МАГІСТЕРСЬКА РОБОТА

МР. ІПм - 43.00.00.000 ПЗ

Група ІПм-23-1

Андрусяк Василь

2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

_____

(прізвище, ім'я, по батькові)

УДК 004.942
(індекс)

МАГІСТЕРСЬКА РОБОТА

Моделі, методи та алгоритми інтеграції чат-ботів з університетськими базами даних та сервісами

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Андрусяк В. М.
(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Романишин Тарас Любомирович, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту
Завідувач кафедри
доц. Бандура В.В.
(посада) (підпис) (дата) (ініціали та прізвище)

Нормоконтроль

доц. Вовк Р.Б.
(посада) (підпис) (дата) (ініціали та прізвище)

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

Івано-Франківськ – 2024

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ ____ ” _____2024 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Андрусяку Василю Миколайовичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи “Моделі, методи та алгоритми інтеграції чат-ботів з університетськими базами даних та сервісами **”**

керівник проекту (роботи) Романишин Т. Л., к.т.н.

затверджені наказом закладу вищої освіти від **“** 22” листопада 2024 р. № 781/7

2. Строк подання студентом проєкту (роботи) 15 грудня 2024 р.

3. Вихідні дані до проекту (роботи) Архітектура, формальний опис та алгоритми інтеграції чат-ботів з університетськими базами даних

4. Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити)

1. Дослідження технологій та інструментів побудови чат-ботів та інтеграції з БД
2. Опис основних можливостей, моделей та архітектура чат-ботів для інтеграції
3. Розробка архітектури та модельй інтеграції університетських БД із чат-ботами
4. Розробка алгоритму та програмної реалізації БД та чат-боту

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. UML діаграма варіантів використання (use case diagram) загальна (Рис. 2.3 ст. 47)
2. UML діаграма послідовностей логіки реєстрації та авторизації (Рис. 2.8 ст. 53)
3. UML діаграма компонентів інтеграції чат-боту із університетськими БД (Рис. 3.1 ст. 57)
4. Модель бази даних (Рис. 3.3 ст. 73)
5. Загальна UML діаграма класів (Рис. 2.4 ст. 74)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 4 вересня 2024 р.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів проекту	Примітка
1	Теоретичне дослідження побудови інтеграцій чат-ботів з університетськими базами даних	20.09.2024	виконано
2	Аналіз технологій розробки чат-ботів для інтеграції з базами даних	01.10.2024	виконано
3	Способи забезпечення безпеки та конфіденційності даних університетів в чат-ботах	12.10.2024	виконано
4	Дослідження алгоритму використання університетських баз даних в чат-боті	25.10.2024	виконано
5	Формулювання вимог та алгоритмів до програмного забезпечення	05.11.2024	виконано
6	Програмна реалізація рішення	22.11.2024	виконано
7	Затвердження пояснювальної записки роботи завідувачем кафедри	15.12.2024	виконано

Студент – магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 139 с., 24 рис., 23 табл., 50 джерел.

Тема: Моделі, методи та алгоритми інтеграції чат-ботів з університетськими базами даних та сервісами.

Об'єкт дослідження: моделі та алгоритми інтеграції університетських баз даних у сучасних чат-ботах

Мета роботи: проектування алгоритмів інтеграції баз даних університетів в чат-ботах, розробка моделей розгортання та масштабування чат-ботів на основі централізованих сховище даних університетів для взаємодії різних типів користувачів чат-боту.

Предмет дослідження: інноваційні чат-боти на різних сучасних платформах таких, як Telegram, Viber, WhatsApp та забечпення в них інтеграції з університетськими базами даних через API взаємодію.

Результати дослідження:

Виконано аналіз існуючих інтеграцій чат-ботів із університетськими базами даних, які призначені для покращення освітнього процесу. Запровадження покращених алгоритмів та моделів налаштування таких інтеграцій.

Висновок:

В результаті дослідження було проаналізовано аналоги чат-ботів університетів та на основі цих досліджень спроектовано оптимізовані алгоритми та моделі взаємодії чат-ботів з базами даних університетів для вдосконалення освітнього процесу та забезпечення взаємодії студентів між собою та викладачами а також моніторинг успішності студентів засобами інноваційних чат-ботів.

ЧАТ-БОТ, ТЕЛЕГРАМ БОТ, УНІВЕРСИТЕТ, БАЗИ ДАНИХ, АДМІНІСТРУВАННЯ УНІВЕРСИТЕТОМ, АРХІВ, ОГОЛОШЕННЯ ТА НОТИФІКАЦІЇ, ІНТЕГРАЦІЙНА ШИНА.

ANNOTATION

Master's work: 139 p., 24 fig., 23 tab., 50 sources.

Topic: Models, methods and algorithms for integrating chatbots with university databases and services.

Object of research: models and algorithms for processing biometric data of users of the system are presented as face images.

Purpose: designing algorithms for integrating university databases into chatbots, developing models for deploying and scaling chatbots based on centralized university data repositories for interaction between different types of chatbot users.

Subject of research: innovative chatbots on various modern platforms such as Telegram, Viber, WhatsApp and ensuring their integration with university databases via API interaction.

Research results:

An analysis of existing chatbot integrations with university databases designed to improve the educational process was performed. Implementation of improved algorithms and models for configuring such integrations.

Conclusion:

As a result of the study, analogues of university chatbots were analyzed and, based on this research, optimized algorithms and models of interaction of chatbots with university databases were designed to improve the educational process and ensure interaction between students and teachers, as well as monitoring student success using innovative chatbots.

CHAT BOT, TELEGRAM BOT, UNIVERSITY, DATABASES, UNIVERSITY ADMINISTRATION, ARCHIVES, ANNOUNCEMENTS AND NOTIFICATIONS, INTEGRATION BUS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХСКОРОЧЕНЬ 7
ВСТУП 8
РОЗДІЛ 1
ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО ЧАТ-БОТИ ТА ІНТЕГРАЦІЮ ІЗ СИСТЕМАМИ УПРАВЛІННЯ БАЗАМИ ДАНИХ
1.1 Теоретичні відомості про чат-боти та їх застосування в різних сферах 13
1.2 Моделі чат-ботів, правила та машинне навчання 17
1.3 Основи інтеграції чат-ботів з системами управління базами даних університетів 25
1.4 Огляд сучасних сервісів для інтеграції чат-ботів та можливості їх застосування у навчальних закладах 30
1.5 Огляд існуючих методів забезпечення безпеки даних при інтеграції чат-ботів 33
1.6 Висновки до розділу 37
РОЗДІЛ 2
ДОСЛІДЖЕННЯ ІСНУЮЧИХ МЕТОДІВ І ТЕХНОЛОГІЙ ІНТЕГРАЦІЇ ЧАТ-БОТІВ З УНІВЕРСИТЕТСЬКИМИ СИСТЕМАМИ І БАЗАМИ ДАНИХ
2.1 Технології та підходи до інтеграції чат-ботів з навчальними та адміністративними системами університетів 38
2.2 Аналіз інтерфейсів для взаємодії з чат-ботами в університетських системах 41
2.3 Визначення вимог до чат-ботів при інтеграції з університетськими базами даних 44
2.4 Розробка алгоритмів авторизації та автентифікацію у чат-боті. Визначення прав користувачів 49
2.5 Висновки до розділу 55

РОЗДІЛ 3

РОЗРОБКА МОДЕЛІ ТА АЛГОРИТМІВ ІНТЕГРАЦІЇ ЧАТ-БОТІВ З УНІВЕРСИТЕТСЬКИМИ БАЗАМИ ДАНИХ ТА СЕРВІСАМИ

3.1 Опис архітектури інтеграції чат-ботів з університетськими базами даних та сервісами 57

3.2 Опис архітектури університетських баз даних для інтеграції з чат-ботами 61

3.3 Модель інтеграції чат-ботів з університетськими базами даних 75

3.4 Проектування та програмна реалізація алгоритму інтеграції чат-боту з університетськими базами даних. Розробка алгоритму чат-боту 82

3.5 Висновки до розділу 98

ВИСНОВКИ 99

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 100

ДОДАТКИ 103

ПЕРЕЛІК УМОВНИХСКОРОЧЕНЬ

API – Application Programming Interface

UML – Unified Modeling Language

CLR – Common Language Runtime

SSMS – SQL Server Management Studio

IL – Intermediate Language

IIS – Internet Information Service

ADO – ActiveX Data Objects

LINQ – Language Integrated Query

MS – Microsoft

RDBMS – Relational Database Management System

JSON – JavaScript Object Notation

БД – База Даних

GUI – Graphical user interface

REST – Representational State Transfer

IB – Integration Bus

ПЗ – програмне забезпечення

CRUD – Create Read Update Delete

СКБД – система керування базами даних

NLP – Natural Language Processing

ВСТУП

Актуальність роботи

Сьогодні, ми живемо в епоху розвитку інформаційних технологій в багатьох сферах людської діяльності, а зокрема в освіті. Все більше університетів впроваджують сучасні системи для автоматизації освітніх процесів навчання, а також управління структурою університетів. І одними із таких засобів оптимізації стали впровадження чат-ботів, які стали невід'ємно важливим інструментом для покращення взаємодії студентів між собою та викладачами, а також для технічного адміністрування університету.

Сучасні чат-боти університетів містять критично важливі можливості для керування освітнім процесом університету, таких, як створення розкладу занять, перегляд та проставляння оцінок викладачами, нотифікації про новини та події університету, які можуть бути корисні для студентів та абітурієнтів. Вони можуть забезпечити швидкий доступ до важливої інформації, допомогти в організації навчального процесу, надати можливість перегляду графіку занять, успішності студентів, автоматизувати консультації та підтримку користувачів.

Проте, для досягнення високої ефективності роботи чат-ботів необхідна їх інтеграція з існуючими університетськими базами даних і сервісами, що дозволить їм автоматично отримувати дані з БД, обробляти їх та надавати актуальну інформацію користувачам чат-боту таку, як інформація про розклад занять, інформація про відвідування, інформація про розташування аудиторій та інше. Зазвичай, університети використовують декілька різних баз даних, такі як електронні журнали, системи управління навчанням (LMS), бібліотечні системи, а також інші внутрішні і зовнішні сервіси.

Враховуючи це все є доцільним розробити чат-боти, які здатні взаємодіяти з цими сервісами, оскільки, вони можуть значно знизити навантаження на адміністрацію та полегшити доступ до навчальних матеріалів для студентів та викладачів.

Інтеграція чат-ботів із університетськими системами зараз є важливим аспектом дослідження для розробників, так як потребує розв’язання низки проблем, таких як забезпечення безпеки передавання даних, швидкості обробки запитів та взаємодії з БД, надійності та масштабованості чат-ботів з зовнішніми системами. Тому надзвичайно важливим є дослідження моделей, методів та алгоритмів інтеграції чат-ботів з університетськими базами даних і сервісами.

Порівняння роботи з відомими розв’язаннями проблеми

На даний момент більшість інтеграцій чат-ботів із базами даних використовують набір API методів для отримання доступу до інформації, інтеграцію через проміжне програмне забезпечення (middleware) або пряме підключення до БД. Перевагою використання RESTful API полягає в тому, що одне і те саме API може використовуватися різними сервісами, кожний з яких може надавати свій власний інтерфейс та функціонал. Завдяки використанню API чат-боти можуть взаємодіяти із університетськими базами через стандартні протоколи. Проте, це також може спричинити негативні наслідки до певних обмежень стосовно швидкодії передачі даних та у збільшенні навантаження на сервери університету, особливо при великих кількостях одночасних запитів. Також є можливість ризику перехоплення даних, тому повинні бути вжиті всі механізми захисту інформації. Прикладом таких чат-ботів є “Чіт-кіт для вступників by LNTU”. Інтеграція через пряме підключення до БД є кращим по швидкодії ніж інтеграція через API, оскільки запити з чат-боту відправляються напряму в БД. Проте, недолік цього підходу полягає у масштабування на інші сервіси. В цьому випадку для кожного сервісу потрібно налаштовувати окрему логіку взаємодії з БД, що є доволі затратним по часу процесом. Прикладом таких чат-ботів є “ NUPP_Bot ”. Інтеграція через проміжне програмне забезпечення (middleware) також забезпечує можливість масштабування сервісів, як у випадку з API, проте комунікація між БД та чат-ботом відбувається не через стандартні API протоколи, а через ПЗ, яке взаємодіє з університетською базою даних. Це ПЗ

може бути на будь-якій мові програмування, наприклад C#, Java, Python, Node.JS. Недолік цього підходу також полягає можливій затримці на проміжному сервері, яке використовується для інтеграції системи з чат-ботом.

Проаналізувавши існуючі методи інтеграції чат-ботів з університетськими базами даних стає очевидним те, що розробка нових методів та алгоритмів механізмів інтеграції, які забезпечують зручну, безпечну та ефективну взаємодію чат-боту з БД є вагомим внеском у розвиток сучасних технологій.

Мета і задачі дослідження

Метою магістерської роботи є розробка моделей, методів та алгоритмів для інтеграції чат-ботів з університетськими базами даних та сервісами, що дозволить покращити взаємодію між ними та сприятиме автоматизації рутинних процесів, які використовуються в університетах. Досліджена модель повинна поєднювати кожну із переваг існуючих методів інтеграцій через пряме підключення до БД, використання API та використання проміжного програмного забезпечення (middleware). Також система повинна відповідати стандартам безпечного передавання даних та, щоб авторизація в систему відбувалася через двохфакторну авторизацію для перешкоджанню злому цієї системи. Окрім цього дана модель повинна забезпечувати високі показники по швидкодії та мати здатність до масштабування, наприклад, на різні платформи чат-ботів, такі, як Telegram, Viber, WhatsApp.

Досягнення мети включало розв’язання таких **задач**:

1. Аналіз існуючих підходів та моделей для інтеграції чат-ботів з різними системами;
2. Дослідження засобів обробки запитів для інтеграції з базами даних університету;
3. Розробка способів інтеграції з різними внутрішніми і зовнішніми сервісами університету;
4. Розробка алгоритмів безпечної передачі даних;

5. Програмна реалізація системи інтеграції чат-бота з університетськими базами даних та сервісами.

Об’єктом дослідження є процеси інтеграції чат-ботів з університетськими базами даних.

Предметом дослідження є архітектура, моделі та алгоритми для інтеграції чат-ботів з університетськими базами даних для розв’язання низки проблем, таких як забезпечення безпеки передавання даних, швидкості обробки запитів та взаємодії з БД, надійності та масштабованості чат-ботів з зовнішніми системами.

Методи дослідження

Для досягнення мети в розробці покращених моделей та алгоритмів інтеграції між БД та чат-ботом було проведено аналіз існуючих рішень інтеграцій та та розроблено рішення для покращення використання цих моделей завдяки поєднанню проміжного програмного забезпечення із API.

Наукова новизна одержаних результатів

Розроблено нові підходи для інтеграції чат-ботів з університетськими базами даних. Розроблено алгоритми за допомогою яких забезпечується безпека передачі даних між інтеграціями та проведено оптимізацію швидкодії обробки запитів.

Практичне значення одержаних результатів

На основі проведеного дослідження було розроблено архітектуру та модель інтеграції чат-боту з університетськими базами даних, які корисні для студентів, викладачів, абітурієнтів та адміністрацію університету. Завдяки цій інтеграції користувачі через чат-бот можуть переглядати та створювати розклад,

виставляти та переглядати оцінки, отримувати актуальну інформацію про навчальний заклад та багато чого іншого.

Особистий внесок

Проведено аналіз існуючих засобів для інтеграції чат-ботів з університетськими системами та базами даних, і на основі цих спостережень спроектовано нові алгоритми для інтеграції з БД, а також розроблено модель системи, що включає безпечну передачу даних та високий рівень швидкодії при обробці запитів.

Публікації

За результатами досліджень, проведених у межах магістерської роботи, підготовлено статтю для участі в науково-технічній конференції для наукового журналу «Інформаційні технології та суспільство» № 1 (12) за 2024 рік. Назва статті “Оптимізація навчального процесу університету за допомогою чат-бота”.

Структура магістерської роботи

Магістерська робота викладена на 139 сторінках друкованого тексту, який складається з вступу, чотирьох розділів, висновків, списку використаних джерел (50 найменувань). Робота містить 24 рисунки та 23 таблиці.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО ЧАТ-БОТИ ТА ІНТЕГРАЦІЮ ІЗ СИСТЕМАМИ УПРАВЛІННЯ БАЗАМИ ДАНИХ

1.1 Теоретичні відомості про чат-боти та їх застосування в різних сферах

1.1.1 Поняття чат-боту

Чат-боти (chatbots) — це програми, які за допомогою тексту або голосу можуть надавати відповіді користувачам на їхні питання. Вони можуть використовувати штучний інтелект (ШІ) для надання відповідей, можуть використовувати машинне навчання (ML) для аналізу отриманих відповідей та обробку природної мови (NLP), щоб надати змогу інтерактивного спілкування з користувачами. Основною метою чат-ботів є автоматизація процесів в різних сферах людського життя, полегшення комунікації між користувачами та оптимізація обслуговування користувачів в різних сферах людської діяльності [1].

1.1.2 Класифікація чат-ботів

Чат-боти можна класифікувати за декількома параметрами:

1) За функціональністю:
    - Сценарні (rule-based) – це примітивні чат-боти, які працюють за допомогою заздалегіть встановленої схеми та алгоритмів. Також вони є доволі простими у впровадженні;
    - Інтелектуальні (AI-powered) – ці типи чат-ботів використовують ШІ для аналізу введених або сказаних даних користувача та на основі цього аналізують відповіді користувача за деякими параметрами із використанням ШІ. Після аналізу отриманих даних ці чат-боти генерують відповіді на основі МН та ШІ [21].

2) За типом інтерфейсу:
   - Текстові — ці типи чат-ботів забезпечують взаємодію між користувачами через текстові повідомлення;
   - Голосові — ці чат-боти інтегровані з голосовими сервісами (наприклад, Siri, Alexa), які забезпечують можливість розпізнання мови а також голосових відповідей на питання користувачів;
   - Гібридні — ці типи чат-ботів комбінують текстовий і голосовий інтерфейси для комунікації з користувачами [21].
3) За сферою застосування:
   - Вузькоспеціалізовані — виконують чітко визначені завдання (наприклад, обслуговування клієнтів, бронювання). Зазвичай вони включають такі вузькі сфери, як готельний бізнес, банківські системи, освітні послуги, медицина та інші;
   - Універсальні — можуть виконувати широкий спектр функцій (наприклад, персональні помічники) користувачів для різних потреб [21].

1.1.3 Основні технології в чат-ботах

Основні технології в чат-ботах включають в себе:

- NLP (Natural Language Processing) — технологія процесингу природної мови аналізує отримані від користувача дані і визначає його сенс. Далі на основі цього аналізу формуються актуальні відповіді для користувачів [15];
- ML (Machine Learning) — технологія машинного навчання дозволяє чат-боту самостійно навчатися на основі даних, які він отримує від користувача. Навчання в цьому випадку відбувається динамічно із накопиченням здобутих від користувача знань, які система записує у свою базу знань, за допомогою якої обробляються наступні відповіді користувачів на схожі запитання [15];

- API (Application Programming Interface) — технологія інтерфейсу прикладного програмування забезпечує інтеграцію з базами даних та іншими сервісами комунікації між програмами [15];
- Хмарні технології — ці технології використовуються для обробки даних і зберігання інформації за допомогою провайдерів хмарних рішень таких, як: AWS, GCP, Azure, Heroku та інші. Принцип функціонування цієї технології полягає в тому, що для обслуговування таких чат-ботів не потрібні фізичні сервери, так як дані зберігаються в хмарних сховищах. Проте, платити за них доведеться за споживання [33].

1.1.4 Застосування чат-ботів у різних сферах

Чат-боти можуть використовувати в таких сферах людської діяльності, як:

1) Освіта – прикладом використання чат-ботів в освіті може бути автоматизація розкладів студентів, оцінювання та перевірка знань студентів. Також важливими аспектами використанян чат-ботів в освіті є підтримка різних типів користувачів, таких, як студенти, викладачі, абітурієнтів, адміністрації університетів, коледжів, ліцеїв, гімназій, ПТУ та шкіл;
2) Бізнес – ця сфера включає в себе множину підсфер, таких, як медицина, промислове виробництво, сільське господарство, сфера послуг, ресторанний бізнес, страхові компанії та інші установи, які призначені для отримання бізнесу. В цій сфері чат-боти призначені для полегшення користувачам взаємодії з сервісами компаній, що сприяє для бізнесу залучення клієнтів та збільшення прибутку;
3) Охорона здоров'я - в цій сфері використанню чат-ботів велике місце приділяється попередньому діагностуванню хвороб, моніторингу стану здоров’я пацієнтів, а також полегшення взаємодії між пацієнтами та клініками для спрощення таких процедур, як запису пацієнтів на візит;

4) Громадські сервіси – чат-боти в даній сфері призначені для підтримки громадян стосовно громадських питань, автоматизацію процесу запису на прийом до органів державної влади та іншу діяльність в державоуправління та надання громадських послуг;
5) HR та рекрутинг – чат-боти цього типу призначені для процесу покращення вибору кандидатів та автоматизації процесу онбордингу нових співробітників [17].
6) Розваги – чат-боти в даній сфері надають змогу для проведення вікторин та надають користувачам інтерактивну допомогу у відеоіграх.

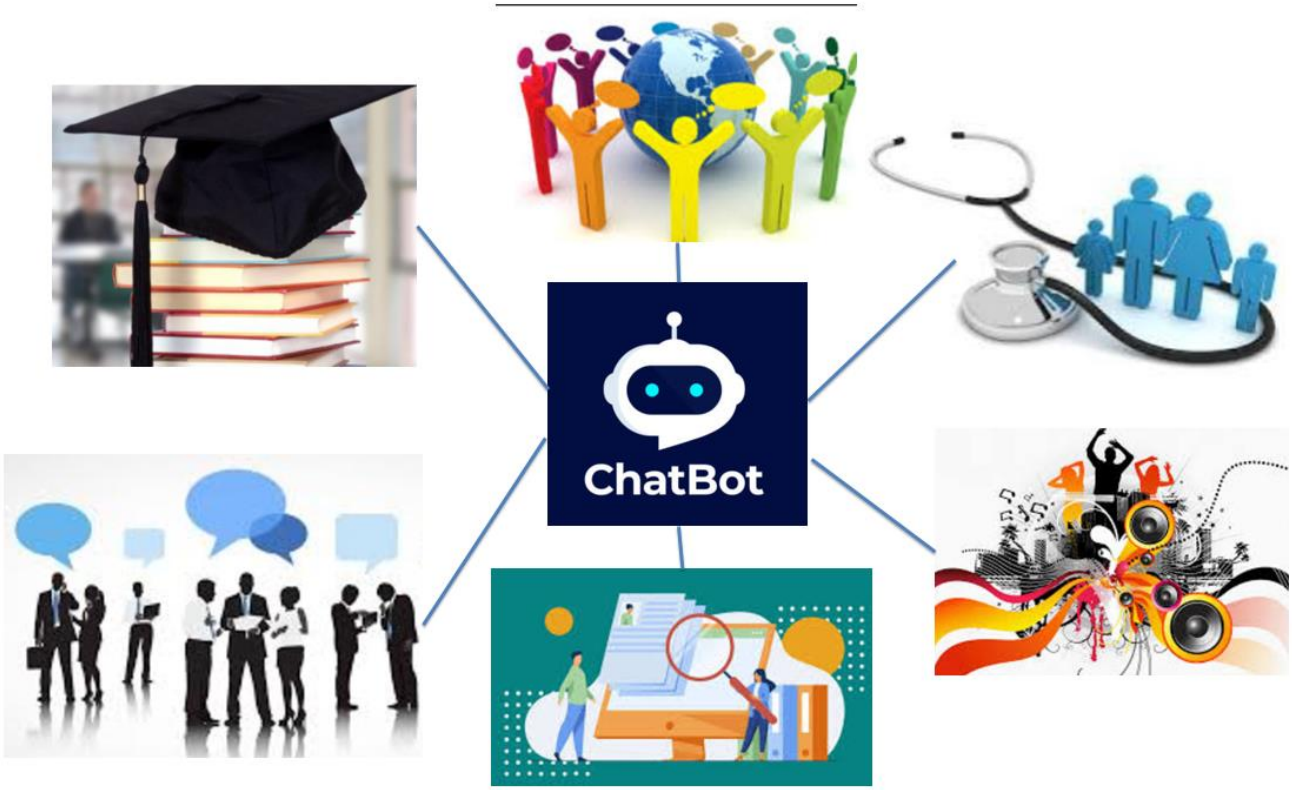

Рис. 1.1. Сфери використання чат-ботів

Окрім зазначених на рис. 1.1 основних сфер використання чат-ботів їх також часто використовують у фінансовій та страховій сферах а також використовують для особистих цілей, таких, як планування сімейного бюджету, ведення звітностей по доходах та витратах комунікація між користувачами в межах встановлених груп та інше [32-36].

1.2 Моделі чат-ботів, правила та машинне навчання

1.2.1 Моделі використання чат-ботів

Моделі, які використовуються для створення чат-ботів, базуються на трьох основних підходах їх розробки: правилоорієнтованих (rule-based), машинному навчанні (machine learning) та комбінований, який включає елементи правилоорієнтованих (rule-based) та чат-ботів на машинному навчанні (machine learning). Кожен з цих підходів має свої переваги, недоліки та області їх застосування [2].

1.2.2 Поняття правилоорієнтованих (rule-based) моделей чат-ботів

Правилоорієнтовані моделі використовують ті чат-боти, які створені на основі правил, працюють за заздалегідь визначеними сценаріями та алгоритмами, які в них закладають розробники. Вони використовують набір правил і умов, за якими обробляється введена користувачем інформація та на основі цих даних генерують відповіді. Правилоорієнтовані чат-боти є більш примітивними та обмежені у функціоналі і не можуть обробити всі запити користувачів, особливо нестандартні питання [16].

1.2.3 Основні характеристики правилоорієнтованих (rule-based) моделей чат-ботів

Основними характеристиками цих чат-ботів є:

- Детермінованість: відповіді є чітко вираженими в залежності від запиту, оскільки ці відповіді генеруються на основі заздалегіть прописаних алгоритмів;
- Сценарність: кожен варіант взаємодії має бути передбачений наперед та для кожного сценарію повинен бути прописаний свій механізм та алгоритм взаємодії з користувачем;
- Передбачуваність: всі правила повинні бути встановлені наперед:

- Простота у реалізації: ці чат-боти не потребують великих обчислювальних ресурсів для алгоритмів обробки запитів користувачів [3].

1.2.4 Переваги, недоліки та варіанти використання правилоорієнтованих (rule-based) моделей чат-ботів

Основні переваги правилоорієнтованих (rule-based) моделей чат-ботів:

- Висока надійність у вузькоспеціалізованих задачах – наприклад коли потрібно розробити чат-бот для надання можливостей запису на візит пацієнтом, що є вузькою задачею, то такий тип чат-боту буде оптимальним вибором. Оскільки такий процес включає заздалегіть сплановані сценарії, такі, як вибір послуги, вибір лікаря та дати і часу візиту;
- Простота налаштування та тестування – ці типи чатботів простіші в реалізації та тестуванні, оскільки орієнтовані здебільшого на вузьку задачі із можливістю написаня алгоритмів для більшості варіантів використання. Також тестування є відносно легким, оскільки, дії є передбачуваними, так як все відбувається за заздалегіть описаними алгоритмами, які заклали розробники програмного забезпечення;
- Легкість контролю за логікою роботи – оскільки ці чат-боти працюють за заздалегіть пропрацьованою моделлю та алгоритмами то результат здебільшого є очікуваний та відповідає логіці, яка була закладена в основу алгоритмів даних чат-ботів [2].

Основні недоліки правилоорієнтованих (rule-based) моделей чат-ботів:

- Обмеженість функціоналу через складність охоплення всіх можливих сценаріїв – оскільки сценарії потрібно заздалегіть опрацьовувати то всі сценарії фактично не можливо пропрацювати. Тому дані чат-боти мають обмежений функціонал та орієнтовані на вузькоспеціалізовані сфери та задачі для яких вони призначені;

- Відсутність гнучкості у відповіді на непередбачувані запити – оскільки алгоритми в цих чатботах повинні бути оброблені заздалегіть, то вони не можуть достатньо правильно відповідати на непередбачувані запитання на які не пропрацьовано алгоритми відповідей [18].

1.2.5 Застосування правилоорієнтованих (rule-based) моделів чат-ботів

Правилоорієнтовані чатботи зазвичай використовують у автоматизації процесів FAQ-сервісів. Прикладами використання цих чатботів є:

- Чат-бот клініки, за допомогою якого можна записатися на візит, переглянути свої візити та документи. Чат-бот може надавати змогу оплачувати візити через інтеграцію з платіжними системами, а також надавати змогу зручної комунікації між пацієнтами та лікарями;
- Чатбот університету, за допомогою якого можна надавати інформацію різній категорії користувачів таким, як студенти, викладачі, адміністрації університету та абітурієнтам. Може надавати студентам можливість переглядати розклад, свою успішність, а також інформацію про структуру університету та новини університету. Для адміністрації чат-бот може надати можливість керування розкладом, налаштування персоналу, налаштування нотифікацій про новини університету та інше. Для викладачів може бути корисним у наданні можливостей оцінювання студентів, перегляду розкладу та інтерактивній комунікації з студентами. Для абітурієнтів може бути корисним, щоб дізнатися інформацію про вступну кампанію та новини університету;
- Чатбот банку, за допомогою якого можна здійснювати транзакції, такі, як оплати комунальних послуг, поповнення мобільних телефонів, перекидування коштів на інші карточки, оплати товарів та послуг, перегляду історії транзакцій та багато іншого [37].

1.2.6 Чат-боти, які працюють на базі машинного навчання

Чат-боти, які працюють на базі машинного навчання, використовують алгоритми для аналізу даних, навчаються на цих даних і призначені для створення гнучких рішень для надання відповідей, як для стандартних так і для нестандартних питань користувачів. Вони здатні розуміти контекст та адаптовуватися до нових непередбачуваних ситуацій та на основі контексту даних приймати рішення для надання актуальних відповідей користувачам [3].

Основні характеристики чат-ботів машинного навчання:

- Адаптивність - здатність до навчання на основі відповідей користувача та вдосконалення процесу генерування відповідей;
- Контекстуальність – здатність чат-боту запам’ятовувати попередні розмови та на основі цих розмов генерувати актуальні відповіді;
- Генеративність – надає можливість створювати нові відповіді під час розмови чи переписки з користувачем, а не тільки обирати з наявних. Така динамічність у наданні згенерованих відповідей дозволяє відповідати на нестандартні запитанні на які заздалегіть важко спланувати відповіді [4].

1.2.7 Типи моделей чат-ботів на базі машинного навчання

На базі машинного навчання можемо класифікувати боти на такі типи:

1) Нейронні мережі:
   - Рекурентні нейронні мережі (RNN) – ці моделі використовуються для обробки послідовних даних таких як текст, звук. При цьому вони зберігають контекст через зворотні зв’язки. Вони добре підходять для завдань перекладу чи прогнозування;
   - Трансформери (наприклад, GPT) - сучасна архітектура для роботи з послідовностями, яка використовує механізм уваги (attention) для ефективного аналізу довгих контекстів. Одним із прикладу таких моделей є чат GPT, який здатний генерувати повноцінний текст [4].

2) Методи обробки природної мови (NLP):
   - Векторизація тексту (word2vec, GloVe) – це методи перетворення слів у числові вектори, які відображають їх значення та взаємозв’язки один між одним в багатовимірному просторі;
   - Аналіз настроїв і семантики – інструменти, які визначають емоційність тексту, можуть відображати позитивний, негативний, нейтральний види та може аналізувати значення слів у контексті [4].

1.2.8 Переваги та недоліки чатботів на базі машинного навчання

Основні переваги чатботів на базі машинного навчання є:

- Гнучкість та універсальність - чат-боти на базі машинного навчання здатні адаптовуватися до різних сценаріїв та навчатися з досвіду введення даних користувачів, що дозволяє використовувати їх у різних галузях, від обслуговування клієнтів до медицини та освіти;
- Можливість обробляти великі обсяги інформації – ці боти можуть швидко аналізувати та обробляти великі обсяги даних, надаючи відповіді чи рекомендації на основі актуальної інформації, що зберігається у великих базах даних, оскільки вони зберігають та аналізують попередні історії чатів користувачів і таким чином збільшують свою базу знань;
- Розуміння складних запитів і врахування контексту - завдяки сучасним методам обробки природної мови (NLP) дані боти можуть розуміти багатозначність, аналізувати складні запити, розпізнавати контекст розмови навіть за умов коли користувач ввів незначні граматичні або пунктуальні помилки та надавати відповіді [31].

Основні недоліки чат-ботів на базі машинного навчання є:

- Високі вимоги до обчислювальних ресурсів - навчання та робота таких ботів потребують потужних серверів або хмарних платформ для обробки великих обсягів даних, що коштує дорого;

- Необхідність великих обсягів даних для навчання - для забезпечення високої точності та функціональності, цим чат-ботам потрібно мати великі та якісні набори даних, в їхній базі знань. Збір цих даних та обробка може бути складним і затратним процесом в часі та ресурсах;
- Потенційна помилковість через невірне навчання - якщо модель була навчена на неправильних даних, то вона може генерувати помилкові або етично неприйнятні відповіді. Це особливо критично у чутливих сферах, як медицина чи освіта де ціна помилки є доволі висока [39].

1.2.9 Застосування чатботів на базі машинного навчання

Дані чатботи найчастіше використовують у якості інтерактивних помічників, наприклад голосові асистенти, такі як Alexa чи Google Assistant, вони здатні відповідати на широкий спектр запитань, включаючи управління пристроями розумного дому, пошук потрібної інформації та організацію завдань різних сфер та складностей реалізації.

Також такі чат-боти часто використовують для підтримки клієнтів у великих компаніях, оскільки вони здатні проводити автоматизацію обробки запитів клієнтів у банках, інтернет-магазинах або службах технічної підтримки. Такі боти здатні працювати 24/7 і знижувати навантаження на працівників компанії [4].

Ще одним важливим прикладом використання чатботів з використанням машинного навчання є особисті освітні та медичні асистенти, які можуть надавати студентам допомогу у вивченні матеріалів, організовувати навчання чи допомагати підготовлюватися до іспитів.

У медицині вони допомагають пацієнтам отримувати інформацію про симптоми, нагадування про прийом ліків чи розклад консультацій.

Важливо зауважити, що чат-боти на основі машинного навчання продовжують розвиватися, і їх застосування поступово охоплює дедалі більше

сфер, що дозволяє вирішувати складні задачі та підвищувати якість обслуговування [16].

1.2.10 Порівняльна характеристика різних моделей чат-ботів

В таблиці нижче наведено порівняльні характеристики основних моделей чатботів, а саме: правилоорієнтованих, які працюють за наперед встановленими алгоритмами та моделей на основі машинного навчання, які навчаються як відповідати на питання від користувача.

Таблиця 1.1

Порівняльна характеристика правилоорієтованої моделі та моделі машинного навчання чат-ботів.

Характеристика	Правилоорієнтовані моделі	Моделі машинного навчання
Гнучкість	Низька	Висока
Необхідність даних	Не потребують великих даних	Потребують великих даних
Швидкість впровадження	Швидка	Середня/повільна
Вартість розробки	Низька	Висока
Застосування	Вузькоспеціалізовані задачі	Широкий спектр задач

1.2.11 Комбінований підхід моделей чат-ботів

Комбінований підхід поєднує сильні сторони правилоорієнтованих моделей надання відповідей і методів машинного навчання, та створює баланс між точністю, гнучкістю та ефективністю надання відповідей на запитання. Такий підхід дозволяє забезпечити базову функціональність, що є корисним у випадку нестандартних питань, а також відповідати на нестандартні запитання при якому буде відпрацьовувати спосіб машинного навчання [24].

Особливості комбінованих підходів, полягає в низці основних чинників:

- Правилоорієнтовані компоненти - використовуються для обробки типових запитів, які легко формалізувати та обробляти. Також вони забезпечують надійність та точність у ситуаціях, де чітко визначені сценарії (наприклад, відповіді на часті та стандартні запитання);
- Компоненти машинного навчання - відповідають за обробку складних та багатозначних запитів із врахування контексту. Призначені в основному для надання відповідей на нестандартні запитання. Вони навчаються на основі історичних даних, дозволяючи покращувати відповіді з часом [29].

Переваги комбінованого підходу є такі:

- Гнучкість - машинне навчання забезпечує адаптивність у незапланованих сценаріях, а правила та алгоритми підтримують функціональність у стандартизованих випадках;
- Зниження помилок - у складних запитах машинне навчання допомагає уникати помилок, пов’язаних із жорсткими правилами;
- Оптимізація ресурсів - частину завдань обробляють правилоорієнтовані алгоритми, що зменшують навантаження на складні моделі машинного навчання і цим зменшується споживання ресурсів
- Можливість поєднання різних технологій;
- Надає можливість гібридного підходу надання відповідей.

Недоліки комбінованого підходу є такі:

- Складність розробки - необхідно розробити та інтегрувати два різних підходи, що вимагає додаткових ресурсів і знань. А також збільшує вартість розробки та впровадження таких чатботів.
- Підтримка та масштабування - комбіновані системи складніше підтримувати через наявність двох паралельних механізмів роботи обробки запитів користувачів [29].

1.2.12 Застосування комбінованих підходів

Комбіновані моделі чатботів зазвичай використовуються для таких випадків:

- Клієнтська підтримка - для стандартних запитів (робочий час, статус замовлення) використовуються правила, а для складних питань — адаптивні моделі.
- Навчальні боти - правила забезпечують базову взаємодію зі студентами (розклад, доступ до матеріалів), а ШІ допомагає аналізувати складніші питання чи контекстні потреби.
- Медичні чат-боти - правила обробляють типові симптоми, тоді як моделі машинного навчання аналізують незвичайні комбінації симптомів або надають додаткові рекомендації [36].
- Чат-боти у фінансовій сфері, які призначені для надання підтримки користувачам. Вони можуть надавати допомогу самостійно за допомогою наперед встановлених правил на базові запитання, або надавати методи за допомогою залучення штучного інтелекту для обробки нестандартних запитань.

Комбінований підхід забезпечує високу ефективність, дозволяючи чат-ботам вирішувати як типові, так і нестандартні завдання. Він особливо корисний у системах, що потребують надійності, масштабованості та адаптивності [23].

1.3 Основи інтеграції чат-ботів з системами управління базами даних університетів

1.3.1 Моделі інтеграції чат-ботів з системами управління базами даних університетів

Інтеграція чат-ботів із системами управління базами даних СУБД університетів є важливим кроком для автоматизації обслуговування користувачів різних типів, таких, як абітурієнтів, студентів, викладачів та

адміністративного персоналу. Це дозволяє забезпечити доступ до актуальної інформації, автоматизувати рутинні операції та покращити взаємодію між користувачами та університетськими службами [38].

Особливості університетських баз даних полягають в їхній структурі. Ці БД зазвичай містять наступні категорії даних:

- Інформація про студентів (контакти, академічна успішність, статус навчання, історія здачі сесій та інше);
- Дані про викладачів (розклади, робоче навантаження, профілі, напрямки діяльності, наукові ступені, регалії та інше) ;
- Навчальні програми, заліки, сесії, групи та курси;
- Розклади занять, екзаменів, заходів;
- Фінансова інформація (оплата навчання, стипендії) [38].

Університетські БД можуть використовувати, як реляційні бази даних (наприклад, MS SQL Server, MySQL, PostgreSQL, Oracle), так і нереляційні (NoSQL, наприклад, MongoDB) для зберігання структурованих і неструктурованих даних.

Безпека в університетських СУБД забезпечується відповідно до законодавства про конфіденційність даних та повинна бути шифрованою при передачі даних через API для того, щоб перешкодити отримання неавторизованого доступу [44].

Архітектура інтеграції чат-ботів із СУБД полягає в таких основних компонентах:

- Інтерфейс користувача - взаємодія зі студентами, викладачами, абітурієнтами чи адміністративним персоналом через текстовий або голосовий канал, який зручний для користувачів;
- Обробник запитів (NLP-двигун) – тут відбувається аналіз запиту, визначення його мети та передача запиту до відповідного модуля системи. Далі система аналізує ці запити та на основі аналізу приймає рішення у відповідях на конкретний запит;

- Сервіс інтеграції - є своєрідним містом між чат-ботом і СУБД, який забезпечує доступ до даних через API;
- СУБД - система, яка зберігає дані університету та відповідає на запити [49].

1.3.2 Механізм інтеграції чатботу з БД

Механізм інтеграції чатботу з БД полягає в тому, що користувач формулює запит (наприклад, "Покажи мій розклад на наступний тиждень").

NLP-двигун визначає намір запиту та ключові дані (наприклад, "розклад", "наступний тиждень"). На основі цих даних сервіс інтеграції відправляє запит до СУБД через API. Після цього СУБД повертає необхідні дані. Чат-бот, на основі цих даних, формує відповідь і надає її користувачу.

Підключення інтеграції чат-ботів до СУБД може здійснюватися за допомогою прямого підключення до СУБД. В цьому випадку чат-бот напряму звертається до бази даних через SQL-запити. Перевагою цього підходу є висока швидкість відповіді та простота реалізації для невеликих систем. Але цей підхід також має деякі недоліки, найкритичнішими з яких є ризик безпеки та складність масштабування.

Інший підхід підключення інтеграції чат-ботів до СУБД полягає у використання API. В цьому випадку СУБД надає доступ до даних через REST або GraphQL API. Далі в чатботі замість прямих викликів до бази даних запити відпрацьовують через набір API методів, які забезпечують CRUD операції. Перевагами цього підходу є безпека, можливість інтеграції з іншими сервісами, а також масштабованість. Коли в нас є набір API методів, який містить CRUD (create, read, update та delete) операції до БД ми можемо його використати в будь-яких чат-ботах, та платформах, наприклад, Telegram, Viber, WhatsApp та інших. Недоліки цього підходу полягають у тому, що система потребує розробки додаткового шару API, який реалізується за допомогою проміжного програмного забезпечення [42, 43].

1.3.3 Вимоги до інтеграції чатботу з БД

Вимоги до інтеграції чатботів з БД обов’язково повинні включати такі аспекти:

1) Безпека даних:
   - Шифрування – потрібне використання сучасних алгоритмів шифрування для передачі даних, таких як TLS, щоб запобігти їх перехопленню зловмисниками чи модифікації; Шифрування зазвичай відбувається на рівні middleware;
   - Аутентифікація та авторизація - реалізація механізмів ідентифікації користувачів (паролі, двофакторна аутентифікація) та обмеження доступу до даних залежно від рівня прав.
2) Масштабованість - система повинна бути здатна обробляти велику кількість запитів одночасно без зниження продуктивності, що забезпечується через хмарні технології або балансування навантаження. Також опціонально повинна бути можливість масштабування робити з БД через API з різними сервісами чат-ботів, таким, як Telegram, Viber, WhatsApp, Facebook та інші сервіси. Також масштабованість може в себе включати розробку зовнішніх застосунків;
3) Сумісність - інтеграція має враховувати особливості існуючих університетських систем, таких як СУБД, наприклад Mongo DB, Cassandra, CRM або LMS. Необхідно використовувати API чи інші методи для забезпечення безперебійної роботи з різними компонентами.
4) Логування та моніторинг - полягає у впровадженні запису дій системи для виявлення потенційних проблем, аналізу помилок та забезпечення прозорості. Реалізація моніторингу продуктивності повинна здійснюватися з використанням відповідних інструментів (наприклад, Prometheus, ELK Stack або власними розробленими сервісами логувань).

1.3.4 Вимоги до інтеграції чатботу з БД

Типовими прикладами використання інтеграції чатботів з університетськими базами даних є

1) Інформаційні запити:
    - Автоматизація перегляду розкладу занять, іспитів, новин університетів чи заходів через чат-бот;
    - Запити статусу виконання завдань, таких як здача курсових чи лабораторних робіт, перегляд оцінок;
    - Для викладачів можливість виставлення оцінок студентів через інтерфейс чат-боту;
    - Для адміністрації створення та можифікація розкладу, налаштування розсилок та інші функціональні можливості для управління університетом [42].
2) Адміністративні задачі:
    - Подання заяв на переведення, академічну відпустку чи зміни групи через інтерактивний інтерфейс чатботу;
    - Замовлення довідок (наприклад, про навчання) та інших офіційних документів із подальшим інформуванням про статус запиту;
    - Оновлення даних студентів та викладачів;
    - Збір статистики [42].
3) Навчальні сервіси:
    - Реєстрація студентів на курси або факультативи з урахуванням їх академічного плану;
    - Пошук навчальних матеріалів у базах даних або бібліотеках університету;
    - Інтеграція чат-боту з міжнародними та національними освітніми платформами [42].

Ці функції дозволяють спростити доступ до даних та послуг університету, підвищуючи ефективність роботи адміністрації й комфорт користувачів.

Інтеграція чат-ботів із університетськими СУБД забезпечує значну автоматизацію процесів і покращує доступність інформації для всіх учасників навчального процесу. Водночас успішна реалізація таких рішень вимагає ретельного планування, врахування вимог безпеки та дотримання найкращих практик розробки [44].

1.4 Огляд сучасних сервісів для інтеграції чат-ботів та можливості їх застосування у навчальних закладах

1.4.1 Огляд сучасних сервісів для інтеграції чат-ботів та можливості їх застосування у навчальних закладах

Сучасні сервіси для розробки та інтеграції чат-ботів значно спрощують процес створення функціональних рішень для навчальних закладів. Вони надають готові інструменти для розробки таких чатботів, обробки запитів, інтеграції з базами даних та зовнішніми сервісами, що робить їх зручними для швидкої розробки та впровадження [13].

1.4.2 Основні категорії сервісів для створення чат-ботів

Основні категорії сервісів для створення чат-ботів:

1) Конструктори чат-ботів (no-code/low-code платформи) – ці чат-боти призначені для розробки чат-ботів без необхідності програмування. Приклади таких чат-ботів: ManyChat, Chatfuel, Tars. Їх основні переваги є простота у використанні та інтуїтивний інтерфейс. Недоліками таких чат-ботів є обмежена функціональність для складних інтеграцій, оскільки вони обмежені функціоналом платформи на якій вони розробляються. Для університетів це дає можливості швидкого створення FAQ-ботів для відповіді на типові запити студентів такі, як розклад, вступна інформація, успішність студентів та інше.

2) Платформи на основі ШІ – ці платформи використовують алгоритми машинного навчання для розуміння природної мови та контексту запитів. Вони

здатні обробляти більш складні та нестандартні питання. Прикладами таких чатботів є: Dialogflow (Google), Microsoft Bot Framework та IBM Watson Assistant. Їх основні переваги це інтеграція з NLP та можливість обробки складних запитів. Недоліками цих чат-ботів є вищі вимоги до ресурсів і складність розробки та налаштування [13].

Для університетів вони можуть бути використаними в якості інтелектуальних помічників для реєстрації на курси, перегляду оцінок чи пошуку навчальних матеріалів.

3) Хмарні платформи інтеграції – ці чат-боти надають API для зв’язку з базами даних і зовнішніми сервісами. Прикладами таких сервісів є AWS Lex, Azure Bot Service та Rasa. Переваги цих сервісів включають гнучкість у кастомізації та масштабованість. Основними недоліками хмарних сервісів розробки чат-ботів є потреба в технічній експертизі для налаштування. Для університетів це надає можливості розробки чат-ботів для обробки запитів студентів та викладачів із доступом до університетських СУБД [13].

4) Омніканальні платформи – ці технології дозволяють створювати ботів, які працюють на різних каналах (Telegram, WhatsApp, Facebook Messenger). Приклади таких сервісів є Twilio та Landbot. Їх основними перевагами є доступність для широкої аудиторії. Серед основних недоліків є обмежена підтримка глибокої інтеграції з університетськими системами. Для університетів вони надають можливості комунікація зі студентами через зручні для них канали зв’язку [14].

1.4.3 Можливості застосування сучасних сервісів у навчальних закладах

Серед основних можливостей застосування сучасних сервісів у навчальних закладах виділяють такі основні можливості:

1) Автоматизація обслуговування студентів, яка може послужити для:
   - Надання інформації про розклад занять, екзаменів, сесій;
   - Підказки щодо процедур вступу чи переведення;

- Відповіді на типові запитання через FAQ-боти.

2) Підтримка навчального процесу, яка призначена для:
   - Допомоги студентам у виборі курсів та реєстрації на них;
   - Інформація про доступні ресурси (бібліотеки, лабораторії);
   - Рекомендації щодо навчальних матеріалів.
3) Управління адміністративними задачами, яке призначена для:
   - Збору даних про студентів для анкетування чи опитувань;
   - Нагадування про дедлайни (оплата навчання, подання документів);
   - Нотифікація про події університету;
   - Сприяння організації заходів (вебінари, конференції).
4) Інклюзія та доступність, яка призначена для:
   - Надання підтримки студентам з особливими потребами;
   - Переклади на різні мови для іноземних студентів;
   - Адаптація для голосових інтерфейсів.
5) Підтримка викладачів та персоналу, яка призначена для:
   - Спрощення взаємодії з адміністрацією;
   - Організація робочих графіків і планування занять;
   - Керування процесом виставлення оцінок;
   - Швидкий доступ до статистики успішності студентів [28].

Таблиця 1.2

Основні сучасні сервіси розробки чатботів.

Сервіс	Особливості	Використання в університетах
Dialogflow	Потужний NLP-двигун від Google, який надає можливість інтеграції з Google Cloud.	Інтелектуальні помічники для студентів і викладачів.

Продовження таблиці 1.2

Microsoft Bot Framework	Широка підтримка різних каналів комунікації, інтеграція з Azure.	Розробка багатофункціональних чат-ботів для університетських задач.
Rasa	Open-source платформа з підтримкою кастомізації та навчання моделей.	Чат-боти з глибокою інтеграцією в університетські СУБД.
ManyChat	Простий у використанні конструктор, орієнтований на маркетинг.	FAQ-боти для швидкої відповіді на запитання абітурієнтів.
Twilio	Омніканальність, інтеграція з SMS та іншими комунікаційними каналами.	Сповіщення студентів через SMS або месенджери.
IBM Watson Assistant	Потужний ШІ, інтеграція з іншими сервісами IBM.	Помічники для пошуку інформації та рекомендацій для студентів.

1.5 Огляд існуючих методів забезпечення безпеки даних при інтеграції чат-ботів

1.5.1 Аналіз існуючих методів та моделей забезпечення безпеки при інтеграції чат-ботів з університетськими базами даних

Інтеграція чат-ботів із системами університетів потребує особливої уваги до захисту даних через обробку конфіденційної інформації різних типів

користувачів, такої як персональні дані студентів, викладачів, розклади та академічні записи, які не можуть бути поширені для загального перегляду. Захист інформації є важливим для забезпечення довіри користувачів та дотримання нормативних вимог [14].

Основні загрози безпеки полягають у таких видах:

- Несанкціонований доступ: це можливість проникнення до баз даних через вразливості чат-бота, тому при підключенні до БД потрібно вжити всіх необхідних заходів для забезпечення безпеки даних;
- Перехоплення даних під час передачі: може виникати при використання незахищених каналів для обміну інформацією, наприклад через API або проміжне програмне забезпечення (middleware).
- Соціальна інженерія: виникає при використання психологічних методів для отримання доступу до конфіденційної інформації. Такі атаки здійснюються на самих користувачів, а не на систему.
- Помилкова автентифікація: виникає при ненадійних механізмах авторизації та автентифікації користувачів. Для вирішення цієї проблеми можна в чат-боті налаштувати двохфакторну авторизацію, яка при кожному вході буде запитувати в користува код підтвердження, яку йому відправило, наприклад, на мобільний номер телефону або електронну пошту.
- Невірне управління доступом: виникає при неправильній сегментації прав доступу до системи. Відповідно для вирішення цієї проблеми можуть бути використані ролі користувачів із різними правами доступу, такими, як абітурієнт, студент, викладач, адміністратор закладу та інші [12].

1.5.2 Існуючі методи забезпечення безпеки даних

Серед основних методів забезпечення безпеки даних є:

1) Шифрування даних – цей метод використовує алгоритми шифрування для захисту інформації під час її передачі та зберігання.

Прикладами таких методів є:

- TLS (Transport Layer Security) для захисту даних під час передачі через мережу;
- AES (Advanced Encryption Standard) для зберігання чутливої інформації.

Перевага даного методу полягає в тому, що він запобігає перехопленню даних зловмисниками. Але недоліком цього методу є те, що він має високі вимоги до апаратного забезпечення.

2) Аутентифікація користувачів – завдяки даному методу система перевіряє користувача та на основі даних користувача визначає його права. Прикладами цього методу є паролі, двохфакторна та багатофакторна авторизація, біометричні дані, які використовуються для входу у систему, використання протоколів, таких як OAuth 2.0 або OpenID, які призначені для налаштування безпечного підключення до системи. Зазвичай, цей метод використовується для захисту від несанкціонованих доступів до системи. Серед його основних недоліків є ризик при використанні слабких паролів або викрадення токенів зловмисниками.

3) Контроль доступу – цей метод надає доступ до системи тільки авторизованим користувачам за допомогою токена, який встановлюється при авторизації та є унікальний для кожної сесії. Користувачі, які мають чітко визначені права, наприклад можуть використовуватися ролі: студент, викладач, абітурієнт, адміністратор та інші. Його, зазвичай, використовують для мінімізації ризиків доступу до інформації користувачам, які призначені для використання іншого функціоналу системи відповідно до їхніх ролей. Проте, недоліком цього методу є можливі помилки, які можуть виникати в процесі налаштування політик доступу для різних користувачів. Ці помилки можуть бути допущені самими користувачами при налаштуванні цих прав доступу.

4) Виявлення та запобігання загрозам – завдяки цьому методу відбувається моніторинг активності чат-бота, яке може допомогти виявити підозрілі дії, наприклад, виклик однотипних запитів або коли є підозріла активність викликів під конкретними користувачами. Такий моніторинг можна реалізувати, наприклад, за допомогою таких систем, як SIEM-систем, які надають можливість аналізувати логи та події активності чат-боту. Серед його переваг слід виділити раннє виявлення потенційних атак. Недоліком даного методу є складність у налаштуванні та аналізі.

5) Логування та аудит - цей метод полягає у налаштуванні запису дій користувачів системи для аналізу помилок, які виникають при використанні системи та виявлення порушень. Логування дій може відбуватися, як у звичайні текстові файли так і в певну базу знань, яка встановлена розробниками системи. Основні переваги полягають у прозорості операцій та швидкому виявленні несанкціонованих дій. Серед недоліків є великий обсяг даних для зберігання, який може впливати на збільшення вартості обслуговування такої системи.

6. Політики конфіденційності – забезпечує відповідності обробки даних законодавчим нормам, таким як GDPR (General Data Protection Regulation) або CCPA (California Consumer Privacy Act). Переваги полягають у дотриманні нормативних вимог і підвищення довіри користувачів. Серед основних недоліків цієї системи варто виділити високі витрати на забезпечення відповідності [28].

1.5.3 Рекомендації з безпеки при інтеграції чат-ботів

При налаштуванні інтеграції чат-ботів необхідно вжити заходів по забезпеченню безпеки передачі даних між системою та користувачами .

Основні рекомендації до безпеки включають такі методи:

- Реалізувати двохфакторну або багатофакторну авторизацію для збільшення безпеки доступу до облікових даних користувачів;
- Використовувати сучасні протоколи шифрування даних;

- Проводити регулярне оновлення системи для того, щоб запобігти відомим вразливостям;
- Налаштувати логування системи та проводити моніторинг системи.

1.6 Висновки до розділу

У цьому розділі було досліджено теоретичні аспекти розробки чат-ботів, їх моделі та особливості інтеграції з університетськими БД і сервісами за допомогою яких їх можна розробляти, тестувати та впроваджувати. Було проведено аналіз сучасних інструментів автоматизації чатботів та їх моделей.

Інтеграція чат-ботів із СУБД університетів сприяє автоматизації багатьох процесів: управління розкладами, доступ до академічної інформації, обслуговування студентів і викладачів. Основними викликами є забезпечення безпеки даних, масштабованість та відповідність нормативним вимогам.

Сучасні платформи, такі як Rasa, Microsoft Bot Framework, спрощують розробку та впровадження чат-ботів. Їх застосування дозволяє створювати функціональні рішення, які адаптовані під потреби освітніх.

Використання чат-ботів у навчальних закладах дозволяє автоматизувати рутинні задачі, покращити доступність інформації та спростити комунікацію між студентами, викладачами та адміністрацією. Сервіси для інтеграції чат-ботів пропонують широкий спектр можливостей, які можна використовувати у навчальних закладах. Їх застосування дозволяє оптимізувати навчальний процес та обслуговування студентів та підвищити ефективність управління університетом.

Отже, розвиток технологій чат-ботів і їх інтеграція з університетськими БД відкриває нові перспективи для автоматизації процесів у сфері освіти. Завдяки їх застосуванню можна значно підвищити ефективність управління навчальними закладами та якість обслуговування користувачів. Це обґрунтовує актуальність дослідження даної теми та розробки відповідних рішень.

РОЗДІЛ 2
ДОСЛІДЖЕННЯ ІСНУЮЧИХ МЕТОДІВ І ТЕХНОЛОГІЙ ІНТЕГРАЦІЇ ЧАТ-БОТІВ З УНІВЕРСИТЕТСЬКИМИ СИСТЕМАМИ І БАЗАМИ ДАНИХ

2.1 Технології та підходи до інтеграції чат-ботів з навчальними та адміністративними системами університетів

2.1.1 Технології інтеграції чат-ботів з навчальними та адміністративними системами університетів

Під інтеграцією чат-ботів із університетськими системами вважається можливість використання сучасних технологій платформ чат-ботів, які взаємодіють із БД та іншими сервісами навчальних закладів. На сьогоднішній день є багато засобів, які надають змогу взаємодії між чат-ботами та університетськими системами. Серед основних методів виділяють:

- API-інтеграція
- Використання веб-хуків (Webhooks)
- Прямий доступ до баз даних
- Інтеграція через хмарні сервіси
- Підхід "Middleware"

1. API-інтеграція - цей метод надає змогу інтеграції за допомогою прикладного програмного інтерфейсу, який надає методи різних типів, які виконуються за допомогою HTTP запитів, таких, як: POST, DELETE, GET, PUT, PATCH. Завдяки вказаним запитом здійснюються CRUD операції. Для прикладу, якщо потрібно отримати дані з БД метод інтеграції виконує GET запит за допомогою, якого користувач отримує інформацію в чат-боті. Аналогічно, якщо потрібно додати або редагувати дані виконуємо POST або DELETE метод із передачею потрібних значень в тіло (body) запиту. Також широко розповсюдженими прикладами використання API-інтеграцій в освітньому

процесі є інтеграція з Moodle або іншими освітніми платформами, а також API бібліотеки часто використовують для пошуку книг і статей.

Перевагами API інтеграції є швидкість розгортання та стандартизований доступ до даних.

Недоліками такої інтеграції є залежність від наявності та якості API, а також можливий ризик неавторизованого доступу, якщо не вжито всіх необхідних засобів безпеки.

2. Використання веб-хуків (Webhooks) - за допомогою веб-хуків можна налаштувати нотифікацію певних подій, які відбуваються в системі за допомогою тригерів. Для прикладу тригер можна встановити, щоб відпрацьовував коли в системі змінюють або додають розклад, виставляють оцінки студентам або додано оголошення університетів, наприклад про день відкритих дверей або інші події на які можуть бути встановлені тригери. Також тригери можна налаштувати, щоб відпрацьовували у встановлену дату та час, наприклад кожного ранку о 9:00 нагадуванння про заняття та інше.

Прикладами таких інтеграцій може бути надсилання сповіщень студентам про нові завдання або зміни в розкладі, масові розсилки про події університету, наприклад день відкритих дверей, проведення змагань та конкурсів та інше. Серед основних переваг використання веб-хуків можна виділити миттєве оновлення даних, оскільки тригери відпрацьовують одразу після виконання певних дій. Серед основних недоліків є потреба у налаштуванні серверу для прийому веб-хуків. Зазвичай це реалізовують за допомогою хмарних сервісів або інших інструментів розробки веб-хуків.

3. Прямий доступ до баз даних - завдяки цьому методу чат-боти можуть мати пряме підключення до бази, наприклад через .config або appsettings.json файл в якому вказується сервер підключення, логін та пароль користувача в БД під яким буде здійснюватися авторизація, назва бази даних та порт, який буде використовуватися для підключення. Прикладами таких інтеграцій можуть бути чат-боти, які роблять напряму через конфігураційні файли взаємодіють із базами

даних для здійснення CRUD операцій, наприклад виставлення оцінок, додавання та редагування розкладу, створення та видалення оголошень та інші важливі події в навчальному процесі, які потребують взаємодії з БД. Серед основних переваг такої інтеграції є максимальний контроль над даними та швидкість відповіді від БД. Недоліком цього методу є ризики безпеки даних, так, оскільки підключення в чат-боті відбувається на пряму до БД

4. Інтеграція через хмарні сервіси – ці інтеграції використовують платформи, такі як Microsoft Azure, AWS, Heroku чи Google Cloud, для зберігання та обробки даних, а також інтеграції з чат-ботами.

Прикладами таких інтеграцій є використання Google Dialogflow для обробки тексту через хмарний сервіс API, а також зберігання студентських даних у захищених хмарних базах для легкого доступу чат-бота. Серед основних переваг даного підходу виділяють масштабованість, оскільки хмарні сервіси використовують модель використання ресурсів у відповідності до потреб споживання, а також у підтримці, оскільки є багато ресурсів де можна отримувати інформацію по налаштуванню хмарних сервісів. Недоліками даної інтеграції є залежність від провайдера та висока вартість, яка зростатиме в залежності від використання чат-боту.

5. Підхід "Middleware" – це процес впровадження проміжного програмного забезпечення (middleware) для координації між чат-ботом і університетськими системами та освітніми ресурсами, які забезпечують оптимізацію освітнього процесу, створення та пошук наукових статтей та інше.

Приклад такого чат-боту є Rasa як middleware для обробки запитів від ботів і передачі їх до СУБД університету. Також можна Використати C#, Java, Python, Node.js чи інші популярні backend технології для створення серверної логіки між чат-ботом та іншими системами. Перевагами цього методу є гнучкість у налаштування інтеграції, а основний недолік полягає у збільшенні складності розробки ПЗ. Також проміжне програмне забезпечення часто реалізується у поєднанні хмарних сервісів.

2.2 Аналіз інтерфейсів для взаємодії з чат-ботами в університетських системах

2.2.1 Поняття інтерфейсу чат-ботів. Ключові вимоги до інтерфейсу чат-ботів

Інтерфейси для взаємодії з чат-ботами є основною складовою їх ефективності. Вони забезпечують зв’язок між користувачами різних видів, таких, як студенти, викладачі, адміністратори, абітурієнти та університетські системи.

Ключові вимоги до інтерфейсів чат-ботів включають:

1) Юзабіліті (зручність користування) - інтерфейс повинен бути інтуїтивно зрозумілим для користувача, із мінімумом кроків для виконання основних дій. Також для відображення інформації в чат-боті варто її візуалізовуватися для кращого сприйняття. Це може бути реалізовано за допомогою графічних елементів, таких, як емоджі, картинки, списки, кнопки, форми та інше.

2) Сумісність - інтерфейс має працювати на різних платформах: мобільних, десктопних, планшетах та бути адаптивним до різних пристроїв та операційних систем. Також інтерфейс повинен бути адаптованим до різних розширень екранів.

3) Доступність – повинен забезпечувати роботу для користувачів з обмеженими можливостями через голосовий інтерфейс, екранні зчитувачі та можливість масштабування для користувачів, які мають вади зору.

4) Інтерактивність – інтерфейс повинен швидко реагувати на запити користувачів і надавати можливість адаптувати відповіді до їхніх потреб та візуально їх представляти.

2.2.2 Типи інтерфейсів

Розглянемо основні типи інтерфейсів та особливості їх застосування.

Типи інтерфейсів для чат-ботів:

1) Текстові інтерфейси – за допомогою цих інтерфейсів відбувається взаємодія з чат-ботом через текстові повідомлення у месенджерах, веб-сайтах або мобільних додатках. Перевагами таких інтерфейсів є зручність для текстового спілкування та підтримка багатьох мов та форматів. Недоліком цього інтерфейсу є обмежена взаємодія з чат-ботом без графічних елементів, таких, як кнопки, діаграми, графіки. На рис. 2.1 наведено приклад текстового інтерфейсу чат-боту.

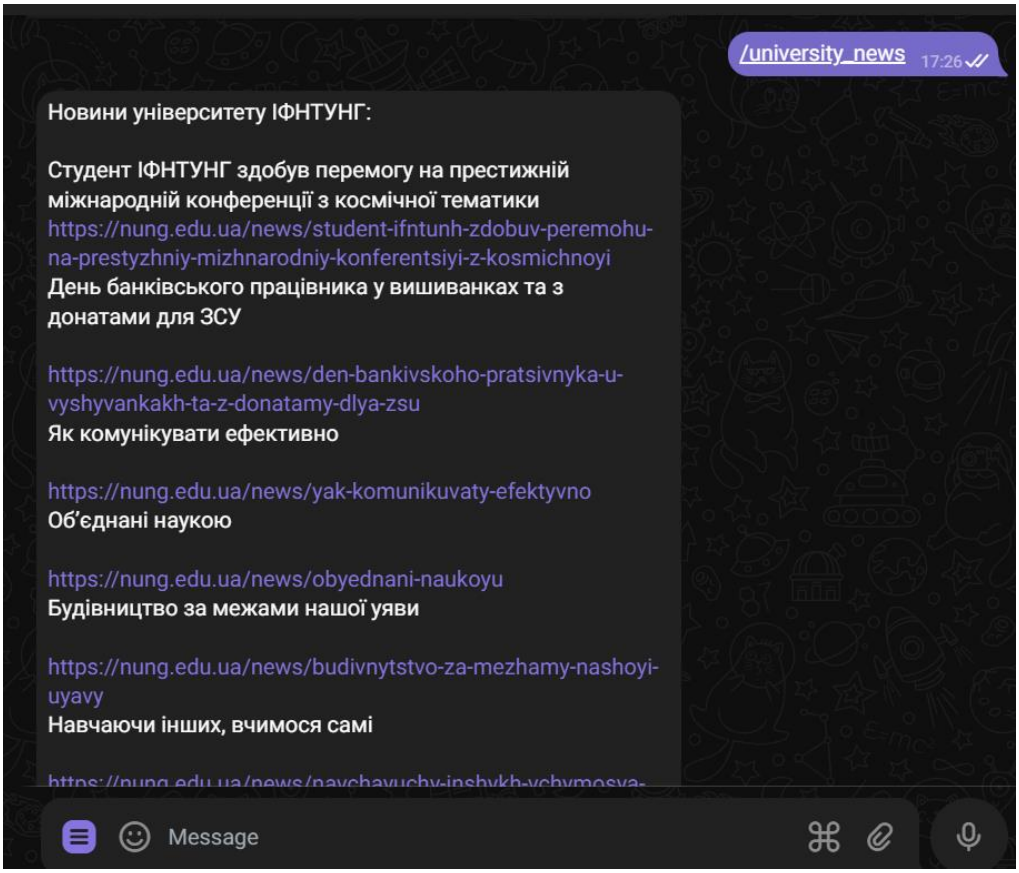

Рис. 2.1. Приклад текстового інтерфейсу чат-бота

2) Голосові інтерфейси – ці інтерфейси надають можливість для користувача спілкуватися з чат-ботом голосом. Вони здатні інтегруватися з голосовими помічниками такими, як Google Assistant та Alexa або спеціалізованими ботами для голосової комунікації. Перевага цих інтерфейсів є в тому, що вони можуть використовуватися коли в користувачів є обмежений

доступ до клавіатури або користувачів надає перевагу природньому спілкуванню. Недолік цього інтерфейсу полягає у складнощах з розпізнаванням акцентів і багатомовністю та високі вимоги до обробки мови.

3) Графічні інтерфейси (GUI) - це інтерактивні елементи такі, як кнопки, списки, форми, які використовуються для спрощення взаємодії через веб або мобільні додатки. Для прикладу коли користувач хоче отримати розклад то замість тексту йому буде зручніше отримати цей розклад списком або у вигляді форми, яка буде розбита на 5 частин (понеділок – п’ятниця) і в кожній цій формі може бути список пар. Перевагою цього інтерфейсу є те, що вони є інтуїтивно зрозумілими для нових користувачів та зменшують кількість можливих помилок при введенні запитів користувачами. Недоліком цього методу є обмеження у складних запитах, які не враховані в графічному інтерфейсі. Приклад інтерфейсу з GUI наведено на рис. 2.2.

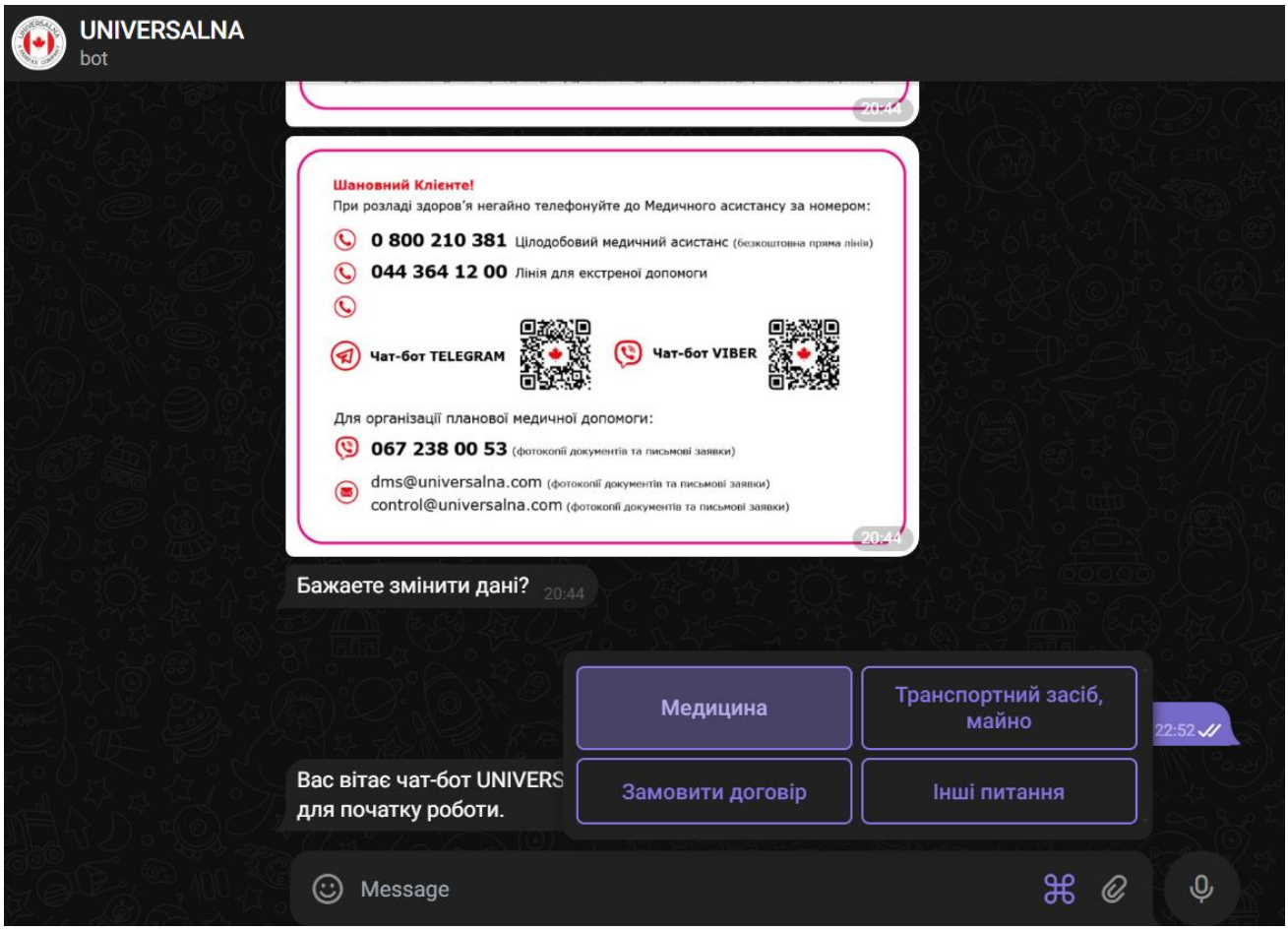

Рис. 2.2. Приклад інтерфейсу чат-боту з GUI

4) Інтеграція з існуючими університетськими системами – цей тип інтерфейсу надає можливість впровадження чат-ботів у внутрішні платформи університету, такі як портали студентів або викладачів. Перевагами цього типу інтерфейлу є доступ до університетських баз даних та сервісів, які викликаються ззовні в чат-боті та єдиний доступ до інформації та функцій. Недолік цього інтерфейсу є в тому, що він залежить від функціональних можливостей основної платформи.

5) Комбіновані – цей тип інтерфейсів включає в себе різні типи інтерфейсів, наприклад голосові та текстові, або текстові та графічці. Ці комбінації інтерфейсів можуть бути різними в залежності від вимог, які ставлять до чат-боту.

2.3 Визначення вимог до чат-ботів при інтеграції з університетськими базами даних

2.3.1 Ключові вимоги для технологій інтеграції

Серед ключових вимог для технологій інтеграції є такі основні вимоги:

- Сумісність - підтримка існуючої інфраструктури університету із чат-ботами. Тобто чат-бот повинен мати змогу доступатися до різних ресурсів університету за допомогою методів інтеграцій;
- Безпека - забезпечення конфіденційності та захисту даних користувачів. Це може бути, зокрема, реалізовано за допомогою двохфакторної авторизації або інших механізмів підтвердження даних користувачів. Відправка кодів підтверджень може бути реалізована на електронну пошту користувача або номер телефону;
- Масштабованість - можливість обробляти великий обсяг запитів. Окрім цього корисним є масштабування на різні платформи чат-ботів, таких, як Telegram, Viber, WhatsApp та інші популярні сервіси;
- Зручніть – інтерфейс повинен бути зрозумілим для користувача;

- Простота впровадження - мінімальні витрати часу та ресурсів для інтеграції чат-ботів із університетськими базами даних для перегляду актуальної інформації та за потреби оновлення даних.

2.3.1 Визначення основних вимог до чат-ботів при інтеграції з університетськими базами даних

Інтеграція між чат-ботами та університетськими базами даних повинна включати функціонал для користувачів, які задіяні в освітньому процесі. Приклад таких користувачів подано в таблиці 2.1.

Таблиця 2.1

Основні користувачі університетських баз даних та опис функціоналу, який призначений для них

Працівники (викладачі, лектори, лаборанти)	Повинні мати доступ до навчальних матеріалів та курсів, а також повинні мати змогу оцінювання студентів та моніторити навчальний процес. Також викладачі повинні мати змогу переглядати графік занять
Студенти	Повинні мати змогу переглядати розклад занять, отримувати інформацію про оцінки з навчальних дисциплін та спілкуватися з викладачами та адміністрацією університету. Також студенти повинні отримувати нотифікації про події університету, такі, як змагання та конкурси
Ректорат, деканат, адміністрація, методисти	Мають змогу проводити адміністрування та управління освітнім процесом, таке, як розподіл занять, збирати інформацію про навчальні досягнення та створювати різні подій університету за допомогою масових розсилок

Продовження таблиці 2.1

Вступники (абітурієнти)	Повинні мати змогу подавати заявки на вступ. Перевіряти статуси заявок та вносити в них зміни та отримувати інформацію про спеціальності університету
Адміністратор системи	Повинні мати змогу управляти базами даних, додавати нових користувачів та редагувати існуючих, проводити обслуговування БД таке, як резервне копіювання.
Незареєстровані користувачі	Повинні мати змогу переглядати загальну інформацію про університет та публічні дані університету

На рис. 2.3 наведено базову UML діаграму варіантів використання основних користувачів системи інтеграції чат-боту з університетськими базами даних. А в UML діаграмах варіантів використання рис. 2.4 – 2.7 показано деталізовано права кожного користувача.

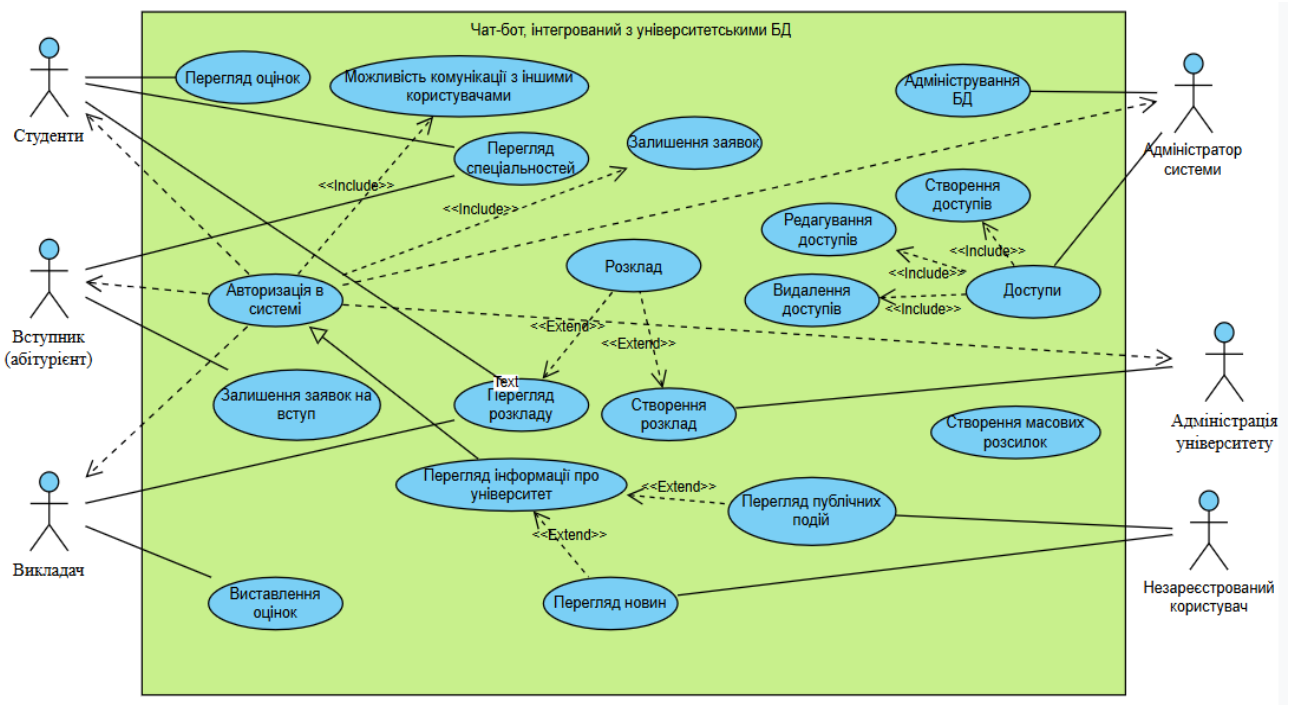

Рис. 2.3. UML діаграма варіантів використання (use case diagram) університетського чат-боту

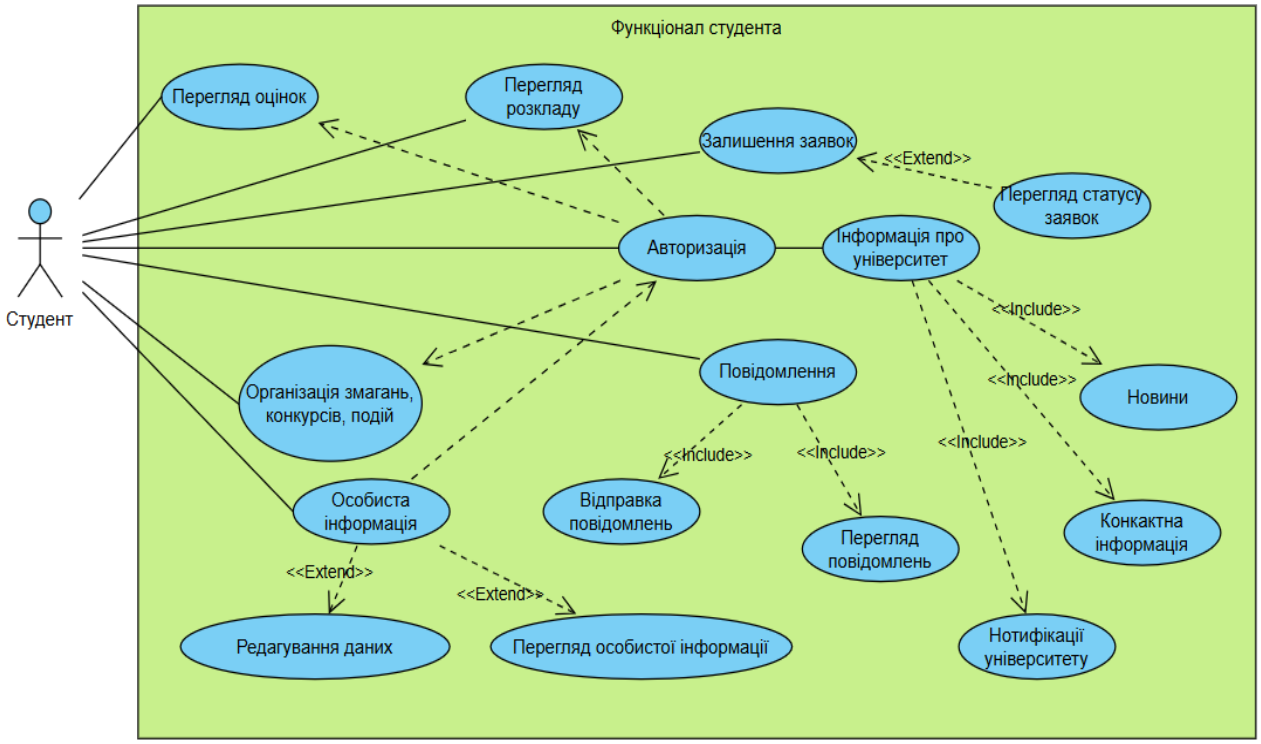

Рис. 2.4. UML діаграма варіантів використання (функціонал студента)

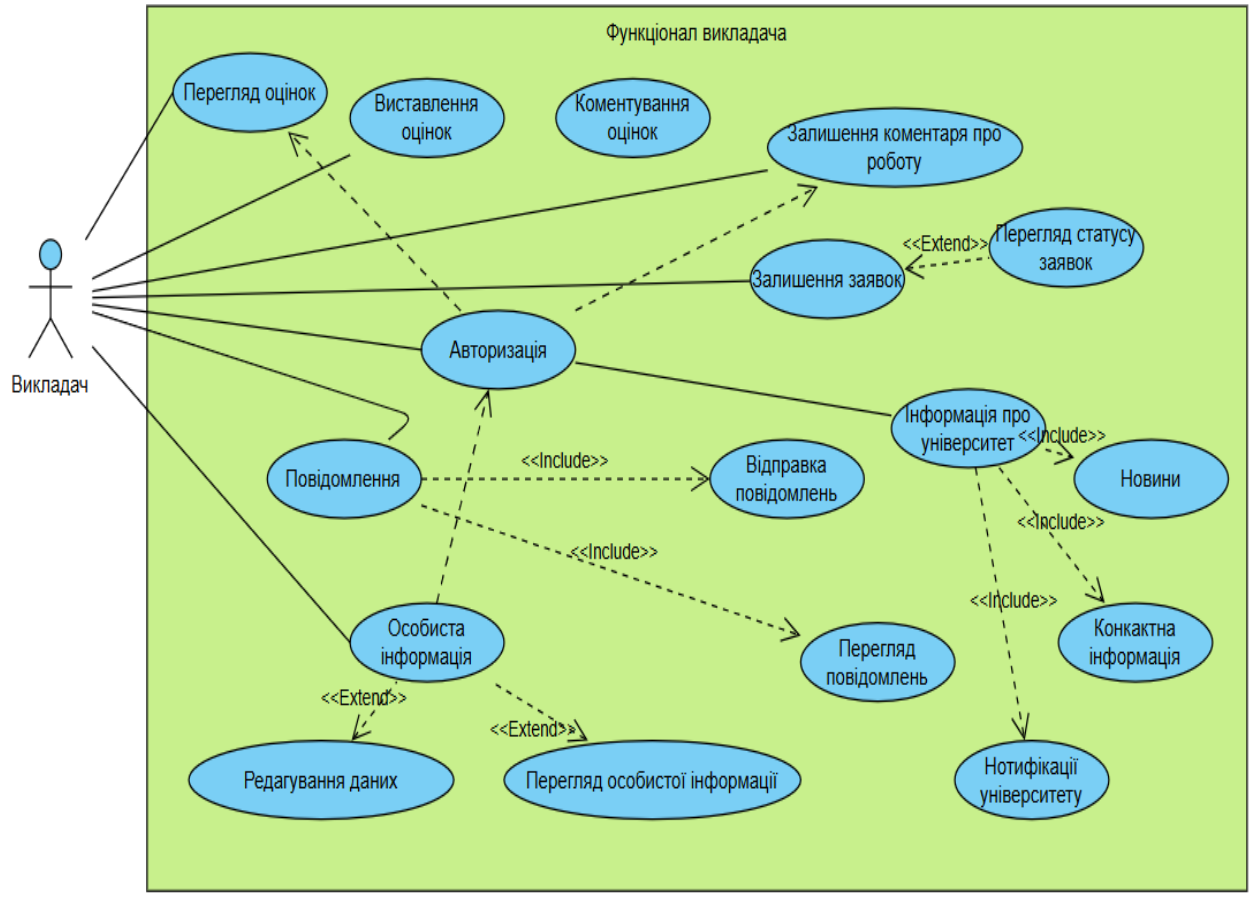

Рис. 2.5. UML діаграма варіантів використання (функціонал викладача)

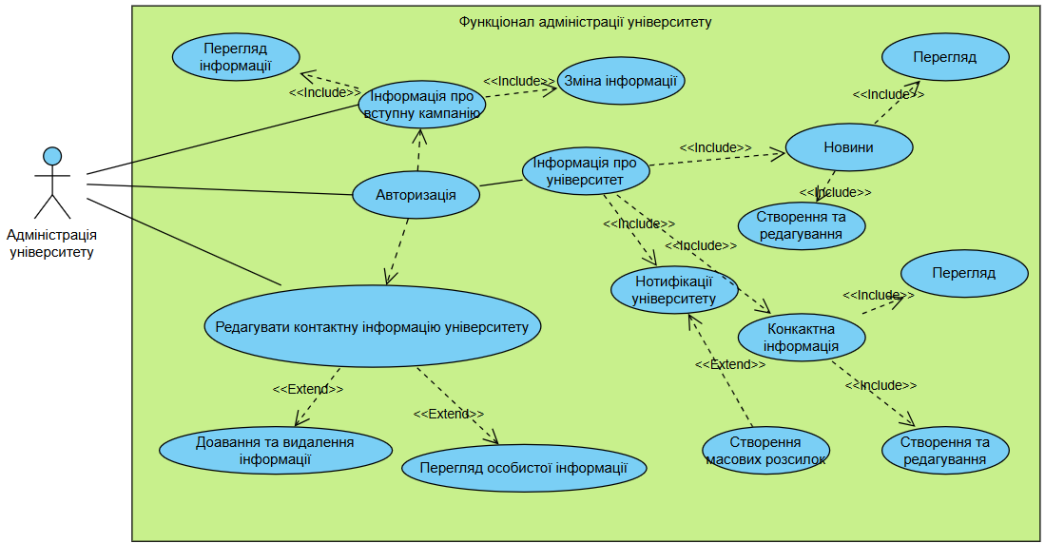

Рис. 2.6. UML діаграма варіантів використання (функціонал адміністрації університету)

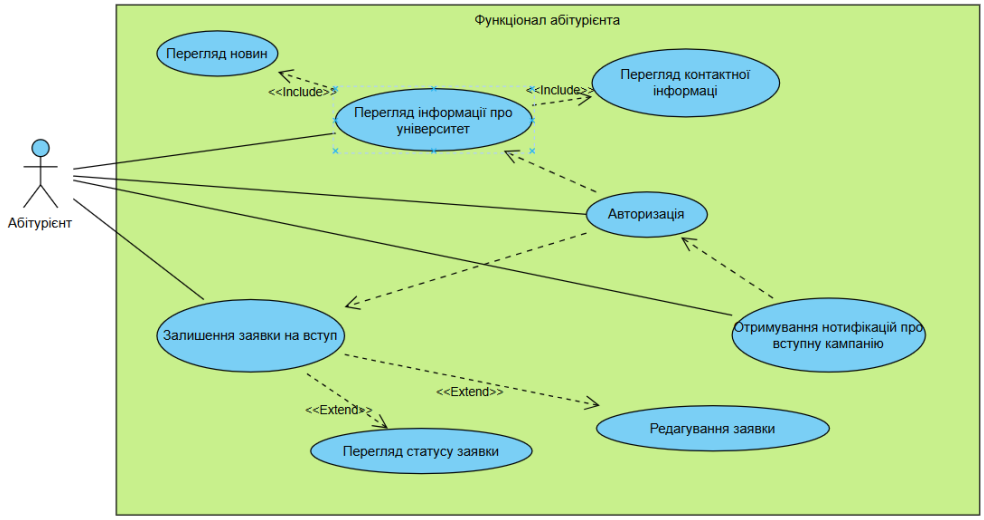

Рис. 2.7. UML діаграма варіантів використання (функціонал абітурієнта)

На наведених вище діаграма ілюстровано права користувачів згідно їхніх ролей. Із цієї діаграми можна помітити, що користувачі після авторизації мають різний функціонал, в залежності, які ролі вони мають. Користувач для використання чат-боту не обов’язково повинен авторизовуватися, але в цьому випадку для нього буде доступна тільки публічна інформація, така, як перегляд інформації про університет, перегляд різних подій університету та інша інформація, яка є в публічному доступі. Після здійснення авторизації в чат-боті

за введеною корпоративною електронною поштою у користувача повинен відкритися функціонал, який доступний для його ролі. На рис. 2.3 наведено загальну діаграму із базовим функціоналом користувачів. Нижче наведено деталізовані UML діаграми варіантів використання для кожного користувача окремо для огляду розширених можливостей користувача.

Ці ролі визначаються на основі корпоративної електронної пошти, якщо це викладач, студент, або представник адміністрації університету. Якщо це абітурієнт по визначає на основі звичайної електронної пошти по якій здійснено авторизацію в системі. В наступному розділі описано деталізований алгоритм процесу авторизації у системі.

2.4 Розробка алгоритмів авторизації та автентифікацію у чат-боті. Визначення прав користувачів

2.4.1 Опис алгоритмів авторизації та автентифікації в чат-боті

Архітектура системи інтеграції чат-боту з університетськими базами даних важливу роль приділяє безпеці даних користувачів. Для забезпечення механізму безпечної обробки та передавання даних спроектовано модель автентифікації користувачів, під час якої користувачам присвоюється відповідна роль на основі введеної електронної пошти. Після цього система перевіряє чи введено комерційну електронну пошту. У випадку, якщо комерційну то система на основі цієї пошти визначає роль користувача (студент, викладач чи адміністрація університету). Якщо введено особисту електронну пошту то система перевіряє чи є вже така пошта зареєстрована в БД. Якщо є то пропонує авторизуватися. Якщо введена пошта ще не зареєстрована в базі то система пропонує здійснити авторизацію абітурієнта. На цьому етапі користувачу потрібно ввести логін та пароль. Після цього йому буде відправлено код підтвердження на електронну пошту. Далі чат-бот попросить користувача ввести цей код підтвердження. У випадку, якщо користувач введе неправильний код то система виведе про це

помилку. Якщо буде введено правильний код то система авторизує користувача із визначеними ролями.

Система авторизації та реєстрації пацієнтів складається із таких основних класів:

1) User – цей клас емулює поведінку користувача. Він має такі поля

- email: String – електронна пошта користувача;
- login: String – логін користувача;
- password: String – пароль користувача;
- verificationCode: String – код підтвердження, який відправлено на електронну пошту;
- isRegistered: Boolean – чи користувач є зареєстрований;
- isVerified: Boolean – чи користувач підтвердив верифікацію;
- lastLoginTime: Date – дата останнього входу;
- roles: List<String> - список ролей, які має користувач.

Також даний клас має такі методи:

- register(): void – цей метод здійснює реєстрацію коирстувача в системі;
- verify(): void – це метод здійснює верифікацію користувача в системі;
- login(): void - це метод здійснює авторизацію користувача;
- resetPassword(newPassword: String): void – за допомогою цього методу користувач може відновити пароль;
- updateProfile(details: Map<String, String>): void – цей метод призначений для редагування особистих даних користувача.

2) Клас System – цей клас призначений для емулювання поведінки системи. Він має такі основні методи:

- logActivity(activity: String): void – цей метод записує активність авторизації/реєстрації в логи для моніторингу потенційно можливих проблем;
- generateAuthToken(userId: String): String - цей метод генерує токен авторизації;

- checkEmail(email: String): Boolean – цей метод перевіряє електронну пошту користувача та на її основі здійснює перевірку чи існує такий користувач в БД та якщо існує, які в нього ролі;
- sendVerificationCode(email: String): void – завдяки цьому методу відбувається двохфакторна авторизація, він відправляє код підтвердження на електронну пошту користувача;
- validateVerificationCode(email: String, code: String): Boolean – цей метод перевіряє чи користувач ввів правильний код підтвердження;
- checkLoginAndPassword(login: String, password: String): Boolean – цей метод здійснює перевірку логіну та паролю;
- invalidateAuthToken(token: String): void – метод призначений для деактивації токену;
- handleFailedLoginAttempts(email: String): void – цей метод призначений для обробки помилок, які виникають на етапі авторизації.

3) TelegramBot – цей клас емулює поведінку чат-боту. Замість нього можуть бути будь-які чат-бот месенджери, наприклад WhatsApp або Viber. Він призначений для того, щоб кумунікувати з пацієнтом, давати йому запитання та відповіді на них. Він має такі методи:

- sendMessage(message: String): void – цей метод відправляє повідомлення користувачу;
- askForInput(prompt: String): String – цей метод призначений для того, щоб давати користувачу запитання;
- notifyUser(userId: String, message: String): void – цей метод використовується для нотифікації користувачів;
- scheduleMessage(userId: String, message: String, time: Date): void – цей метод призначений для запланування відправки повідомлень на певну дату та час;
- retrieveChatHistory(userId: String): List<String> - цей метод використовується для отримання історії чату.

4) Клас EmailService – цей клас емулює поведінку сервісу відправки листів, яка працює на базі MS SQL Server за допомогою компоненти MS SQL Server Agent. Він призначений для відправки листів із кодами підтвердження. Він має такі методи:

- sendEmail(email: String, subject: String, body: String): void – цей метод використовується для відправки електронних листів користувачам;
- sendBulkEmails(emails: List<String>, subject: String, body: String): void – цей метод призначений для відправки масових розсилок користувачам. Він приймає на вхід список електронних листів та повідомлення, яке їм потрібно відправити;
- validateEmailFormat(email: String): Boolean – цей метод здійснює перевірку чи введено електронну пошту правильного формату;
- archiveSentEmails(): void – цей метод архівує листи, які відправлено на електронні пошти користувачів.

5) Клас RoleManagement – цей клас призначений для керування ролями користувачів при роботі з системою. Він має такі основні методи:

- assignRole(userId: String, role: String): void – цей метод призначений для надання ролей користувачам;
- removeRole(userId: String, role: String): void – цей метод призначений для видалення ролей користувачів;
- listUserRoles(userId: String): List<String> - цей метод призначений для визначення ролей користувача;
- validateRolePermission(userId: String, permission: String): Boolean Boolean>): void – цей метод призначений для перевірки дозволів, які надані ролям.

6) Клас NotificationService – призначений для нотифікації користувачів. Він складається з таких основних методів:

- sendPushNotification(userId: String, message: String): void – цей метод призначений для відправки нотифікацій користувачам;

- scheduleNotification(userId: String, message: String, time: Date): void – цей метод призначений для встановлення планових відправок нотифікацій користувачам на визначені дати;
- manageNotificationPreferences(userId: String, preferences: Map<String, Boolean>): void – цей метод колекціонує інформацію про те, що користувачам може бути цікаво на на основі цього встановлює пріоритети відправки.

На рис. 2.8 зображено UML діаграму класів програмних компонентів, які задіяні в процесі авторизації та реєстрації в системі, а також у визначенні ролей користувачів.

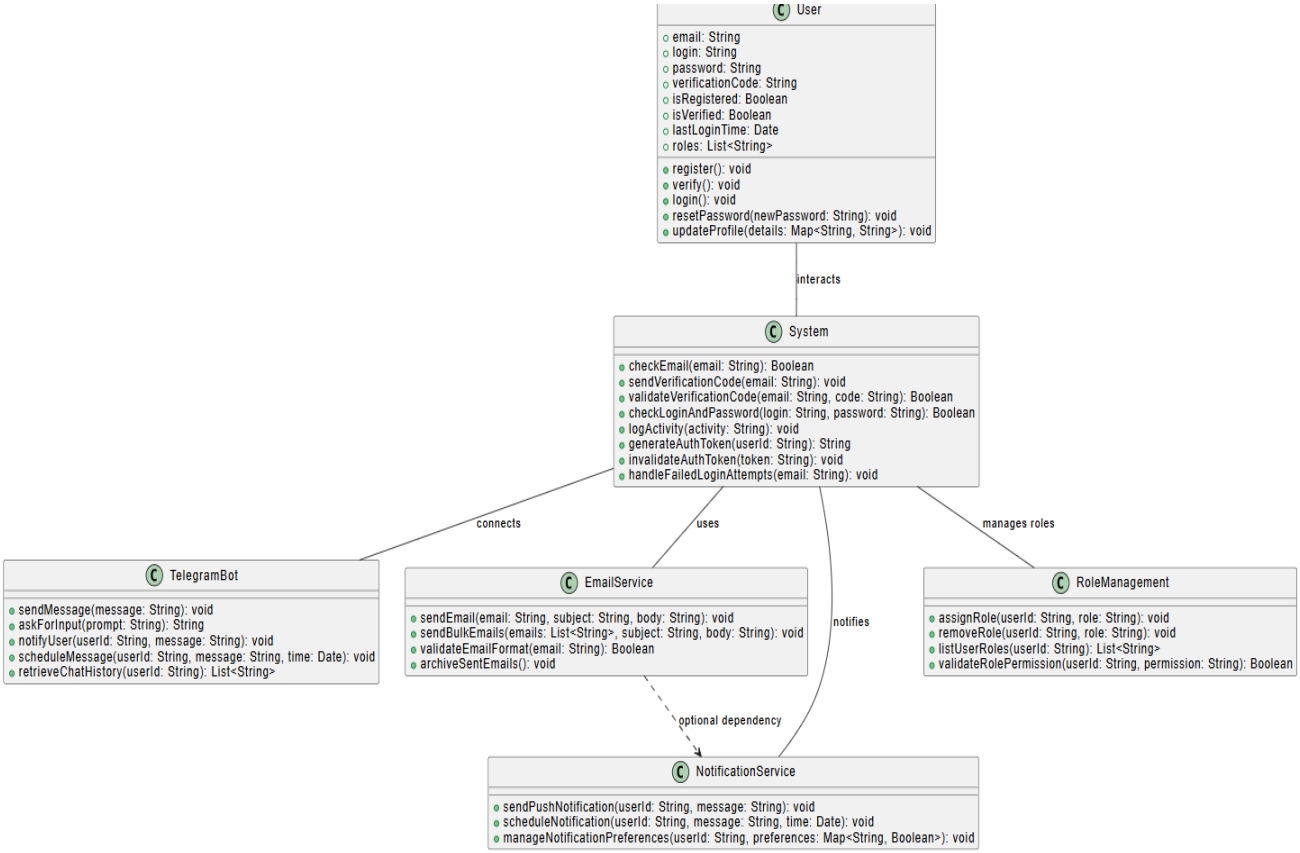

Рис. 2.8. UML діаграма класів логіки роботи компонентів системи, задіяних у двохфакторній авторизації

Послідовність дій, як відбувається процес реєстрації та авторизації в системі зображено на рис. 2.9 і рис. 2.10. На цій UML діаграмі послідовності зображений повний процес авторизації та реєстрації.

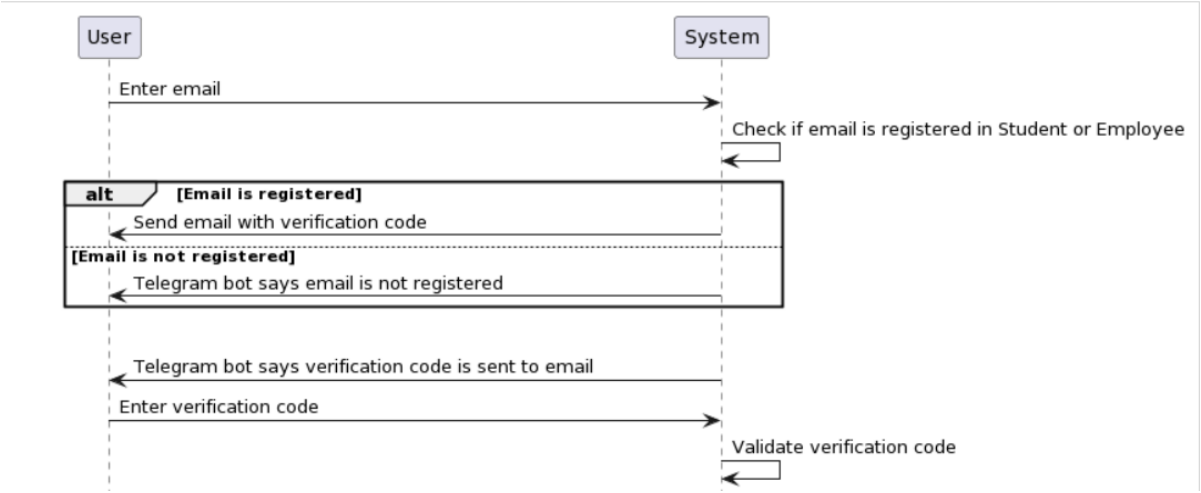

Рис. 2.9. UML sequence diagram (діаграма послідовності)

Нижче наведено рисунок із продовженням UML діаграми послідовності.

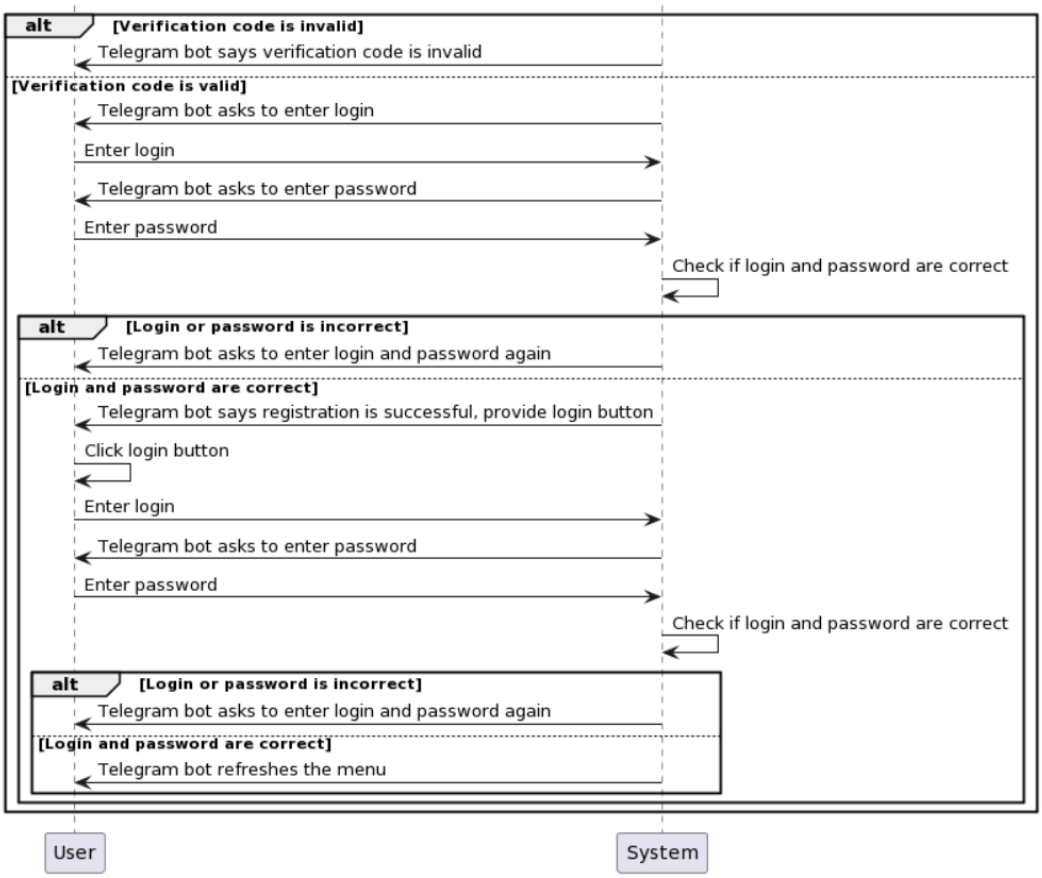

Рис. 2.10. UML sequence diagram (діаграма послідовності)

Із діаграми послідовності помітно, що користувач, якщо бажає ввійти в обобистий кабінет, то повинен ввести свою електронну пошту. Якщо користувач

викладач, студент або хтось із адміністрації університету то повинен ввести корпоративну електронну пошту. Якщо користувач вступник (абітурієнт) то повинен ввести особисту електронну пошту. Після введення пошти чат-бот приймає ці дані та передає до проміжного програмного забезпечення (middleware). Далі проміжне ПЗ викликає процедуру на стороні БД, яка перевіряє, чи в базі даних існує такий користувач. Якщо не існує то пропонує здійснити реєстрацію. Якщо користувач за введеною електронною поштою знайдений в БД то система відправляє на цю електронну пошту код підтвердження. Далі чат-бот просить користувача ввести цей код. Після того, як користувач ввів у чат-боті код підтвердження система перевіряє чи він відповідає тому, який було відправлено на електронну пошту користувача. Якщо код введено неправильно то middleware видає інформацію чат-боту інформацію про те, що введений код є неправильний. Після цього чат-бот виводить помилку та просить повторно ввести код. Якщо користувач ввів правильний код то система його авторизовує в особистий кабінет. Після цього він зможе користуватися функціоналом, який стане доступним відповідно до його ролі.

2.5 Висновки до розділу

У цьому розділі проведено аналіз існуючих методів, технологій та вимог, які необхідні для інтеграції чат-ботів з університетськими базами даних та системами. Розглянуто ключові аспекти, які забезпечують ефективність і безпеку роботи інтегрованих рішень чат-ботів з університетськими БД а також розглянуто методи інтеграцій та безпеку даних при використанні чат-ботів, які доступаються до БД університетів. Також розглянули функціональні та нефункціональні вимоги до чат-ботів та інтерфейси взаємодії чат-ботів з користувачами.

Виділили функціональні (автентифікація, обробка запитів, інтеграція з базами даних) і нефункціональні (масштабованість, продуктивність, зручність

використання) аспекти, які повинні враховуватися при розробці та спроектували алгоритми авторизації та автентифікації користувачів, розробили алгоритм авторизації за допомогою якого створюються користувачі в системі та створюються для них відповідні ролі. Також проаналізували сучасні способи взаємодії з користувачами, такі як текстові платформи, голосові асистенти та мультимодальні системи, які забезпечують зручність і доступність.

Результати аналізу показали, що інтеграція чат-ботів з університетськими системами має великий потенціал для автоматизації та покращення якості обслуговування різних груп користувачів, таких, як абітурієнти, адміністрації університету, студентів і викладачів. Для успішного впровадження необхідно забезпечити дотримання всіх вимог до безпеки, масштабованості та сумісності, а також використовувати сучасні технології для створення адаптивних та надійних рішень.

РОЗДІЛ 3
РОЗРОБКА МОДЕЛІ ТА АЛГОРИТМІВ ІНТЕГРАЦІЇ ЧАТ-БОТІВ З УНІВЕРСИТЕТСЬКИМИ БАЗАМИ ДАНИХ ТА СЕРВІСАМИ

3.1 Опис архітектури інтеграції чат-ботів з університетськими базами даних та сервісами

3.1.1 Опис архітектури інтеграції чат-боту з університетськими БД

Для проектування архітектури інтеграції чат-боту з університетською БД використано трьохшарову модель, яка складається із таких основних рівнів:

1) База даних – на цьому рівні відбувається керування даних, їх збереження, обробка транзакцій, запитів та інше. Також на цьому рівні працює частково backend за допомогою збережених процедур, функцій та представлень. До БД підключено 2 рівень, проміжне програмне забезпечення (middleware).

2) Проміжне програмне забезпечення (middleware) – на цьому рівні відбувається взаємодія між БД та чат-ботом. Проміжне ПЗ складається з набору API методів, які здатні взаємодіяти з БД, вичитувати з таблиць дані для отримання інформації, яку потім передає в чат-бот. Також проміжне ПЗ може виконувати збережені процедури БД, які виконують транзакції, наприклад додавання, видалення або редагування розкладу та інше. Ці всі дані, які проміжне ПЗ отримує з БД передає до 3 рівня чат-боту.

3) Чат-бот – на цьому рівні відбувається взаємодія системи з користувачем. Користувач пише повідомлення, чат-бот обробляє це повідомлення та робить запит на middleware, яке вичитує з БД дані або виконує транзакції. Після цього чат-бот повертає відповідь користувачу.

4) Різні розширення, такі, як інтеграція із аналітичними системами, сервісами нотифікацій, сервісами безпеки та інших.

На рис. 3.1 зображено UML component diagram (діаграму компонентів) цієї архітектури

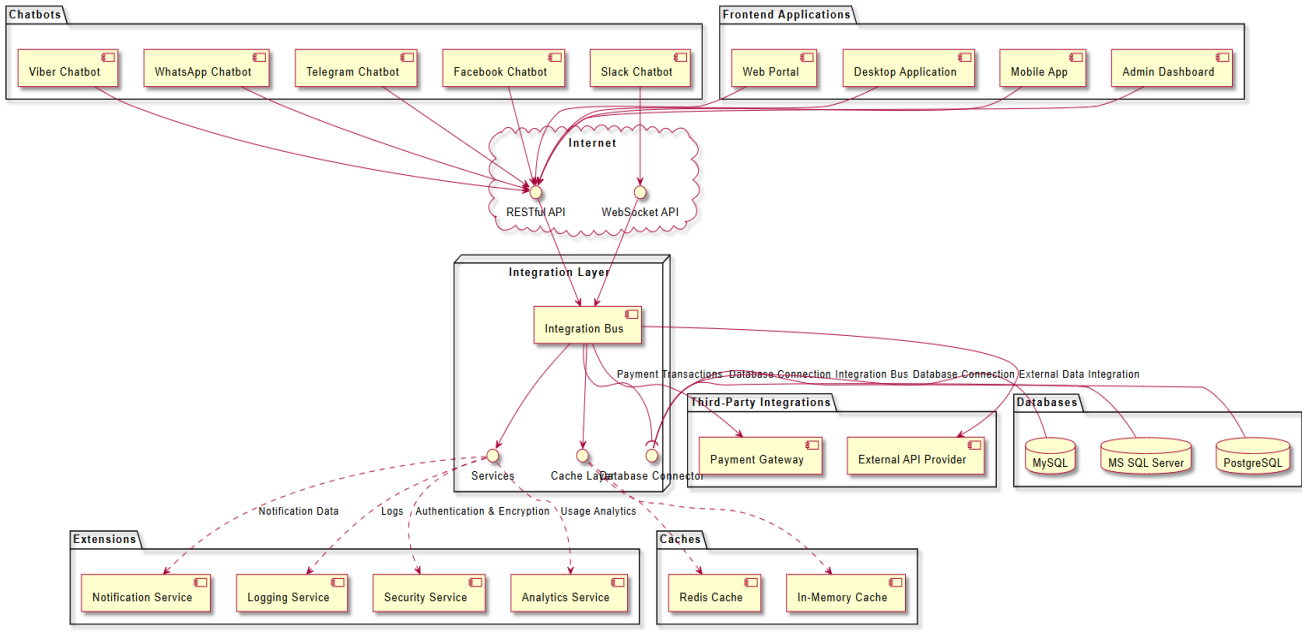

Рис. 3.1. UML діашоама компонентів. Архітектура інтеграції чат-боту із університетськими базами даних

Як ми бачимо із UML діаграми компонентів, зображеної на рис. 3.1 дана модель має найвищий шар чат-боти або веб чи мобільні застосунки. Вони працюють через набір RESTful API методи, які подають запити на проміжне програмне забезпечення. Далі проміжне ПЗ передає ці запити до БД та сторонні додатки, якщо такі підключені. Після цього БД та сторонні застосунки надають зворотну відповідь до middleware, який здатний обробляти ці відповіді та конвертує їх у вигляді API відповідей до чат-боту. Далі чат-бот приймає ці відповіді та виводить їх користувачу. В цій архітектурі обов’язковими є 3 рівні: чат-бот, проміжне ПЗ та БД. Інші компоненти додаються при потребі.

Аналізуючи вище подану архітектуру можна одразу зробити висновки про переваги цієї моделі, основна з яких є масштабованість. Як бачимо з діаграми, що окрім верхнього шару чат-бот може бути також доданий шар застосунків. Тобто маючи централізовану БД та проміжне ПЗ ми можемо їх інтегрувати з будь-яким сервісом. Сам шар чат-бот може в себе включати різні платформи. Найпопулярнішими з них на сьогоднішній день є: Telegram, Facebook, WhahtsApp, Viber та можна додавати інші. Відмінність полягатиме тільки у

використанні бібліотек конкретної платформи. Шар фронтенд аплікацій може включати в себе різні застосунки, такі, як мобільні, десктопні та веб-застосунки.

Всі ці компоненти взаємодіють із БД через API методи. API методи обробляються на проміжному ПЗ де відбувається їх валідація та передача до зовнішніх систем та БД. Серед зовнішніх систем, які можуть бути залученими до інтеграції є такі:

- Сервіс нотифікації користувачів – для відправки масових розсилок, наприклад про новини університету, про змагання, які заплановані в університеті або інші важливі події;
- Сервіс логувань – для моніторингу активності використання система та можливості виявлення потенційних проблем в безпеці;
- Сервіси безпеки – для забезпечення надійної роботи та перешкоджанню перехоплення даних зловмисниками;
- Сервіси аналітики – вони забезпечують аналітику даних, наприклад успішності студентів, відвудування занять, залучення у навчальні дисципліни та багато чого іншого;
- Сервіси кешування – ці сервіси використовуються для кешування даних з метою покращення оптимізації надання відповідей;
- Сервіси зовнішніх API – ці сервіси можуть бути корисними для прикладу при пошуку навчальних ресурсів, отримання наукових статей та інших популярних сервісів

Друга важлива перевага цієї архітектури полягає у її гнучкості. Оскільки кожен рівень цієї архітектури можу бути побудований на довільних технологіях. БД можна використовувати різних СУБД таких, як MS SQL Server, MySQL, Postgres, Firebird та інші. Проміжний сервер далі зможе підключатися до цих БД за допомогою ODBC провайдерів або напряму за допомогою конфігів, в яких прописується рядок підключення до БД. Відмінність у взаємодії із цими СУБД полягає тільки у підключенню да цих СУБД та стандартних бібліотек взаємодій із компонентами чих СУБД. Проміжне програмне забезпечення може бути

побудоване на будь-якій мові backend розробки, таких, як: Java, .NET, NodeJS, Python та інші. Стосовно фронтенду чат-ботів та їх взаємодії з проміжним ПЗ то відмінність полягає тільки у платформі, на якій чат-бот розробляється та в бібліотеках, які надає ця платформа. Всі API методи є ідентичними для всіх чат-ботів, тому їх можна створювати на будь-яких платформах, які дозволяють виконувати HTTP-запити.

Також бувають випадки, коли потрібно університет інтегрувати з іншими університетами, які працюють в межах договорів співпраці. За допомогою цієї моделі ми модемо з легкістю цього досягти. На рис. 3.2 показано приклад такої інтеграції між університетами.

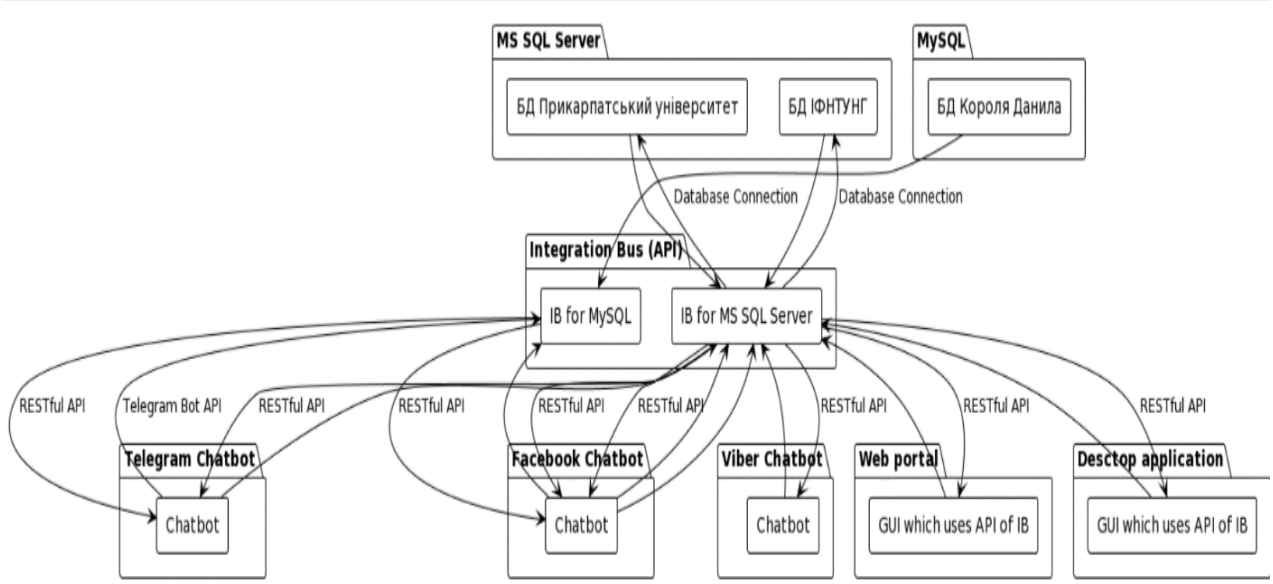

Рис. 3.2. Модель інтеграції університетів

На цій діаграмі для прикладу наведено, Що 2 університети мають БД в СУБД MS SQL Server та один університет має БД на СУБД наприклад MySQL або Oracle. Далі в нас є проміжне ПЗ, яке обробляє підключення до цих БД та взаємодіє з їхніми бібліотеками для записів з БД та отримання транзакцій. Це все відюувається за допомогою використання API запитів. Таким чином реалізується інтеграції між базами даних різних університетів. Всі доступи керуються ролями користувачів.

3.2 Опис архітектури університетських баз даних для інтеграції з чат-ботами

3.2.1 Опис архітектури університетських баз даних для інтеграції з чат-ботами

Університетська база даних повинна забезпечувати можливість зберігання, обробки та керування всією інформацією університету для різної групи користувачів. Сутності університетської БД повинні визначати інфраструктуру університету, містити інформацію про користувачів (студентів, викладачів, адміністрацію та абітурієнтів університету), містити інформацію про форми контролю, заліки, успішність студентів. Також БД повинна містити розклад занять студентів та можливість керування ним. Нижче наведено опис основних сутностей університетської БД.

Університетська база даних складається із таких основних таблиць:

Groups – містить інформацію про групи. Поля цієї таблиці показано в таблиці 3.1.

Таблиця 3.1

Поля таблиці Groups

Назва поля	Тип даних	Опис
GroupID	int	Первинний ключ таблиці
GroupName	nvarchar	Назва форми контролю (залік або екзамен)
GroupDesc	nvarchar	Опис форми контролю
GroupSpecialityRef	int	Зовнішній ключ, який посилається на таблицю Specialities

ControlForms – ця таблиця містить інформацію про форми контролю оцінювання. Поля цієї таблиці показано в таблиці 3.2.

Таблиця 3.2

Поля таблиці ControlForms

Назва поля	Тип даних	Опис
ControlFormID	int	Первинний ключ таблиці
ControlFormName	nvarchar	Назва форми контролю (залік або екзамен)
ControlFormDesc	nvarchar	Опис форми контролю

Courses – містить інформацію про курси. Поля цієї таблиці показано в таблиці 3.3.

Таблиця 3.3

Поля таблиці Courses

Назва поля	Тип даних	Опис
CourseID	int	Первинний ключ таблиці
CourseName	nvarchar	Назва курсу
CourseDesc	nvarchar	Опис курсу

Degree – містить інформацію про наукові ступені викладачів. Поля цієї таблиці показано в таблиці 3.4.

Таблиця 3.4

Поля таблиці Degree

Назва поля	Тип даних	Опис
DegreeID	int	Первинний ключ таблиці
DegreeName	nvarchar	Назва наукового ступеня
DegreeDesc	nvarchar	Опис наукового ступеня

Departments – містить інформацію про відділення (кафедру). Поля цієї таблиці показано в таблиці 3.5.

Таблиця 3.5

Поля таблиці Degree

Назва поля	Тип даних	Опис
DepartmentID	int	Первинний ключ таблиці
DepartmentName	nvarchar	Назва відділення (кефедри)
DepartmentDesc	nvarchar	Опис відділення (кефедри)
DepartmentFacultyRef	int	Зовнішній ключ, який посилається на таблицю Faculties
DepartmentLogo	nvarchar	Посилання на логотип
DepartmentHTMLDesc	nvarchar	HTML сторінка кафедри
DepartmentPhone1	nvarchar	Номер телефону №1
DepartmentPhone2	nvarchar	Номер телефону №2
DepartmentPhone3	nvarchar	Номер телефону №3
DepartmentPhone4	nvarchar	Номер телефону №4
DepartmentPhone5	nvarchar	Номер телефону №5
DepartmentAddress	nvarchar	Адреса де розміщена кафедра
DepartmentEmail	nvarchar	Електронна пошта кафедри
DepartmentSocialNetwork1	nvarchar	Посилання на соціальну мережу №1
DepartmentSocialNetwork2	nvarchar	Посилання на соціальну мережу №2
DepartmentSocialNetwork3	nvarchar	Посилання на соціальну мережу №3
DepartmentSocialNetwork4	nvarchar	Посилання на соціальну мережу №4
DepartmentSocialNetwork5	nvarchar	Посилання на соціальну мережу №5

DiplomaThesis – містить інформацію про дипломні або магістерські роботи. Поля цієї таблиці показано в таблиці 3.6.

Таблиця 3.6

Поля таблиці DiplomaThesis

Назва поля	Тип даних	Опис
DiplomaThesisID	int	Первинний ключ таблиці
DiplomaThesisTheme	nvarchar	Тема дипломної роботи
DiplomaThesisDesc	nvarchar	Опис дипломної роботи
DiplomaThesisScore	int	Оцінка за дипломну роботу
DiplomaThesisDate	datetime	Дата захисту дипломної роботи
StudentRef	int	Зовнішній ключ, посилання на Students
EducationalSubjectRef	int	Зовнішній ключ, посилання на EducationalSubjects
EmployeeRef	int	Зовнішній ключ, посилання на Employees

EducationalSubjects – містить інформацію про навчальні дисципліни. Поля цієї таблиці показано в таблиці 3.7.

Таблиця 3.7

Поля таблиці EducationalSubjects

Назва поля	Тип даних	Опис
EducationalSubjectID	int	Первинний ключ таблиці
EducationalSubjectName	nvarchar	Назва навчальної дисципліни
EducationalSubjectDesc	nvarchar	Опис навчальної дисципліни
EducationalSubjectCountHours	int	Кількість годин дисципліни
EducationalSubjectSpecialityRef	int	Зовнішній ключ, посилання на таблицю Specialities

Exams – містить інформацію про екзамени. Поля цієї таблиці показано в таблиці 3.8.

Таблиця 3.8

Поля таблиці Exams

Назва поля	Тип даних	Опис
ExamID	int	Первинний ключ таблиці
ExamName	nvarchar	Назва екзамену
ExamDesc	nvarchar	Опис екзамену
ExamSemester	int	Семест в якому відбувся екзамен
ExamDate	datetime	Дата екзамену
ExamEducationalSubjectRef	int	Зовнішній ключ, посилання на таблицю EducationalSubjects
ExamEmployeeRef	int	Зовнішній ключ, посилання на таблицю Employees
ExamGroupRef	int	Зовнішній ключ, посилання на таблицю Groups

Students – містить інформацію про студентів. Поля цієї таблиці показано в таблиці 3.9.

Таблиця 3.9

Поля таблиці Students

Назва поля	Тип даних	Опис
StudentID	int	Первинний ключ таблиці
StudentGroupRef	int	Зовнішній ключ, посилання на таблицю Groups
StudentFacultyRef	int	Зовнішній ключ, посилання на таблицю Faculties

Продовження таблиці 3.9

StudentCityRef	int	Зовнішній ключ, посилання на таблицю Cities
StudentGenderRef	int	Зовнішній ключ, посилання на таблицю Gender
StudentSpecialityRef	int	Зовнішній ключ, посилання на таблицю Specialities
StudentName1	nvarchar	Прізвище студента
StudentName2	nvarchar	Ім’я студента
StudentName3	nvarchar	По батькові студента
StudentPhoto	nvarchar	Фото студента
StudentBirthDdate	datetime	Дата народження студента
StudentPhone1	nvarchar	Номер телефону студента №1
StudentPhone2	nvarchar	Номер телефону студента №2
StudentEmail	nvarchar	Електронна пошта студента
StudentAddress	nvarchar	Адреса проживання студента
StudentHasScholarship	bit	Чи має студент стипендію
StudentScholarship	money	Розмір стипендії студента

TypeOfClasses – містить інформацію про типи занять. Поля цієї таблиці показано в таблиці 3.10.

Таблиця 3.10

Поля таблиці TypeOfClasses

Назва поля	Тип даних	Опис
TypeOfClassID	int	Первинний ключ таблиці
TypeOfClassName	nvarchar	Назва типу заняття
TypeOfClassDesc	nvarchar	Опис типу заняття

Employees – містить інформацію про викладачів. Поля цієї таблиці показано в таблиці 3.11.

Таблиця 3.11

Поля таблиці Employees

Назва поля	Тип даних	Опис
EmployeeID	int	Первинний ключ таблиці
EmployeePhoto	nvarchar	Фото працівника
EmployeeName1	nvarchar	Прізвище працівника
EmployeeName2	nvarchar	Ім’я працівника
EmployeeName3	nvarchar	По батькові працівника
EmployeeBirthDate	datetime	Дата народження
EmployeePhoneNumber1	nvarchar	Номер телефону №1
EmployeePhoneNumber2	nvarchar	Номер телефону №2
EmployeeEmail	nvarchar	Електронна пошта
EmployeeAddress	nvarchar	Адреса проживання
EmployeeSalaryPerHour	money	Зарплата погодинна
EmployeeSalaryPerMonth	money	Зарплата місячна
EmployeeAlmaMater	nvarchar	Приналежність працівника
EmployeeScientificInterest	nvarchar	Наукові інтереси
EmployeeDesc	nvarchar	Опис працівника
EmployeeDescHTML	nvarchar	HTML сторінка працівника
EmployeeDegreeRef	int	Зовнішній ключ, посилання на таблицю Degrees
EmployeeSpecialityRef	int	Зовнішній ключ, посилання на таблицю Specialities
EmployeePositionRef	int	Зовнішній ключ, посилання на таблицю Positions

Продовження таблиці 3.11

EmployeeGenderRef	int	Зовнішній ключ, посилання на таблицю Gender
EmployeeCityRef	int	Зовнішній ключ, посилання на таблицю Cities
EmployeeFacultyRef	int	Зовнішній ключ, посилання на таблицю Faculties
EmployeeDepartmentRef	int	Зовнішній ключ, посилання на таблицю Departments

Faculties – містить інформацію про інститути. Поля цієї таблиці показано в таблиці 3.12.

Таблиця 3.12

Поля таблиці Faculties

Назва поля	Тип даних	Опис
FacultyID	int	Первинний ключ таблиці
FacultyName	nvarchar	Назва інституту
FacultyDesc	nvarchar	Опис інституту
FacultyLogo	nvarchar	Логотип інституту
FacultyHTMLDesc	nvarchar	HTML сторінка інституту
FacultyPhone1	nvarchar	Номер телефону №1 інституту
FacultyPhone2	nvarchar	Номер телефону №2 інституту
FacultyPhone3	nvarchar	Номер телефону №3 інституту
FacultyPhone4	nvarchar	Номер телефону №4 інституту
FacultyPhone5	nvarchar	Номер телефону №5 інституту
FacultyAddress	nvarchar	Адреса розташування інституту
FacultyEmail	nvarchar	Електронна пошта інституту
FacultySocialNetwork1	nvarchar	Посилання на соціальну мережу №1

Продовження таблиці 3.12

FacultySocialNetwork2	nvarchar	Посилання на соціальну мережу №2
FacultySocialNetwork3	nvarchar	Посилання на соціальну мережу №3
FacultySocialNetwork4	nvarchar	Посилання на соціальну мережу №4
FacultySocialNetwork5	nvarchar	Посилання на соціальну мережу №5
FacultyDirectorRef	int	Зовнішній ключ, посилання на таблицю Employees

StudentSesion – містить інформацію про сесії студентів. Поля цієї таблиці показано в таблиці 3.13.

Таблиця 3.13

Поля таблиці StudentSesion

Назва поля	Тип даних	Опис
StudentSesionID	int	Первинний ключ таблиці
StudentRef	int	Зовнішній ключ, посилання на таблицю Students
EducationalSubjectRef	int	Зовнішній ключ, посилання на таблицю EducationalSubjects
TypeOfClassesRef	int	Зовнішній ключ, посилання на таблицю TypeOfClasses
EmployeeRef	int	Зовнішній ключ, посилання на таблицю Employees
StudentSesionName	nvarchar	Назва сесії
StudentSesionDesc	nvarchar	Опис сесії
StudentSesionScore	int	Оцінка за сесію
StudentSesionDate	datetime	Дата сесії
Semester	int	Семестр в якому відбулася сесія

Users – містить інформацію про користувачів. Поля цієї таблиці показано в таблиці 3.14.

Таблиця 3.14

Поля таблиці Users

Назва поля	Тип даних	Опис
UserID	int	Первинний ключ таблиці
UserName	nvarchar	ПІБ користувача
UserDesc	nvarchar	Опис користувача
UserLogin	nvarchar	Логін користувача
UserPassword	nvarchar	Пароль користувача
UserEmail	nvarchar	Електронна пошта
UserRoleRef	int	Зовнішній ключ, посилання на таблицю UserRole
UserStudentRef	int	Зовнішній ключ, посилання на таблицю Students
UserEmployeeRef	int	Зовнішній ключ, посилання на таблицю Employees

Gender – містить дані про статі людей. Поля цієї таблиці показано в таблиці 3.15.

Таблиця 3.15

Поля таблиці Gender

Назва поля	Тип даних	Опис
GenderID	int	Первинний ключ таблиці
GenderName	nvarchar	Назва статі
GenderCode	nvarchar	Код статі (FEM або MAL)

Schedule – містить інформацію про розклад занять. Поля цієї таблиці показано в таблиці 3.16.

Таблиця 3.16

Поля таблиці Schedule

Назва поля	Тип даних	Опис
ScheduleID	int	Первинний ключ таблиці
ScheduleName	nvarchar	Назва розкладу
ScheduleDesc	nvarchar	Опис розкладу
ScheduleDateFrom	datetime	Дата та час початку заняття
ScheduleDateTo	datetime	Дата та час закінчення заняття
GroupRef	int	Зовнішній ключ, посилання на таблицю Groups
EmployeeRef	int	Зовнішній ключ, посилання на таблицю Employees
EducationalSubjectRef	int	Зовнішній ключ, посилання на таблицю EducationalSubjects
TypeOfClassRef	int	Зовнішній ключ, посилання на таблицю TypeOfClasses

ThesisTypes – містить інформацію про дисертації. Поля цієї таблиці показано в таблиці 3.17.

Таблиця 3.17

Поля таблиці ThesisTypes

Назва поля	Тип даних	Опис
ThesisTypeID	int	Первинний ключ таблиці
ThesisTypeName	nvarchar	Назва дисертації

UserRoles – містить інформацію про ролі користувачів. Поля цієї таблиці показано в таблиці 3.18.

Таблиця 3.18

Поля таблиці UserRoles

Назва поля	Тип даних	Опис
UserRoleID	int	Первинний ключ таблиці
UserRoleName	nvarchar	Назва ролі
UserRoleDesc	nvarchar	Опис ролі
UserRoleXML	xml	Налаштування прав користувача у форматі xml

StudentExams - містить інформацію про екзамени студентів. Поля цієї таблиці показано в таблиці 3.19.

Таблиця 3.19

Поля таблиці StudentExams

Назва поля	Тип даних	Опис
StudentExamID	int	Первинний ключ таблиці
StudentExamScore	int	Назва спеціальності
StudentExamDate	datetime	Код спеціальності
StudentRef	int	Опис спеціальності
ExamRef	int	Зовнішній ключ, посилання на таблицю Faculties
StudentRef	int	Зовнішній ключ, посилання на таблицю Students
ExamRef	int	Зовнішній ключ, посилання на таблицю Exams

Positions – містить інформацію про посади. Поля цієї таблиці показано в таблиці 3.20.

Таблиця 3.20

Поля таблиці Positions

Назва поля	Тип даних	Опис
PositionID	int	Первинний ключ таблиці
PositionName	nvarchar	Назва посади
PositionDesc	nvarchar	Опис посади
PositionCanLearnAtLaboratory	bit	Може навчатися в лабораторіях
PositionCanLeadScientificTopic	bit	Чи може вести наукові статті
PositionCanLeadScientificDirection	bit	Чи може бути керівником аспірантів
PositionCanGiveLecture	bit	Чи може давати лекції
PositionCanGiveLaboratory	bit	Чи може проводити лабораторні заняття

Specialities – містить інформацію про спеціальності. Поля цієї таблиці показано в таблиці 3.21.

Таблиця 3.21

Поля таблиці Specialities

Назва поля	Тип даних	Опис
SpecialityID	int	Первинний ключ таблиці
SpecialityName	nvarchar	Назва спеціальності
SpecialityCode	nvarchar	Код спеціальності
SpecialityDesc	nvarchar	Опис спеціальності
SpecialityFacultyRef	int	Зовнішній ключ, посилання на Faculties

В цьому розділі описані основні таблиці та їх поля. Додаткових є значно більше та при потребі цю базу даних можна розширювати іншими таблицями та полями. Також для кожної таблиці створено збережену процедуру для виконання CRUD операцій, перегляди для запитів даних з БД та функції, які виконують логіку калькуляції та визначення даних та повертають значення. Ці функції використовуються в збережених процедурах та представленнях для запиту даних. В наступній частині цього розділу зображено ERD діаграму зв'язків таблиць університетської БД.

3.2.2 Архітектура університетської бази даних

На рис. 3.3 зображено зв'язки між основними таблицями бази даних чат-боту ІФНТУНГ-помічник. Дану діаграму створено засобами SSMS.

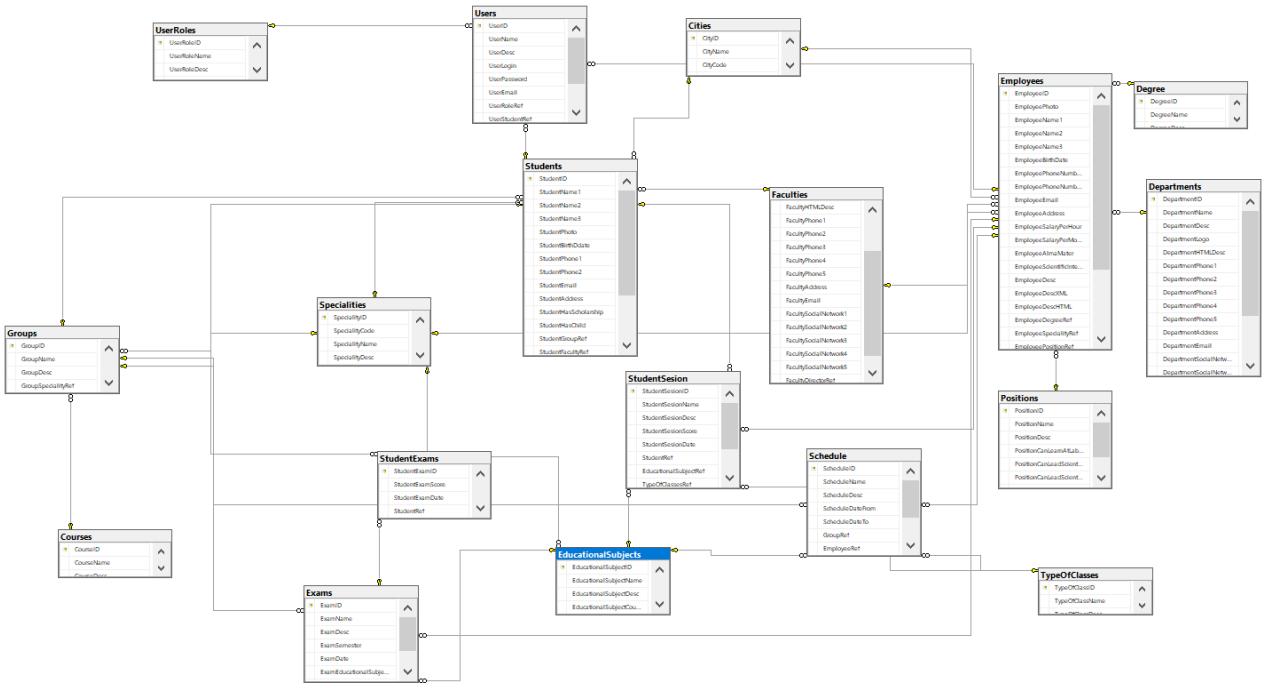

Рис. 3.3. Модель БД

Для опису архітектури БД розроблено модель БД показану на рис. 3.3, яка показує список таблиць (сутностей), їх поля та зв'зки між таблиця. На діаграмі додані тільки основні таблиці, є ще інші які рідше використовуються тому в

межах цієї роботи не описані. Дана діаграма відображає тільки загальну архітектуру, яку використано для проектування бази даних.

3.3 Модель інтеграції чат-ботів з університетськими базами даних

3.3.1 Опис моделі інтеграції чат-ботів з університетськими базами даних

Для візуального представлення моделі інтеграції БД університетів з чат-ботами побудовано UML діаграму класів, яка зображена на рис. 3.4. На цьому рисунку зображено загальну діаграму класів взаємодії. Окремі елементи UML діаграми класів подано на рис. 3.5 (клас Chatbot та його зв’язки з іншими сутностями системи) та рис. 3.6 (клас Middleware та його зв’язки з іншими сутностями системи).

Як видно із діаграм модель даної інтеграції побудована на основі таких класів: User, Chatbot, Middleware, WhatsApp/Telegram/Viber/Facebook, Database, AnalyticsService, NotificationService, CacheService, SecurityService, LogService, MS SQL Server/Oracle/Інші СКБД.

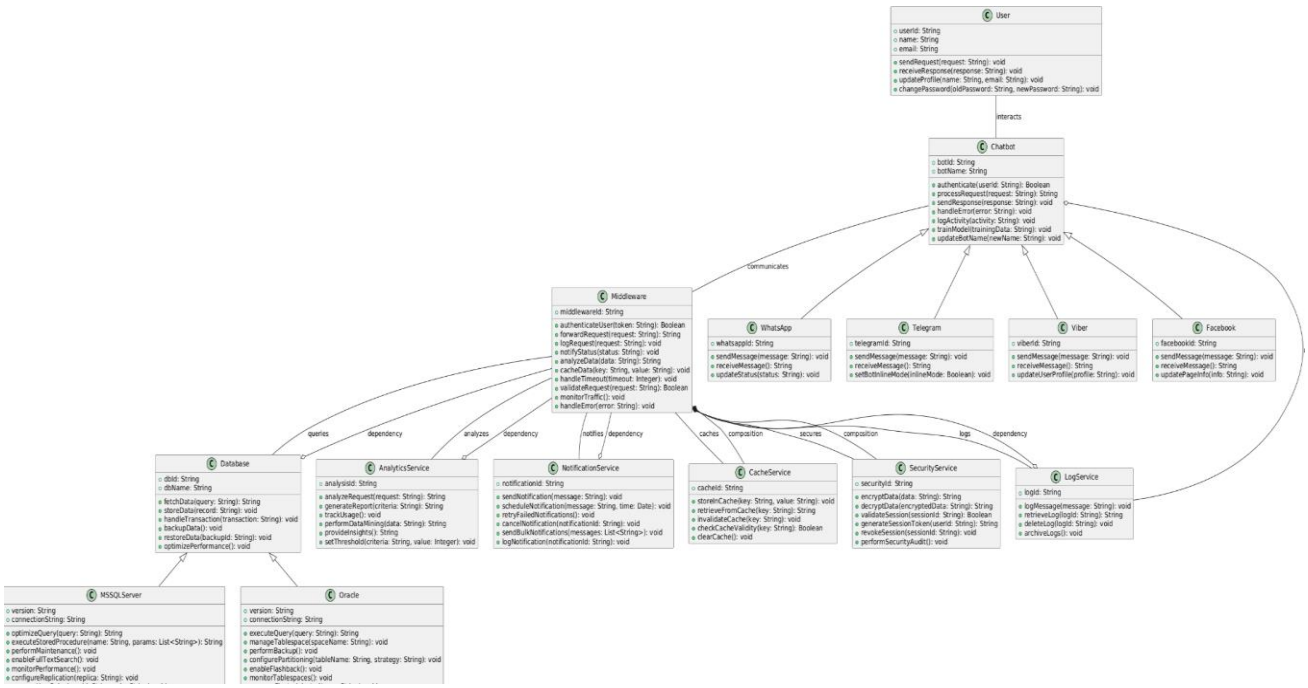

Рис. 3.4. Загальна UML діаграма класів

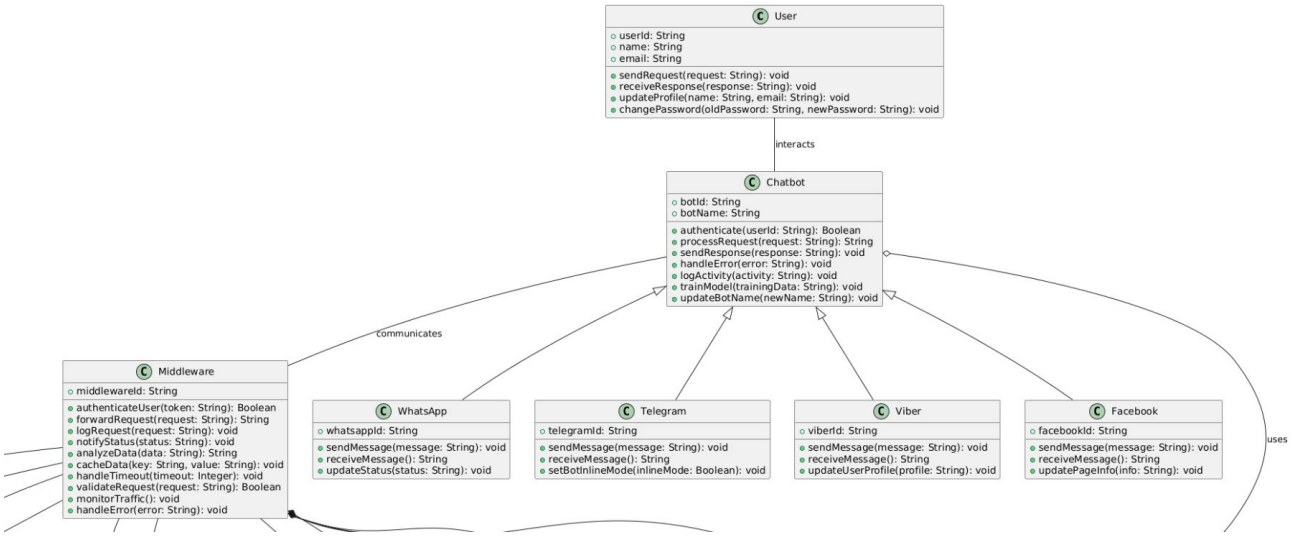

Рис. 3.5. UML діаграма класів (клас Chatbot та його зв’язки з іншими сутностями системи)

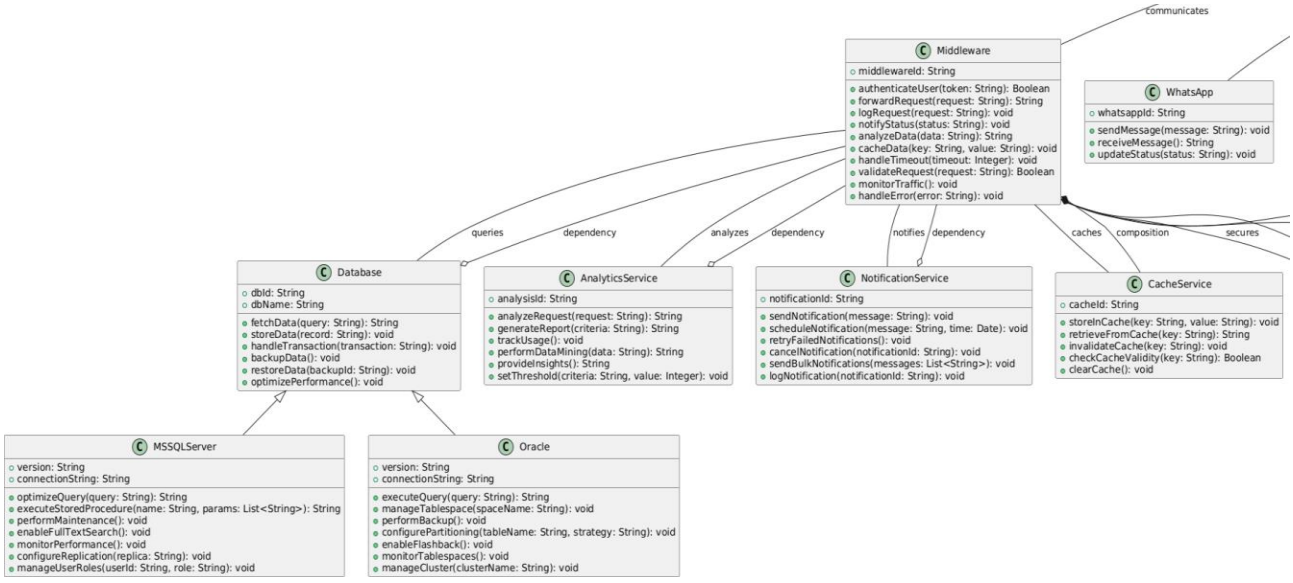

Рис. 3.6. UML діаграма класів (клас Middleware та його зв’язки з іншими сутностями системи)

3.3.2 Клас User

Клас User відповідальний за емуляцію поведінки користувача. Цей метод взаємодіє із класом Chatbot. Він має 3 основні поля:

- UserID - унікальний ідентифікатор користувача. По даному ідентифікатору користувачу присвоюються відповідні ролі;

- Name – ПІБ користувача ;
- Email – комерційна електронна пошта користувача за допомогою якої здійснено реєстрацію та авторизацію в чат-боті.

Клас User також має 4 основні методи:

sendRequest – цей метод призначений для відправки листів від користувачів. Він має вхідний параметр request, який приймає введений текст користувача;

- receiveResponse – цей метод призначений для надання відповідей користувачам. Він приймає вхідний параметр response, який міститиме текст відповіді для користувача на його запитання;
- updateProfile – цей метод приймає вхідні параметри такі, як ПІБ та електронну пошту користувача. За допомогою переданих цих параметрів даний метод редагує інформацію користувача в системі.
- changePassword – цей метод призначений для зміни паролю. Він приймає такі вхідні параметри, як oldPassword – старий пароль, та newPassword – новий пароль.

3.3.2 Клас Chatbot

Клас Chatbot призначений для представлення моделі чат-боту. Він має основні вхідні поля botId – ідентифікатор чат-боту та botName. Наприклад коли в нас є декілька платформ чат-ботів може бути 1 - WhatsApp, 2 – Viber і т. д. Він має такі основні методи:

- Authenticate – цей метод призначений для автентифікації в систему та приймає вхідний параметр userId, тобто ідентифікатор користувача на основі якого визначаються його права;
- processRequest – цей метод призначений для обробки повідомлень користувачів за допомогою заздалегіть оброблених алгоритмів. Він приймає вхідний параметр request, який приймає текст користувача, який потребує обробки;

- sendResponse – цей метод призаначений для відправки відповідей користувачу. Він приймає вхідний параметр response, який містить текст відповіді, яку потрібно відправити користувачу;
- handleError – цей метод призначений для обробки помилок. Він приймає вхідний параметр error, який містить опис помилки;
- logActivity – цей метод призначений для запису логувань дій чат-боту. Він приймає вхідний параметр activity, який містить інформацію про дію, яка відбулася та яку потрібно записати у лог файл;
- trainModel – цей метод призначений для навчання моделі відповідати на запитання користувачів використовуючи базу знань за допомогою якої штучний інтелект навчає модель надавати відповіді на запитання. Метод має вхідний параметр trainingData, який містить дані для тренування надання відповіді.

Метод Chatbot взаємодіє із класом User та його наслідують класи: WhatsApp, Telegram, Viber та Facebook. Також цей чат-бот агрегує клас LogService за допомогою якого здійснюється логування дій користувача.

3.3.3 Клас Middleware

Клас Middleware моделює проміжне програмне забезпечення за допомогою якою відбувається взаємодія між чат-ботом та базою даних.

Він має такі методи:

- authenticateUser – метод авторизації користувачів, який перевіряє відповідність токена;
- forwardRequest – цей метод передає запити користувачів до БД. Він має вхідний параметр request, який мість текст повідомлення користувача;
- logRequest – цей метод записує запитання користувача у логи. Містить вхідний параметр request, який містить текст запитання, який потрібно записати в логи;

- notifyStatus – цей метод призначений для перевірки статусу нотивікації чи запит отримав відповідь. Він має вхідний параметр status, який може містити значення (доставлено/в обробці/не вдалося доставити);
- analyzeData – цей метод призначений для аналізу введених даних. Приймає параметр data, який містить дані, які потрібно проаналізувати;
- cacheData – цей метод призначений для кешування даних, щоб пришвидшити процес надання відповідей. Має вхідний параметр key, за допомогою якою користувач зможе отримати кешовану відповідь;
- handleTimeout – цей метод призначений для обробки затримок з відповідями, які виникають під час використання чат-боту. Приймає параметр timeout, кількість мілісекунд затримки відповіді.
- validateRequest – цей метод призначений для валідації запитів. Приймає параметр request, який містить запит, який потрібно валідувати;
- monitorTraffic – призначений для моніторингу трафіку системи;
- handleError – призначений для обробки помилок.

3.3.4 Класи WhatsApp/Telegram/Viber/Facebook

Класи WhatsApp/Telegram/Viber/Facebook – презентують моделі платформ чат-ботів. Їх може бути скільки завгодно. В даному прикладі розглянуто 4 популярні платформи на яких будують чат-боти. Ці класи наслідуються від класу Chatbot. Вони містять такі методи:

- sendMessage – цей метод призначений дляя відправки повідомлень. Він приймає вхідний параметр message – повідомлення, яке потрібно доставити та UserID – користувача, якому потрібно відправити повідомлення;
- receiveMessage – метод призначений для отримання повідомлень від користувача. Він приймає текстові повідомлення та зворотні відповіді користувача;
- setBotInlineMode – метод призначений для встановлення вбудованого режиму відповідей.

3.3.5 Клас Database

Клас Database представляє загальну модель університетської бази даних, яка може складатися із компонентів, розроблених на різних СКБД. Він призначений для надання можливості взаємодії із БД. Цей клас асоціюється із класом Middleware, а також має дочірні класи MS SQL Server та Oracle. Можуть бути додані інші СКБД при потребі. Цей клас містить такі методи:

- fetchData – метод для отримання даних від користувача;
- storeData – метод для збереження даних;
- handleTransaction – метод для обробки транзакцій;
- backupData – метод для бекапу даних;
- restoreData – метод для відновлення даних;
- optimizePerformance – метод для оптимізації швидкодії БД, який може виконувати операції ребілду та реорганайзу індексів та оновлення статистики.

3.3.6 Клас AnalyticsService

Клас AnalyticsService призначений для аналітики використання системи. Він має залежність від класу Middleware, оскільки для аналітики потребує даних з бази даних, які може отримати за допомогою класу Middleware. Цей клас містить такі методи:

- analyzeRequest – призначений для аналізу запиту;
- generateReport – призначений для генерування звіту;
- trackUsage – призначений для відслідковування використання системи;
- performDataMining – призначений для видобутку даних по продуктивності системи;
- provideInsights – призначений для надання основних подій інтеграції;
- setThreshold – призначений для встановлення порогу граничних значень системи, таких, як затримки відповідей та інші конфігурації системи.

3.3.6 Клас NotificationService

Клас NotificationService призначений для обробки нотифікацій системи. Він має залежність від класу Middleware, оскільки сповіщення зберігаються та створюються в базі даних, а дані з БД цей клас може отримати за допомогою класу Middleware. Він має такі методи:

- sendNotification – метод відправки сповіщень;
- scheduleNotification – метод відправки сповіщений, який дозволяє запланувати розсилку на встановлений час;
- retryFailedNotifications – метод, який намагається повторно відправити повідомлення, які не вдалося доставити;
- cancelNotification – метод скасування сповіщень;
- sendBulkNotifications – метод масових розсилок, наприклад про новини університету, оновлення розкладу і т. д.;
- logNotification – метод логування сповіщень.

3.3.7 Клас CacheService

Клас CacheService використовується для кешування даних користувача з метою пришвидшення надання відповідей. Цей метод має залежність від класу Middleware, так, як без цього класу не зможе отримувати інформацію з БД і відповідно не буде мати даних для кешування. Він має такі методи:

- storeInCache – метод збереження кешу, який призначений для оптимізації надання відповідей користувачам;
- retrieveFromCache – метод отримання даних з кешу по ключу;
- invalidateCache – метод деактивації кешу для видалення астарілих даних із кеш-пам’яті;
- checkCacheValidity – перевірка правильності кешу, який здійснюється на основ перевірки кеш-пам’яті;
- clearCache – метод очищення кешу, який цілком очищує кеш для користувача системи.

3.3.8 Клас SecurityService

Клас SecurityService забезпечує безпеку системи. Він має відношення до класу Middleware через який проходить весь трафік системи. Цей клас має такі методи:

- encryptData – метод шифрування даних;
- decryptData – метод розшифрування даних;
- validateSession – метод перевірки сесій;
- generateSessionToken – метод генерації токенів;
- revokeSession – метод відхилення сесії;
- performSecurityAudit – метод, який забезпечує аудит для безпеки системи.

3.3.9 Клас LogService

Клас LogService призначений для ведення логів системи. Він має зв’язок із класом Middleware, за допомогою якою здійснюються логування. Цей клас має такі методи:

- logMessage – методу запису в логи;
- retrieveLog – метод отримання інформації з логів
- deleteLog – метод видалення інформації з логів
- archiveLogs – метод архівування логів

3.4 Проектування та програмна реалізація алгоритму інтеграції чат-боту з університетськими базами даних. Розробка алгоритму чат-боту

3.4.1 Проектування та програмна реалізація алгоритму інтеграції чат-боту з університетськими базами даних. Опис технологій та компонентів, які використані при розробці інтеграції

Програму реалізовано да допомогою технологій стеку .NET та хмарних сервісів на Azure. Опис використаних технологій для кожного рівень:

1) База даних – університетську БД розроблено в СУБД MS SQL Server. В ній створено таблиці для зберігання даних, зв’язки між ними та набір об’єктів для взаємодії БД із 2 рівнем проміжним ПЗ. Серед цих об’єктів є збережені процедури, функції, перегляди, тригери та джоби. Нижче надено опис цих об’єктів та їхнє призначення:

- Збережені процедури – використовуються для виконання CRUD операцій, а також за їх допомогою забезпечується логіка роботи бекенду. Ці збережені процедури мають набір параметрів за допомогою яких виконують призначену для них логіку;
- Функції – виконують встановлену для них логіку згідно переданих вхідних параметрів. Відрізняються від процедур тим, що повертають значення, які можна використовувати у вибірках даних;
- Представлення – ці об’єкти БД використовуються для полегшення вибірки даних та виконання запитів із БД
- Тригери – використовують для автоматичного виконання подій. В даному випадку використані також для логування дій користувачів. Тобто коли виконується запит, наприклад вставлення даних в таблицю Students відпрацьовує тригер, який логує інформацію про те хто та коли створив студента в БД.
- Джоби – використовуються для обслуговування БД, ребілду та реорганайзу індексів, налаштування автоматичних бекапів БД та їх відновлень. Також за допомогою SQL Server Agent налаштовано відправку електронних листів за допомогою вбудованого mail сервісу.

2) Проміжне програмне забезпечення (middleware) – побудоване в основному за допомогою .NET технологій та хмарних сервісів Azure. Зокрема використано ADO.NET – для роботи з БД, викликів процедур, виконання запитів, LINQ – для роботи з колекціями та структурами даних, CLR – для виконання C# коду в збережених процедурах та Entity Framework для роботи із сутностями.

3) Чат-бот – при розробці чат-боту використано також технології .NET та бібліотеки API телеграм платформи, які дозволяють створювати інтерфейс в

чат-боті та надають змогу обробки запитів в чат-боті. Серед основних бібліотек використано такі:

- Telegram.Bot – це стандартна бібліотека телеграм платформи, яка дозволяє інтегрувати чат-бот у застосунок на .NET за допомогою токен, який реєструється в чат-боті BotFather;
- Telegram.BotTypes – за допомогою цієї біліотеки здійснено керування різними типами повідомлень чат-боту ткелеграму, такі, як списки, каруселі, тексти, зображення та інше
- Telegram.Bot.Types.Enums – за допомогою цієї бібліотеки вібувається взаємодія
- Telegram.Net – за допомогою цієї бібліотеки реалізовуємо можливість виконання в чат-боті HTTP-запитів для взаємодії з сторонніми сервісами. Для цього також додаємо 2 суміжні бібліотеки NetTelegramBotApi та NetTelegramBotApi.Types для виконання API запитів різних типів.

3.4.2 Програмна реалізація логіки backend на стороні БД

Для забезпечення логіки роботи ситеми на стороні БД розроблено процедури. Для прикладу при реєстрації користувача система перевіряє чи є такий користувач у системі і, якщо немає тоді вирішиє створити нового. Створення нового користувача відбувається за допомогою CRUD процедур для створення, додавання, редагування та зчитування даних.

Results | Messages

	CRUDOperationID	CRUDOperationName	CRUDOperationDesc
1	1	Create	Додавання нових даних
2	2	Select	Виведення даних
3	3	Update	Редагування існуючих даних
4	4	Delete	Видалення даних

Рис. 3.7. Таблиця із CRUD операціями

Для забезпечення роботи CRUD операцій створено окрему таблицю, яка містить 4 основні операції: видалення, створення, редагування та вибірка. Для кожної сутності створено процедуру для виконання цих операцій. Назви процедур прописуються по конвеншину sp + сутність + CRUD. Приклад такої процедури є spEmployeeCRUD. Ця процедура призначена для створення, редагування, видалення та вибірки даних із таблиці Employees, яка містить інформацію про викладачів та працівників університету. Вона має такі обов'язкові параметри, які подано нижче:

- @EmployeeID int = null out - цей параметр позначений, як out, тобто він є вихідний. Коли виконуємо операції update/delete його потрібно вказувати, але коли виконуємо операцію insert/select то процедура поверне ідентифікатор існуючого або новоствореного працівника;
- @CRUDOperationID int – цей вхідний параметр може бути цілим числом від 1 до 4, і він вказує на операцію, яку нам потрібно виконати. Ці опеарції визначаються на основі таблиці 3.7;
- @Login nvarchar (200) – логін користувача під яким виконується метод.

Далі слідують поля таблиці для якої сутності виконуємо CRUD, наприклад, якщо виконуємо операції для таблиці Employees то передаємо поля цієї таблиці, такі, як: EmployeeName1, EmployeeName2, EmployeeName3 і т.д. Типи даних цих полів відповідають типам даних із таблиць, які наведено в попередьому розділі

Після цього оголошуємо тимчасову табличну змінну, яку буде зберігати ідентифікатор сутності. Вона оголошується таким чином.

Лістинг 3.1 Оголошення тимчасової табличної змінної для зберігання доданих значень

```
declare @EmployeeesTemp table (EmployeeID int)
```

Після цього відбувається перевірка чи вказані правильні зовнішні ключі для забезпечення цілісності даних та зв'язків між таблицями.

Лістинг 3.2 Перевірка обмежень зовнішніх ключів.

```
if @EmployeeDegreeRef is not null and not exists (
```

```
        select * from Degree with (nolock)
        where DegreeID = @EmployeeDegreeRef
      )
      begin
        select N'Не вдалося додати працівника університету.
Вказаний не існуючий науковий ступінь'
        return 0
      end

      if @EmployeeSpecialityRef is not null and not exists (
        select * from Specialities with (nolock)
        where SpecialityID = @EmployeeSpecialityRef
      )
      begin
        select N'Не вдалося додати працівника університету. Вказано
не існуючу спеціальність'
        return 0
      end
```

Після того, як відбулася перевірка правильності зовнішніх ключів виконуються самі CRUD операції. Якщо параметром CRUD операції є 1 то відбувається створення нового запису.

Лістинг 3.3 Імплементація команди додавання даних.

```
      if @CRUDOperationID = 1
      begin
        insert Employees
          (EmployeePhoto,                          EmployeeName1,
EmployeeName2,          EmployeeName3,
          EmployeeBirthDate,                 EmployeePhoneNumber1,
EmployeePhoneNumber2,   EmployeeEmail,
          EmployeeAddress,                 EmployeeSalaryPerHour,
EmployeeSalaryPerMonth, EmployeeAlmaMater,
          EmployeeScientificInterest,              EmployeeDesc,
EmployeeDescXML,        EmployeeDescHTML,
          EmployeeDegreeRef,               EmployeeSpecialityRef,
EmployeePositionRef,    EmployeeGenderRef,
          EmployeeCityRef,                    EmployeeFacultyRef,
EmployeeDepartmentRef,  EmployeeHasChild,
          EmployeeChildCount)
        select
          @EmployeePhoto,                          @EmployeeName1,
@EmployeeName2,          @EmployeeName3,
          @EmployeeBirthDate,               @EmployeePhoneNumber1,
@EmployeePhoneNumber2,   @EmployeeEmail,
          @EmployeeAddress,               @EmployeeSalaryPerHour,
@EmployeeSalaryPerMonth, @EmployeeAlmaMater,
          @EmployeeScientificInterest,             @EmployeeDesc,
@EmployeeDescXML,        @EmployeeDescHTML,
```

```
          @EmployeeDegreeRef,              @EmployeeSpecialityRef,
@EmployeePositionRef,    @EmployeeGenderRef,
          @EmployeeCityRef,                   @EmployeeFacultyRef,
@EmployeeDepartmentRef,  @EmployeeHasChild,
          @EmployeeChildCount

        select @EmployeeID = EmployeeID from @EmployeeesTemp
      end
```

Якщо параметром CRUD операції є 2 то відбувається зчитування даних на основі введених вхідних параметрів.

Лістинг 3.4 Імплементація команди зчитування даних.

```
      else if @CRUDOperationID = 2

      begin
        select column1, … from Employees with (nolock)
        where (EmployeeID = @EmployeeID or @EmployeeID is null)
          and   (EmployeeBirthDate   =   @EmployeeBirthDate   or
@EmployeeBirthDate is null)
          and  (EmployeePhoneNumber1  =  @EmployeePhoneNumber1  or
@EmployeePhoneNumber1 is null)
          and  (EmployeePhoneNumber2  =  @EmployeePhoneNumber2  or
@EmployeePhoneNumber2 is null)
          and (EmployeeEmail = @EmployeeEmail or @EmployeeEmail is
null)
          and  (EmployeeName1  like  concat(N'%',  @EmployeeName1,
N'%') or @EmployeeName1 is null)
          and  (EmployeeName2  like  concat(N'%',  @EmployeeName2,
N'%') or @EmployeeName2 is null)
          and  (EmployeeName3  like  concat(N'%',  @EmployeeName3,
N'%') or @EmployeeName3 is null)
          and   (EmployeeDegreeRef   =   @EmployeeDegreeRef   or
@EmployeeDegreeRef is null)
          and (EmployeeSpecialityRef = @EmployeeSpecialityRef or
@EmployeeSpecialityRef is null)
          and   (EmployeePositionRef   =   @EmployeePositionRef   or
@EmployeePositionRef is null)
          and   (EmployeeGenderRef   =   @EmployeeGenderRef   or
@EmployeeGenderRef is null)
          and   (EmployeeFacultyRef   =   @EmployeeFacultyRef   or
@EmployeeFacultyRef is null)
          and (EmployeeDepartmentRef = @EmployeeDepartmentRef or
@EmployeeDepartmentRef is null)
          and    (EmployeeCityRef    =    @EmployeeCityRef    or
@EmployeeCityRef is null)
          and    (EmployeeHasChild    =    @EmployeeHasChild    or
@EmployeeHasChild is null)
          and   (EmployeeChildCount   =   @EmployeeChildCount   or
@EmployeeChildCount is null)
      end
```

Якщо параметром CRUD операції є 3 то відбувається редагування існуючих даних на основі введених вхідних параметрів.

Лістинг 3.5 Імплементація команди редагування даних.

```
else if @CRUDOperationID = 3
  begin
    update Employees set
      EmployeePhoto              = @EmployeePhoto,
      EmployeeName1              = @EmployeeName1,
      EmployeeName2              = @EmployeeName2,
      EmployeeName3              = @EmployeeName3,
      EmployeeBirthDate          = @EmployeeBirthDate,
      EmployeePhoneNumber1       = @EmployeePhoneNumber1,
      EmployeePhoneNumber2       = @EmployeePhoneNumber2,
      EmployeeEmail              = @EmployeeEmail,
      EmployeeAddress            = @EmployeeAddress,
      EmployeeSalaryPerHour      = @EmployeeSalaryPerHour,
      EmployeeSalaryPerDay       = @EmployeeSalaryPerDay,
      EmployeeSalaryPerMonth     = @EmployeeSalaryPerMonth,
      EmployeeAlmaMater          = @EmployeeAlmaMater,
      EmployeeScientificInterest                              =
@EmployeeScientificInterest,
      EmployeeDesc               = @EmployeeDesc,
      EmployeeDescXML            = @EmployeeDescXML,
      EmployeeDescHTML           = @EmployeeDescHTML,
      EmployeeDegreeRef          = @EmployeeDegreeRef,
      EmployeeSpecialityRef      = @EmployeeSpecialityRef,
      EmployeePositionRef        = @EmployeePositionRef,
      EmployeeGenderRef          = @EmployeeGenderRef,
      EmployeeCityRef            = @EmployeeCityRef,
      EmployeeFacultyRef         = @EmployeeFacultyRef,
      EmployeeDepartmentRef      = @EmployeeDepartmentRef,
      EmployeeHasChild           = @EmployeeHasChild,
      EmployeeChildCount         = @EmployeeChildCount
     where EmployeeID = @EmployeeID
  end
```

Якщо параметром CRUD операції є 4 то відбувається редагування існуючих даних на основі введених вхідних параметрів. Для видалення рекомендовано перевіряти по ідентифікатору.

Лістинг 3.6 Імплементація команди видалення даних.

```
else if @CRUDOperationID = 4
  begin
    delete from Employees
    where EmployeeID = @EmployeeID
  end
```

В даному прикладі наведено тільки фрагменти логіки, як відбуваються CRUD операції на стороні БД за допомогою збережених процедур на прикладі таблиці Employees. Повний опис коду цих процедур додано в додаток А.

3.4.2 Програмна реалізація проміжного програмного забезпечення (middleware)

На цьому рівні реалізовано взаємодію елементів БД та чатботу через набір API методів. Також для прикладу наведено логіку роботи для сутності Employees.

Першим етапом потрібно додати всі поля цієї сутності.

Лістинг 3.7 програмне оголошення полів.

```
public int EmployeeID { get; set; } [MaxLength(200)]
public string? EmployeeName1 { get; set; }
```

Після цього за допомогою технології Entity Framework додаємо зовнішні ключі для цієї сутності.

Лістинг 3.8 програмне оголошення полів.

```
property for Gender
        [ForeignKey("Department")]
        public int? EmployeeDepartmentRef { get; set; }
        public Department? Department { get; set; }
```

Для встановлення зв’язків між основною та зв’язаною таблицею використано бібліотеки DataAnnotations та DataAnnotations.Schema, яка представляє схему цих сутностей та дозволяють встановлювати анотації такі, як [ForeignKey("Department")]. Ці дії повторюємо для кожного поля таблиці та після цього генеруємо модель класів. Вона повинна відповідати змісту цих таблиць.

Лістинг 3.9 Додавання набору даних.

```
public DbSet<Employee> Employees { get; set; }
```

Далі потрібно перевизначити метод. При цьому нам треба перевизначити модель цього класу.

Лістинг 3.10 Перевизначення класу.

```
    protected override void OnModelEditing(ModelBuilder
modelBuilder)
    {
```

```
                modelBuilder.Entity<Employee>()
                    .HasOne<Departments>(s => s. Department)
                    .WithMany(g => g.Employees)
                    .HasForeignKey(s => s.EmployeeDepartmentRef);
    }
```

Після цього за допомогою технології Entity Framework виконуємо міграцію сутностей на основі створених моделей класів. Коли ми мігрували схему сутностей можемо розпочати роботу над створенням контролера. Для цього створюємо окремий клас EmployeeController, який буде містити обробку API запитів. Треба для кожного виду операцій створити свій запит, для видалення DELETE, для створення – POST, для редагування – PUT, для отримання даних – GET. Для виконання API запитів підключаємо бібліотеку Http та створюємо клас контекст таблиці, який буде вміщати вміст даної таблиці.

Лістинг 3.11 Створення контексту.

```
protected static EmployeeContext _EmployeeContext;
public EmployeesController(EmployeeContext EmployeeContext)
{
     _EmployeeContext = EmployeeContext;
}
```

Далі клас-контект потрібно використати безпосередньо в API методі. За допомогою анотації Http + тип запиту(Delete, Post, Put, Get). Приклад такого GET-запиту наведено нижче.

Лістинг 3.12 Створення асинхронного методу.

```
private    async    Task<ActionResult<IEnumerable<Employee>>>    Get
Employee s()
{
     if (_Employee Context. Employee e == null)
     {
          return NotFound(); // якщо не знайдено такого виклкдача
     }
     else
     {
          // відбувається очікування на відповідь серверу
          return await _ EmployeeContext. Employees.ToListAsync();
     }
}
```

Інші види API запитів є аналогічні для всіх сутностей, змінюється тільки анотація в залежності від типу запиту та тіло моделі. Також, якщо тип запиту GET по не передається тіло у метод, а тільки параметри

Лістинг 3.13 Створення API методу додавання даних.

```
            [HttpPost]
            public async Task<ActionResult<Employee>> Post Employee (Employees Employee)
            {
                _ EmployeeContext.Employees.Add(Employee);
                await _ Employee Context.SaveChangesAsync();
                return CreatedAtAction(nameof(Get Employee), new { id = Employee. Employee ID }, Employee);
        }
```

API методи DELETE та PUT робомо аналогічно, як GET та POST тільки із іншим тілом запиту. Для передавання тіла запиту використовується формат передачі даних JSON.

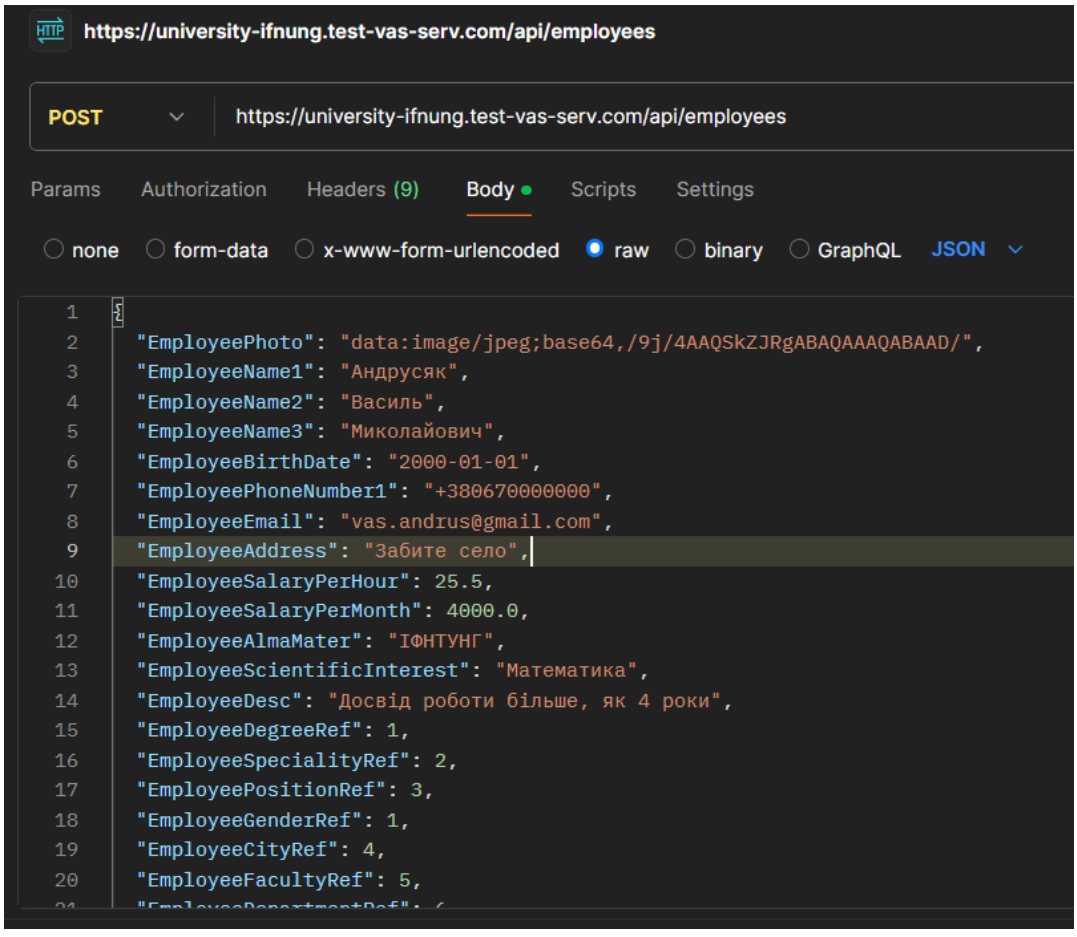

Рис. 3.8. Створення нового працівника через POST запит

Після цього можна протестувати дані API методи за допомогою таких інструментів, як Postman або Soap UI. На рис. 3.8 показано, як створити нового

працівника. Для цього потрібно вставити посилання по якому будемо робити запит до серверу та вибрати метод POST. Далі переходимо на вкладку raw та додаємо тіло в якому вказуємо інформацію про працівника. Тіло вказуємо у форматі JSON та в рядку під Body обираємо JSON формат для того, щоб Postman його розпізнав.

На рис. 3.9 зображено, як робити вибірку працівників. Для цього робимо запит на посиланням проміжного серверу (middleware) із типом GET. Body залишаємо порожнє, оскільки GET запити не містять body, а мають тільки паарметри. В посилання після api/ дописуємо назву сутності, в даному випадку employees та після цього, як нас цікавить працівник під конкретним ідентифікатором вказуємо після слеша його ідентифікатор. Інші методи редагування та видалення робимо так само, як вказані вище, тільки змінюємо тип запиту, якщо редагуємо вказуємо PUT, якщо видаляємо вказуємо DELETE.

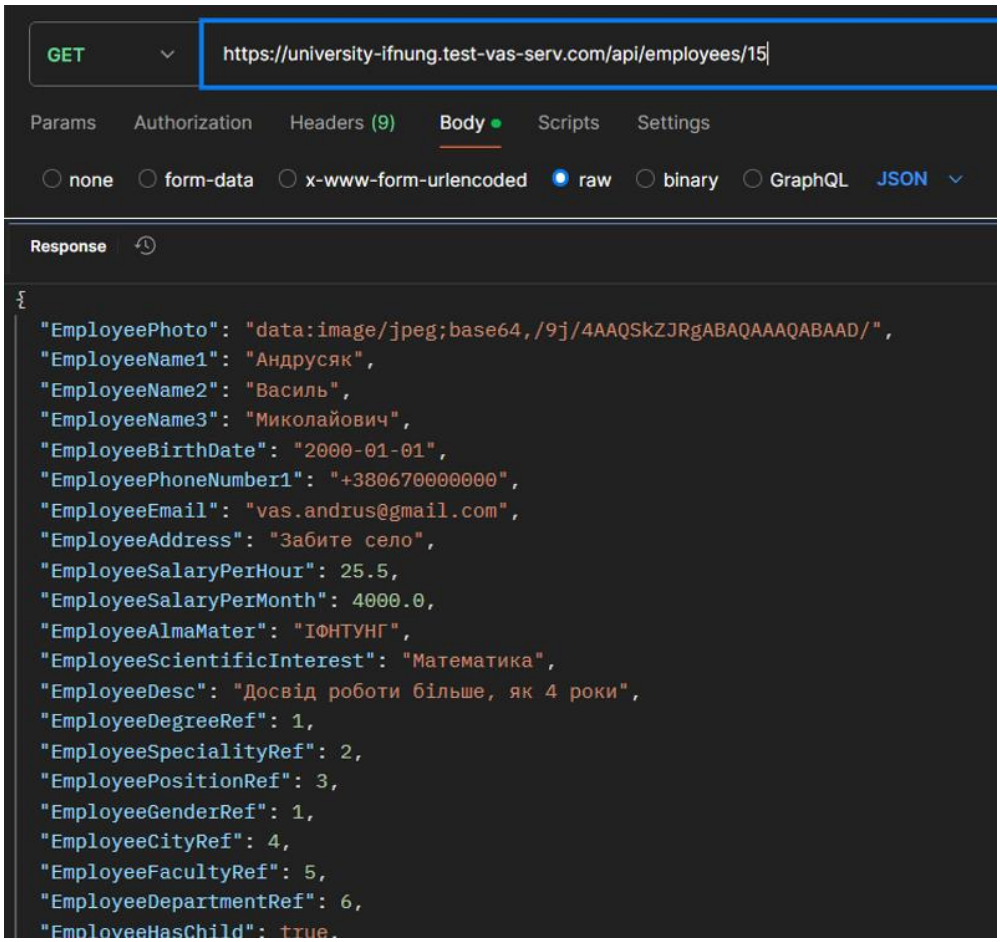

Рис. 3.9. Отримання інформації про працівника через GET запит

3.4.3 Програмна реалізація алгоритму роботи чат-боту

Даний чат-бот реалізовано на платформі Telegram, яка має набір вбудованих бібліотек. При налаштуванні цієї інтеграції використано основні пакети: Telegram, TelegramBotTypes, TelegramApi та Telegram.Client.Api. Для авторизації використано бібліотеку RestSharp, яка надає змогу вбудованої роботи із токенами. Також для серіалізації та десеріалізації об’єктів, оскільки API методи приймають body в JSON форматі та в цьому формані надають відповіді, використано бібліотеку Newtonsoft.Json. Налаштування системи здійснено за допомогою бібліотеки Microsoft.Extensions.Configuration.

Після додавання цих пакетів додаємо бібліотеки Telegram API. Приклади цих бібліотек включають: Bot, Exceptions, Polling, Types, Enums, ReplyMarkups. Після підключення цих бібліотек потрібно згенерувати токен для чат-боту. Це можна зробити за допомогою бота BotFather. В ньому потрібно ввести команду \token та після цього вказати назву чат-боту. Потім за допомогою команди /mybots перевіряємо список наших чат-ботів та можемо звідси взяти токен.

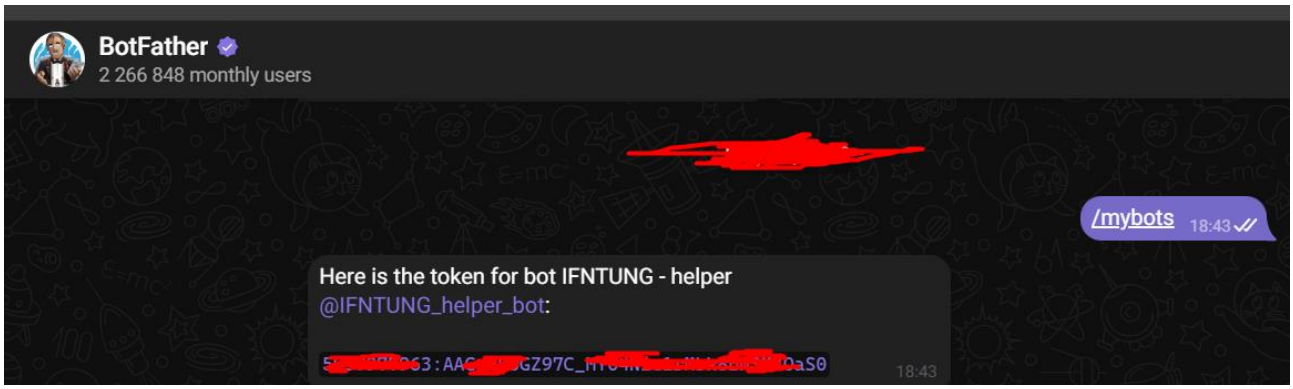

Рис. 3.10. Створення токена

Після створення токена його потрібно використати у конфігураційному файлі appsettings.json. Створюємо змінну, яка буде містити цей токен та призначаємо їй значення з цього поля. Приклад подано нижче.

Лістинг 3.14 Присвоєння токена чат-боту.

```
var conf = new ConfigurationBuilder()
```

```
        .AddJsonFile("appsettings.json", optional: true,
reloadOnEdit: true)
        .Build();
    BotToken = conf ["BotToken"];
```

Далі в чат-боті BotFather додаємо команди меню за допомогою команди /setcommands + текст команди + (-) + інформаційний текст для користувача. Приклад створення команд меню:

- /setcommands universityInfo - Інформація про університет
- /setcommands universityNews - Новини уніеврситету
- /setcommands authorization - Авторизація
- /setcommands registration - Реєстрація

Після створення меню обробляємо логіку роботи чат-боту в методі main, яка виконує підключення чат-бота до системи. Приклад підключення подано нижче.

Лістинг 3.15 Підключення чат-боту до застосунку.

```
clientTelegram = new BotClient(BotClient);

var cancellationToken = new CancellationTokenSource();
var receiverOptions = new ReceiverOptions
{
    AllowedUpdates = { } // тут вказуються апдейти
};
client.StartReceiving(
    HandleUpdateAsync, receiverOptions,HandleErrorAsync,
    cancellationToken: cts.Token
);
Console.ReadLine();
cts.Cancel();
```

Після цього імплементуємо логіку, обробки повідомлень, які чат-бот отримує від користувача. Для цього створюємо клас QueuedUpdateReceiver.

Лістинг 3.16 Створення класу обробки подій.

```
    public      class      QueuedUpdateReceiver(ITelegramBotClient
botClient, ReceiverOptions? receiverOptions = default,
        Func<Exception,         CancellationToken,         Task>?
pollingErrorHandler = default) : IAsyncEnumerable<Update>
```

Далі в цьому класі додаємо асинхронний метод обробки винятків, які виникають при запитах користувачів.

Лістинг 3.17 Створення методу асинхронної обробки подій.

```
    public                                    IAsyncEnumerator<Update>
GetAsyncEnumerator(CancellationToken cancellationToken = default)
        {
            if (Interlocked.CompareExchange(ref _inProcess, 1, 0)
is 1)
                throw                                              new
InvalidOperationException(nameof(GetAsyncEnumerator) + " may only be
called once");
            return    _enumerator    =    new(receiver:    this,
cancellationToken: cancellationToken);
        }
```

Цей метод здійснює валідацію повідомлень, передає їх на middleware, яке після цього виконує API метод.

В цьому коді використовуємо об’єкт client класу BotClient, та ініціалізуємо його токеном нашого бота. Після цього налаштовуємо метод отримання даних від чат-боту та відповідні їхні обробки. Цю логік подано нижче в цій роботі.

Далі взаємодія із користувачем відюується за допомогою методів OnUpdateHandler та OnMessageHandler.

Метод OnUpdateHandler відпрацьовує при кліку користувача на об’якти чат-боту, наприклад списки, форми, діаграми та кнопки. Метод OnMessageHandler відпрацьовує коли користувач самостійно вводить повідомлення.

Лістинг 3.18 Оголошення методів обробки подій та відправки повідомлень:

```
      public event OnUpdateHandler? OnUpdate { add { _onUpdate +=
value; StartEventReceiving(); } remove { _onUpdate -= value;
StopEventReceiving(); } }
        public event OnMessageHandler? OnMessage { add { _onMessage
+= value; StartEventReceiving(); } remove { _onMessage -= value;
StopEventReceiving(); } }
```

Код чат-боту можна подивитися у додатку В.

3.4.4 Результати інтеграції чат-боту із уніерситетськими БД

В результаті інтеграції розроблено чат-бот, який вміє працювати із БД, виконувати запити та обробляти в ній транзакції. Для початку роботи з чат-ботом потрібно зареєструватися. При реєстрації чат-бот просить ввести користувача

свою електронну пошту. Після введення електронної пошти система виводить користувачу повідомлення про те, що на вказану електронну пошту відправлено код підтвердження. Після підтвердження коду ситема просить ввести логін та пароль. Після того, як користувач пройшов всі кроки реєстрації отримує повідомлення, що він успішно зареєструвався та оновлюється нижнє меню. Приклад показано на рис. 3.11

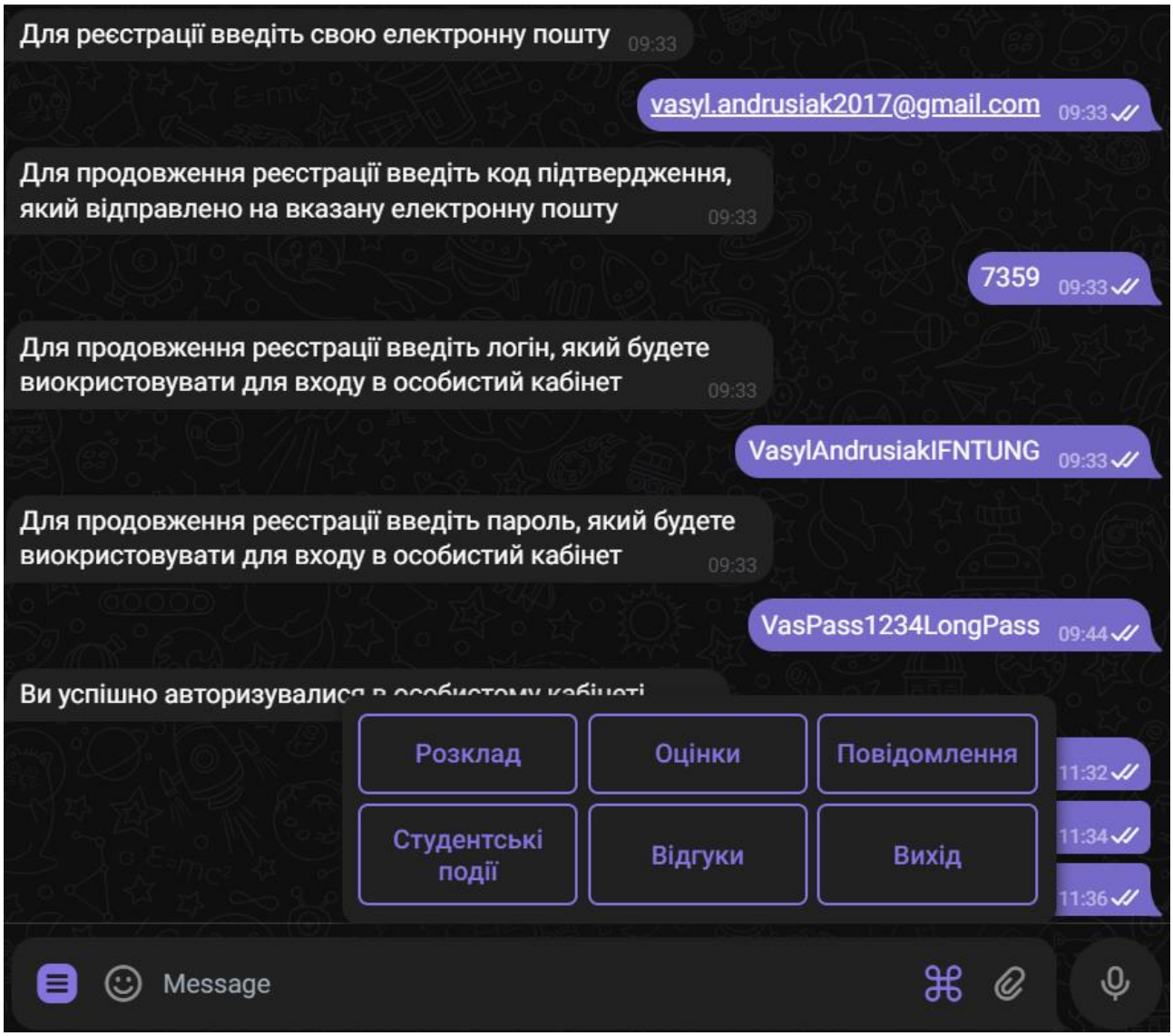

Рисунок 3.11. Переписка з чат-ботом при реєстрації

Далі користувач може використовувати функціонал, який доступний для його ролі. В цьому прикладі зайшов під користувачем з роллю студент, для якого доступна функція перегляду розкладу. Користувач із меню натискає на розклад

та отримує відповідь від чат-бота про те, що потрібно ввести дату за яку він хоче переглянути розклад. Користувач вводить дату та у відповідь отримує свій розклад. Показано на рис. 3.12. Як можна помітити із рисунка нижче, що чат-бот виводить інформацію у зручному для користувача вигляді. В даному прикладі також використані емоджі. Можна також додавати інші графічні елементи, різні зоюраження, графіки і т.д. Для перегляду іншого функціоналу розроблено нижнє меню за допомогою якого користувач може переглянути свої оцінки, написати та переглянути свої повідомлення, переглянути студентські події та переглядати і залишати відгуки.

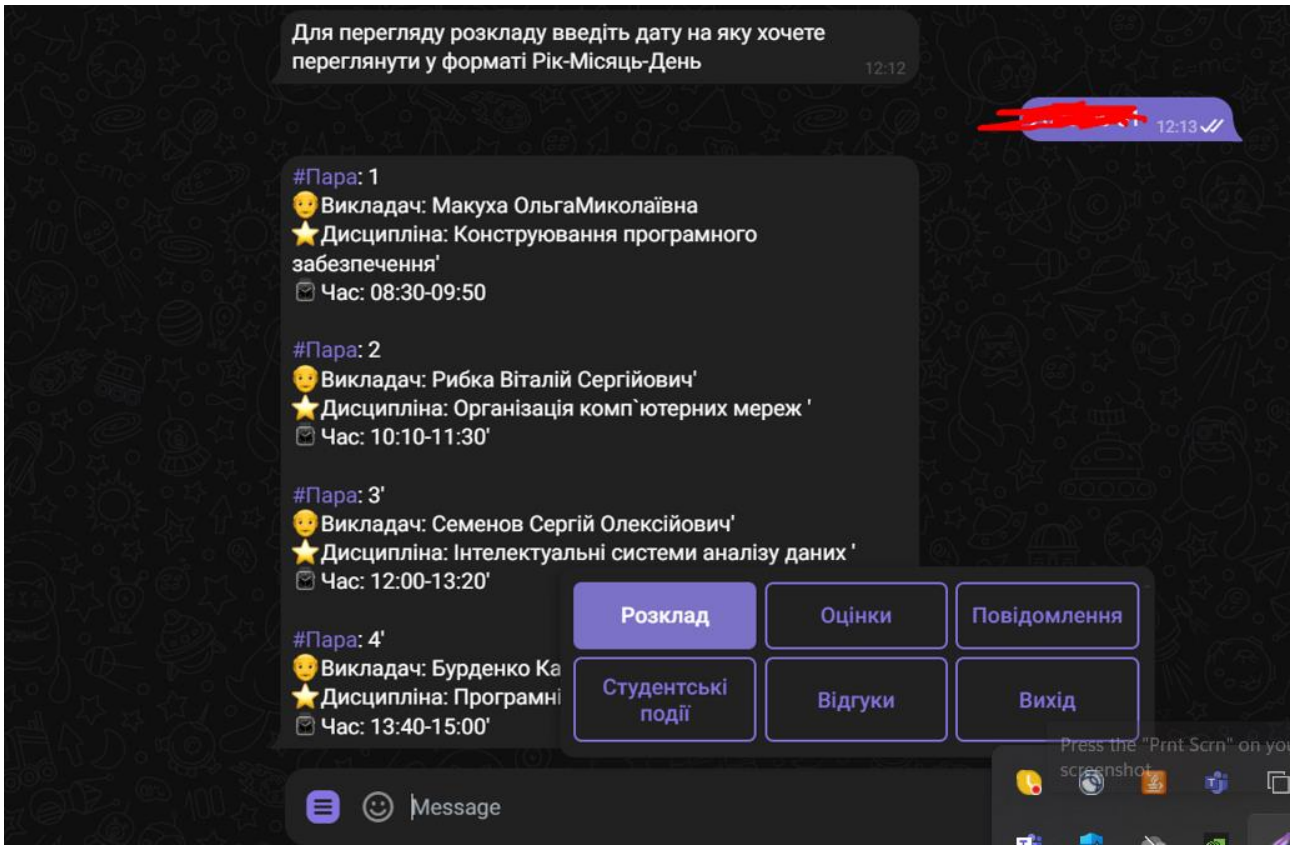

Рисунок 3.12 Перегляд розкладу в чат-боті

Інший функціонал аналогічно, так як розклад доступний тільки для користувачів, які здійснили реєстрацію в чат-боті та авторизувалися у систему. Інакше для них будуть доступні тільки ті пункти меню, які доступні для

незареєстрованих користувачів, такі, як перегляд публічної інформації, перегляд контактних даних та кнопки реєстрації та авторизації.

3.5 Висновки до розділу

У цьому розділі було розглянуто ключові аспекти розробки моделі та алгоритмів інтеграції чат-ботів з університетськими базами даних та сервісами. Було розроблено архітектуру БД та створено таблиці для зберігання сутностей університету. Окрім цього розроблено проміжне програмне забезпечення (middleware) за допомогою якого ми доступаємося до бази даних з чатботу за допомогою API методів.

Для забезпечення безпеки інтеграції чат-боту з університетською БД спроектовано та розроблено алгоритм авторизації за допомогою якого користувачі чат-боту можуть авторизуватися в чатбот по двохфакторній авторизації та на основі ролей бачити відповідний для них функціонал та пункти меню

Описана багаторівнева архітектура включає клієнтський рівень, шар інтеграції, бекенд та базу даних. Такий підхід забезпечує масштабованість, гнучкість, безпеку та зручність у використанні.

Запропоновані методи забезпечення безпеки, включаючи шифрування даних, аутентифікацію та авторизацію користувачів, відповідають сучасним стандартам захисту інформації.

Виконана розробка та впровадження чат-бота, який здатний виконувати основні функції, зокрема надавати доступ до розкладу, статусів заявок, реєстрації на курси та інших адміністративних послуг.

Розроблена модель інтеграції чат-ботів з університетськими базами даних і сервісами є ефективним рішенням для автоматизації навчальних і адміністративних процесів. Вона забезпечує безпечну та зручну взаємодію користувачів із системою, адаптуючись до потреб сучасного університету.

ВИСНОВКИ

Результатом виконання магістерської роботи є розроблена архітектура, модель та алгоритми інтеграції чат-ботів з університетськими БД. Також розроблено базу даних університету, чат-бот та проміжне ПЗ (Middleware), яке призначена для взаємодії чат-боту із БД. Ці програмні компоненти було створено шляхом проведення детального аналізу предметної області, розробки специфікації до програмного продукту та чіткої постановки завдання.

Даний чат-бот чудовий варіант використання для студентів, викладачів, абітурієнтів та адміністраторів університету, оскільки він містить декілька основних функцій, які роблять його особливим. А саме:

- Можливість перегляду та створення розкладу
- Можливість виставлення оцінок
- Можливість перегляду оцінок
- Можливість залишення та перегляд відгуків
- Отримання нотифікацій по зміні навчального процесу
- Отримання інформації про новини університету
- Отримання інформації про вступну кампанію

Крім цього в межах цієї роботи було спроектовано розподілену архітектуру розробки програмного забезпечення, яка складається із 3 шарів: БД, інтеграційної шини та чат-боту, які взаємодіють між собою через REST API запити. На даний момент чат-бот працює тільки в месенджері Telegram, проте така архітектура дозволяє нам з легкістю додавати нові чат боти для прикладу Viber чи Faccebook. Крім цього спроектована архітектура є гнучкою до вибору технологій, наприклад наш Telegram чат-бот створений із використанням технології .NET, .NET, проте Viber можна для прикладу зробити на Python, а Facebook на JavaScript.

Результатом виконання цієї роботи є чат-бот університету, який вміє взаємодіяти із БД через інтеграційну шину.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is a Chatbot?. Oracle. Retrieved 2023-02-15.
2. Jurafsky, D., Martin, J.H. (2023). Speech and Language Processing. Pearson.
3. Goodfellow, I., Bengio, Y., Courville, A. (2016). Deep Learning. MIT Press.
4. Introduction to Natural Language Processing. Stanford NLP Group. Retrieved 2023-01-20.
5. Vaswani, A. et al. (2017). Attention Is All You Need. NeurIPS.
6. Google Dialogflow Documentation. Google Cloud. Retrieved 2024-11-15. Mark Richards Fundamentals of Software Architecture: An Engineering Approach, 2016. – 298 с.
7. Adrian Ostrowski Software Architecture with C++: Design modern systems using effective architecture concepts, design patterns, and techniques with C#, 2018. – 405 с.
8. Brown, T.B., et al. (2020). Language Models are Few-Shot Learners. arXiv:2005.14165.
9. Mikolov, T. et al. (2013). Efficient Estimation of Word Representations in Vector Space. arXiv:1301.3781.
10. Pennington, J., Socher, R., Manning, C.D. (2014). GloVe: Global Vectors for Word Representation. EMNLP.
11. Applications of NLP in Higher Education. AI Magazine, 2023.
12. Turing, A. (1950). "Computing Machinery and Intelligence". Mind, 59(236), pp. 433-460.
13. Microsoft Bot Framework Documentation. Microsoft. Retrieved 2024-10-25.
14. Chollet, F. (2018). Deep Learning with Python. Manning Publications.
15. Hochreiter, S., Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), pp. 1735-1780.
16. Sutskever, I., Vinyals, O., Le, Q.V. (2014). Sequence to Sequence Learning with Neural Networks. NeurIPS.

17. API Design Guidelines. OpenAPI Initiative. Retrieved 2024-09-10.
18. Security in Machine Learning Systems. IBM Research. Retrieved 2024-05-18.
19. Dierks, T., Rescorla, E. (2008). The Transport Layer Security (TLS) Protocol. RFC 5246.
20. OAuth 2.0 Authorization Framework. IETF. Retrieved 2024-03-05.
21. Jensen, T. et al. (2021). Building Scalable Chatbots. O'Reilly Media.
22. GDPR Compliance for Chatbots. European Commission. Retrieved 2024-06-12.
23. AWS Lambda for Serverless Chatbots. Amazon Web Services. Retrieved 2024-07-08.
24. Serban, I.V. et al. (2016). Building End-To-End Dialogue Systems Using Generative Hierarchical Neural Network Models. AAAI.
25. Ruder, S. (2019). "Transfer Learning in NLP". arXiv:1903.11260.
26. Node.js for Chatbot Development. Node.js Foundation. Retrieved 2024-08-01.
27. Data Encryption Standards for University Systems. EDUCAUSE. Retrieved 2024-04-20.
28. Pavlyshenko, B. (2020). AI-Powered Educational Systems. Springer.
29. Mithun Pattankar Mastering ASP.NET Web API: Build powerful HTTP services and make the most of the ASP.NET Core Web API platform 1st Edition, 2017. – 332 c.
30. REST API Best Practices. API Academy. Retrieved 2023-12-10.
31. Alpaydin, E. (2020). Introduction to Machine Learning. MIT Press.
32. Azure Cognitive Services Documentation. Microsoft. Retrieved 2024-09-05.
33. Secure Data Storage Solutions for Chatbots. Cloud Security Alliance. Retrieved 2023-11-22.
34. Bengio, Y., LeCun, Y., Hinton, G. (2015). Deep Learning. Nature, 521, pp. 436-444.
35. OpenAI API Documentation. OpenAI. Retrieved 2024-10-12.
36. Xu, B. et al. (2023). GPT Applications in Education Systems. AI Journal.
37. Google Cloud for Higher Education. Google Cloud. Retrieved 2024-06-30.

38. Trivedi, P. et al. (2019). Scalable Knowledge Graph Embeddings. arXiv:1906.02945.
39. Database Management Systems. Oracle University. Retrieved 2024-02-15.
40. Fayyad, U., Piatetsky-Shapiro, G., Smyth, P. (1996). Knowledge Discovery and Data Mining. AI Magazine, 17(3).
41. BERT, J. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805.
42. Integration Patterns for SaaS Systems. Salesforce. Retrieved 2024-05-25.
43. Ramesh, A. et al. (2021). Zero-Shot Learning in Chatbots. ICML.
44. Data Privacy in AI Systems. AI Ethics Journal, 2023.
45. Kingma, D.P., Ba, J. (2015). Adam: A Method for Stochastic Optimization. arXiv:1412.6980.
46. MongoDB for Scalable Chatbots. MongoDB University. Retrieved 2024-07-20.
47. Real-Time Chatbot Analytics. Google Firebase. Retrieved 2024-04-01.
48. Bishop, C.M. (2006). Pattern Recognition and Machine Learning. Springer.
49. TensorFlow Guide. TensorFlow.org. Retrieved 2024-09-14.
50. Apache Kafka for Messaging in Chatbot Systems. Apache Software Foundation. Retrieved 2024-02-20.

ДОДАТКИ

Додаток А

Програмні коди бази даних

```
USE [University]
GO
/****** Object:  StoredProcedure [dbo].[spCityCRUD]    Script Date: 14.12.2024
16:10:21 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE proc [dbo].[spCityCRUD]
  @CityID int = null out,
  @CRUDOperationID int,
  @CityName nvarchar (200),
  @CityCode nvarchar (200),
  @CityDesc nvarchar (max),
  @Login nvarchar (200)
as
begin
  declare @CitiesTemp table (CityID int)

  if @CRUDOperationID = 1
  begin
    insert into Cities (CityName, CityCode, CityDesc)
    output inserted.CityID into @CitiesTemp (CityID)
    select @CityName, @CityCode, @CityDesc

    select @CityID = CityID from @CitiesTemp
  end
  else if @CRUDOperationID = 2
  begin
    select * from Cities with (nolock)
    where (CityCode = @CityCode or @CityCode is null)
      and CityName like concat(N'%', @CityName, N'%')
      and CityDesc like concat(N'%', @CityDesc, N'%')
  end
  else if @CRUDOperationID = 3
  begin
    update Cities
    set
      CityCode = @CityCode,
      CityName = @CityName,
      CityDesc = @CityDesc
    where CityID = @CityID
  end
  else if @CRUDOperationID = 4
  begin
    delete from Cities
    where CityID = @CityID
  end
  return 0
end
/*
  creation date: 2024-12-02 09:47 by Vasyl Andrusiak -- ANDRUS-123
  modification date:

  declare @CityID int
```

```
  exec spCityCRUD
    @CityID = @CityID out,
    @CRUDOperationID = 1,
    @CityName = N'Івано-Франківськ',
    @CityCode = N'IF',
    @CityDesc = N'м. Івано-Франківськ, Іавно-Франківька область, Україна',
    @Login = N'ChatbotAdmin'

  select @CityID as CityID


  select getdate()
*/


GO
/****** Object:   StoredProcedure [dbo].[spControlFormCRUD]       Script Date:
14.12.2024 16:10:21 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
create proc [dbo].[spControlFormCRUD]
  @ControlFormID int = null out,
  @CRUDOperationID int,
  @ControlFormName nvarchar (200),
  @ControlFormDesc nvarchar (max),
  @Login nvarchar (200)
as
begin
  declare @CitiesTemp table (ControlFormID int)

  if @CRUDOperationID = 1
  begin
    insert into ControlForms (ControlFormName, ControlFormDesc)
    output inserted.ControlFormID into @CitiesTemp (ControlFormID)
    select @ControlFormName, @ControlFormDesc

    select @ControlFormID = ControlFormID from @CitiesTemp
  end
  else if @CRUDOperationID = 2
  begin
    select * from ControlForms with (nolock)
    where ControlFormName like concat(N'%', @ControlFormName, N'%')
      and ControlFormDesc like concat(N'%', @ControlFormDesc, N'%')
  end
  else if @CRUDOperationID = 3
  begin
    update ControlForms
    set
      ControlFormName = @ControlFormName,
      ControlFormDesc = @ControlFormDesc
    where ControlFormID = @ControlFormID
  end
  else if @CRUDOperationID = 4
  begin
    delete from ControlForms
    where ControlFormID = @ControlFormID
  end
  return 0
end
/*
  creation date: 2024-12-02 09:47 by Vasyl Andrusiak -- ANDRUS-123
```

```
  modification date:

  declare @ControlFormID int
  exec spControlFormCRUD
    @ControlFormID = @ControlFormID out,
    @CRUDOperationID = 1,
    @ControlFormName = N'Івано-Франківськ',
    @ControlFormCode = N'IF',
    @ControlFormDesc = N'м. Івано-Франківськ, Іавно-Франківька область, Україна',
    @Login = N'ChatbotAdmin'

  select @ControlFormID as ControlFormID


  select getdate()
*/


GO
/****** Object:  StoredProcedure [dbo].[spCourseCRUD]    Script Date: 14.12.2024
16:10:21 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
create proc [dbo].[spCourseCRUD]
  @CourseID int = null out,
  @CRUDOperationID int,
  @CourseName nvarchar (200),
  @CourseDesc nvarchar (max),
  @Login nvarchar (200)
as
begin
  declare @CitiesTemp table (CourseID int)

  if @CRUDOperationID = 1
  begin
    insert into Courses (CourseName, CourseDesc)
    output inserted.CourseID into @CitiesTemp (CourseID)
    select @CourseName, @CourseDesc

    select @CourseID = CourseID from @CitiesTemp
  end
  else if @CRUDOperationID = 2
  begin
    select * from Courses with (nolock)
    where CourseName like concat(N'%', @CourseName, N'%')
      and CourseDesc like concat(N'%', @CourseDesc, N'%')
  end
  else if @CRUDOperationID = 3
  begin
    update Courses
    set
      CourseName = @CourseName,
      CourseDesc = @CourseDesc
    where CourseID = @CourseID
  end
  else if @CRUDOperationID = 4
  begin
    delete from Courses
    where CourseID = @CourseID
  end
  return 0
```

```
end
/*
  creation date: 2024-12-02 09:47 by Vasyl Andrusiak -- ANDRUS-123
  modification date:

  declare @CourseID int
  exec spCourseCRUD
    @CourseID = @CourseID out,
    @CRUDOperationID = 1,
    @CourseName = N'Івано-Франківськ',
    @CourseCode = N'IF',
    @CourseDesc = N'м. Івано-Франківськ, Іавно-Франківька область, Україна',
    @Login = N'ChatbotAdmin'

  select @CourseID as CourseID


  select getdate()
*/



GO
/****** Object:  StoredProcedure [dbo].[spDegreeCRUD]    Script Date: 14.12.2024
16:10:21 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
create proc [dbo].[spDegreeCRUD]
  @DegreeID int = null out,
  @CRUDOperationID int,
  @DegreeName nvarchar (200),
  @DegreeDesc nvarchar (max),
  @Login nvarchar (200)
as
begin
  declare @CitiesTemp table (DegreeID int)

  if @CRUDOperationID = 1
  begin
    insert into Degree (DegreeName, DegreeDesc)
    output inserted.DegreeID into @CitiesTemp (DegreeID)
    select @DegreeName, @DegreeDesc

    select @DegreeID = DegreeID from @CitiesTemp
  end
  else if @CRUDOperationID = 2
  begin
    select * from Degree with (nolock)
    where DegreeName like concat(N'%', @DegreeName, N'%')
      and DegreeDesc like concat(N'%', @DegreeDesc, N'%')
  end
  else if @CRUDOperationID = 3
  begin
    update Degree
    set
      DegreeName = @DegreeName,
      DegreeDesc = @DegreeDesc
    where DegreeID = @DegreeID
  end
  else if @CRUDOperationID = 4
  begin
```

```
    delete from Degree
    where DegreeID = @DegreeID
  end
  return 0
end
/*
  creation date: 2024-12-02 09:47 by Vasyl Andrusiak -- ANDRUS-123
  modification date:

  declare @DegreeID int
  exec spDegreeCRUD
    @DegreeID = @DegreeID out,
    @CRUDOperationID = 1,
    @DegreeName = N'Івано-Франківськ',
    @DegreeCode = N'IF',
    @DegreeDesc = N'м. Івано-Франківськ, Іавно-Франківька область, Україна',
    @Login = N'ChatbotAdmin'

  select @DegreeID as DegreeID


  select getdate()
*/


GO
/****** Object:  StoredProcedure [dbo].[spDepartmentCRUD]  Script Date:
14.12.2024 16:10:21 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE proc [dbo].[spDepartmentCRUD]
  @DepartmentID int = null out,
  @CRUDOperationID int,
  @DepartmentName nvarchar (200),
  @DepartmentDesc nvarchar (max),
  @DepartmentHTMLDesc nvarchar (max),
  @DepartmentPhone1 nvarchar (200),
  @DepartmentPhone2 nvarchar (200),
  @DepartmentPhone3 nvarchar (200),
  @DepartmentPhone4 nvarchar (200),
  @DepartmentPhone5 nvarchar (200),
  @DepartmentAddress nvarchar (max),
  @DepartmentEmail nvarchar (max),
  @DepartmentLogo nvarchar (max),
  @DepartmentSocialNetwork1 nvarchar (max),
  @DepartmentSocialNetwork2 nvarchar (max),
  @DepartmentSocialNetwork3 nvarchar (max),
  @DepartmentSocialNetwork4 nvarchar (max),
  @DepartmentSocialNetwork5 nvarchar (max),
  @DepartmentFacultyRef int,
  @Login nvarchar (200)
as
begin
  declare @DepartmentTemp table (DepartmentID int)

  if @DepartmentFacultyRef is not null and not exists (
    select * from Faculties with (nolock)
    where FacultyID = @DepartmentFacultyRef
  )
  begin
    select N'Не вдалося додати кафедру, вказаний не існуючий інститут'
```

```
    return 0
  end

  if @CRUDOperationID = 1
  begin
    insert Departments
      (DepartmentName,             DepartmentDesc,               DepartmentLogo,
DepartmentHTMLDesc,
      DepartmentPhone1,           DepartmentPhone2,           DepartmentPhone3,
DepartmentPhone4,
      DepartmentPhone5,            DepartmentAddress,           DepartmentEmail,
DepartmentSocialNetwork1,
      DepartmentSocialNetwork2,                          DepartmentSocialNetwork3,
DepartmentSocialNetwork4, DepartmentSocialNetwork5,
      DepartmentFacultyRef)
    output inserted.DepartmentID into @DepartmentTemp (DepartmentID)
    values
      (@DepartmentName,            @DepartmentDesc,             @DepartmentLogo,
@DepartmentHTMLDesc,
      @DepartmentPhone1,          @DepartmentPhone2,          @DepartmentPhone3,
@DepartmentPhone4,
      @DepartmentPhone5,           @DepartmentAddress,          @DepartmentEmail,
@DepartmentSocialNetwork1,
      @DepartmentSocialNetwork2,                         @DepartmentSocialNetwork3,
@DepartmentSocialNetwork4, @DepartmentSocialNetwork5,
      @DepartmentFacultyRef)

    insert into Departments (DepartmentName, DepartmentDesc)
    select @DepartmentName, @DepartmentDesc

    select @DepartmentID = DepartmentID from @DepartmentTemp
  end
  else if @CRUDOperationID = 2
  begin
    select * from Departments with (nolock)
    where DepartmentName like concat(N'%', @DepartmentName, N'%')
      and DepartmentDesc like concat(N'%', @DepartmentDesc, N'%')
      and (DepartmentID = @DepartmentID or @DepartmentID is null)
  end
  else if @CRUDOperationID = 3
  begin
    update Departments set
      DepartmentName           = @DepartmentName,
      DepartmentDesc           = @DepartmentDesc,
      DepartmentLogo           = @DepartmentLogo,
      DepartmentHTMLDesc       = @DepartmentHTMLDesc,
      DepartmentPhone1         = @DepartmentPhone1,
      DepartmentPhone2         = @DepartmentPhone2,
      DepartmentPhone3         = @DepartmentPhone3,
      DepartmentPhone4         = @DepartmentPhone4,
      DepartmentPhone5         = @DepartmentPhone5,
      DepartmentAddress        = @DepartmentAddress,
      DepartmentEmail          = @DepartmentEmail,
      DepartmentSocialNetwork1 = @DepartmentSocialNetwork1,
      DepartmentSocialNetwork2 = @DepartmentSocialNetwork2,
      DepartmentSocialNetwork3 = @DepartmentSocialNetwork3,
      DepartmentSocialNetwork4 = @DepartmentSocialNetwork4,
      DepartmentSocialNetwork5 = @DepartmentSocialNetwork5,
      DepartmentFacultyRef     = @DepartmentFacultyRef
     where DepartmentID = @DepartmentID

  end
  else if @CRUDOperationID = 4
```

```
  begin
    delete from Departments
    where DepartmentID = @DepartmentID
  end

  return 0
end
/*
  creation date: 2024-12-02 09:47 by Vasyl Andrusiak -- ANDRUS-123
  modification date:

  select getdate()
*/


GO
/****** Object:  StoredProcedure [dbo].[spDiplomaThesisCRUD]    Script Date:
14.12.2024 16:10:21 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
create proc [dbo].[spDiplomaThesisCRUD]
  @DiplomaThesisID int = null out,
  @CRUDOperationID int,
  @DiplomaThesisTheme nvarchar (200) = null,
  @DiplomaThesisDesc nvarchar (max) = null,
  @DiplomaThesisScore int = null,
  @DiplomaThesisDate datetime = null,
  @StudentRef int = null,
  @EducationalSubjectRef int = null,
  @EmployeeRef int = null,
  @Login nvarchar (200)
as
begin
  declare @DiplomaThesisesTemp table (DiplomaThesisID int)

  if @StudentRef is not null and not exists (
    select * from Faculties with (nolock)
    where FacultyID = @StudentRef
  )
  begin
    select N'Не вдалося додати дипломну роботу, вказаний не існуючий студент'
    return 0
  end

  if @EducationalSubjectRef is not null and not exists (
    select * from Faculties with (nolock)
    where FacultyID = @EducationalSubjectRef
  )
  begin
    select N'Не вдалося додати дипломну роботу, вказаний не існуючу навчальну
дисципліну'
    return 0
  end

  if @EmployeeRef is not null and not exists (
    select * from Employees with (nolock)
    where EmployeeID = @EmployeeRef
  )
  begin
    select N'Не вдалося додати дипломну роботу, вказаний не існуючий викладач'
    return 0
```

```
  end


  if @CRUDOperationID = 1
  begin
    insert DiplomaThesis
      (DiplomaThesisTheme,   DiplomaThesisDesc,              DiplomaThesisScore,
DiplomaThesisDate,
      StudentRef,          EducationalSubjectRef, EmployeeRef)
    output inserted.DiplomaThesisID into @DiplomaThesisesTemp (DiplomaThesisID)
    select
      @DiplomaThesisTheme,   @DiplomaThesisDesc,            @DiplomaThesisScore,
@DiplomaThesisDate,
      @StudentRef,          @EducationalSubjectRef, @EmployeeRef

    select @DiplomaThesisID = DiplomaThesisID from @DiplomaThesisesTemp
  end
  else if @CRUDOperationID = 2
  begin
    select * from DiplomaThesis with (nolock)
    where  (DiplomaThesisTheme  like  concat(N'%',  DiplomaThesisTheme,  N'%')  or
@DiplomaThesisDesc is null)
      and  (DiplomaThesisDesc  like  concat(N'%',  DiplomaThesisDesc,  N'%')  or
@DiplomaThesisDesc is null)
      and  (DiplomaThesisScore  =  @DiplomaThesisScore  or  @DiplomaThesisScore  is
null)
      and (DiplomaThesisDate = @DiplomaThesisDate or @DiplomaThesisDate is null)
      and (StudentRef = @StudentRef or @StudentRef is null)
      and      (EducationalSubjectRef      =      @EducationalSubjectRef      or
@EducationalSubjectRef is null)
      and (EmployeeRef = @EmployeeRef or @EmployeeRef is null)
  end
  else if @CRUDOperationID = 3
  begin
    update DiplomaThesis set
      DiplomaThesisTheme    = @DiplomaThesisTheme,
      DiplomaThesisDesc     = @DiplomaThesisDesc,
      DiplomaThesisScore    = @DiplomaThesisScore,
      DiplomaThesisDate     = @DiplomaThesisDate,
      StudentRef            = @StudentRef,
      EducationalSubjectRef = @EducationalSubjectRef,
      EmployeeRef           = @EmployeeRef
     where DiplomaThesisID = @DiplomaThesisID

  end
  else if @CRUDOperationID = 4
  begin
    delete from DiplomaThesis
    where DiplomaThesisID = @DiplomaThesisID
  end
  return 0
end
/*
  creation date: 2024-12-02 09:47 by Vasyl Andrusiak -- ANDRUS-123
  modification date:


  select getdate()
*/
```

```
GO
/****** Object:  StoredProcedure [dbo].[spEducationalSubjectCRUD]    Script Date:
14.12.2024 16:10:21 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
create proc [dbo].[spEducationalSubjectCRUD]
  @EducationalSubjectID int = null out,
  @CRUDOperationID int,
  @EducationalSubjectName nvarchar (200) = null,
  @EducationalSubjectDesc nvarchar (max) = null,
  @EducationalSubjectCountHours int = null,
  @EducationalSubjectSpecialityRef int = null,
  @Login nvarchar (200)
as
begin
  declare @EducationalSubjectesTemp table (EducationalSubjectID int)

  if @EducationalSubjectSpecialityRef is not null and not exists (
    select * from Faculties with (nolock)
    where FacultyID = @EducationalSubjectSpecialityRef
  )
  begin
    select N'Не вдалося додати навчальну дисципліну. Вказаний не існуючий студент'
    return 0
  end

  if @CRUDOperationID = 1
  begin
    insert EducationalSubjects
      (EducationalSubjectName,                              EducationalSubjectDesc,
EducationalSubjectCountHours, EducationalSubjectSpecialityRef)
    values
      (@EducationalSubjectName,                          @EducationalSubjectDesc,
@EducationalSubjectCountHours, @EducationalSubjectSpecialityRef)

    select      @EducationalSubjectID      =      EducationalSubjectID      from
@EducationalSubjectesTemp
  end
  else if @CRUDOperationID = 2
  begin
    select * from EducationalSubjects with (nolock)
    where (EducationalSubjectID = @EducationalSubjectID or @EducationalSubjectID
is null)
      and (EducationalSubjectID = @EducationalSubjectID or @EducationalSubjectID
is null)
      and  (EducationalSubjectName  like  concat(N'%',  @EducationalSubjectDesc,
N'%') or @EducationalSubjectDesc is null)
      and  (EducationalSubjectDesc  like  concat(N'%',  @EducationalSubjectDesc,
N'%') or @EducationalSubjectDesc is null)
      and   (EducationalSubjectCountHours   =   @EducationalSubjectCountHours   or
@EducationalSubjectCountHours is null)
      and (EducationalSubjectSpecialityRef = @EducationalSubjectSpecialityRef or
@EducationalSubjectSpecialityRef is null)



  end
  else if @CRUDOperationID = 3
  begin
    update EducationalSubjects set
```

```
      EducationalSubjectName          = @EducationalSubjectName,
      EducationalSubjectDesc          = @EducationalSubjectDesc,
      EducationalSubjectCountHours    = @EducationalSubjectCountHours,
      EducationalSubjectSpecialityRef = @EducationalSubjectSpecialityRef
     where EducationalSubjectID = @EducationalSubjectID

  end
  else if @CRUDOperationID = 4
  begin
    delete from EducationalSubjects
    where EducationalSubjectID = @EducationalSubjectID
  end
  return 0
end
/*
  creation date: 2024-12-02 09:47 by Vasyl Andrusiak -- ANDRUS-123
  modification date:


  select getdate()
*/




GO
/****** Object:  StoredProcedure [dbo].[spTypeOfClassCRUD]  Script Date:
14.12.2024 16:10:21 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
create proc [dbo].[spTypeOfClassCRUD]
  @TypeOfClassID int = null out,
  @CRUDOperationID int,
  @TypeOfClassName nvarchar (200),
  @TypeOfClassDesc nvarchar (max),
  @Login nvarchar (200)
as
begin
  declare @CitiesTemp table (TypeOfClassID int)

  if @CRUDOperationID = 1
  begin
    insert into TypeOfClasses (TypeOfClassName, TypeOfClassDesc)
    output inserted.TypeOfClassID into @CitiesTemp (TypeOfClassID)
    select @TypeOfClassName, @TypeOfClassDesc

    select @TypeOfClassID = TypeOfClassID from @CitiesTemp
  end
  else if @CRUDOperationID = 2
  begin
    select * from TypeOfClasses with (nolock)
    where TypeOfClassName like concat(N'%', @TypeOfClassName, N'%')
      and TypeOfClassDesc like concat(N'%', @TypeOfClassDesc, N'%')
  end
  else if @CRUDOperationID = 3
  begin
    update TypeOfClasses
    set
      TypeOfClassName = @TypeOfClassName,
      TypeOfClassDesc = @TypeOfClassDesc
```

```
    where TypeOfClassID = @TypeOfClassID
  end
  else if @CRUDOperationID = 4
  begin
    delete from TypeOfClasses
    where TypeOfClassID = @TypeOfClassID
  end
  return 0
end
/*
  creation date: 2024-12-02 09:47 by Vasyl Andrusiak -- ANDRUS-123
  modification date:

  declare @TypeOfClassID int
  exec spTypeOfClassCRUD
    @TypeOfClassID = @TypeOfClassID out,
    @CRUDOperationID = 1,
    @TypeOfClassName = N'Івано-Франківськ',
    @TypeOfClassCode = N'IF',
    @TypeOfClassDesc = N'м. Івано-Франківськ, Іавно-Франківька область, Україна',
    @Login = N'ChatbotAdmin'

  select @TypeOfClassID as TypeOfClassID


  select getdate()
*/



GO
ALTER proc spDepartmentCRUD
  @DepartmentID int = null out,
  @CRUDOperationID int,
  @DepartmentName nvarchar (200),
  @DepartmentDesc nvarchar (max),
  @DepartmentHTMLDesc nvarchar (max),
  @DepartmentPhone1 nvarchar (200),
  @DepartmentPhone2 nvarchar (200),
  @DepartmentPhone3 nvarchar (200),
  @DepartmentPhone4 nvarchar (200),
  @DepartmentPhone5 nvarchar (200),
  @DepartmentAddress nvarchar (max),
  @DepartmentEmail nvarchar (max),
  @DepartmentLogo nvarchar (max),
  @DepartmentSocialNetwork1 nvarchar (max),
  @DepartmentSocialNetwork2 nvarchar (max),
  @DepartmentSocialNetwork3 nvarchar (max),
  @DepartmentSocialNetwork4 nvarchar (max),
  @DepartmentSocialNetwork5 nvarchar (max),
  @DepartmentFacultyRef int,
  @Login nvarchar (200)
as
begin
  declare @DepartmentTemp table (DepartmentID int)

  if @DepartmentFacultyRef is not null and not exists (
    select * from Faculties with (nolock)
    where FacultyID = @DepartmentFacultyRef
  )
  begin
    select N'Не вдалося додати кафедру, вказаний не існуючий інститут'
    return 0
```

```
  end

  if @CRUDOperationID = 1
  begin
    insert Departments
      (DepartmentName,             DepartmentDesc,               DepartmentLogo,
DepartmentHTMLDesc,
      DepartmentPhone1,           DepartmentPhone2,           DepartmentPhone3,
DepartmentPhone4,
      DepartmentPhone5,            DepartmentAddress,          DepartmentEmail,
DepartmentSocialNetwork1,
      DepartmentSocialNetwork2,                        DepartmentSocialNetwork3,
DepartmentSocialNetwork4, DepartmentSocialNetwork5,
      DepartmentFacultyRef)
    output inserted.DepartmentID into @DepartmentTemp (DepartmentID)
    values
      (@DepartmentName,            @DepartmentDesc,             @DepartmentLogo,
@DepartmentHTMLDesc,
      @DepartmentPhone1,          @DepartmentPhone2,          @DepartmentPhone3,
@DepartmentPhone4,
      @DepartmentPhone5,           @DepartmentAddress,         @DepartmentEmail,
@DepartmentSocialNetwork1,
      @DepartmentSocialNetwork2,                       @DepartmentSocialNetwork3,
@DepartmentSocialNetwork4, @DepartmentSocialNetwork5,
      @DepartmentFacultyRef)

    insert into Departments (DepartmentName, DepartmentDesc)
    select @DepartmentName, @DepartmentDesc

    select @DepartmentID = DepartmentID from @DepartmentTemp
  end
  else if @CRUDOperationID = 2
  begin
    select * from Departments with (nolock)
    where DepartmentName like concat(N'%', @DepartmentName, N'%')
      and DepartmentDesc like concat(N'%', @DepartmentDesc, N'%')
      and (DepartmentID = @DepartmentID or @DepartmentID is null)
  end
  else if @CRUDOperationID = 3
  begin
    update Departments set
      DepartmentName           = @DepartmentName,
      DepartmentDesc           = @DepartmentDesc,
      DepartmentLogo           = @DepartmentLogo,
      DepartmentHTMLDesc       = @DepartmentHTMLDesc,
      DepartmentPhone1         = @DepartmentPhone1,
      DepartmentPhone2         = @DepartmentPhone2,
      DepartmentPhone3         = @DepartmentPhone3,
      DepartmentPhone4         = @DepartmentPhone4,
      DepartmentPhone5         = @DepartmentPhone5,
      DepartmentAddress        = @DepartmentAddress,
      DepartmentEmail          = @DepartmentEmail,
      DepartmentSocialNetwork1 = @DepartmentSocialNetwork1,
      DepartmentSocialNetwork2 = @DepartmentSocialNetwork2,
      DepartmentSocialNetwork3 = @DepartmentSocialNetwork3,
      DepartmentSocialNetwork4 = @DepartmentSocialNetwork4,
      DepartmentSocialNetwork5 = @DepartmentSocialNetwork5,
      DepartmentFacultyRef     = @DepartmentFacultyRef
     where DepartmentID = @DepartmentID

  end
  else if @CRUDOperationID = 4
  begin
```

```
    delete from Departments
    where DepartmentID = @DepartmentID
  end

  return 0
end
/*
  creation date: 2024-12-02 09:47 by Vasyl Andrusiak -- ANDRUS-123
  modification date:

  select getdate()
*/

create proc spEmployeeCRUD
  @EmployeeID int = null out,
  @CRUDOperationID int,
  @EmployeePhoto nvarchar (max) = null,
  @EmployeeName1 nvarchar (200) = null,
  @EmployeeName2 nvarchar (200) = null,
  @EmployeeName3 nvarchar (200) = null,
  @EmployeeBirthDate datetime = null,
  @EmployeePhoneNumber1 nvarchar (200) = null,
  @EmployeePhoneNumber2 nvarchar (200) = null,
  @EmployeeEmail nvarchar (max) = null,
  @EmployeeAddress nvarchar (max) = null,
  @EmployeeSalaryPerHour int = null,
  @EmployeeSalaryPerMonth int = null,
  @EmployeeAlmaMater nvarchar (max) = null,
  @EmployeeScientificInterest nvarchar (max) = null,
  @EmployeeDesc nvarchar (max) = null,
  @EmployeeDescXML xml = null,
  @EmployeeDescHTML nvarchar (max) = null,
  @EmployeeDegreeRef int = null,
  @EmployeeSpecialityRef int = null,
  @EmployeePositionRef int = null,
  @EmployeeGenderRef int = null,
  @EmployeeCityRef int = null,
  @EmployeeFacultyRef int = null,
  @EmployeeDepartmentRef int = null,
  @EmployeeHasChild bit = null,
  @EmployeeChildCount int = null,
  @Login nvarchar (200)
as
begin
  declare @EmployeeesTemp table (EmployeeID int)

  if @EmployeeDegreeRef is not null and not exists (
    select * from Degree with (nolock)
    where DegreeID = @EmployeeDegreeRef
  )
  begin
    select N'Не вдалося додати працівника університету. Вказаний не існуючий
науковий ступінь'
    return 0
  end

  if @EmployeeSpecialityRef is not null and not exists (
    select * from Specialities with (nolock)
    where SpecialityID = @EmployeeSpecialityRef
  )
  begin
    select N'Не вдалося додати працівника університету. Вказано не існуючу
спеціальність'
```

```
    return 0
  end

  if @EmployeePositionRef is not null and not exists (
    select * from Positions with (nolock)
    where PositionID = @EmployeePositionRef
  )
  begin
    select N'Не вдалося додати працівника університету. Вказано не існуючу посаду
користсувача'
    return 0
  end

  if @EmployeeGenderRef is not null and not exists (
    select * from Gender with (nolock)
    where GenderID = @EmployeeGenderRef
  )
  begin
    select N'Не вдалося додати працівника університету. Вказано не існуючу стать'
    return 0
  end

  if @EmployeeCityRef is not null and not exists (
    select * from Cities with (nolock)
    where CityID = @EmployeeCityRef
  )
  begin
    select N'Не вдалося додати працівника університету. Вказано не існуюче місто'
    return 0
  end

  if @EmployeeFacultyRef is not null and not exists (
    select * from Faculties with (nolock)
    where FacultyID = @EmployeeFacultyRef
  )
  begin
    select N'Не вдалося додати працівника університету. Вказано не існуючий
інститут'
    return 0
  end

  if @EmployeeDepartmentRef is not null and not exists (
    select * from Departments with (nolock)
    where DepartmentID = @EmployeeDepartmentRef
  )
  begin
    select N'Не вдалося додати працівника університету. Вказано не існуючу
кафедру'
    return 0
  end

  if @CRUDOperationID = 1
  begin
    insert Employees
      (EmployeePhoto,                EmployeeName1,            EmployeeName2,
EmployeeName3,
      EmployeeBirthDate,           EmployeePhoneNumber1,  EmployeePhoneNumber2,
EmployeeEmail,
      EmployeeAddress,           EmployeeSalaryPerHour, EmployeeSalaryPerMonth,
EmployeeAlmaMater,
      EmployeeScientificInterest,  EmployeeDesc,               EmployeeDescXML,
EmployeeDescHTML,
```

```
      EmployeeDegreeRef,             EmployeeSpecialityRef, EmployeePositionRef,
EmployeeGenderRef,
      EmployeeCityRef,             EmployeeFacultyRef,    EmployeeDepartmentRef,
EmployeeHasChild,
      EmployeeChildCount)
    select
      @EmployeePhoto,                 @EmployeeName1,            @EmployeeName2,
@EmployeeName3,
      @EmployeeBirthDate,         @EmployeePhoneNumber1,  @EmployeePhoneNumber2,
@EmployeeEmail,
      @EmployeeAddress,          @EmployeeSalaryPerHour, @EmployeeSalaryPerMonth,
@EmployeeAlmaMater,
      @EmployeeScientificInterest, @EmployeeDesc,             @EmployeeDescXML,
@EmployeeDescHTML,
      @EmployeeDegreeRef,          @EmployeeSpecialityRef, @EmployeePositionRef,
@EmployeeGenderRef,
      @EmployeeCityRef,           @EmployeeFacultyRef,    @EmployeeDepartmentRef,
@EmployeeHasChild,
      @EmployeeChildCount

    select @EmployeeID = EmployeeID from @EmployeeesTemp
  end
  else if @CRUDOperationID = 2
  begin
    select * from Employees with (nolock)
    where (EmployeeID = @EmployeeID or @EmployeeID is null)
      and (EmployeeBirthDate = @EmployeeBirthDate or @EmployeeBirthDate is null)
      and (EmployeePhoneNumber1 = @EmployeePhoneNumber1 or @EmployeePhoneNumber1
is null)
      and (EmployeePhoneNumber2 = @EmployeePhoneNumber2 or @EmployeePhoneNumber2
is null)
      and (EmployeeEmail = @EmployeeEmail or @EmployeeEmail is null)
      and (EmployeeName1 like concat(N'%', @EmployeeName1, N'%') or @EmployeeName1
is null)
      and (EmployeeName2 like concat(N'%', @EmployeeName2, N'%') or @EmployeeName2
is null)
      and (EmployeeName3 like concat(N'%', @EmployeeName3, N'%') or @EmployeeName3
is null)
      and (EmployeeDegreeRef = @EmployeeDegreeRef or @EmployeeDegreeRef is null)
      and      (EmployeeSpecialityRef      =      @EmployeeSpecialityRef      or
@EmployeeSpecialityRef is null)
      and (EmployeePositionRef = @EmployeePositionRef or @EmployeePositionRef is
null)
      and (EmployeeGenderRef = @EmployeeGenderRef or @EmployeeGenderRef is null)
      and  (EmployeeFacultyRef  =  @EmployeeFacultyRef  or  @EmployeeFacultyRef  is
null)
      and      (EmployeeDepartmentRef      =      @EmployeeDepartmentRef      or
@EmployeeDepartmentRef is null)
      and (EmployeeCityRef = @EmployeeCityRef or @EmployeeCityRef is null)
      and (EmployeeHasChild = @EmployeeHasChild or @EmployeeHasChild is null)
      and  (EmployeeChildCount  =  @EmployeeChildCount  or  @EmployeeChildCount  is
null)


  end
  else if @CRUDOperationID = 3
  begin
    update Employees set
      EmployeePhoto              = @EmployeePhoto,
      EmployeeName1              = @EmployeeName1,
      EmployeeName2              = @EmployeeName2,
      EmployeeName3              = @EmployeeName3,
      EmployeeBirthDate          = @EmployeeBirthDate,
```

```
      EmployeePhoneNumber1       = @EmployeePhoneNumber1,
      EmployeePhoneNumber2       = @EmployeePhoneNumber2,
      EmployeeEmail              = @EmployeeEmail,
      EmployeeAddress            = @EmployeeAddress,
      EmployeeSalaryPerHour      = @EmployeeSalaryPerHour,
      EmployeeSalaryPerMonth     = @EmployeeSalaryPerMonth,
      EmployeeAlmaMater          = @EmployeeAlmaMater,
      EmployeeScientificInterest = @EmployeeScientificInterest,
      EmployeeDesc               = @EmployeeDesc,
      EmployeeDescXML            = @EmployeeDescXML,
      EmployeeDescHTML           = @EmployeeDescHTML,
      EmployeeDegreeRef          = @EmployeeDegreeRef,
      EmployeeSpecialityRef      = @EmployeeSpecialityRef,
      EmployeePositionRef        = @EmployeePositionRef,
      EmployeeGenderRef          = @EmployeeGenderRef,
      EmployeeCityRef            = @EmployeeCityRef,
      EmployeeFacultyRef         = @EmployeeFacultyRef,
      EmployeeDepartmentRef      = @EmployeeDepartmentRef,
      EmployeeHasChild           = @EmployeeHasChild,
      EmployeeChildCount         = @EmployeeChildCount
     where EmployeeID = @EmployeeID


  end
  else if @CRUDOperationID = 4
  begin
    delete from Employees
    where EmployeeID = @EmployeeID
  end
  return 0
end
/*
  creation date: 2024-12-02 09:47 by Vasyl Andrusiak -- ANDRUS-123
  modification date:


  select getdate()
*/




create proc spStudentCRUD
  @StudentID int = null out,
  @CRUDOperationID int,
  @StudentName1   nvarchar (200) = null,
  @StudentName2   nvarchar (200) = null,
  @StudentName3   nvarchar (200) = null,
  @StudentPhoto nvarchar (max) = null,
  @StudentBirthDdate    datetime = null,
  @StudentPhone1  nvarchar (200) = null,
  @StudentPhone2  nvarchar (200) = null,
  @StudentEmail   nvarchar (200) = null,
  @StudentAddress nvarchar (max) = null,
  @StudentHasScholarship      bit = null,
  @StudentHasChild      bit = null,
  @StudentGroupRef      int = null,
  @StudentFacultyRef    int = null,
  @StudentCityRef int = null,
  @StudentGenderRef     int = null,
  @StudentSpecialityRef int = null,
  @StudentScholarship   money = null,
  @Login nvarchar (200)
```

```
as
begin
  declare @StudentTemp table (StudentID int)


  if @StudentGroupRef is not null and not exists (
    select * from Groups with (nolock)
    where GroupID = @StudentGroupRef
  )
  begin
    select N'Не вдалося додати студента. Вказано не існуючу групу'
    return 0
  end

  if @StudentSpecialityRef is not null and not exists (
    select * from Specialities with (nolock)
    where SpecialityID = @StudentSpecialityRef
  )
  begin
    select N'Не вдалося додати студента. Вказано не існуючу спеціальність'
    return 0
  end

  if @StudentGenderRef is not null and not exists (
    select * from Gender with (nolock)
    where GenderID = @StudentGenderRef
  )
  begin
    select N'Не вдалося додати студента. Вказано не існуючу стать'
    return 0
  end

  if @StudentCityRef is not null and not exists (
    select * from Cities with (nolock)
    where CityID = @StudentCityRef
  )
  begin
    select N'Не вдалося додати студента. Вказано не існуюче місто'
    return 0
  end

  if @StudentFacultyRef is not null and not exists (
    select * from Faculties with (nolock)
    where FacultyID = @StudentFacultyRef
  )
  begin
    select N'Не вдалося додати студента. Вказано не існуючий інститут'
    return 0
  end


  if @CRUDOperationID = 1
  begin
    insert Students
      (StudentName1,       StudentName2,          StudentName3,     StudentPhoto,
      StudentBirthDdate,   StudentPhone1,         StudentPhone2,    StudentEmail,
      StudentAddress,                  StudentHasScholarship,   StudentHasChild,
StudentGroupRef,
      StudentFacultyRef,       StudentCityRef,                  StudentGenderRef,
StudentSpecialityRef,
      StudentScholarship)
    output inserted.StudentID into @StudentTemp (StudentID)
    values
```

```
      (@StudentName1,      @StudentName2,        @StudentName3,    @StudentPhoto,
      @StudentBirthDdate,       @StudentPhone1,                    @StudentPhone2,
@StudentEmail,
      @StudentAddress,                @StudentHasScholarship,   @StudentHasChild,
@StudentGroupRef,
      @StudentFacultyRef,      @StudentCityRef,                @StudentGenderRef,
@StudentSpecialityRef,
      @StudentScholarship)


    select @StudentID = StudentID from @StudentTemp
  end
  else if @CRUDOperationID = 2
  begin
    select * from Students with (nolock)
    where (StudentID = @StudentID or @StudentID is null)
      and (StudentEmail = @StudentEmail or @StudentEmail is null)
      and (StudentBirthDdate = @StudentBirthDdate or @StudentBirthDdate is null)
      and (StudentPhone1 = @StudentPhone1 or @StudentPhone1 is null)
      and (StudentPhone2 = @StudentPhone2 or @StudentPhone2 is null)
      and (StudentName1 like concat(N'%', @StudentName1, N'%') or @StudentName1
is null)
      and (StudentName2 like concat(N'%', @StudentName2, N'%') or @StudentName2
is null)
      and (StudentName3 like concat(N'%', @StudentName3, N'%') or @StudentName3
is null)
      and (StudentGroupRef = @StudentGroupRef or @StudentGroupRef is null)
      and (StudentFacultyRef = @StudentFacultyRef or @StudentFacultyRef is null)
      and (StudentCityRef = @StudentCityRef or @StudentCityRef is null)
      and (StudentGenderRef = @StudentGenderRef or @StudentGenderRef is null)
      and (StudentSpecialityRef = @StudentSpecialityRef or @StudentSpecialityRef
is null)
      and  (StudentScholarship  =  @StudentScholarship  or  @StudentScholarship  is
null)


  end
  else if @CRUDOperationID = 3
  begin
    update Students set
      StudentName1          = @StudentName1,
      StudentName2          = @StudentName2,
      StudentName3          = @StudentName3,
      StudentPhoto          = @StudentPhoto,
      StudentBirthDdate     = @StudentBirthDdate,
      StudentPhone1         = @StudentPhone1,
      StudentPhone2         = @StudentPhone2,
      StudentEmail          = @StudentEmail,
      StudentAddress        = @StudentAddress,
      StudentHasScholarship = @StudentHasScholarship,
      StudentHasChild       = @StudentHasChild,
      StudentGroupRef       = @StudentGroupRef,
      StudentFacultyRef     = @StudentFacultyRef,
      StudentCityRef        = @StudentCityRef,
      StudentGenderRef      = @StudentGenderRef,
      StudentSpecialityRef  = @StudentSpecialityRef,
      StudentScholarship    = @StudentScholarship
     where StudentID = @StudentID


  end
  else if @CRUDOperationID = 4
  begin
```

```
    delete from Students
    where StudentID = @StudentID
  end
  return 0
end
/*
  creation date: 2024-12-02 09:47 by Vasyl Andrusiak -- ANDRUS-123
  modification date:


  select getdate()
*/
```

Додаток В

Програмні коди чат-боту

```
using System;
using Telegram.Bot;
using Telegram.Bot.Exceptions;
using Telegram.Bot.Polling;
using Telegram.Bot.Types;
using Telegram.Bot.Types.Enums;
using System.Threading;
using System.Threading.Tasks;
using Telegram.Bot.Types.ReplyMarkups;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Extensions.Configuration;
using Newtonsoft.Json;
using System.IO;

namespace IFNTUNGHelper
{
    class Program
    {
        public static string TelegramBotToken;
        public static string UserLogin;
        public static string UserPassword;
        public static string InstallationGUID;
        public static string GrandType;
        public static string BaseRunUrl;
        public static string BaseTokenUrl;
        public static string BaseUrl;

        public static string selectedMenuTab { get; set; }
        public static string bearerToken { get; set; }
        public static string bearerPatientToken { get; set; }

        private static TelegramBotClient client;



////////////////////////////////////////////////////////////////////////////
        // Menu
        public static string menuTabName = "mainMenu";
        public static bool profileTabIsActive = false;
        public static bool recordToVisitTabIsActive = false;
        public static bool visitTabIsActive = false;
        public static bool documentTabIsActive = false;
        public static bool newsTabIsActive = false;
        public static bool contactsTabIsActive = false;

        public static bool loginIsEntered = false;
        public static bool loginNeedToEnter = false;
        public static bool passwordIsEntered = false;
        public static bool passwordNeedToEnter = false;
        public static string username;
        public static string password;

        public static string patientLogin;
```

```
////////////////////////////////////////////////////////////////////////
        /// Record to the visit Menu
        public static bool recordToVisitIsActive = false;

        public static bool recortBySpecialityIsActive = false;
        public static bool recortToServiceIsActive = false;
        public static bool recortToUserIsActive = false;

        public static string userSpecialityCode;
        public static string userLogin;
        public static string serviceID;
        public static string venueID;
        public static string startTime;
        public static string endTime;


////////////////////////////////////////////////////////////////////////


        static Program()
        {
            var configuration = new ConfigurationBuilder()
                .AddJsonFile("appsettings.json", optional: true, reloadOnChange:
true)
                .Build();

            TelegramBotToken = configuration["TelegramBotToken"];
            UserLogin = configuration["UserLogin"];
            UserPassword = configuration["UserPassword"];
            InstallationGUID = configuration["InstallationGUID"];
            GrandType = configuration["GrandType"];
            BaseRunUrl = configuration["BaseRunUrl"];
            BaseTokenUrl = configuration["BaseTokenUrl"];
            BaseUrl = configuration["BaseUrl"];

            string                        appSettingsFile                        =
Path.Combine(Directory.GetCurrentDirectory(), "appsettings.json");
            string directoryPath = Path.GetDirectoryName(appSettingsFile);
            string fileName = Path.GetFileName(appSettingsFile);
            appSettingsFile                                                      =
Path.Combine(directoryPath.Replace(@"\bin\Debug\netcoreapp3.1", ""), fileName);

            if (System.IO.File.Exists(appSettingsFile))
            {
                string json = System.IO.File.ReadAllText(appSettingsFile);
                AppConfig                          config                        =
JsonConvert.DeserializeObject<AppConfig>(json);

                TelegramBotToken = config.TelegramBotToken;
                UserLogin = config.UserLogin;
                UserPassword = config.UserPassword;
                InstallationGUID = config.InstallationGUID;
                GrandType = config.GrandType;
                BaseRunUrl = config.BaseRunUrl;
                BaseTokenUrl = config.BaseTokenUrl;
                BaseUrl = config.BaseUrl;

            }
            else
            {
                Console.WriteLine("Cannot find a file appsettings.json");
            }
```

```
        }


        static void Main(string[] args)
        {
            client = new TelegramBotClient(TelegramBotToken);

            var cts = new CancellationTokenSource();
            var cancellationToken = cts.Token;

            var receiverOptions = new ReceiverOptions
            {
                AllowedUpdates = { } // receive all update types
            };

            client.StartReceiving(
                HandleUpdateAsync,
                HandleErrorAsync,
                receiverOptions,
                cancellationToken: cts.Token
            );

            Console.WriteLine("Bot is started!");
            Console.ReadLine();

            cts.Cancel();


            Console.WriteLine("Hello World!");
        }


        public static async Task HandleUpdateAsync(ITelegramBotClient client,
Update update, CancellationToken cancellationToken)
        {
            if (update.Type == UpdateType.Message && update?.Message?.Text !=
null)
            {
                await HandleMessage(client, update.Message);
                return;
            }

            if (update.Type == UpdateType.CallbackQuery)
            {
                await HandleCallbackQuery(client, update.CallbackQuery);
                return;
            }
        }

        public static async Task HandleMessage(ITelegramBotClient client, Message
message)
        {
            Console.WriteLine($"message.Text = {message.Text}");

            if (message.Text == null || message.Text == "")
            {
                Console.WriteLine("test");
            }
            else if (message.Text == "/start")
            {
                await    client.SendTextMessageAsync(message.Chat.Id,    "Select
command: /keyboard | /authorization");
```

```
                return;
            }
            else if (message.Text == "/menu")
            {
                recordToVisitIsActive = false;

                recortBySpecialityIsActive = false;
                recortToServiceIsActive = false;
                recortToUserIsActive = false;

                userSpecialityCode = null;
                userLogin = null;
                serviceID = null;
                venueID = null;
                startTime = null;
                endTime = null;
                selectedMenuTab = null;

                await client.SendTextMessageAsync(message.Chat.Id, "Ви перейшли в
головне меню");
                return;
            }
            else if (message.Text == "/freeDataList")
            {
                await    client.SendTextMessageAsync(message.Chat.Id,    "Select
command: /keyboard | /authorization");
                return;
            }

            else  if  (message.Text  ==  "/authorization"  ||  message.Text  ==
"Авторизація")
            {
                loginNeedToEnter = true;
                passwordNeedToEnter = false;
                loginIsEntered = false;
                passwordIsEntered = false;
            }

            /// Authorization (entering and check credentials)
            if (loginNeedToEnter)
            {

                await client.SendTextMessageAsync(message.Chat.Id, $"Введіть Ваш
логін");

                loginNeedToEnter = false;
                loginIsEntered = true;
                passwordNeedToEnter = true;
            }
            else if (passwordNeedToEnter && !passwordIsEntered)
            {
                username = message.Text;

                passwordNeedToEnter = false;
                passwordIsEntered = true;

                await client.SendTextMessageAsync(message.Chat.Id, $"Введіть Ваш
пароль");

            }
        }
```

```
        public static async Task HandleCallbackQuery(ITelegramBotClient client,
CallbackQuery callbackQuery)
        {
            int documentID, scheduleID;
            bool isNumber;
            string documentUrl, scheduleBookingInfo, userListAll, serviceListAll;

            Console.WriteLine($"selectedMenuTab = {selectedMenuTab}");

            if (selectedMenuTab == "Documents")
            {
                isNumber = int.TryParse(callbackQuery.Data, out documentID);

            }
            else
            {
                await client.SendTextMessageAsync(callbackQuery.Message.Chat.Id,
$"You selected : {callbackQuery.Data}");

                return;
            }

        }

        public static Task HandleErrorAsync(ITelegramBotClient client, Exception
exception, CancellationToken cancellationToken)
        {
            var ErrorMessage = exception switch
            {
                ApiRequestException apiRequestException
                    =>                      $"Error                      Telegram
API:\n{apiRequestException.ErrorCode}\n{apiRequestException.Message}",
                _ => exception.ToString()
            };
            Console.WriteLine(ErrorMessage);
            return Task.CompletedTask;
        }


    }
}

using System;
using System.Collections.Generic;
using Newtonsoft.Json;
using System.IO;
using System.Net;

namespace IFNTUNGHelper
{
    public class AppConfig
    {
        public ConnectionStrings ConnectionStrings { get; set; }
        public string TelegramBotToken { get; set; }
        public string UserLogin { get; set; }
        public string UserPassword { get; set; }
        public string InstallationGUID { get; set; }
        public string GrandType { get; set; }
        public string BaseRunUrl { get; set; }
        public string BaseTokenUrl { get; set; }
        public string BaseUrl { get; set; }
        public string AppsettingsPath { get; set; }
    }
```

```
    public class ConnectionStrings
    {
        public string DefaultConnection { get; set; }
    }
}

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from  sklearn.metrics  import  accuracy_score,  precision_score,  recall_score,
confusion_matrix
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from mlxtend.frequent_patterns import apriori, association_rules
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
import networkx as nx
from networkx.algorithms.community import greedy_modularity_communities

# 1. Завантаження та підготовка даних
# Завантаження даних
data_url           =           'https://archive.ics.uci.edu/ml/machine-learning-
databases/00222/bank.zip'
data                                                                          =
pd.read_csv(r"D:\MyStudy\University\Master\2_Semester\DataExtractionMethodsAndTo
ols\Laboratory\bank\bank.csv", delimiter=';')

# Очищення даних
data = data.dropna()

# Підготовка даних
X = data.drop('y', axis=1)
y = data['y'].apply(lambda x: 1 if x == 'yes' else 0)

# Кодування категоріальних ознак
X_encoded = pd.get_dummies(X, columns=['job', 'marital', 'education', 'default',
'housing', 'loan', 'contact', 'month', 'poutcome'])

# Розділення даних
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.3,
random_state=42)

# Нормалізація даних
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# 2. Побудова моделі дерев вирішень для прогнозування
# Побудова моделі
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)

# Прогнозування
y_pred = model.predict(X_test)

# Оцінка ефективності моделі
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
```

```
specificity = cm[0,0] / (cm[0,0] + cm[0,1])

print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'Specificity: {specificity}')

# 3. Кластеризація клієнтів за допомогою k-середніх
# Кластеризація
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_train)

# Прогнозування кластерів
clusters = kmeans.predict(X_train)

# Аналіз результатів
plt.scatter(X_train[:, 0], X_train[:, 1], c=clusters)
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('K-means Clustering')
plt.show()

# 4. Пошук асоціативних правил
# Перетворення даних для пошуку правил
basket = (X > 0).astype(int)

# Пошук частих наборів елементів
frequent_itemsets = apriori(basket, min_support=0.1, use_colnames=True)

# Пошук асоціативних правил
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.0)
print(rules.head())

# 5. Байесовська класифікація текстових даних
# Приклад текстових даних
texts = ["text data 1", "text data 2", "text data 3"]
labels = [0, 1, 0]

# Перетворення текстових даних
vectorizer = CountVectorizer()
X_texts = vectorizer.fit_transform(texts)

# Байесовська класифікація
nb_model = MultinomialNB()
nb_model.fit(X_texts, labels)

# Прогнозування
text_pred = nb_model.predict(X_texts)

# Оцінка ефективності моделі
accuracy_text = accuracy_score(labels, text_pred)
precision_text = precision_score(labels, text_pred)
recall_text = recall_score(labels, text_pred)

print(f'Accuracy (text): {accuracy_text}')
print(f'Precision (text): {precision_text}')
print(f'Recall (text): {recall_text}')

# 6. Асоціативна кластеризація
# Створення графу
G = nx.karate_club_graph()

# Асоціативна кластеризація
```

```
communities = list(greedy_modularity_communities(G))

# Аналіз результатів
print(f'Number of communities: {len(communities)}')
print(f'Communities: {communities}')

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
namespace IFNTUNGHelperIB.Models
{
    public class Student
    {
        public int StudentID { get; set; }
        [MaxLength(200)]
        public string? StudentName1 { get; set; }
        [MaxLength(200)]
        public string? StudentName2 { get; set; }
        [MaxLength(200)]
        public string? StudentName3 { get; set; }
        public string? StudentPhoto { get; set; }
        public DateTime? StudentBirthDdate { get; set; }
        public string? StudentPhone1 { get; set; }
        public string? StudentPhone2 { get; set; }
        [EmailAddress]
        public string? StudentEmail { get; set; }
        public string? StudentAddress { get; set; }
        public bool? StudentHasScholarship { get; set; }
        public bool? StudentHasChild { get; set; }
        [ForeignKey("Group")]
        public int? StudentGroupRef { get; set; }
        public Group? Group { get; set; } // Navigation property for Group
        [ForeignKey("Faculty")]
        public int? StudentFacultyRef { get; set; }
        public Faculty? Faculty { get; set; } // Navigation property for Faculty
        [ForeignKey("City")]
        public int? StudentCityRef { get; set; }
        public City? City { get; set; } // Navigation property for City
        [ForeignKey("Gender")]
        public int? StudentGenderRef { get; set; }
        public Gender? Gender { get; set; } // Navigation property for Gender
        [ForeignKey("Speciality")]
        public int? StudentSpecialityRef { get; set; }
        public Speciality? Speciality { get; set; } // Navigation property for
Speciality
        public int? StudentScholarship { get; set; }
    }
}



using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;
using System.Threading.Tasks;
using IFNTUNGHelperIB.Models;
using IFNTUNGHelperIB.Controllers;
namespace IFNTUNGHelperIB.Models
{
    public class StudentContext : DbContext
    {
        public   StudentContext(DbContextOptions<StudentContext>   options)   :
base(options)
        {
```

```
        }
        public DbSet<Student> Students { get; set; }
        public DbSet<Group> Groups { get; set; }
        public DbSet<Faculty> Faculties { get; set; }
        public DbSet<City> Cities { get; set; }
        public DbSet<Gender> Genders { get; set; }
        public DbSet<Speciality> Specialities { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Student>()
                .HasOne<Group>(s => s.Group)
                .WithMany(g => g.Students)
                .HasForeignKey(s => s.StudentGroupRef);

            modelBuilder.Entity<Student>()
                .HasOne<Faculty>(s => s.Faculty)
                .WithMany(f => f.Students)
                .HasForeignKey(s => s.StudentFacultyRef);

            modelBuilder.Entity<Student>()
                .HasOne<City>(s => s.City)
                .WithMany(c => c.Students)
                .HasForeignKey(s => s.StudentCityRef);

            modelBuilder.Entity<Student>()
                .HasOne<Gender>(s => s.Gender)
                .WithMany(g => g.Students)
                .HasForeignKey(s => s.StudentGenderRef);

            modelBuilder.Entity<Student>()
                .HasOne<Speciality>(s => s.Speciality)
                .WithMany(sp => sp.Students)
                .HasForeignKey(s => s.StudentSpecialityRef);
        }
    }
}


using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using IFNTUNGHelperIB.Models;
namespace IFNTUNGHelperIB.Controllers

public async Task<ActionResult<IEnumerable<Student>>> GetStudents()
{
if (_StudentContext.Students == null)
{
return NotFound();
}
else
{
return await _StudentContext.Students.ToListAsync();
}
}



[HttpGet("{id}")]
public async Task<ActionResult<IEnumerable<Student>>> GetStudent(int id)
{
if (_StudentContext.Students == null)
```

```
{
return NotFound();
}
else
{
var Student = await _StudentContext.Students.FindAsync(id);
     if (Student == null)
     {
        return NotFound();
     }
     else
     {
        // return Student;
        return Ok(new List<Student> { Student });
     }
}
}


HttpPost]
        public async Task<ActionResult<Student>> PostStudent(Student Student)
        {
            _StudentContext.Students.Add(Student);
            await _StudentContext.SaveChangesAsync();
            return    CreatedAtAction(nameof(GetStudent),    new    {    id    =
Student.StudentID }, Student);


[HttpPut]
public async Task<ActionResult> PutStudent(int id, Student Student)
{

if (id != Student.StudentID)
{
return BadRequest();
}
{
_StudentContext.Entry(Student).State = EntityState.Modified;
}
try
{
     await _StudentContext.SaveChangesAsync();
}
catch (DbUpdateConcurrencyException)
{
     if (!StudentAvailable(id))
{
     return NotFound();
}
else
{
     throw;
}
return Ok();
}



[HttpDelete]
public async Task<ActionResult> DeleteStudent(int id)
{
if (_StudentContext.Students == null)
```

```
{
return NotFound();
}
else
{
var Student = _StudentContext.Students.FindAsync(id);
     if (Student == null)
     {
            return NotFound();
     }
     else
     {
          _StudentContext.Students.Remove(await Student);
          await _StudentContext.SaveChangesAsync();
          return Ok();
      }
}
}
```

```
using System;
using System.Collections.Generic;

namespace AgileScrumExample
{
    class Program
    {
        static void Main()
        {
            // Sprint planning meeting
            Console.WriteLine("Sprint 1 - Planning Meeting");
            Console.WriteLine("Discussing and prioritizing user stories...");

            // Creating a list of user stories (tasks)
            List<UserStory> backlog = new List<UserStory>
            {
                new UserStory("As a user, I want to log in to the system", 5),
                new UserStory("As a user, I want to view my profile", 3),
                new UserStory("As a user, I want to search for products", 8)
            };

            // Sprint backlog planning
            List<UserStory> sprintBacklog = new List<UserStory>();
            int sprintCapacity = 10;
            int remainingCapacity = sprintCapacity;

            foreach (UserStory story in backlog)
            {
                if (story.StoryPoints <= remainingCapacity)
                {
                    sprintBacklog.Add(story);
                    remainingCapacity -= story.StoryPoints;
                }
            }

            // Sprint execution
            Console.WriteLine("\nSprint 1 - Execution");
            foreach (UserStory story in sprintBacklog)
            {
                Console.WriteLine($"Working on: {story.Description}");
                // Simulating work completion
                story.Complete();
            }
```

```
            // Sprint review and retrospective
            Console.WriteLine("\nSprint 1 - Review and Retrospective");
            foreach (UserStory story in sprintBacklog)
            {
                if (story.IsCompleted)
                    Console.WriteLine($"{story.Description} - Completed");
                else
                    Console.WriteLine($"{story.Description} - Incomplete");
            }

            Console.WriteLine("Identifying improvements and discussing what went
well.");

            Console.WriteLine("\nEnd of Sprint 1.");
        }
    }

    // UserStory class representing a task
    public class UserStory
    {
        public string Description { get; }
        public int StoryPoints { get; }
        public bool IsCompleted { get; private set; }

        public UserStory(string description, int storyPoints)
        {
            Description = description;
            StoryPoints = storyPoints;
            IsCompleted = false;
        }

        public void Complete()
        {
            // Simulating completion of the user story
            IsCompleted = true;
        }
    }
}
const test = require('ava');
const fs = require('fs');
const app = require('../app');
const common = require('../lib/common');
const request = require('supertest');
const agent = request.agent(app);

// Get test data to compare in tests
const rawTestData = fs.readFileSync('./bin/testdata.json', 'utf-8');
const testData = JSON.parse(rawTestData);

// Setup some global DB objects for comparison
let db;
let config;
let products;
let customers;
let users;

// Start up app and wait for it to be ready
test.before.cb(t => {
    app.on('appStarted', async () => {
        // Set some stuff now we have the app started
        config = app.config;
        db = app.db;
```

```
        // Get some data from DB to use in compares
        await common.testData(app);
        products = await db.products.find({}).toArray();
        customers = await db.customers.find({}).toArray();
        users = await db.users.find({}).toArray();
        agent
            .post('/admin/login_action')
            .send({
                email: users[0].userEmail,
                password: 'test'
            })
            .expect(200)
            .end((err, res) => {
                if(err){
                    t.fail();
                    t.end();
                }
                t.end();
            });
    });
});

test.cb('[Success] Get products JSON', t => {
    agent
        .get('?json=true')
        .expect(200)
        .end((err, res) => {
            if(err){
                t.fail();
                t.end();
            }

            if(res.body.length < config.productsPerPage){
                t.is(res.body.length, testData.products.length);
            }else{
                t.is(res.body.length, config.productsPerPage);
            }
            t.pass();
            t.end();
        });
});

test.cb('[Success] User Login', t => {
    agent
        .post('/admin/login_action')
        .send({
            email: users[0].userEmail,
            password: 'test'
        })
        .expect(200)
        .end((err, res) => {
            if(err){
                t.fail();
                t.end();
            }

            t.deepEqual(res.body.message, 'Login successful');
            t.end();
        });
});

test.cb('[Fail] Incorrect user password', t => {
    agent
```

```
        .post('/admin/login_action')
        .send({
            email: users[0].userEmail,
            password: 'test1'
        })
        .expect(400)
        .end((err, res) => {
            if(err){
                t.fail();
                t.end();
            }

            t.deepEqual(res.body.message, 'Access denied. Check password and try
again.');
            t.end();
        });
});

test.cb('[Fail] Customer login with incorrect email', t => {
    agent
        .post('/customer/login_action')
        .send({
            loginEmail: 'test1@test.com',
            loginPassword: 'test'
        })
        .expect(400)
        .end((err, res) => {
            if(err){
                t.fail();
                t.end();
            }

            t.deepEqual(res.body.message, 'A customer with that email does not
exist.');
            t.end();
        });
});

test.cb('[Success] Customer login with correct email', t => {
    agent
        .post('/customer/login_action')
        .send({
            loginEmail: 'test@test.com',
            loginPassword: 'test'
        })
        .expect(200)
        .end((err, res) => {
            if(err){
                t.fail();
                t.end();
            }

            t.deepEqual(res.body.message, 'Successfully logged in');
            t.end();
        });
});

test.cb('[Success] Add product to cart', t => {
    agent
        .post('/product/addtocart')
        .send({
            productId: products[0]._id,
            productQuantity: 1,
```

```
            productOptions: JSON.stringify(products[0].productOptions)
        })
        .expect(200)
        .end((err, res) => {
            if(err){
                t.fail();
                t.end();
            }

            t.deepEqual(res.body.message, 'Cart successfully updated');
            t.end();
        });
});

test.cb('[Fail] Add incorrect product to cart', t => {
    agent
        .post('/product/addtocart')
        .send({
            id: 'fake_product_id',
            state: false
        })
        .expect(400)
        .end((err, res) => {
            if(err){
                t.fail();
                t.end();
            }
            t.deepEqual(res.body.message,  'Error  updating  cart.  Please  try
again.');
            t.end();
        });
});

test.cb('[Sucess] Change product publish status', t => {
    agent
        .post('/admin/product/published_state')
        .send({
            id: products[0]._id,
            state: false
        })
        .expect(200)
        .end((err, res) => {
            if(err){
                t.fail();
                t.end();
            }
            t.end();
        });
});
```