Міністерство освіти і науки України

Луцький національний технічний університет
(повне найменування закладу вищої освіти)

Факультет комп'ютерних та інформаційних технологій
(повне найменування факультету)

Кафедра комп'ютерної інженерії та безпеки
(повне найменування кафедри)

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

ЦЕНТР ДОВІДКОВОЇ ІНФОРМАЦІЇ СТУДЕНТА З ВИКОРИСТАННЯМ RASPBERRY PI ТА PYTHON

STUDENT HELP CENTER USING RASPBERRY PI AND PYTHON

спеціальність 123 Комп'ютерна інженерія
(шифр і назва спеціальності)

освітня програма Комп'ютерна інженерія
(назва освітньої програми)

Виконав: здобувач вищої освіти
групи КІс-21
Грицюк Назар Олександрович

(підпис)

Керівник:
асистент
Конкевич Людмила Миколаївна

(підпис)

Кваліфікаційну роботу
допущено до захисту
« 10 » червня 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Лавренчук Світлана Василівна

(підпис)

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерної інженерії та безпеки*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *123 Комп'ютерна інженерія*
Освітня програма: *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ доц. Тарас ТЕРЛЕЦЬКИЙ
« 10 » 01 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Грицюк Назар Олександрович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Центр довідкової інформації студента з використанням Raspberry Pi та Python*

Керівник роботи *асистент Конкевич Людмила Миколаївна*

затверджені наказом закладу вищої освіти від «04» січня 2025 року № 11/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи 10.06.2025р.

3. Вихідні дані до роботи *джерелом розробки є науково-технічна література та публікації в періодичних виданнях з даного питання, опубліковані зарубіжні та вітчизняні роботи в даній області та різні інтернет-ресурси технічного спрямування*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Вступ
Аналіз предметної області
Проєктування системи
Реалізація та тестування
Висновки

5. Перелік графічного (ілюстративного) матеріалу:

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області та наявних рішень	*Конкевич Л.М., асистент*		
Проєктування системи	*Конкевич Л.М., асистент*		
Реалізація та тестування системи	*Конкевич Л.М., асистент*		
Нормоконтроль	*Багнюк Н.В., доцент*		
Гарант ОП	*Лавренчук С.В., доцент*		
Показник запозичень тексту	***___%***		
Академічна доброчесність	*Міскевич О.І.,ст.викладач*		

7. Дата видачі завдання 10.01.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	*Огляд літератури із досліджуваної проблеми, аналіз предметної області та наявних рішень*	до 10.02.2025 р.	Виконано
2.	*Проєктування системи*	до 02.03.2025 р.	Виконано
3.	*Реалізація та тестування системи*	до 02.04.2025 р.	Виконано
4.	*Висновки та пропозиції*	до 10.04.2025 р.	Виконано
5.	*Формування списку використаних джерел*	до 15.04.2025 р.	Виконано
6.	*Формування додатків*	до 02.05.2025 р.	Виконано
7.	*Оформлення ілюстративного матеріалу*	до 10.05.2025 р.	Виконано
8.	*Представлення остаточного варіанту кваліфікаційної роботи бакалавра керівникові*	до 15.05.2025 р.	Виконано
9.	*Нормоконтроль*	до 30.05.2025 р.	Виконано
10	*Інструментальна перевірка на академічний плагіат*	до 03.06.2025 р	Виконано
11.	*Здача кваліфікаційної роботи та всіх супровідних документів на кафедру*	до 10.06.2025 р.	Виконано

Здобувач вищої освіти _____ (підпис) Грицюк Н.О. (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ (підпис) Конкевич Л.М. (прізвище, ініціали)

АНОТАЦІЯ

Грицюк Н.О. Центр довідкової інформації студента з використанням Raspberry Pi та Python. Рукопис.

Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр» за освітньою програмою «Комп'ютерна інженерія» спеціальності 123 Комп'ютерна інженерія. Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатку.

У першому розділі подано огляд предметної області, обґрунтовано актуальність розробки інформаційного центру для студентів, розглянуто використання месенджерів в освітньому середовищі та проаналізовано існуючі рішення в цій галузі, зокрема інформаційні системи на основі чат-ботів.

Другий розділ присвячений вибору апаратних та програмних засобів. Розглянуто архітектуру мікрокомп'ютера Raspberry Pi, мову програмування Python, фреймворк Aiogram для розробки Telegram-бота, веб-фреймворк Flask, систему управління базами даних SQLite та платформу для контейнеризації сервісів Docker.

У третьому розділі описано процес розробки Telegram-бота, реалізацію модульної архітектури системи, створення веб-інтерфейсу адміністрування, тестування системи, а також можливі напрямки подальшого вдосконалення та адаптації для інших навчальних закладів.

Ключові слова: Telegram-бот, Raspberry Pi, Python, Aiogram Flask, Docker.

ANNOTATION

Hrytsiuk N. Student Help Center using Raspberry Pi and Python. Manuscript.

Bachelor's qualification thesis for the Educational Program «Computer Engineering» of specialty 123 Computer Engineering. Lutsk National Technical University. Lutsk, 2025.

The qualification thesis consists of an introduction, three chapters, conclusions, a list of references, and an appendix.

The first chapter provides an overview of the subject area, substantiates the relevance of developing an information center for students, examines the use of messengers in the educational environment, and analyzes existing solutions in the field, particularly chatbot-based information systems.

The second chapter focuses on the selection of hardware and software tools. It covers the architecture of the Raspberry Pi microcomputer, the Python programming language, the Aiogram framework for Telegram bot development, the Flask web framework, the SQLite database management system, and the Docker platform for service containerization.

The third chapter describes the development process of the Telegram bot, the implementation of a modular system architecture, the creation of a web-based administration interface, system testing, and possible directions for future improvements and adaptation for other educational institutions.

Keywords: Telegram bot, Raspberry Pi, Python, Aiogram Flask, Docker.

ЗМІСТ

ВСТУП

У сучасному освітньому середовищі питання ефективного доступу студентів до довідкової інформації набуває особливої актуальності. Зростання обсягу навчальних матеріалів, зміни в розкладах, наявність важливої адміністративної інформації потребують оперативного інструменту, який дозволяє студентам швидко отримувати відповіді на типові запити. У зв'язку з цим виникає потреба в розробці автоматизованої системи, яка б надавала студентам простий, швидкий і доступний канал отримання довідкової інформації.

Метою кваліфікаційної роботи є розробка Центру довідкової інформації студента на базі Telegram бота, який функціонує на Raspberry Pi та реалізований з використанням Python. Така система дозволяє студентам отримувати необхідну інформацію безпосередньо у своєму смартфоні через зручний та звичний інтерфейс.

Об'єктом дослідження є інформаційні системи для підтримки студентів у закладах вищої освіти. Предметом дослідження є Telegram бот з інформаційною базою, реалізований на Raspberry Pi із застосуванням Python.

Для досягнення поставленої мети в роботі передбачено виконання таких завдань:

– реалізувати Telegram бота, який дозволяє студентам отримувати довідкову інформацію щодо навчальних корпусів, аудиторій, викладачів і контактів за допомогою інтерактивного меню в месенджері Telegram;

– розробити інтерфейс адміністратора на базі веб-фреймворку Flask, що забезпечує можливість керування довідковою інформацією: додавання, редагування та видалення записів через веб-інтерфейс із базовою автентифікацією;

– створити структуру бази даних у системі управління базами даних SQLite для зберігання довідкових матеріалів у вигляді пов'язаних між собою таблиць із забезпеченням логічної цілісності даних;

– розгорнути систему на Raspberry Pi, використовуючи платформу Docker та Docker Compose для контейнеризації сервісів, що забезпечує ізоляцію, зручність обслуговування та портативність рішень;

– провести тестування функціональності системи, включаючи функціональні, навантажувальні та юзабіліті-випробування, з метою перевірки стабільності, продуктивності та зручності використання Telegram бота та веб-інтерфейсу.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд аналогічних систем або рішень

У багатьох закладах вищої освіти впроваджуються цифрові системи для забезпечення студентів довідковою інформацією. Серед них можна виділити мобільні додатки, офіційні вебпортали, чат-боти у месенджерах та спеціалізовані платформи для розкладу занять. Однак ефективність та зручність використання цих інструментів значно варіюється.

Одним із найпоширеніших рішень є мобільні додатки університетів, які дозволяють студентам переглядати розклад, знаходити аудиторії, отримувати сповіщення про зміни. Так, у Луцькому національному технічному університеті використовується додаток [9] uSched, що був розроблений із підтримкою адміністрації. Цей додаток [9] дає змогу переглядати розклад занять, здійснювати пошук за групами, предметами або викладачами, а також зберігати улюблені розклади для швидкого доступу. Важливою функцією стала можливість перегляду плану корпусів з інтерактивною позначкою аудиторій, що значно полегшує орієнтування в університеті (рис. 1.1).

Незважаючи на очевидну користь, такі додатки мають кілька недоліків. По-перше, студент повинен завантажити окремий застосунок, що не завжди зручно. По-друге, підтримка додатка залежить від розробників: при їх відсутності застосунок швидко застаріває. Крім того, оновлення контенту може потребувати втручання технічного персоналу, що ускладнює адміністрування.

Ще одним типом рішень є вебпортали університетів. Вони часто містять необхідну інформацію, однак інтерфейс не завжди зручний для мобільних пристроїв. Студенту доводиться переходити по багатьох сторінках, щоб знайти потрібну інформацію, що не відповідає сучасним вимогам швидкості доступу. Крім того, через перевантаженість таких сайтів функціональність часто втрачається під час пікових навантажень.

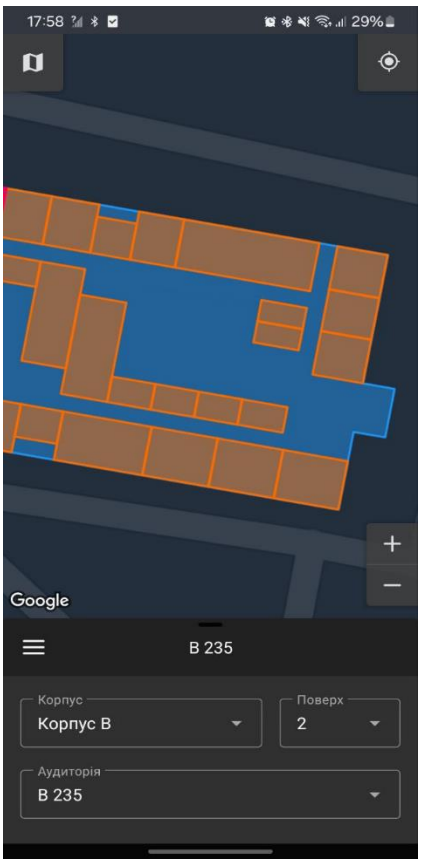

Рисунок 1.1 – Зображення інтерфейсу мобільного додатку uSched з мапою аудиторій [9]

Більш гнучким і сучасним підходом є використання чат-ботів у Telegram [12], що дозволяють отримувати інформацію безпосередньо в месенджері, яким користуються більшість студентів. Такі боти забезпечують швидку взаємодію, не вимагають встановлення додаткового програмного забезпечення та можуть працювати цілодобово. Наприклад, у деяких українських університетах уже реалізовані боти, які відповідають на запити типу «розклад», «викладачі», «контакти». Проте більшість із них не підтримують гнучкого оновлення бази даних або потребують складної серверної інфраструктури.

Таким чином, більшість існуючих рішень мають або обмежену функціональність, або складну структуру адміністрування. Запропонована ж у роботі система базується на Telegram-боті, що працює на мінікомп’ютері Raspberry Pi з використанням Python та SQLite [11, 10]. Це рішення забезпечує:

– доступність (бот працює без встановлення окремих додатків);

– гнучкість (інформація оновлюється через адмін-панель);

– мінімальні витрати (завдяки використанню відкритого ПЗ і недорогого апаратного забезпечення);

– швидкий відгук (інтерфейс працює через звичний Telegram [12]).

Така система забезпечує високу автономність, низький поріг входу для адміністрування і зручність для студентів, що робить її ефективною альтернативою іншим інструментам

1.2 Розгляд технологій

Для реалізації центру довідкової інформації студента обрано поєднання перевірених і водночас сучасних технологій, які забезпечують стабільну роботу, простоту обслуговування та масштабованість. Основу реалізації складають: Aiogram, Raspberry Pi, SQLite, Flask [4] та Docker.

Aiogram – це сучасний асинхронний фреймворк для створення Telegram-ботів на Python. Він побудований на основі модуля asyncio і дозволяє ефективно працювати з великою кількістю одночасних запитів без блокування виконання програми [5]. На відміну від використання Telegram Bot API напряму, Aiogram забезпечує більш високу гнучкість та зручний синтаксис для маршрутизації команд, обробки кнопок, побудови FSM (finite state machines) тощо. У нашому випадку бот має лише дві текстові команди: /start та /help, а вся подальша взаємодія з користувачем відбувається через меню-кнопки, побудоване за допомогою ReplyKeyboardMarkup та InlineKeyboardMarkup. Асинхронна модель Aiogram дозволяє обробляти десятки запитів паралельно, що критично важливо при використанні на обмеженій апаратній платформі, такій як Raspberry Pi [10]. Крім того, Aiogram добре документований і має активну спільноту, що полегшує впровадження нових функцій [1, 5].

Raspberry Pi – це мініатюрний одноплатний комп’ютер, який підтримує повноцінну операційну систему на базі Linux (зокрема, Raspberry Pi OS). У даному проєкті використовується модель Raspberry Pi 4, яка має чотириядерний

процесор та обсяг оперативної пам’яті до 8 ГБ. Цього достатньо для паралельного запуску Telegram-бота, веб-сервера та бази даних. Головною перевагою є компактність пристрою, низьке енергоспоживання і можливість працювати 24/7 без потреби у системі охолодження. Raspberry Pi чудово підходить для розгортання локального інформаційного сервісу з мінімальними витратами (рис. 1.2) [10].

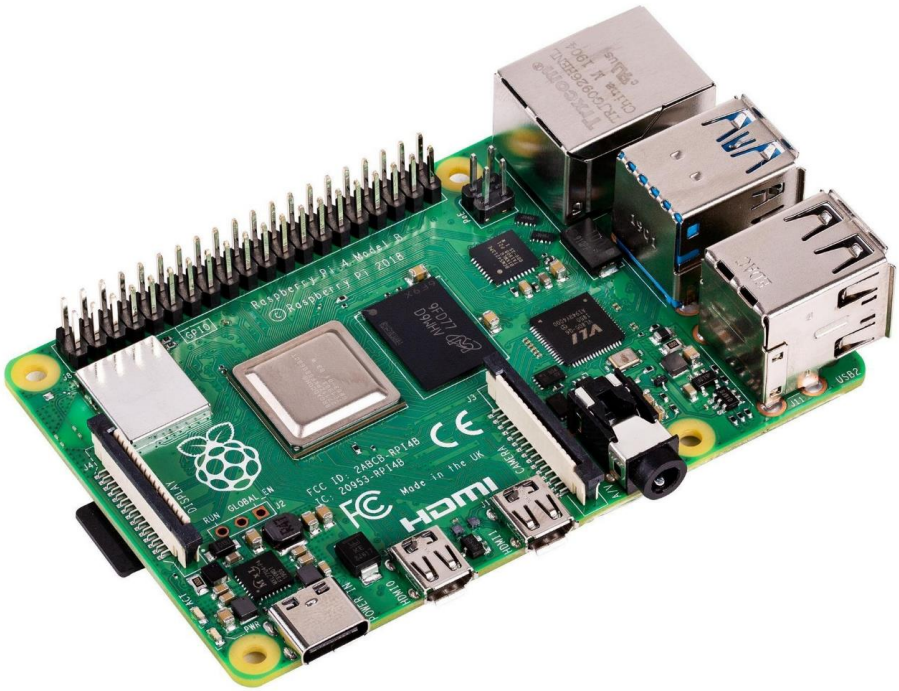

Рисунок 1.2 – Raspberry Pi 4 [10]

SQLite – це вбудована СУБД, яка не вимагає окремого серверного середовища. Вся база даних зберігається у вигляді одного файлу, що значно спрощує адміністрування [11]. Для нашої системи, де кількість транзакцій невелика, а зміна даних відбувається рідко, SQLite є ідеальним рішенням. Через стандартний модуль sqlite3, який постачається з Python, можна виконувати всі необхідні SQL-операції: читання, додавання, редагування та видалення записів. Структура бази включає таблиці: buildings (корпуси), rooms (аудиторії), teachers (викладачі), contacts (контактні дані) та logs (історія змін). Це дозволяє швидко знаходити потрібну інформацію за запитом користувача або змінювати її через адмін-панель [11].

Flask – мікрофреймворк на Python, що дозволяє швидко розгорнути веб-інтерфейс або REST API [1]. У нашому проєкті він використовується для створення адмін-панелі, через яку можна керувати довідковою інформацією: додавати нові аудиторії, викладачів, контакти. Flask відомий своєю простотою, для запуску повноцінного веб-сервісу достатньо кількох десятків рядків коду. Підтримка шаблонізатора Jinja2 дозволяє створювати динамічні сторінки, а інтеграція з ORM-бібліотекою SQLAlchemy спрощує роботу з базою даних. Крім того, Flask легко адаптується під різні пристрої, тож адмінка буде доступна як з ноутбука, так і з мобільного пристрою [11].

Docker – ізоляція середовища та зручне розгортання. Щоб забезпечити зручне розгортання проєкту та уникнути проблем із сумісністю залежностей, використовується технологія контейнеризації Docker. Усі компоненти системи Telegram-бот, Flask-додаток , база даних SQLite – ізольовані в окремих Docker контейнерах [6, 11]. Це дозволяє:

– зберігати однакове середовище як у розробці, так і на продакшн-сервері;

– швидко запускати або зупиняти систему кількома командами (docker-compose up -d);

– оновлювати компоненти без порушення роботи інших;

– переносити проєкт на інший Raspberry Pi або сервер без складної переустановки [10].

Для управління декількома контейнерами одночасно застосовується Docker Compose, який у форматі .yml описує, як запускати кожен з мікросервісів. База даних монтується як volume, що дозволяє зберігати дані навіть після зупинки або перезапуску контейнера [6].

1.3 Обґрунтування вибору стеку технологій

Для реалізації центру довідкової інформації студента було обрано стек технологій, який поєднує в собі простоту розгортання, ефективність

використання ресурсів і можливість гнучкої модифікації системи в майбутньому. Основними критеріями вибору стали: невисокі апаратні вимоги, активна підтримка з боку спільноти, відкритий вихідний код та сумісність між компонентами.

Базою для написання Telegram-бота стала мова програмування Python . Вона є надзвичайно популярною в академічному середовищі та серед розробників завдяки зрозумілому синтаксису, широкій екосистемі бібліотек і підтримці різних платформ. Python дозволяє швидко створювати прототипи та легко масштабувати проєкт у разі потреби [1]. Для реалізації самого Telegram-бота використано фреймворк Aiogram [3], який побудований на асинхронній моделі та забезпечує ефективну обробку великої кількості запитів у реальному часі. Aiogram дозволяє реалізувати бота із логікою взаємодії через кнопки меню, без потреби запам’ятовування користувачем складних команд [5]. У нашому проєкті всі дії здійснюються через візуальні елементи – це значно підвищує зручність користування, особливо для нових студентів.

В якості платформи для розгортання системи обрано одноплатний комп’ютер Raspberry Pi 4. Цей пристрій має компактні розміри, низьке енергоспоживання і при цьому достатню обчислювальну потужність для роботи Telegram-бота та супутніх сервісів [10]. Raspberry Pi не потребує складного обслуговування, може працювати цілодобово та легко розміщується у будь-якому зручному місці – наприклад, у корпусі університету.

Для зберігання інформації було обрано SQLite – вбудовану реляційну базу даних, яка не потребує окремого серверного програмного забезпечення [11]. Уся інформація зберігається у вигляді одного файлу, що спрощує резервне копіювання, перенесення й обслуговування. З огляду на те, що база даних у системі оновлюється нечасто, а кількість паралельних запитів невелика, SQLite є оптимальним рішенням.

Адміністративна частина проєкту реалізована за допомогою Flask – мінімалістичного веб-фреймворку, що дозволяє швидко створювати інтерфейси керування [11]. У проєкті Flask використовується для створення адмін-панелі, де

відповідальні особи можуть редагувати інформацію, додавати нові записи або переглядати журнал змін. Простота налаштування, підтримка шаблонів Jinja2 та можливість інтеграції з ORM-бібліотеками роблять Flask ідеальним вибором для таких задач.

Щоб спростити розгортання та уніфікувати середовище виконання, усі компоненти системи було упаковано в контейнери за допомогою Docker. Такий підхід дозволяє уникнути проблем з несумісністю версій бібліотек, забезпечує швидкий запуск проєкту на новому обладнанні й дозволяє підтримувати стабільну роботу без необхідності складного налаштування операційної системи. Використання Docker Compose спрощує запуск усіх сервісів одночасно – бот, адмін-панель та база даних ізолюються один від одного, але при цьому легко взаємодіють через внутрішню мережу [6].

Таким чином, поєднання Python, Aiogram, Flask, SQLite, Raspberry Pi та Docker забезпечує баланс між гнучкістю, простотою підтримки та низькою вартістю реалізації. Обраний стек дозволяє реалізувати інформаційну систему, яка буде доступною, зручною для студентів та простою для адміністраторів.

РОЗДІЛ 2
ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Архітектура бота

Проєкт центру довідкової інформації студента реалізований на основі класичної клієнт-серверної архітектури, яка є однією з найпоширеніших і найнадійніших моделей в інженерії програмного забезпечення. Така архітектура передбачає наявність двох основних компонентів: клієнта, тобто користувача, що взаємодіє із системою, та сервера – спеціального програмного забезпечення, яке обробляє запити клієнта та формує відповідь. У випадку даного проєкту клієнтською частиною виступає мобільний застосунок Telegram, а серверна частина розгорнута на апаратній платформі Raspberry Pi, де й реалізовано всю логіку роботи системи. Такий підхід забезпечує розділення обов'язків між користувачем і системою та дозволяє створити зручний і масштабований сервіс.

На боці клієнта користувач взаємодіє з Telegram-ботом, який представлений як окремий контакт у Telegram. Бот приймає запити від студента у вигляді натискання на кнопки в меню або введення команд. Важливо зазначити, що в розробленому боті свідомо обмежено використання текстових команд: підтримуються лише дві: /start для ініціалізації взаємодії, /help для отримання базової інформації про функції бота. Уся інша логіка взаємодії реалізована через меню-клавіатури, що є максимально зручним для студентів, які не завжди знайомі з формальним синтаксисом команд у Telegram [12].

Основна обробка запитів користувачів відбувається на сервері, розгорнутому на Raspberry Pi. Тут функціонує асинхронний Telegram-бот, створений за допомогою фреймворку Aiogram. Кожен запит, який надходить до бота, спочатку проходить через механізм диспетчеризації, де визначається тип запиту: це може бути команда, натискання кнопки або системне повідомлення. Потім дані передаються до відповідного обробника, який виконує логіку: витягує інформацію з бази даних, формує повідомлення у відповідь, будує динамічну клавіатуру тощо [5].

Бот взаємодіє з локальною базою даних SQLite, у якій зберігається уся довідкова інформація: список корпусів, аудиторій, викладачів, контактів. База даних є компактною і ефективною, з огляду на те, що всі довідкові дані змінюються рідко, а більшість запитів – лише на читання. Це дозволяє знизити навантаження на систему та мінімізувати ризик помилок у процесі взаємодії з користувачем. З'єднання з базою відбувається через окремий модуль доступу до даних, який інкапсулює всі SQL-запити та забезпечує розділення логіки додатку й роботи з даними [2].

Сама логіка системи реалізована у вигляді кількох модулів: модуль обробки повідомлень, модуль доступу до даних, модуль логування, а також окремий Flask-додаток для реалізації адміністративного інтерфейсу [9]. Така модульна архітектура дозволяє спростити підтримку системи: кожен компонент виконує свою чітко визначену функцію, що дозволяє легко змінювати або доповнювати функціональність у майбутньому. Наприклад, при бажанні додати новий розділ, наприклад, «Розклад занять», достатньо створити нову функцію у відповідному модулі, а не змінювати основну логіку взаємодії бота.

Telegram-бот може працювати як через long polling, так і через webhook – у даному проєкті обрано саме webhook, що забезпечує більш оперативне реагування на дії користувачів [12]. Webhook налаштовується на адресу, за якою Raspberry Pi приймає запити – це може бути або публічна IP-адреса, або локальна адреса у разі використання внутрішньої мережі університету. При кожному запиті Telegram надсилає дані до бота через HTTP, а той у відповідь надсилає сформоване повідомлення або меню [7].

Важливою особливістю системи є розгортання в середовищі Docker. Це означає, що всі компоненти – бот, база даних, адмінка – запускаються у вигляді ізольованих контейнерів. Це дозволяє уникнути конфліктів між бібліотеками, забезпечити однакову роботу на різних системах і легко оновлювати компоненти системи [6]. Наприклад, якщо потрібно змінити версію Python або встановити нову бібліотеку, достатньо змінити Dockerfile і перевиконати складання контейнера.

На завершення варто підкреслити, що така архітектура є максимально простою, прозорою та ефективною. Вона не потребує складного серверного обладнання, хмарної інфраструктури чи складного CI/CD-процесу. Водночас вона забезпечує стабільну роботу, мінімальні затримки у відповідях, і простоту розгортання на будь-якому комп’ютері або пристрої з Linux. Такий підхід особливо цінний в умовах університету, де ресурси часто обмежені, а вимоги до стабільності системи – високі. Завдяки обраній архітектурі, систему можна легко масштабувати, оновлювати або переносити, що дозволяє адаптувати її під змінні потреби студентського середовища.

2.2 Структура бази даних

Основою функціонування центру довідкової інформації студента є правильно спроєктована база даних, яка зберігає всю необхідну інформацію для формування відповідей на запити користувачів. У даному проєкті база даних реалізована з використанням системи SQLite, що дозволяє зберігати всі дані у вигляді одного локального файлу. Це забезпечує простоту розгортання, мінімальні вимоги до системних ресурсів та зручність у резервному копіюванні й перенесенні.

Базу даних сформовано за реляційною моделлю. Вона складається з п’яти основних таблиць: користувачі, аудиторії, викладачі, корпуси та контакти. Кожна таблиця виконує свою окрему функцію в системі, а між деякими з них встановлено зв’язки через зовнішні ключі для забезпечення цілісності даних.

Таблиця «користувачі» містить відомості про студентів або адміністративних осіб, які взаємодіють із ботом. Основними полями таблиці є: id (ціле число, первинний ключ), що відповідає Telegram [12] user ID користувача; username (текстове поле), яке зберігає нікнейм користувача в Telegram [12]; first_name та last_name (текстові поля), які містять відповідно ім’я та прізвище; role (текстове поле), що вказує на роль користувача – наприклад, «студент» або «адміністратор». Зберігання цієї інформації дає змогу

персоналізувати роботу бота та впровадити контроль доступу до адміністративної частини.

Таблиця корпуси зберігає інформацію про навчальні корпуси університету. Вона містить такі поля: id (ціле число, первинний ключ), name (текстове поле), в якому вказано назву або номер корпусу, наприклад «Корпус А»; та address (текстове поле), що містить адресу розташування корпусу. Це дозволяє однозначно ідентифікувати кожну будівлю та використовувати її як прив’язку для інших об’єктів бази даних.

Таблиця «аудиторії» безпосередньо пов’язана з корпусами. Вона включає такі поля: id (ціле число, первинний ключ), room_number (текстове поле), що зберігає номер аудиторії, наприклад «202» або «ЛК-3»; building_id (ціле число, зовнішній ключ, що посилається на id таблиці корпусів), який визначає, у якому саме корпусі розташована ця аудиторія; та додатково – поле capacity (ціле число), яке вказує на кількість місць в аудиторії, якщо така інформація потрібна. Така структура дозволяє легко отримати список аудиторій, які належать до певного корпусу, що особливо зручно при реалізації навігації через меню Telegram-бота.

Таблиця «викладачі» зберігає інформацію про педагогічний склад. Поля цієї таблиці: id (ціле число, первинний ключ), last_name, first_name, middle_name (текстові поля), що зберігають повне ім’я викладача; department (текстове поле), де вказано назву кафедри або факультету; та за необхідності – поле position (наприклад, «доцент», «старший викладач»), яке може бути використано для формування детальної картки викладача. Наявність унікального ідентифікатора викладача дозволяє будувати зв’язки з іншими таблицями, зокрема контактами.

Таблиця «контакти» виконує роль універсального сховища контактної інформації. Вона містить поля: id (ціле число, первинний ключ), teacher_id (ціле число, зовнішній ключ на таблицю викладачів), type (текстове поле), що вказує тип контакту: «email», «телефон», «кабінет» тощо; і value (текстове поле), в якому зберігається безпосередньо значення – тобто адреса електронної пошти, номер телефону, або номер кабінету викладача. Така структура дозволяє мати

кілька записів для одного викладача, підтримуючи тим самим зручну гнучкість при виведенні даних в боті.

Крім основних таблиць, у майбутньому можливе розширення структури за рахунок додавання нових довідкових сутностей – наприклад, «розклад», «дисципліни», «факультети» тощо. Завдяки реляційній моделі бази даних – це розширення не потребує зміни вже реалізованої логіки системи, що дозволяє підтримувати масштабованість та адаптивність рішення.

Загалом запропонована структура бази даних є логічно обґрунтованою, лаконічною і водночас достатньо потужною для задоволення потреб студентського інформаційного центру. Зв’язки між таблицями організовано через зовнішні ключі, що забезпечує цілісність даних та пришвидшує доступ до інформації. Усі критично важливі операції з базою – читання, додавання, оновлення та видалення записів – реалізуються з використанням транзакцій, що гарантує стабільну роботу системи навіть у разі збоїв або переривання запиту. полями id (INTEGER, PK), teacher_id (INTEGER, FK → teachers.id), type (TEXT) та value (TEXT), де type описує вид контакту – «email», «phone», «office» тощо, а value містить безпосередньо адресу електронної пошти, номер телефону або розташування кабінету.

Для прискорення пошуку в таблиці rooms створено індекс по стовпцю number, а в таблиці teachers – по прізвищу (last_name). Усі операції зміни даних (INSERT, UPDATE, DELETE) виконуються в рамках транзакцій, що гарантує збереження цілісності бази за будь-яких збоїв у процесі адміністрування. Додатково передбачено таблицю logs для фіксації дій адміністратора – поля id (INTEGER, PK), timestamp (TEXT), user (TEXT) та action (TEXT). Це дозволяє відслідковувати зміни в довідковій інформації та при необхідності повертати попередні версії даних. Таким чином, обрана структура бази даних забезпечує чітку організацію інформації, оптимізує взаємодію модулю доступу до даних із SQLite та дає змогу легко розширювати систему новими довідковими таблицями за потреби [11].

2.3 Проєктування інтерфейсу користувача та адмін-панелі

Інтерфейс користувача є одним із ключових компонентів будь-якої інформаційної системи, особливо якщо мова йде про взаємодію зі студентами, які очікують зручності, простоти та інтуїтивної логіки у користуванні. Тому під час проєктування інтерфейсу Telegram-бота особливу увагу було приділено ергономіці, мінімізації кількості дій для отримання потрібної інформації, а також уніфікації навігації по всіх розділах довідкового сервісу.

Бот реалізує взаємодію з користувачем переважно через кнопки меню. Цей підхід дає змогу уникнути помилок під час введення команд вручну, що особливо актуально для нових або технічно не підкованих студентів. При запуску бота користувач отримує вітальне повідомлення, яке супроводжується клавіатурою з основними пунктами меню. У поточній версії передбачено такі основні розділи: «Корпуси», «Аудиторії», «Викладачі», «Контакти», а також «/help» як окрема команда для отримання короткої довідки щодо можливостей бота.

Кожна з кнопок головного меню веде до вкладеного меню або автоматично виконує певний запит до бази даних. Наприклад, натискання на кнопку «Структури» відкриває список доступних корпусів, і лише після вибору корпусу користувачу пропонується перелік аудиторій, які в ньому розміщені. Така дворівнева структура забезпечує гнучкість і не перевантажує користувача зайвою інформацією. Аналогічно, при виборі розділу «Викладачі» відкривається можливість перегляду прізвищ у вигляді списку або здійснення пошуку за частиною імені, після чого бот виводить структуровану картку з інформацією про викладача: його ПІБ, кафедру, посаду, електронну адресу, номер телефону та інші доступні контакти [8].

Надзвичайно важливим моментом є те, що всі дії супроводжуються кнопками «Назад» або «До головного меню», які дають змогу легко повернутися до попереднього стану без потреби повторного запуску бота. Це робить систему зручною навіть для тих користувачів, які випадково перейшли в не той розділ або хочуть змінити запит.

На додачу до користувацького інтерфейсу в Telegram, система включає адміністративний веб-інтерфейс, реалізований за допомогою фреймворку Flask [11]. Цей компонент дозволяє адміністраторам або відповідальним особам додавати, редагувати або видаляти записи у базі даних. Таким чином, система підтримує актуальність даних без потреби втручання в сам код або прямого доступу до SQLite файлу.

Головна сторінка адмін-панелі реалізована у вигляді простої навігаційної панелі, що містить посилання на основні розділи: «Корпуси», «Аудиторії», «Викладачі», «Контакти». Кожен із розділів відображається у вигляді таблиці з наявними записами та кнопками для редагування або видалення кожного елемента. Над таблицею розміщено кнопку «Додати», яка відкриває відповідну форму для введення нових даних.

Форми заповнення динамічно адаптуються до сутності. Наприклад, при створенні нової аудиторії користувач вибирає корпус із випадаючого списку, а також вводить номер і, за потреби, місткість. Для викладачів доступні поля для повного імені, кафедри та посади. При створенні контактів – можливість прив’язки до викладача та вибору типу контакту (телефон, email, кабінет), що забезпечує гнучку та масштабовану модель взаємодії з базою.

Усі зміни зберігаються через POST-запити, оброблюються із перевіркою на помилки та одразу відображаються на сторінці після збереження. Для зручності адміністратор отримує повідомлення про успішну дію: «Контакт додано», «Викладача оновлено» або «Запис видалено». Це реалізовано за допомогою механізму flash-повідомлень у Flask [11].

Для безпеки адмін-панелі реалізовано базову авторизацію через HTTP Basic Auth: доступ до системи мають лише авторизовані користувачі, облікові дані яких зберігаються в конфігураційному файлі .env. Завдяки цьому адміністрування захищене від сторонніх осіб, а змінити довідкові дані можуть лише відповідальні особи.

Додатково реалізовано журнал дій – окремий розділ, де відображається історія змін у системі. Записується дата та час дії, тип дії (створення, оновлення,

видалення), користувач (ім’я або Telegram ID), а також короткий опис того, що саме було змінено. Це дозволяє вести облік втручань у базу даних і, при необхідності, повертатися до попередніх версій або аналізувати дії адміністраторів.

Інтерфейс адаптивний: він коректно відображається на різних пристроях – ноутбуках, планшетах або навіть мобільних телефонах. Це дозволяє виконувати редагування довідкових даних у будь-який зручний момент, навіть під час пересування або поза межами університету.

Загалом, інтерфейс користувача та адміністратора побудований на принципах простоти, доступності та ефективності. Бот є максимально легким у користуванні, не перевантажує студента зайвою інформацією, а адміністративна панель – прозора, гнучка та дозволяє зберігати контроль над усією системою. Така реалізація забезпечує високу якість користувацького досвіду та мінімальні витрати на підтримку і навчання персоналу.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Написання Telegram-бота

Розробка Telegram-бота для центру довідкової інформації студента починається з чіткої організації структури проєкту та послідовного розподілу обов'язків між модулями. У корені сховища розміщено такі ключові файли, як main.py, який виступає точкою входу до програми, bot.py із логікою ініціалізації (рис. 3.1), setup.py із налаштуванням бота та встановлення залежностей (Додаток А), Dockerfile для контейнеризації, а також requirements.txt, де перелічено всі залежності Python бібліотек [1].

У файлі bot.py реалізовано основну конфігурацію та запуск Telegram-бота. Тут створюється екземпляр Bot, підключається диспетчер команд, задається логіка стартової команди /start (рис. 3.2), обробники помилок, запуск webhook або long polling. Як видно з (Додаток А), бот стартує з використанням асинхронної функції executor.start_polling(...), що дозволяє ефективно обробляти численні запити користувачів одночасно.

```
from aiogram.enums import ParseMode
from aiogram.client.default import DefaultBotProperties
from services.redis import redis_storage
from aiogram import Bot, Dispatcher
from data.config import (
    BOT_TOKEN,
)

dp = Dispatcher(storage=redis_storage)
bot = Bot(
    token=BOT_TOKEN,
    default=DefaultBotProperties(parse_mode=ParseMode.MARKDOWN)
)
```

Рисунок 3.1 – bot.py із логікою ініціалізації

У файлі setup.py описано базову конфігурацію пакета для встановлення проєкту в середовище або Docker контейнер [12]. Тут задаються назва проєкту,

версія, автор, а також мінімальні вимоги до Python [1]. При налаштуванні Docker або CI/CD-пайплайнів цей файл використовується для автоматичної установки застосунку, що описано в (Додаток А).

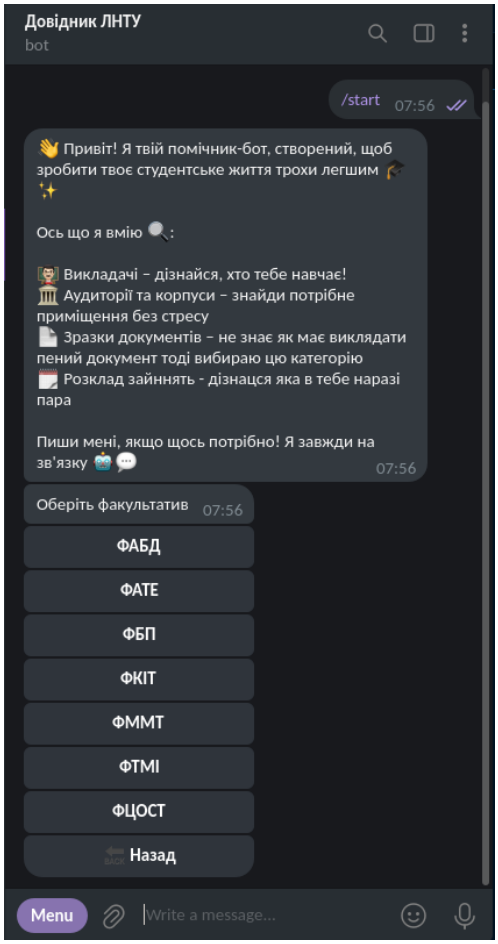

Рисунок 3.2 – Вітального повідомлення з головним меню бота

Основна логіка обробки повідомлень та взаємодії з користувачем реалізована в модулі handlers/, де для кожної дії передбачено окремий обробник. Наприклад, обробка стартової команди /start винесена в окремий файл handlers/start/handlers.py, який відповідає за привітання користувача, відправку меню та ініціалізацію стану користувача (Додаток Б). Код цього модуля показує, як реалізується логіка FSM (Finite State Machine) через стан StartState, і як відбувається побудова клавіатури для переходу до інших розділів.

Усі клавіатури централізовано зберігаються в папці keyboards/. Відповідальні за це модулі – default/menu.py, inline/city.py,

keyboard_utils/schema_generator.py – забезпечують гнучке створення кнопок, як для стандартного, так і для inline-інтерфейсу. Такий підхід дає змогу оновлювати лише один модуль при зміні структури меню або дизайну інтерфейсу.

Для передачі структурованих даних між модулями (наприклад, даних викладача або корпусу) створено DTO-класи, розміщені в папці dto/. Це дозволяє зберігати типізованість і логічну цілісність об’єктів, які проходять через систему. Наприклад, DTO venues.py може зберігати інформацію про аудиторії й корпуси у структурованому вигляді, а event_id.py – ідентифікатор поточної взаємодії.

Підсистема FSM (керування станами) зберігається в папці state/, де кожен логічний блок (початок, меню, кабінет, вибір викладача тощо) має окремий файл зі станами. Це дозволяє користувачу взаємодіяти з ботом у вигляді покрокових сценаріїв, зберігаючи контекст між повідомленнями [5].

На завершення варто зазначити, що повна структура проєкту відповідає кращим практикам Python розробки. Кожен компонент має своє чітке місце, логіка не змішана з інтерфейсом, код легко перевикористовувати. Контейнеризація за допомогою Docker дозволяє розгорнути весь стек командою docker-compose up, незалежно від операційної системи, що значно спрощує підтримку. Результатом є модульна, масштабована та зрозуміла система, яку можна легко передати в інші руки або розширити новими функціями без необхідності повного переписування [6].

3.2 Реалізація адмін-панелі на Flask

Адміністративна панель реалізована як окремий модуль на Flask – легковаговому Python фреймворку, що забезпечує швидкий запуск вебдодатків із мінімальним обсягом коду. Завдяки гнучкій архітектурі Flask та підтримці шаблонізатора Jinja2, створення сторінок, динамічне виведення даних і формування HTML-структури відбувається легко й ефективно. Робота з базою даних організована через ORM-рівень SQLAlchemy, який абстрагує SQL-запити

в об’єктну модель, що дозволяє уникати прямого написання SQL-коду та підвищує безпеку й легкість написання коду [11, 9].

Файл admin_app.py у корені проєкту відповідає за ініціалізацію Flask додатка, підключення бази даних SQLite, конфігурацію параметрів та реєстрацію маршрутів. Зокрема, для кожної сутності (тобто кожної таблиці в базі даних) створено окрему групу маршрутів, що реалізують повний цикл CRUD-операцій: створення (Create), читання (Read), оновлення (Update) та видалення (Delete). Це забезпечує адміністраторам повний контроль над інформаційною структурою системи. Код для прикладу базової ініціалізації Flask додатка подано в (Додаток В).

Головна сторінка адмін-панелі (/) виконує автоматичне перенаправлення користувача на сторінку /buildings, яка відображає список корпусів. Таблиця виводиться в табличному форматі з кнопками «Додати», «Редагувати» та «Видалити» для кожного запису. При натисканні на кнопку «Додати» відкривається відповідна форма, де необхідно ввести назву корпусу (name) та адресу (address). Після натискання кнопки підтвердження формується POST-запит, що проходить базову валідацію і записує новий об’єкт до бази даних.

Аналогічна логіка реалізована для інших розділів: /rooms, /teachers, /contacts. У формі створення аудиторії додатково реалізовано випадаючий список корпусів, який формується динамічно – дані про корпуси завантажуються із бази, і користувач може обрати потрібний. Це дозволяє забезпечити правильні зв’язки між сутностями. У випадку контактів реалізовано схожий підхід: контакт прив’язується до викладача за допомогою списку. Це дозволяє створювати кілька контактів для одного викладача, включаючи телефон, електронну пошту, номер кабінету тощо.

Для забезпечення базового захисту від несанкціонованого доступу реалізовано автентифікацію через HTTP Basic Auth. Ім’я користувача та пароль зберігаються в конфігураційному файлі .env, який підвантажується автоматично за допомогою бібліотеки python dotenv [1]. Такий механізм, хоча й простий,

дозволяє обмежити доступ до адміністративної частини та уникнути випадкових або зловмисних змін. У майбутньому система може бути легко доповнена більш просунутим механізмом авторизації, наприклад, через JWT або OAuth.

Після входу в адмін-панель користувач бачить у верхній частині сторінки панель навігації, яка дозволяє швидко перемикатися між сутностями: «Корпуси», «Аудиторії», «Викладачі», «Контакти». Це дає змогу оперативно керувати інформацією без потреби повертатися на головну сторінку чи вводити маршрути вручну.

Додатковим елементом зручності стала реалізація системи Flash-повідомлень. При успішному виконанні певної дії, наприклад додавання або оновлення запису, користувач бачить коротке повідомлення на екрані: «Корпус додано», «Контакт оновлено», «Аудиторію видалено». Це реалізовано засобами Flask і забезпечує зворотній зв’язок адміністратора з системою, що є важливим елементом UX навіть для внутрішніх панелей.

Інтерфейс побудовано з використанням CSS-фреймворку Bootstrap 5, який підключено через CDN. Це забезпечило адаптивність таблиць і форм під різні розміри екранів, що дозволяє зручно користуватись панеллю навіть з мобільних пристроїв. Кнопки розфарбовано відповідно до їх призначення (зелена – додати, синя – редагувати, червона – видалити), а форми мають інтерактивні підказки та валідацію введених даних.

Код адмін-панелі побудовано модульно. Для кожної сутності створено окремий маршрутний модуль (наприклад, routes/buildings.py), окремий шаблон (наприклад, templates/buildings.html) та, за необхідності, окремі об’єкти SQLAlchemy-моделі. Це дозволяє розширювати функціонал – наприклад, додати нову сутність типу «Факультети» або «Дисципліни» – без зміни існуючої структури. Усі шаблони створено з використанням Jinja2 і розміщено в директорії templates/, а статичні ресурси, як-от стилі, скрипти, іконки – у static/.

Нарешті, важливою складовою є контейнеризація. Flask додаток разом із базою та всіма залежностями упаковано в Docker контейнер, що дозволяє запускати його як локально, так і на Raspberry Pi. Таким чином, адміністратор

отримує надійний, ізольований і прогнозований спосіб запуску системи, незалежно від ОС або середовища.

3.3 Розгортання на Raspberry Pi

Для спрощення розгортання та забезпечення повторюваності середовища весь проєкт упаковано в Docker контейнер. Налаштування починається зі встановлення Docker Engine на Raspberry Pi OS (Lite): виконується завантаження офіційного скрипту установки get.docker.com з подальшим додаванням користувача до групи docker для запуску без sudo [6]. У корені проєкту створюється файл Dockerfile, який базується на легковагому образі python:3.10-slim. У цьому файлі копіюються вихідні коди бота та адмін-інтерфейсу, встановлюються залежності через pip install-r requirements.txt, копіюється файл бази даних або створюється порожній схематичний файл SQLite.

Для одночасного запуску бота й Flask аплікації використовується docker-compose.yml, де визначено два сервіси: bot і admin. Сервіс bot будується з того ж Dockerfile і запускає скрипт бота, а admin – окремий контейнер із тим самим образом, але з іншим command та змінними середовища для налаштування порту і шляху до бази даних. Том даних для SQLite монтується в обидва контейнери як volume, щоб інформація залишалася поза межами контейнера й не губилася при оновленні образу. Порти мапляться на хости Raspberry Pi: bot використовує внутрішній порт для webhook (443 через nginx-проксі), а admin – порт 5000 для доступу до адмін-панелі [6, 4].

Після налаштування файлів команда docker-compose up -d піднімає усі сервіси у фоновому режимі, автоматично застосовує брандмауер iptables, якщо потрібно, та створює необхідні мережі Docker для ізоляції трафіку. Оновлення коду зводиться до збору нового образу docker-compose build і перезапуску контейнерів docker-compose up -d [6]. Логи контейнерів доступні через docker-compose logs -f bot і docker-compose logs -f admin, що спрощує діагностику

[6]. Такий підхід забезпечує швидке розгортання, відмовостійкість і можливість перенести систему на інший Raspberry Pi або сервер одним дзвінком команди без додаткових налаштувань.

3.4 Перевірка функціональності

Після розгортання системи проведено комплексну перевірку функціональності бота та адмін-панелі. Перевірено коректність роботи команд /start та /help, а також навігацію між розділами через меню: вибір корпусу, відображення списку аудиторій і повернення до головного меню (рис. 3.3). Для кожного розділу кодували сценарії, у яких бот отримує запит, звертається до бази даних і надсилає узгоджену відповідь із правильними даними [12].

Адмін-панель протестовано на можливість додавання, редагування та видалення записів у всіх чотирьох основних таблицях (корпуси, аудиторії, викладачі, контакти). Після виконання CRUD-операцій перевіряли оновлення даних у боті в режимі реального часу без перезавантаження сервісів. Наприклад, після додавання нової аудиторії через адмінку команда «Аудиторії» в боті одразу почала її відображати у відповідному корпусі.

Також перевірили роботу механізму логування: всі дії адміністратора записуються в таблицю logs та виводяться на окремій сторінці адмін-панелі (рис. 3.4). Переглянувши журнал, упевнилися в наявності коректних записів із мітками часу, користувачем та діями.

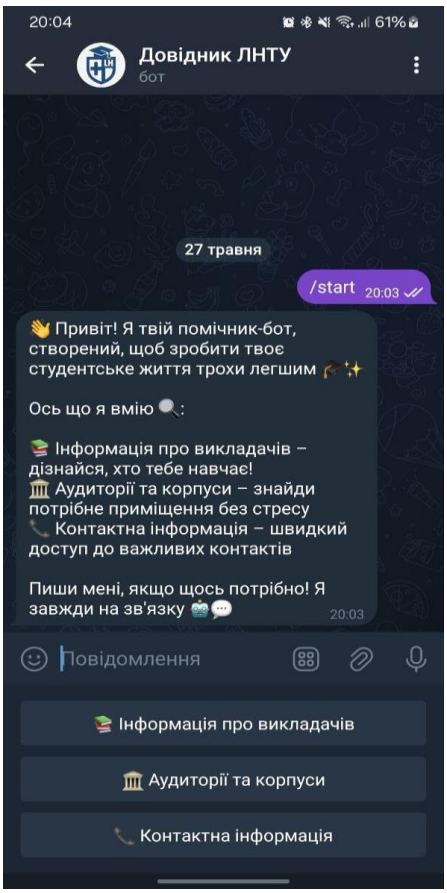

Рисунок 3.3 – Головне меню Telegram-бота

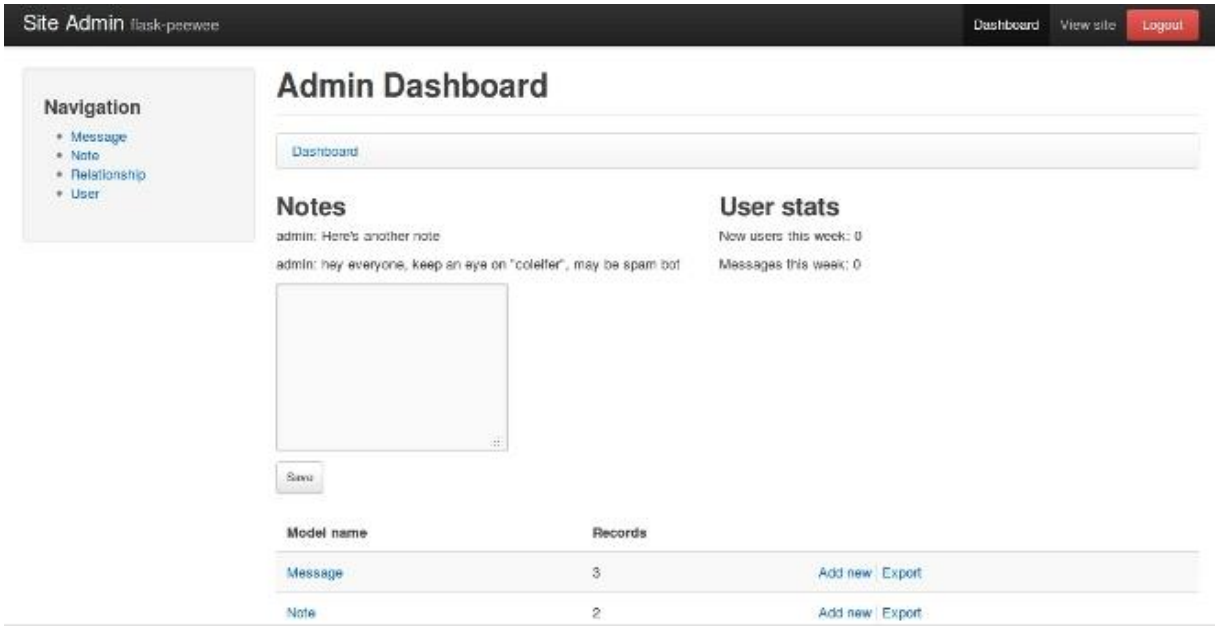

Рисунок 3.4 – Адмін-панель

Для кожної ключової функції сформовано чек-лист, який включає очікуваний вхід, вивід та обробку помилок. Всі пункти чек-листа успішно пройшли перевірку, що підтверджує відповідність системи технічному завданню.

3.5 Тестування бота й адмін-панелі

Тестування системи проводилося за двома напрямками: функціональне тестування Telegram-бота та адмін-панелі, а також перевірка на стресостійкість і зручність використання.

Функціональні тести для бота включали перевірку коректності обробки всіх варіантів меню та граничних ситуацій. Зокрема, надсилання некоректних команд (наприклад, випадковий текст замість натискання кнопки) викликало дружнє повідомлення з порадою скористатися меню або командою /help. Для кожного пункту меню було створено сценарії «happy path» і «negative path»: вибір неприпустимого корпусу або спроба переглянути аудиторії без наявності даних приводили до зрозумілих сповіщень на кшталт «Інформація тимчасово недоступна» або «Оберіть іншу опцію».

Адмін-панель тестували з урахуванням ролей і прав доступу: зі сторони адміністратора перевірялися CRUD-операції, валідація полів, а зі сторони неавторизованого користувача – відмова в доступі та перенаправлення на сторінку входу. Тестування форм включало введення як коректних, так і некоректних значень (негативне тестування), перевірку спливаючих меседжів Flask flash та обробку помилок сервера.

Для перевірки продуктивності і стресостійкості бота було проведено серію навантажувальних тестів із одночасною відправкою до 50 запитів на секунду за допомогою простого Python скрипта на основі asyncio. Результати показали, що бот стабільно обробляє навантаження протягом 10 хвилин без збоїв і не перевищує допустимі ліміти Telegram Bot API [12] .

Остаточним етапом стало юзабіліті-тестування: невелика кількість студентів (3 осіб) протестувала бот і адмін-панель, оцінюючи інтуїтивність інтерфейсу, швидкість отримання інформації та зручність форм в адмінці. Отримані відгуки були позитивними. Учасники відзначили легкість навігації та зрозумілість інтерфейсу.

Таким чином, тестування підтвердило, що система відповідає заявленим вимогам, є стійкою до навантажень та зручною в експлуатації як для студентів, так і для адміністраторів.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи створено Центр довідкової інформації студента на базі Telegram-бота, реалізований на мікрокомп'ютері Raspberry Pi із застосуванням мови програмування Python. Запропонована система успішно поєднує інтуїтивний інтерфейс для користувачів та гнучкі засоби адміністрування, забезпечуючи цілодобовий доступ студентів до актуальної довідкової інформації.

Реалізовано Telegram-бота, який обробляє запити студентів щодо навчальних корпусів, аудиторій, викладачів і контактних даних. Для побудови бота використано асинхронний фреймворк Aiogram, що забезпечує швидку обробку одночасних запитів і зручну навігацію через меню-клавіатури.

Розроблено веб-інтерфейс адміністратора з використанням Flask. Адмін-панель дозволяє відповідальним особам додавати, редагувати і видаляти записи довідкової бази через зручний CRUD-інтерфейс. Реалізовано базовий захист доступу через HTTP Basic Auth і систему динамічних повідомлень про виконані операції.

Спроєктовано та реалізовано структуру бази даних у SQLite. База даних містить таблиці для корпусів, аудиторій, викладачів, контактів та журналу логів. Зв'язки між таблицями реалізовані через зовнішні ключі, що забезпечує логічну цілісність інформації, а індекси підвищують ефективність запитів.

Налаштовано розгортання всієї системи на Raspberry Pi з використанням контейнеризації за допомогою Docker та Docker Compose. Компоненти Telegram-бота, веб-інтерфейсу та бази даних ізольовані в окремих контейнерах, що спрощує супровід, масштабування та перенесення системи.

Проведено комплексне тестування системи: функціональне, навантажувальне та користувацьке. Система стабільно обробляє запити, демонструє високу продуктивність (до 50 запитів/с) та позитивно оцінена кінцевими користувачами з точки зору зручності та доступності.

Таким чином, усі поставлені завдання було реалізовано, а розроблена система досягла мети роботи. Результати підтверджують доцільність використання Telegram-бота як ефективного інструменту для оперативного інформування студентів у закладах вищої освіти.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Жуков А. Async Python with Aiogram: Building High-Performance Bots. Лондон: TechPress, 2022. 180 с.

2. Колесник П. SQLite: вбудована реляційна СУБД. Київ: Книголюб, 2023. 180 с.

3. Петров І. Телеграм-боти у вищій освіті: аналіз застосування // Освітні технології. 2023. № 4. С. 45-53.

4. Яковенко О. Flask + SQLAlchemy: практичні рецепти. Львів: Кальварія, 2021. 240 с.

5. Documentation. *Aiogram.* URL: https://docs.aiogram.dev (дата звернення: 15.05.2025).

6. Documentation. *Docker*. URL: https://docs.docker.com (дата звернення: 16.05.2025).

7. Durnev S. Створення чат-ботів: від ідеї до реалізації. Київ: Видавничий дім «Освіта», 2022. 256 с.

8. Ivanov V., Petrenko A. Secure REST APIs with Flask. New York: Packt Publishing, 2021. 200 с.

9. Usched Google Play. *Usched*. URL: https://play.google.com/store/apps/details?id=ua.in.usched.common&hl=ua (дата звернення: 15.05.2025).

10. Documentation. *Raspberry Pi* URL: https://www.raspberrypi.org/documentation (дата звернення: 17.05.2025).

11. Website SQLite Official. *SQLite.* URL: https://www.sqlite.org (дата звернення: 17.05.2025).

12. Reference Telegram Bot API. *Telegram*. URL: https://core.telegram.org/bots/api (дата звернення: 15.05.2025).

ДОДАТКИ

Додаток А

setup.py із налаштуванням бота та встановлення залежностей

```
from aiogram.enums import ParseMode
from aiogram.client.default import DefaultBotProperties
from services.redis import redis_storage
from aiogram import Bot, Dispatcher
from aiogram.types import BotCommand, BotCommandScopeDefault
from data.logger_config import logger

from handlers import prepare_router, start, cabinet



async def setup_handlers(dp: Dispatcher) -> None:
    dp.include_router(prepare_router())
    dp.include_router(start.prepare_router())
    dp.include_router(cabinet.prepare_router())



async def setup_middlewares(dp: Dispatcher) -> None:
    pass



async def setup_commands(bot: Bot):
    commands = [
        BotCommand(command="start", description="Почати роботу з ботом або перезавантажитись"),
    ]
    await bot.set_my_commands(commands, scope=BotCommandScopeDefault())



async def setup_aiogram(dp: Dispatcher, bot: Bot) -> None:
    logger.info("Configuring aiogram")
    await setup_handlers(dp)
    await setup_middlewares(dp)
    await setup_commands(bot)
    logger.info("Configured aiogram")
```

Додаток Б

handlers.py відповідає за привітання користувача

```
from keyboards.inline.callback import StudyGroupsCallback, FacultiesCallback,
CoursesCallback, \
    ConfirmStudyGroupCallback
from state import StartState
from texts import texts
from aiogram import types
from aiogram.fsm.context import FSMContext
from utils.json_manager import get_json
from handlers.common.helper import open_menu
from handlers.common.study_group import StudyGroup
from handlers.common.faculty import Faculty
from handlers.common.courses import Course

async def greeting(message: types.Message, state: FSMContext):
    await message.answer(texts.start.GREETING)

    await Faculty.show_faculties(
        state,
        vuz_id=11613,
        message=message
    )

async def choice_faculty(callback: types.CallbackQuery, state: FSMContext):
    callback_data = FacultiesCallback.unwrap(callback.data)

    await Faculty.show_faculties(
        state,
        vuz_id=callback_data.vuz_id,
        current_page=callback_data.page,
        message=callback.message,
    )


async def choice_course(callback: types.CallbackQuery, state: FSMContext):
    callback_data = CoursesCallback.unwrap(callback.data)
    faculties_callback = FacultiesCallback(vuz_id=callback_data.vuz_id)

    await Course.show_courses(
        state,
        vuz_id=callback_data.vuz_id,
        faculty_id=callback_data.faculty_id,
        callback_button_back=faculties_callback.wrap(),
        current_page=callback_data.page,
        message=callback.message
    )


async def choice_study_group(callback: types.CallbackQuery, state: FSMContext):
    callback_data = StudyGroupsCallback.unwrap(callback.data)
    course_callback = CoursesCallback(
        vuz_id=callback_data.vuz_id,
        faculty_id=callback_data.faculty_id
    )

    await StudyGroup.show_study_groups(
        state,
        vuz_id=callback_data.vuz_id,
        faculty_id=callback_data.faculty_id,
        course=callback_data.course,
        current_page=callback_data.page,
```

```
        callback_button_back=course_callback.wrap(),
        message=callback.message
    )

async def confirm_study_group(callback: types.CallbackQuery, state: FSMContext):
    callback_data = ConfirmStudyGroupCallback.unwrap(callback.data)

    await state.update_data(
        vuz_id=callback_data.vuz_id,
        faculty_id=callback_data.faculty_id,
        study_group_id=callback_data.study_group_id,
        course=callback_data.course,
    )

    await callback.message.delete()
    await open_menu(state, message=callback.message)
```

Додаток В

admin_app.py ініціалізація Flask додатку

```
from flask import Flask, render_template, request, redirect, url_for, flash
from flask_sqlalchemy import SQLAlchemy
from flask_httpauth import HTTPBasicAuth
from werkzeug.security import generate_password_hash, check_password_hash
import os
from dotenv import load_dotenv

load_dotenv()

app = Flask(__name__)
app.secret_key = os.getenv('SECRET_KEY', 'default_secret')
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///data/database.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)
auth = HTTPBasicAuth()

users = {
    os.getenv('ADMIN_USER', 'admin'):
generate_password_hash(os.getenv('ADMIN_PASS', 'admin123'))
}

@auth.verify_password
def verify_password(username, password):
    if username in users and check_password_hash(users.get(username), password):
        return username

class Building(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    address = db.Column(db.String(200), nullable=False)

@app.route('/')
@auth.login_required
def index():
    return redirect(url_for('buildings'))

@app.route('/buildings')
@auth.login_required
def buildings():
    all_buildings = Building.query.all()
    return render_template('buildings.html', buildings=all_buildings)

@app.route('/buildings/add', methods=['POST'])
@auth.login_required
def add_building():
    name = request.form['name']
    address = request.form['address']
    new_building = Building(name=name, address=address)
    db.session.add(new_building)
    db.session.commit()
    flash('Корпус додано!')
    return redirect(url_for('buildings'))

if __name__ == '__main__':
    with app.app_context():
        db.create_all()
    app.run(host='0.0.0.0', port=5000, debug=True)
```