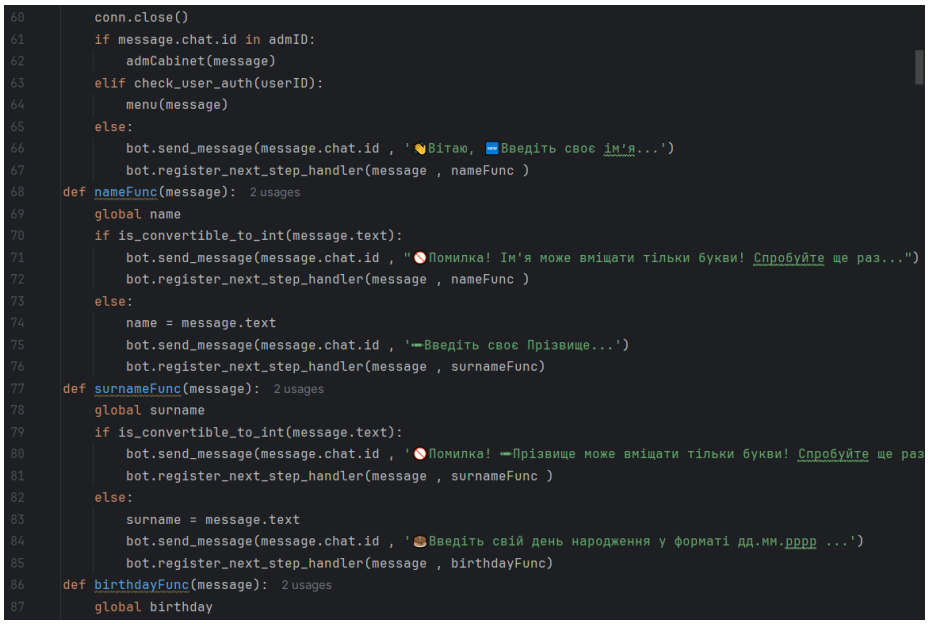
15. Патерни комунікації у мікросервісних архітектурах: синхронні та асинхронні моделі. *Технічний огляд*. 2024. URL: https://blog.conf.vntu.edu.ua/index.php/mn/mn2024/paper/viewFile/19300/16151

ДОДАТКИ

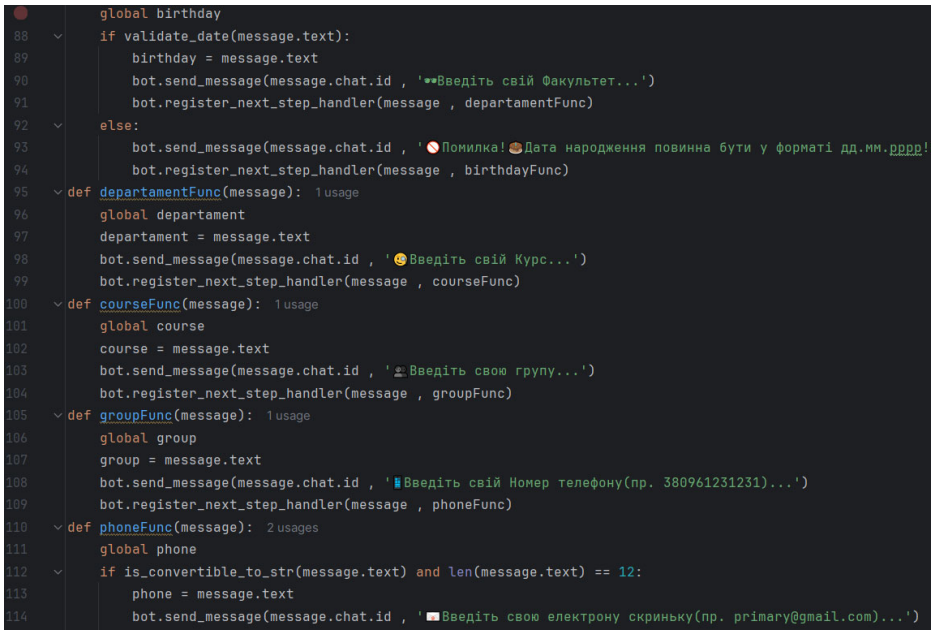
Додаток А. Програмний код

```python
import telebot
from telebot import types
import time
from random import randint
import sqlite3
import re
import asyncio
import os.path
from pathlib import Path
group , phone , email , departament , surname , name , userID , course , birthday  = '' , ''
bot = telebot.TeleBot('7049824274:AAEPN7qLbCZRt3kCbN63aPm_AmQemXdNpmM')
admID = [6113448235]
def is_convertible_to_int(str):
    any(char.isdigit() for char in str)
def is_convertible_to_str(string):
    try:
        int(string)
        return True
    except ValueError:
        return False
def check_user_auth(user_id):
    conn = sqlite3.connect('database.sql')
    cur = conn.cursor()
    cur.execute( sql: 'SELECT userID FROM students WHERE userID = ?',  parameters: (user_id,))
    result = cur.fetchone()
    cur.close()
    conn.close()
    return result is not None
def validate_date(data):
        pattern = r'^\d{2}\.\d{2}\.\d{4}$'  # Паттерн для дати у форматі "дд.мм.рррр"
        if re.match(pattern, data):
            return True
```

```python
            return True
        else:
            return False

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

    web_app_url = f"https://ol-academy-frontend.vercel.app?userId={userID}"
    web_button = types.KeyboardButton(
            text="🌐 Відкрити веб-додаток",
            web_app=types.WebAppInfo(url=web_app_url)
    )

    markup.add(web_button)

    bot.send_message(message.chat.id , '✅Ви авторизувались як адміністратор', reply_markup=markup)
@bot.message_handler(commands = ['start'])
def hello(message):
    global userID
    userID = message.chat.id
    conn = sqlite3.connect('database.sql')
    cur = conn.cursor()
    cur.execute('CREATE TABLE IF NOT EXISTS students (userID int(50) ,name varchar(20) , surname varchar(20)
    cur.close()
    conn.close()
```



```
    conn.close()
    if message.chat.id in admID:
        admCabinet(message)
    elif check_user_auth(userID):
        menu(message)
    else:
        bot.send_message(message.chat.id , '👋Вітаю, 🆕Введіть своє ім'я...')
        bot.register_next_step_handler(message , nameFunc )
def nameFunc(message):  2 usages
    global name
    if is_convertible_to_int(message.text):
        bot.send_message(message.chat.id , "🚫Помилка! Ім'я може вміщати тільки букви! Спробуйте ще раз...")
        bot.register_next_step_handler(message , nameFunc )
    else:
        name = message.text
        bot.send_message(message.chat.id , '➖Введіть своє Прізвище...')
        bot.register_next_step_handler(message , surnameFunc)
def surnameFunc(message):  2 usages
    global surname
    if is_convertible_to_int(message.text):
        bot.send_message(message.chat.id , '🚫Помилка! ➖Прізвище може вміщати тільки букви! Спробуйте ще раз
        bot.register_next_step_handler(message , surnameFunc )
    else:
        surname = message.text
        bot.send_message(message.chat.id , '🎂Введіть свій день народження у форматі дд.мм.рррр ...')
        bot.register_next_step_handler(message , birthdayFunc)
def birthdayFunc(message):  2 usages
    global birthday
```



```
    global birthday
    if validate_date(message.text):
        birthday = message.text
        bot.send_message(message.chat.id , '🏫Введіть свій Факультет...')
        bot.register_next_step_handler(message , departamentFunc)
    else:
        bot.send_message(message.chat.id , '🚫Помилка!🎂Дата народження повинна бути у форматі дд.мм.рррр!
        bot.register_next_step_handler(message , birthdayFunc)
def departamentFunc(message):  1 usage
    global departament
    departament = message.text
    bot.send_message(message.chat.id , '🤓Введіть свій Курс...')
    bot.register_next_step_handler(message , courseFunc)
def courseFunc(message):  1 usage
    global course
    course = message.text
    bot.send_message(message.chat.id , '👥Введіть свою групу...')
    bot.register_next_step_handler(message , groupFunc)
def groupFunc(message):  1 usage
    global group
    group = message.text
    bot.send_message(message.chat.id , '📱Введіть свій Номер телефону(пр. 380961231231)...')
    bot.register_next_step_handler(message , phoneFunc)
def phoneFunc(message):  2 usages
    global phone
    if is_convertible_to_str(message.text) and len(message.text) == 12:
        phone = message.text
        bot.send_message(message.chat.id , '📧Введіть свою електрону скриньку(пр. primary@gmail.com)...')
```

```
        bot.send_message(message.chat.id , '📧Введіть свою електрону скриньку(пр. primary@gmail.com)...')
        bot.register_next_step_handler(message , emailFunc)
    else:
        bot.send_message(message.chat.id , '🚫Помилка! 📱Номер телефону може вміщати тільки цифри (пр. 380961
        bot.register_next_step_handler(message , phoneFunc )
def emailFunc(message):  2 usages
    global email
    if '@' in message.text:
        email = message.text
        insertIntoDB(message)
    else:
        bot.send_message(message.chat.id , "🚫Помилка! 📧Електрона скринька має обов'язково вміщати @! Спробу
        bot.register_next_step_handler(message , emailFunc )
def insertIntoDB(message):  1 usage
    global group , phone , email , departament , surname , name , course , birthday
    student_info = [userID , name , surname , birthday , departament , group , course , phone , email]
    conn = sqlite3.connect('database.sql')
    cur = conn.cursor()
    cur.execute( sql: 'INSERT INTO students (userID , name, surname, birthday , department, studentGroup, cours
    conn.commit()
    cur.close()
    conn.close()
    bot.send_message(message.chat.id , '✅Ви успішно зареєструвались')
    menu(message)
def menu(message):  22 usages
    if message.chat.id in admID:
        admCabinet(message)
    else:
```

```
    else:
        markup = types.ReplyKeyboardMarkup(row_width=2)
        item1 = types.KeyboardButton('👥Моя інформація')
        item2 = types.KeyboardButton('ℹСтатус')
        item3 = types.KeyboardButton('✨Соц. Мережі')
        item4 = types.KeyboardButton('🕐Розклад')
        markup.add(item1, item2, item3, item4)



        web_app_url = f"https://ol-academy-frontend.vercel.app?userId={userID}"
        web_button = types.KeyboardButton(
            text="🌐 Відкрити веб-додаток",
            web_app=types.WebAppInfo(url=web_app_url)
        )

        markup.add(web_button)
        bot.send_message(message.chat.id , '🏞Головне меню' , reply_markup=markup)
@bot.message_handler(func=lambda message: message.text == '👥Моя інформація')
def send_info(message):
    global userID
    userID = message.chat.id
    markup = types.ReplyKeyboardMarkup(row_width=2)
    item1 = types.KeyboardButton('➖Змінити дані')
    item2 = types.KeyboardButton('🚫Назад')
    markup.add(item1 , item2)
    conn = sqlite3.connect('database.sql')
    cur = conn.cursor()
    cur.execute( sql: 'SELECT * FROM students WHERE userID = ?', parameters: (userID,))
```

```
    cur.execute( sql: 'SELECT * FROM students WHERE userID = ?', parameters: (userID,))
    result = cur.fetchone()
    cur.close()
    conn.close()
    name, surname, birthday, department, group, course , phone, email , status = result[1:]
    info_message = f"ℹ️Інформація\n🆕Ім'я: {name}\n➖Прізвище: {surname}\n🎂Дата народження: {birthday}\n👓
    bot.send_message(message.chat.id, info_message, parse_mode='HTML' , reply_markup=markup)
@bot.message_handler(func=lambda message: message.text == '🚫Назад')
def returnToMenu(message):
    menu(message)
@bot.message_handler(func=lambda message: message.text == '➖Змінити дані')
def changeInfo(message):
    markup = types.InlineKeyboardMarkup()
    button1 = types.InlineKeyboardButton("🆕Ім'я", callback_data='name')
    button2 = types.InlineKeyboardButton("➖Прізвище", callback_data='surname')
    button3 = types.InlineKeyboardButton("🎂Дата народження", callback_data='birthday')
    button4 = types.InlineKeyboardButton("👓Факультет", callback_data='departament')
    button5 = types.InlineKeyboardButton("👥Група", callback_data='group')
    button6 = types.InlineKeyboardButton("😉Курс", callback_data='course')
    button7 = types.InlineKeyboardButton("📱Номер телефону", callback_data='phone')
    button8 = types.InlineKeyboardButton("📧Електрона скринька", callback_data='email')
    button9 = types.InlineKeyboardButton("🚫Назад", callback_data='menu')
    markup.add(button1, button2 , button3 , button4 , button5 , button6 , button7 , button8 , button9 )
    bot.send_message(message.chat.id , '➖Оберіть що ви хочете змінити' , reply_markup=markup)
@bot.callback_query_handler(func=lambda call: True)
def callback_query(call):
    markup = types.ReplyKeyboardMarkup(row_width=2)
    item1 = types.KeyboardButton('🚫Назад')
```

```
    item1 = types.KeyboardButton('🚫Назад')
    markup.add(item1)
    if call.data == 'phoneDecanat':
        bot.send_message(call.message.chat.id , 'Номер телефону деканату: <code>+380123123123</code>' , parse
    if call.data == 'menu':
        menu(call.message)
    if call.data == 'name':
        bot.send_message(call.message.chat.id, "Введіть нове 🆕Ім'я...",reply_markup=markup)
    elif call.data == 'surname':
        bot.send_message(call.message.chat.id, "Введіть нове ➖Прізвище...",reply_markup=markup)
    elif call.data == 'departament':
        bot.send_message(call.message.chat.id, "Введіть новий 👓Факультет...",reply_markup=markup)
    elif call.data == 'group':
        bot.send_message(call.message.chat.id, "Введіть нову 👥групу...",reply_markup=markup)
    elif call.data == 'phone':
        bot.send_message(call.message.chat.id, "Введіть новий 📱Номер телефону...",reply_markup=markup)
    elif call.data == 'email':
        bot.send_message(call.message.chat.id, "Введіть нову 📧електрону скриньку...",reply_markup=markup)
    elif call.data == 'course':
        bot.send_message(call.message.chat.id, "Введіть новий 😉Курс...",reply_markup=markup)
    elif call.data == 'birthday':
        bot.send_message(call.message.chat.id, "Введіть нову 🎂дату народження...",reply_markup=markup)
    bot.register_next_step_handler(call.message, insertNewData, call.message.chat.id , call)

def insertNewData(message, chat_id , call):  7 usages
    newData = message.text
    conn = sqlite3.connect('database.sql')
    cur = conn.cursor()
    global userID
```

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

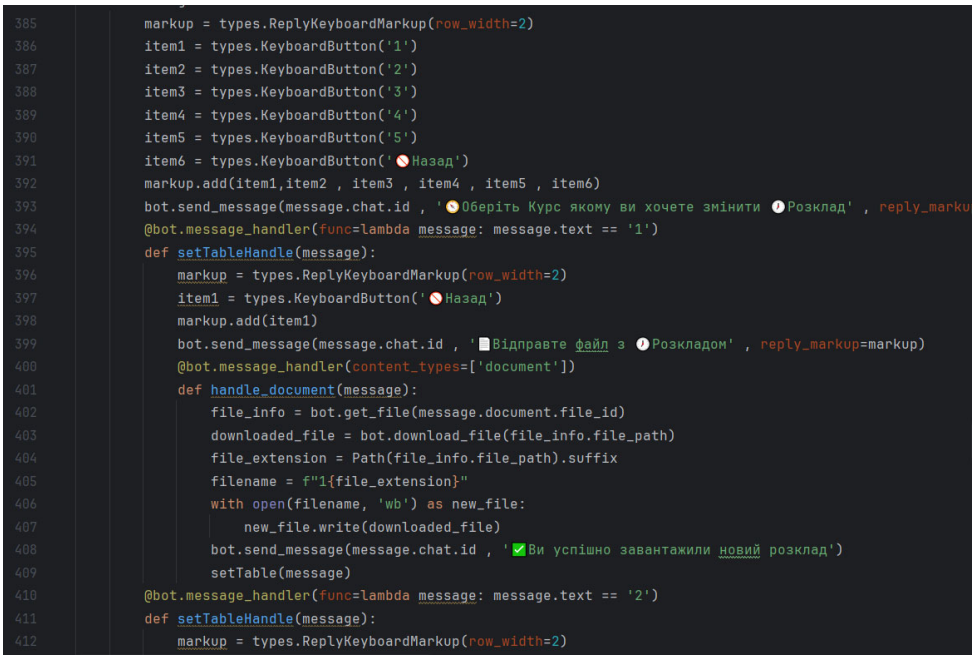
```
            menu(message)
    elif call.data == 'phone':
        if is_convertible_to_str(message.text) and message.text != '🚫Назад' :
            if len(message.text) == 12:
                cur.execute( sql: "UPDATE students SET phoneNumber = ? WHERE userID = ?", parameters: (newData,
                bot.send_message(chat_id, "✅Ви успішно оновили дані!")
                menu(message)
            else:
                bot.send_message(message.chat.id , "🚫Помилка! Номер телефону може вміщати тільки цифри та 12
                bot.register_next_step_handler(message, insertNewData, call.message.chat.id , call )
        elif message.text != '🚫Назад' :
            bot.send_message(message.chat.id , "🚫Помилка! Номер телефону може вміщати тільки цифри та 12 зна
            bot.register_next_step_handler(message, insertNewData, call.message.chat.id , call )
    elif call.data == 'email':
        if '@' in message.text:
            cur.execute( sql: "UPDATE students SET email = ? WHERE userID = ?", parameters: (newData, userID))
            bot.send_message(chat_id, "✅Ви успішно оновили дані!")
            menu(message)
        elif message.text != '🚫Назад':
            bot.send_message(message.chat.id , "🚫Помилка! 📧Електрона скринька має обов'язково вміщати @! Сп
            bot.register_next_step_handler(message , insertNewData, call.message.chat.id , call )
    elif call.data == 'course':
        cur.execute( sql: "UPDATE students SET course = ? WHERE userID = ?", parameters: (newData, userID))
        bot.send_message(chat_id, "✅Ви успішно оновили дані!")
        menu(message)
    elif call.data == 'birthday':
        if validate_date(message.text):
            cur.execute( sql: "UPDATE students SET birthday = ? WHERE userID = ?", parameters: (newData, userID)
```

```
            bot.send_message(chat_id, "✅Ви успішно оновили дані!")
            menu(message)
        else:
            bot.send_message(message.chat.id , '🚫Помилка!💩Дата народження повинна бути у форматі дд.мм.рррр
            bot.register_next_step_handler(message ,insertNewData, call.message.chat.id , call)

    conn.commit()
    cur.close()
    conn.close()
@bot.message_handler(func=lambda message: message.text == '↓Статус')
def status(message):
    global userID
    userID = message.chat.id
    conn = sqlite3.connect('database.sql')
    cur = conn.cursor()
    cur.execute( sql: 'SELECT status FROM students WHERE userID = ?', parameters: (userID,))
    result = cur.fetchone()
    cur.close()
    conn.close()
    if result == (None,):
        markup = types.ReplyKeyboardMarkup(row_width=2)
        item1 = types.KeyboardButton('Змінити ↓Статус')
        item2 = types.KeyboardButton('🚫Назад')
        markup.add(item1 , item2)
        bot.send_message(message.chat.id , 'Ви ще не встановили ↓Статус' , reply_markup=markup)
    else:
        markup =  types.ReplyKeyboardMarkup(row_width=2)
```

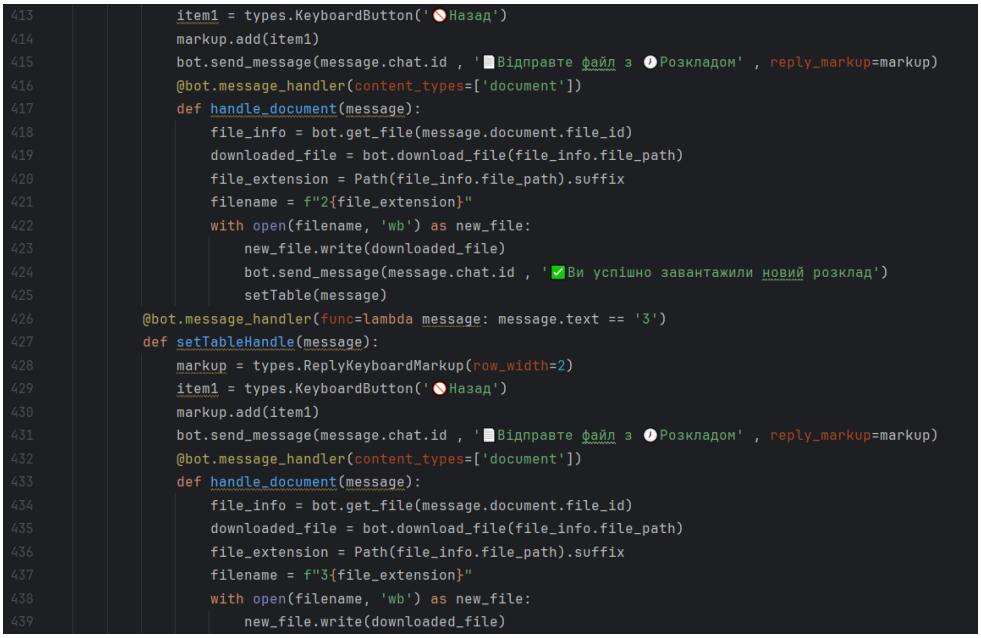
```
        markup =  types.ReplyKeyboardMarkup(row_width=2)
        item1 = types.KeyboardButton('Змінити ↓Статус')
        item2 = types.KeyboardButton('🚫Назад')
        markup.add(item1 , item2)
        result = str(result)
        result = result.replace( _old: "('", _new: "").replace( _old: "',)", _new: "")
        bot.send_message(message.chat.id , f'Ваш ↓Статус:{result}'  , reply_markup=markup)
@bot.message_handler(func=lambda message: message.text == 'Змінити ↓Статус')
def changeStatus(message):
    markup =  types.ReplyKeyboardMarkup(row_width=2)
    item1 = types.KeyboardButton('🏠Вдома')
    item2 = types.KeyboardButton('🤒Хворий')
    item3 = types.KeyboardButton('💼Працюю')
    item4 = types.KeyboardButton('🛥️За кордоном')
    markup.add(item1 , item2 , item3 ,item4)
    bot.send_message(message.chat.id , 'Оберіть один із доступних ↓Статусів'  , reply_markup=markup)
    @bot.message_handler(func=lambda message: message.text == '🏠Вдома')
    def temp(message):setStatus(message , statusStr: '🏠Вдома')
    @bot.message_handler(func=lambda message: message.text == '💼Працюю')
    def temp(message):setStatus(message , statusStr: '💼Працюю')
    @bot.message_handler(func=lambda message: message.text == '🤒Хворий')
    def temp(message):setStatus(message , statusStr: '🤒Хворий')
    @bot.message_handler(func=lambda message: message.text == '🛥️За кордоном')
    def temp(message):setStatus(message , statusStr: '🛥️За кордоном')
def setStatus(message , statusStr):  4 usages
    global userID
    userID = message.chat.id
    conn = sqlite3.connect('database.sql')
```

```
        cur = conn.cursor()
        cur.execute( sql: "UPDATE students SET status = ? WHERE userID = ?", parameters: (statusStr, userID))
        conn.commit()
        cur.close()
        conn.close()
        bot.send_message(message.chat.id , '✅Новий Статус успішно встановлено!')
        menu(message)
    @bot.message_handler(func=lambda message: message.text == '🕐Розклад')
    def selectCourse(message):
        markup =  types.ReplyKeyboardMarkup(row_width=2)
        item1 = types.KeyboardButton('1 Курс')
        item2 = types.KeyboardButton('2 Курс')
        item3 = types.KeyboardButton('3 Курс')
        item4 = types.KeyboardButton('4 Курс')
        item5 = types.KeyboardButton('5 Курс')
        item6 = types.KeyboardButton('🚫Назад')
        markup.add(item1 , item2 , item3 ,item4 , item5 , item6)
        bot.send_message(message.chat.id , 'Оберіть 🧐Курс' , reply_markup=markup)
        @bot.message_handler(func=lambda message: message.text == '1 Курс')
        def temp(message):
            sendFile(message , num: '1')
        @bot.message_handler(func=lambda message: message.text == '2 Курс')
        def temp(message):
            sendFile(message , num: '2')
        @bot.message_handler(func=lambda message: message.text == '3 Курс')
        def temp(message):
            sendFile(message , num: '3')
```

```
            sendFile(message , num: '3')
        @bot.message_handler(func=lambda message: message.text == '4 Курс')
        def temp(message):
            sendFile(message , num: '4')
        @bot.message_handler(func=lambda message: message.text == '5 Курс')
        def temp(message):
            sendFile(message , num: '5')
        def sendFile(message , num):
            suffixes = ["txt", "pdf", "docx" , "csv" , "xls" , "xlsx", "png"]
            file_found = False
            for suffix in suffixes :
                print(num , suffix)
                try:
                    with open(f"{num}.{suffix}", 'rb') as file:
                        bot.send_document(message.chat.id, file)
                        file_found = True
                        menu(message)

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

    @bot.message_handler(func=lambda message: message.text == '🕐Встановити розклад')  5 usages
    def setTable(message):
        if message.chat.id in admID:
            markup = types.ReplyKeyboardMarkup(row_width=2)
```



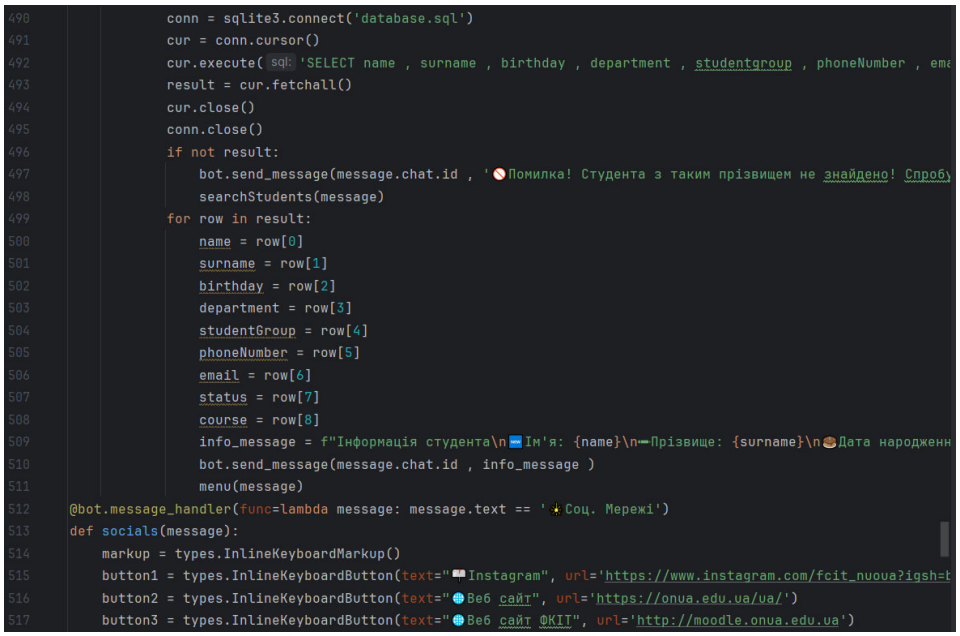
```
    markup = types.ReplyKeyboardMarkup(row_width=2)
    item1 = types.KeyboardButton('1')
    item2 = types.KeyboardButton('2')
    item3 = types.KeyboardButton('3')
    item4 = types.KeyboardButton('4')
    item5 = types.KeyboardButton('5')
    item6 = types.KeyboardButton('🚫Назад')
    markup.add(item1,item2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



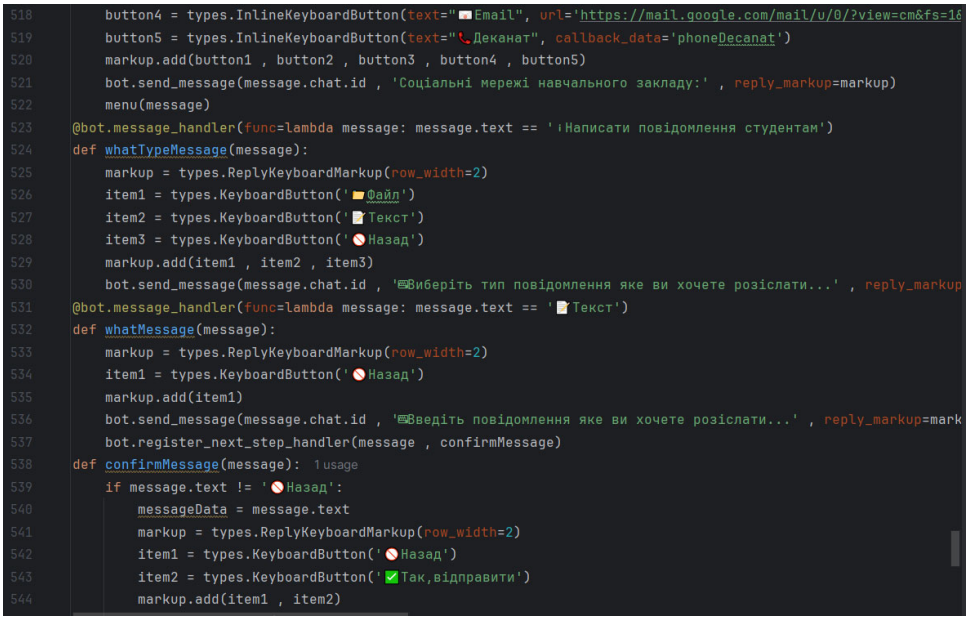
```
        item1 = types.KeyboardButton('🚫Назад')
        markup.add(item1)
        bot.send_message(message.chat.id , '📄Відправте файл з 🕐Розкладом' , reply_markup=markup)
        @bot.message_handler(content_types=['document'])
        def handle_document(message):
            file_info = bot.get_file(message.document.file_id)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

```
                    bot.send_message(message.chat.id , '✅Ви успішно завантажили новий розклад')
                    setTable(message)
        @bot.message_handler(func=lambda message: message.text == '4')
        def setTableHandle(message):
            markup = types.ReplyKeyboardMarkup(row_width=2)
            item1 = types.KeyboardButton('🚫Назад')
            markup.add(item1)
            bot.send_message(message.chat.id , '📄Відправте файл з 🕐Розкладом' , reply_markup=markup)
            @bot.message_handler(content_types=['document'])
            def handle_document(message):
                file_info = bot.get_file(message.document.file_id)
                downloaded_file = bot.download_file(file_info.file_path)
                file_extension = Path(file_info.file_path).suffix
                filename = f"4{file_extension}"
                with open(filename, 'wb') as new_file:
                    new_file.write(downloaded_file)
                    bot.send_message(message.chat.id , '✅Ви успішно завантажили новий розклад')
                    setTable(message)
        @bot.message_handler(func=lambda message: message.text == '5')
        def setTableHandle(message):
            markup = types.ReplyKeyboardMarkup(row_width=2)
            item1 = types.KeyboardButton('🚫Назад')
            markup.add(item1)
            bot.send_message(message.chat.id , '📄Відправте файл з 🕐Розкладом' , reply_markup=markup)
```

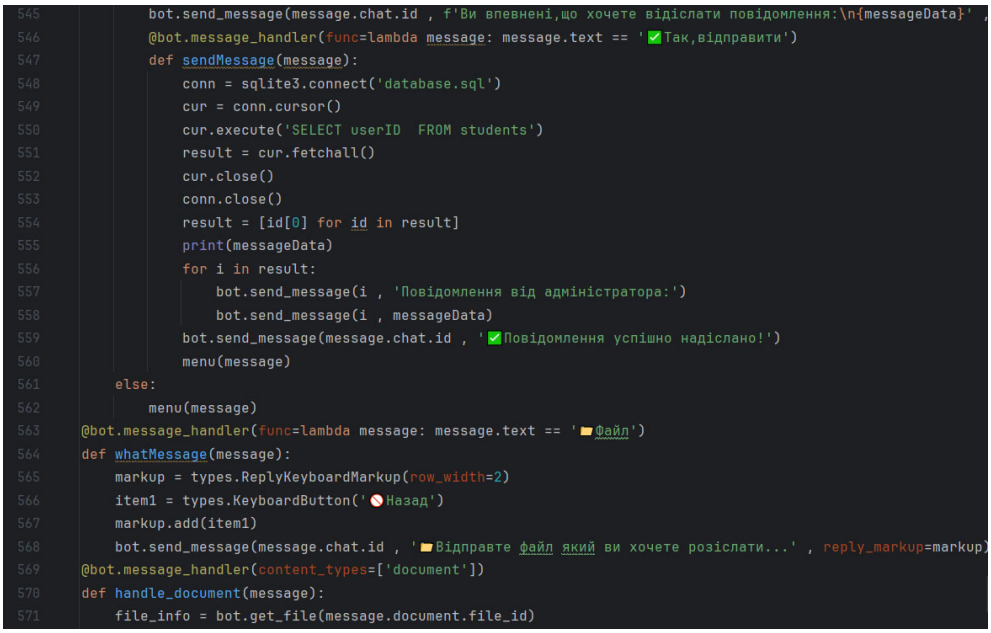
```
            @bot.message_handler(content_types=['document'])
            def handle_document(message):
                file_info = bot.get_file(message.document.file_id)
                downloaded_file = bot.download_file(file_info.file_path)
                file_extension = Path(file_info.file_path).suffix
                filename = f"5{file_extension}"
                with open(filename, 'wb') as new_file:
                    new_file.write(downloaded_file)
                    bot.send_message(message.chat.id , '✅Ви успішно завантажили новий розклад')
                    setTable(message)
    else:
        bot.send_message(message.chat.id , '🚫Вам відмовлено в доступі')
        menu(message)
@bot.message_handler(func=lambda message: message.text == '🔍Пошук студентів')  1 usage
def searchStudents(message):
    if message.chat.id in admID:
        markup = types.ReplyKeyboardMarkup(row_width=2)
        item1 = types.KeyboardButton('🚫Назад')
        markup.add(item1)
        bot.send_message(message.chat.id , 'Введіть ➖Прізвище студента...' , reply_markup=markup)
        bot.register_next_step_handler(message, searchStudentsHandle)
    else:
        bot.send_message(message.chat.id , '🚫Вам відмовлено в доступі')
        menu(message)
def searchStudentsHandle(message):  1 usage
    if message.text != '🚫Назад':
            conn = sqlite3.connect('database.sql')
```



```
            conn = sqlite3.connect('database.sql')
            cur = conn.cursor()
            cur.execute( sql: 'SELECT name , surname , birthday , department , studentgroup , phoneNumber , ema
            result = cur.fetchall()
            cur.close()
            conn.close()
            if not result:
                bot.send_message(message.chat.id , '🚫Помилка! Студента з таким прізвищем не знайдено! Спробу
                searchStudents(message)
            for row in result:
                name = row[0]
                surname = row[1]
                birthday = row[2]
                department = row[3]
                studentGroup = row[4]
                phoneNumber = row[5]
                email = row[6]
                status = row[7]
                course = row[8]
                info_message = f"Інформація студента\n🆔Ім'я: {name}\n➖Прізвище: {surname}\n🍩Дата народженн
                bot.send_message(message.chat.id , info_message )
                menu(message)
@bot.message_handler(func=lambda message: message.text == '✨Соц. Мережі')
def socials(message):
    markup = types.InlineKeyboardMarkup()
    button1 = types.InlineKeyboardButton(text="📱Instagram", url='https://www.instagram.com/fcit_nuoua?igsh=t
    button2 = types.InlineKeyboardButton(text="🌐Веб сайт", url='https://onua.edu.ua/ua/')
    button3 = types.InlineKeyboardButton(text="🌐Веб сайт ФКІТ", url='http://moodle.onua.edu.ua')
```



```
    button4 = types.InlineKeyboardButton(text="📩Email", url='https://mail.google.com/mail/u/0/?view=cm&fs=1&
    button5 = types.InlineKeyboardButton(text="📞Деканат", callback_data='phoneDecanat')
    markup.add(button1 , button2 , button3 , button4 , button5)
    bot.send_message(message.chat.id , 'Соціальні мережі навчального закладу:' , reply_markup=markup)
    menu(message)
@bot.message_handler(func=lambda message: message.text == '↓Написати повідомлення студентам')
def whatTypeMessage(message):
    markup = types.ReplyKeyboardMarkup(row_width=2)
    item1 = types.KeyboardButton('📁Файл')
    item2 = types.KeyboardButton('📝Текст')
    item3 = types.KeyboardButton('🚫Назад')
    markup.add(item1 , item2 , item3)
    bot.send_message(message.chat.id , '⌨Виберіть тип повідомлення яке ви хочете розіслати...' , reply_markup
@bot.message_handler(func=lambda message: message.text == '📝Текст')
def whatMessage(message):
    markup = types.ReplyKeyboardMarkup(row_width=2)
    item1 = types.KeyboardButton('🚫Назад')
    markup.add(item1)
    bot.send_message(message.chat.id , '⌨Введіть повідомлення яке ви хочете розіслати...' , reply_markup=mark
    bot.register_next_step_handler(message , confirmMessage)
def confirmMessage(message):  1 usage
    if message.text != '🚫Назад':
        messageData = message.text
        markup = types.ReplyKeyboardMarkup(row_width=2)
        item1 = types.KeyboardButton('🚫Назад')
        item2 = types.KeyboardButton('✅Так,відправити')
        markup.add(item1 , item2)
```



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

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

Додаток Б. Копії матеріалів апробації

УДК 004.588

РОЗРОБКА СИСТЕМИ ОПЕРАТИВНОГО ІНФОРМУВАННЯ СТУДЕНТІВ ТА АДМІНІСТРАЦІЇ ЗАСОБАМИ МЕСЕНДЖЕРА TELEGRAM

Сорокін В. С., Манаков С.Ю.
Національний університет «Одеська юридична академія» (Україна)

1. Актуальність дослідження та постановка проблеми

Сучасний освітній простір вимагає високої оперативності та ефективності комунікації між адміністративним апаратом і здобувачами освіти [1]. Впровадження цифрових інструментів для оптимізації комунікаційних процесів є ключовим напрямом діджиталізації вищої освіти в Україні [2]. Проте, існуюча практика інформування у багатьох закладах вищої освіти (ЗВО) залишається повільною та багатоетапною.

Основні проблеми, що вирішуються:

- Неоперативність інертність: Критично важливі зміни (перенесення занять, зміни в розкладі, терміни подання документів) часто доводяться до студентів із запізненням через необхідність оновлення офіційного сайту та нерегулярний перегляд електронної пошти.
- Інформаційний хаос: Інформація є децентралізованою та розсіяною між різними джерелами (сайт, внутрішні портали, неформальні групи), що ускладнює її верифікацію та використання.
- Високе адміністративне навантаження: Співробітники деканатів витрачають надмірний час на рутинні операції, що призводить до зниження загальної ефективності управління.

Мета роботи — розробити та реалізувати централізовану, мобільно-орієнтовану та оперативну інформаційну систему, здатну мінімізувати комунікаційні бар'єри.

2. Аналіз існуючих рішень та переваги пропонованої системи

Традиційні академічні інформаційні системи (АІС) та системи управління навчанням (LMS) орієнтовані на складні внутрішні процеси і не завжди забезпечують високу швидкість та зручність мобільного доступу до оперативних даних. Неформальні комунікації через месенджери є швидкими, але несуть ризики дезінформації та не мають інтеграції з офіційною базою даних університету.

Пропонована система на базі Telegram Bot та Mini App є кращою за існуючі методи, оскільки вирішує проблему компромісу між швидкістю та достовірністю:

- **Достовірність**: Інтеграція з централізованою Базою Даних (БД) університету забезпечує актуальність усіх даних (розклад, профіль, оголошення).
- **Оперативність**: Використання механізму push-сповіщень Telegram забезпечує миттєве доведення критичної інформації.

- **Зручність (UX)**: Застосування Telegram Mini App дозволяє замінити незручні текстові команди бота на повноцінний, графічний і візуально привабливий інтерфейс [3].

Таким чином, розроблений програмний комплекс поєднує надійність офіційного джерела з гнучкістю та доступністю платформи, якою студенти користуються щоденно.

3. Архітектура та функціонал пропонованого рішення

Система є дворівневим програмним комплексом, що функціонує як зв'язок між клієнтською частиною (Telegram) і серверною частиною.

3.1 Технічна реалізація та архітектура

Система складається з чотирьох основних компонентів:

- **Telegram Bot API**: Відповідає за обробку вхідних запитів та миттєве відправлення сповіщень (навіть коли користувач не в мережі).
- **Серверний додаток** (Backend): Реалізує бізнес-логіку системи, авторизацію (через унікальний ID користувача Telegram) та управління правами доступу (студент/адміністратор).
- **База Даних** (БД): Централізоване сховище для всіх даних. Дана архітектура гарантує, що зміни, внесені адміністратором, миттєво стають доступними для студентів через інтерфейс.
- **Telegram Mini App**: Реалізовано як вбудований веб-додаток. Він служить основним інтерфейсом для студентів, надаючи зручний доступ до:
  a) **Актуального розкладу** занять.
  b) **Особистого академічного профілю** (інформація про студента, група).
  c) **Списку минулих та поточних сповіщень**.

3.2 Ключовий функціонал

Система забезпечує двосторонній зв'язок та управління інформацією.

Для адміністрації (Деканату):

- **Таргетована розсилка**: Функціонал для швидкого створення та відправки цільових повідомлень (SMS/сповіщень) обраній аудиторії (курс, група, усі студенти), що значно знижує інформаційний шум.
- **Оперативне управління розкладом**: Інтерфейс для швидкого внесення змін у розклад.

Для студентів:

- **Миттєве інформування**: Отримання push-сповіщень про термінові оголошення.
- **Зручний доступ**: Міні-додаток спрощує навігацію та візуалізацію складних даних (розклад, оцінки).

Висновки

Розроблена система оперативного інформування на базі технологій Telegram вирішує ключову проблему неефективності комунікацій у сфері вищої освіти, запропоновану в розділі 1. Інтеграція Telegram Bot та Mini App забезпечила створення інструменту, який є одночасно швидким, достовірним та зручним у користуванні.

Практична цінність системи полягає у:

1. Зниженні адміністративного навантаження на співробітників деканату за рахунок автоматизації рутинних операцій.
2. Підвищенні рівня інформованості студентів та мінімізації пропусків занять через своєчасне доведення критичних даних.
3. Демонстрації можливостей використання мобільних платформ для створення офіційних, надійних та привабливих для молоді каналів комунікації у ЗВО.

Таким чином, результати дослідження можуть бути рекомендовані до впровадження як ефективна складова цифрової трансформації університетського управління.

2. Міністерство освіти і науки України. Стратегія розвитку вищої освіти України до 2030 року. — К.: МОН, 2021. — 78 с. https://zakon.rada.gov.ua/laws/show/286-2022-%D1%80#Text
3. Ткаченко І.О. Інноваційні підходи до розробки користувацьких інтерфейсів мобільних додатків // Технологічні науки. — 2022. — Т. 12, №3. — С. 45-50. https://essuir.sumdu.edu.ua/server/api/core/bitstreams/53040424-7c38-4982-aba1-ef4a152a0ce6/content