Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

_____Інститут електроенергетики_____
(навчально-науковий інститут)

_____факультет інформаційних технологій____
(факультет)

Кафедра_____інформаційних технологій та комп'ютерної інженерії_____
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи ступеня __бакалавра_

(бакалавра, магістра)

Здобувача вищої освіти_____Артемова Володимира Олександровчиа_____
(ПІБ)

академічної групи_____126-21-2_____
(шифр)

спеціальності _____126 «Інформаційні системи та технології»_____
(код і назва спеціальності)

спеціалізації за освітньо-професійною (освітньо-науковою) програмою _____
(за наявності) _____ Інформаційні системи та технології_____
(офіційна назва)

на тему _____ Розробка рекомендаційної системи вибору курсів мережної академії Cisco_____
(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	Доц. Соколова Н. О.			
розділів:				

Рецензент				

Нормоконтролер	Проф. Коротенко Г.М.			

Дніпро
20_

ЗАТВЕРДЖЕНО:
завідувач кафедри
інформаційних технологій та комп'ютерної інженерії
(повна назва)

_____ __В.В. Гнатушенко__
(підпис) (ініціали та прізвище)

«_____» _____20___року

ЗАВДАННЯ

на кваліфікаційну роботу
ступеня бакалавра
(бакалавра, магістра)

здобувача вищої освіти ____Артемов В.О.____ **академічної групи**___126-21-2___
(прізвище та ініціали) (шифр)

спеціальності_____126 «Інформаційні системи та технології»_____

спеціалізації за освітньою-професійною програмою _____
(за наявності)

_____

на тему___Розробка рекомендаційної системи вибору курсів мережної академії Cisco_____

затверджену наказом ректора НТУ «Дніпровська політехніка» від 05.05.2025 № 336-с

Розділ	Зміст	Термін виконання
Розділ 1	Аналіз стану області рішення завдання	01.02.2025 – 03.03.2025
Розділ 2	Проектування та розробка інформаційної системи	04.03.2025– 17.04.2025
Розділ 3	Тестування та результати роботи	18.04.25 – 21.05.2025

Завдання видано _____ __Соколова Н. О.___
(підпис керівника) (ініціали та прізвище)

Дата видачі 03.02.2025

Дата подання до екзаменаційної комісії _____

Прийнято до виконання _____ _____Артемов В.О.____
(підпис здобувача вищої освіти) (ініціали та прізвище)

РЕФЕРАТ

Пояснювальна записка: 61 с., 24 рис., 3 табл., 2 додаток, 31 джерела.

ІНФОРМАЦІЙНА СИСТЕМА, ЧАТ-БОТ, ПЕРСОНАЛІЗАЦІЯ, РЕКОМЕНДАЦІЙНА СИСТЕМА, CISCO NETWORKING ACADEMY, PYTHON, TELEGRAM BOT API, POSTGRESQL, ШТУЧНИЙ ІНТЕЛЕКТ.

Об'єкт кваліфікаційної роботи: інформаційна система у вигляді інтелектуального чат-бота для персоналізованого підбору курсів на платформі Cisco Networking Academy.

Предмет кваліфікаційної роботи: методи та засоби реалізації чат-бота для рекомендації навчальних курсів з урахуванням інтересів і рівня знань користувача.

Мета роботи: розробка інтелектуального чат-бота, який допомагає користувачам Cisco Networking Academy ефективно орієнтуватися в навчальному контенті, враховуючи їхні інтереси, рівень знань та основні цілі.

У вступі розглядається актуальність проблеми інформаційного перенавантаження в онлайн-освіті, зокрема на платформах з великою кількістю навчальних матеріалів, таких як Cisco Networking Academy. Обгрунтовується необхідність створення інструментів для персоналізованого підбору курсів.

У першому розділі описується в подробицях аналіз існуючих освітніх платформ, проблеми інформаційного перенавантаження та труднощі у виборі релевантних курсів. Розглядаються сучасні підходи до вирішення цих проблем, зокрема використання інтелектуальних чат-ботів.

У другому розділі обговорюється в подробицях процес розробки інтелектуального чат-бота: вибір технологій, проектування архітектури, реалізація функціоналу та інтеграція з платформою Cisco Networking Academy. Наводяться результати тестування та оцінка ефективності розробленого рішення.

У висновках узагальнені результати проекту, підтверджено доцільність застосування інтелектуального чат-бота для підвищення ефективності навчального процесу, запропоновано напрями подальшого вдосконалення системи. Актуальність платформи обумовлюється великим попитом на якісні оналайн-курси в галузі інформаційних технологій та мережевих технологій, а також необхідністю ефективного управління навчальним контентом для забезпечення індивідуального підходу до кожного користувача.

Практичне значення кваліфікаційної роботи полягає у створенні інструменту, який спрощує вибір навчальних курсів, знижує навантаження на користувачів платформи Cisco Networking Academy та сприяє персоналізації навчання.

Розроблене програмне забезпечення може бути запроваджено у рамках навчального процесу з платформою Cisco Networking Academy, а також адаптоване до інших сервісів або систем з подібною структурою.

Abstract

Explanatory note: 61 p., 24 fig., 3 tables, 2 appendices, 31 sources.

INFORMATION SYSTEM, CHATBOT, PERSONALIZATION, RECOMMENDATION SYSTEM, CISCO NETWORKING ACADEMY, PYTHON, TELEGRAM BOT API, POSTGRESQL, ARTIFICIAL INTELLIGENCE.

Object of qualification work: an information system in the form of an intelligent chatbot for personalized course selection on the Cisco Networking Academy platform.

Subject of the qualification work: methods and means of implementing a chatbot for recommending training courses based on the interests and level of knowledge of the user.

Purpose: to develop an intelligent chatbot that helps Cisco Networking Academy users to effectively navigate the learning content, taking into account their interests, level of knowledge and main goals.

The introduction discusses the relevance of the problem of information overload in online education, in particular on platforms with a large number of educational materials, such as Cisco Networking Academy. The necessity of creating tools for personalized course selection is substantiated.

The first section describes in detail the analysis of existing educational platforms, the problems of information overload and difficulties in choosing relevant courses. It also discusses modern approaches to solving these problems, including the use of intelligent chatbots.

The second section discusses in detail the process of developing an intelligent chatbot: technology selection, architecture design, functionality implementation, and integration with the Cisco Networking Academy platform. The results of testing and evaluation of the developed solution are presented.

The conclusions summarize the results of the project, confirm the feasibility of using an intelligent chatbot to improve the efficiency of the educational process, and suggest areas for further improvement of the system. The relevance of the platform is due to the high demand for high-quality online courses in the field of information technology and networking, as well as the need for effective management of educational content to ensure an individual approach to each user.

The practical significance of the qualification work is to create a tool that simplifies the selection of training courses, reduces the burden on users of the Cisco Networking Academy platform, and promotes personalization of training.

The developed software can be implemented as part of the educational process with the Cisco Networking Academy platform, as well as adapted to other services or systems with a similar structure.

Зміст

ВСТУП

Проблема складності навчання та його ефективності має глибоке коріння в історії людства. Багато поколінь навчання було й залишається невід'ємною частиною життя людини. З дитинства ми пізнаємо світ, розвиваємось і форм як особистості. Щороку процес навчання стає більш насиченим, технологічним і доступним – ресурси зростають, але виклики залишаються. Навчання – це шлях безперервного росту, де кожен крок відкриває нові горизонти, а знання перетворюють потенціал на силу.

Сучасне суспільство диктує нові вимоги до рівня обізнаності, гнучкості та швидкості засвоєння інформації. У зв'язку з цим проблеми ефективного навчання набуває особливої актуальності. Однак, разом із розширенням можливостей доступу до інформації виникає й нова перешкода – інформаційне перевантаження. Ресурси для навчання збільшуються щодня: платформи, відео лекції, онлайн-курс, методичні матеріали, інтерактивні завдання, тести, форуми тощо. Усе це створює ситуацію, коли бажання вчитися часто стикається з проблемою вибору. Людина, яка хоче опанувати нову тему або професію, стикається з надлишком інформації та контенту, серед якого важко знайти справді якісні, структуровані та відповідні матеріали. Це призводить до плутанини, зниження мотивації, а іноді навіть до повної відмови від навчального процесу.

Окремо слід виділити платформу Cisco Networking Academy – одну з провідних освітніх систем, що надає доступ до великої кількості курсів у сфері мережевих технологій, кібербезпеки, програмування та суміжних напрямків. Курси мають різні рівні складності та різну тривалість, що дає змогу користувачам з різним досвідом знаходити для себе відповідні варіанти. Проте саме велика кількість контенту на платформі створює певні труднощі: новачкам складно зрозуміти, з чого почати, який курс обрати, як сформувати ефективну освітню траєкторію. Без зовнішньої допомоги багато користувачів

витрачають значний час на пошук потрібного курсу або залишають платформу, не почавши навчання.

У відповідь на ці виклики з’являється потреба в інтелектуальних рішеннях, які здатні ефективно аналізувати потреби користувача, його рівень знань, інтереси та освітні цілі, і на основі цього формувати персоналізовані рекомендації. Одним із перспективних підходів є використання чат-ботів на основі штучного інтелекту, які забезпечують інтерактивну взаємодію з користувачем, адаптацію під індивідуальні особливості та швидке надання релевантної інформації.

Чат-боти можуть не лише заощадити час користувача, а й підвищити ефективність навчального процесу шляхом надання чіткої освітньої маршрутизації. Вони здатні задавати уточнюючі питання, пропонувати курси відповідно до обраного напряму, а також адаптувати поради на основі зворотного зв’язку. У контексті платформи Cisco створення такого інструменту має потенціал значно покращити досвід користувача та сприяти більш усвідомленому вибору навчального контенту.

Мета дипломного проекту – розробка інтелектуального чат-бота, який допомагатиме користувачам платформи Cisco Network Academy у виборі навчальних курсів, враховуючи їхні інтереси, рівень підготовки та освітні цілі.

Об’єкт дослідження – процес взаємодії користувача з освітньою платформою Cisco Network Academy.

Предмет дослідження – методи та засоби реалізації інтелектуального чат-бота для персоналізованого підбору навчальних курсів на платформі Cisco.

1 АНАЛІЗ СТАНУ ОБЛАСТІ РІШЕННЯ ЗАВДАННЯ

1.1 Загальні відомості з предметної галузі

У сучасному світі цифрові технології відіграють ключову роль у трансформації освітнього процесу. Однією з провідних платформ у галузі інформаційних технологій є Cisco Networking Academy – глобальна освітня програма, започаткована компанією Cisco System у 1997 році[10]. Програма надає доступ до безкоштовних онлайн-курсів, сертифікацій та навчальних ресурсів з таких напрямів, як мережеві технології, кібербезпека, програмування та інші суміжні галузі. З моменту заснування понад 17,5 мільйонів студентів у 190 країнах світу скористалися можливостями Cisco Networking Academy для здобуття цифрових навичок.

Зі зростанням обсягу доступної інформації та кількості навчальних курсів виникає проблема інформаційного перевантаження [1], коли користувачам складно самостійно обрати релевантні курси, що відповідають їхнім потребам та рівню підготовки. Це знижує ефективність навчального процесу та може призводити до втрати мотивації.

У відповідь на ці виклики в освітньому середовищі набувають популярності інтелектуальні чат-боти – програмні агенти, що використовують технології штучного інтелекту для взаємодії з користувачами в режимі реального часу [5]. Чат-боти можуть виконувати різноманітні функції: від надання інформації та рекомендацій до проведення оцінювання знань та підтримки навчального процесу.

Використання чат-ботів у навчанні має низку переваг [9]:

- Персоналізація навчання: чат-боти можуть адаптувати навчальний контент відповідно до індивідуальних потреб та рівня знань користувача.
- Доступність: забезпечують цілодобовий доступ до навчальних матеріалів та підтримки.

- Інтерактивність: сприяють активному залученню користувачів до навчального процесу через інтерактивні діалоги та завдання.
- Автоматизація: дозволяють автоматизувати рутинні завдання, такі як реєстрація на курси, надання довідкової інформації, оцінювання тощо.

Проте впровадження чат-ботів в освітній процес також пов’язане з певними викликами, зокрема необхідністю забезпечення високої якості взаємодії обробки природної мови та адаптації до різноманітних освітніх контекстів.

У контексті платформи Cisco Networking Academy розробка інтелектуального чат-бота для допомоги у виборі навчальних курсів є актуальним завданням, що сприятиме підвищенню ефективності навчання, зменшенню інформаційного перевантаження та покращенню користувацького досвіду [14].

1.2. Призначення розробки та галузь застосування

Розробка інтелектуального чат-бота зумовлена необхідністю подолання проблеми інформаційного перенавантаження в онлайн-освіті, що особливо актуально для платформ із великим обсягом навчального контенту, таких як Cisco Networking Academy. Аналіз сучасного стану предметної області вказує на потребу у створенні інструментів, здатних адаптивно орієнтувати користувача в освітньому середовищі.

Запропонована система чат-бота має на меті надати персоналізовану підтримку при виборі навчальних курсів відповідно до інтересів, рівня підготовки та професійних цілей користувача. Застосування інтелектуальних алгоритмів дозволяє автоматично аналізувати вхідні дані та генерувати релевантні рекомендації, що сприяє підвищенню мотивації до навчання та якості освітньої взаємодії.

Розроблена система призначена для використання в освітній сфері, передусім на платформі Cisco Networking Academy, а також може бути адаптована для інтеграції з іншими навчальними середовищами та системами управління навчанням (LMS). Потенційними користувачами є студенти, викладачі, освітні консультанти й адміністратори.

Особливу увагу приділено адаптації чат-бота до потреб студентів НТУ «Дніпровська політехніка». Для цього передбачено можливість реалізації окремого функціонального модуля, що забезпечить швидкий та зручний доступ до сервісу безпосередньо для студентів університету [25]. Такий підхід дозволить підвищити ефективність навігації освітнім контентом, а також посилити індивідуалізацію освітнього процесу в межах конкретного закладу вищої освіти.

Чат-бот забезпечує [5]:

- автоматизовану обробку запитів щодо навчальних програм;
- рекомендації курсів відповідно до профілю користувача;
- скорочення часу на пошук релевантного контенту;
- зниження когнітивного навантаження під час взаємодії з платформою.

Таким чином, розробка має чітко визначене функціональне призначення та охоплює перспективну галузь застосування — сучасні цифрові освітні середовища з підтримкою персоналізованих технологій навчання.

1.3. Постановка завдання

Завдання: розробити чат-бота мовою Python із використанням фреймворку OpenCV, підключити базу даних PostgreSQL для зберігання інформації про користувачів та курси. Інтеграція з API Cisco Networking Academy для отримання актуальної інформації про курси.

Завдання дипломної роботи:

1. Проаналізувати сучасні проблеми інформаційного перевантаження в онлайн-освіті та труднощі у виборі навчальних курсів.
2. Дослідити функціональні можливості та структуру платформи Cisco Network Academy.
3. Вивчити існуючі рішення щодо використання чат-бота для платформи Cisco.
4. Розробити концепцію інтелектуального чат-бота для платформи Cisco.
5. Реалізувати прототип чат-бота з використанням сучасних технологій штучного інтелекту.
6. Провести тестування розробленого чат-бота та оцінити його ефективність у підборі навчальних курсів.

Функціонал:

У таблиці 1.1 наведено можливості користувачів (об’єктів) при роботі з чат-бота.

Також передбачається, що платформа налічуватиме певні об’єкти з певними властивостями (наведено у табиці. 1.2):

Таблиця 1.1 - Можливості користувачів роботи з чат-ботом.

Об’єкт	Можливості
Студент	-Реєстрація та авторизація в системі -Заповнення профілю (інтереси, рівень знань) -Отримання персоналізованих рекомендацій курсу -Перегляд описів курсів та їх структури -Збереження обраних курсів до списку «Обране» -Отримання сповіщень про нові курси та оновлення -Надсилання зворотного зв’язку щодо роботи чат-бота
Адміністратор	-Керування базою даних курсів (додавання, редагування, видалення)

	-Модернізація оголошень та повідомлень -Аналіз статистики використання чат-бота -Управління доступу користувачів -Налаштування інтеграції з платформою Cisco Networking Academy
Гість	-Ознайомлення з функціоналом бота -Перегляд загальної інформації про доступні курси -Отримання загальних рекомендацій без персоналізації -Можливості реєстрації для повного доступу до функцій

Таблиця 1.2 – Об’єкти платформи та їх властивості

Об’єкт	Властивості
Користувач (студент, адміністратор, гість)	- Ідентифікатор користувача -Роль (студент, адміністратор, гість) -Ім’я та прізвище -Електрона пошта -Пароль (для студентів та адміністраторів) -Інтереси та цілі навчання (для студентів) -Історія взаємодії з чат-ботом
Оголошення	-Заголовок -Текст оголошення -Дата та час публікації -Автор оголошення (адміністратор) -Статус (активне\неактивне)
Чат	-Унікальний ідентифікатор сесії -Історія повідомлень -Дата та час початку сесії -Тип користувача (студент, гість) -Взаємодія з курсами (переглянуті, обрані)

1.4. Вимоги до програми або програмного виробу

1.4.1. Вимоги до функціональних характеристик

Функціональні вимоги до чат-бота для рекомендації навчальних курсів на платформі Cisco Networking Academy передбачають створення інтерактивної системи, яка дозволяє студентам отримувати персоналізовані поради щодо вибору курсів. Користувачі повинні мати можливості реєструватися, заповнювати профілі з інформацією про свої інтереси та рівень підготовки, після чого отримувача рекомендації, що відповідають їхнім начальним цілям.

Чат-бот також має забезпечувати доступ до актуальної інформації про доступні курси, їх зміст та вимоги, дозволяючи користувачам переглядати та обирати відповідні програми. Крім того, система повинна обробляти запити користувачів, надаючи відповіді на поширені запитання та підтримуючи зворотний зв’язок.

З технічної точки зору, реалізація чат-бота передбачає використання мови програмування Python. Інтеграція з Telegram здійснюється за допомогою бібліотеки python-telegram-bot [30], яка забезпечує взаємодію з Telegram Bot API. Для зберігання даних користувача та курсів використовується реляційна база даних PostgreSQL, з’єднання з якою реалізується через бібліотеку psycopg2 [23].

Обробка повідомлень від Telegram може здійснюватися через вебхуки, які дозволяють отримувати оновлення в реальному часі. Для цього потрібно налаштувати HTTPS-з’єднання, за допомогою Let`s Encrypt.

Таким чином, поєднання зазначених технологій дозволяє створити ефективний та масштабний чат-бот, який забезпечує користувачам зручний доступ до персоналізованих рекомендацій навчальних курсів.

1.4.2 Вимоги до інформаційної безпеки

Основні вимоги до інформаційної безпеки, конфіденційність даних – необхідно забезпечити захист персональних даних користувачів, таких як імена, електрона адреса та історія взаємодії з ботом. Це досягається шляхом шифрування даних під час передачі та зберігання, а також обмеженням доступу до них лише уповноваженим особам [24].

Цілісність інформації – система гарантує, що дані не можуть бути змінені або знищені несанкціонованими особами. Це включає використання механізмів контролю версій, журналювання змін та регулярного резервного копіювання даних.

Доступність сервісу – чат-бот доступен для користувачів у будь-який час. Це вимагає впровадження заходів протидії атакам типу «відмова в обслуговуванні» (DDoS) [24], а також забезпечення надійної інфраструктури з можливістю масштабування.

Аутентифікація та авторизація – впровадження механізмів перевірки автентичності користувачів та обмеження їхніх прав доступу до функцій бота. Це включає використання токенів доступу, двофакторної аутентифікації та ролей користувачів.

Відповідність стандартам – система відповідає міжнародним стандартам інформаційної безпеки, таким як ISO\IEC 27001, що визначає вимоги до систем управління інформаційною безпекою.

1.4.3 Вимоги до складу та параметрів технічних засобів

Для забезпечення повноцінного доступу до функціональності інтелектуального чат-бота та інформаційної системи, розробленої в рамках проекту, користувачу достатньо мати будь-який сучасний пристрій, що підтримує роботу з веббраузерами або застосунками. Це може бути ноутбук, персональний комп’ютер, планшет або смартфон.

Мінімальні технічні вимоги для пристрою користувача (на прикладі роботи з браузерами Google Chrome на ноутбуці):

- Версія браузера: Google Chrome версії 81.0.4044.20 або вище;
- Операційна система: Windows 7 або новіша (також можливе використання macOS чи Linux, за умови підтримки сучасного браузера);
- Підтримка архітектури: 32- або 64-розрядна ОС та процесор;
- Оперативна пам’ять (RAM): не менше 2 ГБ;
- Процесор: з тактовою частотою від 1 ГГц;
- Накопичувач: HDD або SSD з обсягом вільного простору не менше 10 ГБ;
- Пристрої введення: клавіатура та миша\тачпад.

Такі параметри дозволять забезпечити стабільну роботу чат-бота через вебінтерфейс або месенджер Telegram, включаючи завантаження сторінок, обробку запитів, збереження даних та інтерактивну взаємодію з користувачем.

1.4.4 Вимоги до інформаційної та програмної сумісності

Розроблена інформаційна система у вигляді інтелектуального чат-бота повинна бути сумісною з іншими компонентами освітнього середовища, а також відповідати сучасним стандартам програмної взаємодії.

Програмна сумісність:

- Операційні системи: підтримка клієнтських пристроїв з ОС Windows 7 і вище, macOS, Linux, Android та iOS (через вебінтерфейс або Telegram)
- Браузери: чат-бот працює у браузерах із підтримкою сучасних вебтехнологій – Google Chrome (81.0.4044.20+), Mozilla Firefox, Safari, Microsoft Edge.

- Месенджери: повна сумісність з Telegram через використання Telegram Bot API та бібліотеки python-telegram-bot.
-
- Мови програмування та бібліотеки: системи реалізована на мові Python з використанням сумісних бібліотек (psycopg2, python-telegram-bot, OpenCV, ін.) [27].

Інформаційна сумісність:

- API Cisco Network Academy: системи інтегрується з API освітньої платформи Cisco, що забезпечує отримання актуальної інформації про курси та їх структуру.
- Структура даних: дані передаються у форматах JSON або XML – загальноприйнятих форматах обміну інформацією між вебсервісами.
- Інтероперабельність: структура даних дозволяє взаємодіяти з іншими освітніми системами (наприклад, LMS) при необхідності, що забезпечцє можливість подальної інтеграції.

1.5 Огляд існуючих освітніх чат-ботів

У зв'язку з розвитком цифрових технологій в освітньому середовищі все більшого поширення набувають інтелектуальні чат-боти, які відіграють роль асистентів, наставників або навігаторів у процесі навчання. Аналіз практичного застосування таких систем у відомих освітніх проектах дозволяє оцінити переваги та обмеження подібних рішень.

Приклади успішних реалізацій:

- Google Assistant for Education – бот, що допомагає учням керувати завданнями в Google Classroom, отримувати нагадування та знаходити навчальні матеріали.

- Doulingo Bot – чат-бот, вбудований у застосунок Doulingo, який моделює реальну розмову і підлаштовується під рівень знань користувача.
- Watson Assistant (IBM) – застосовується в університетах США для відповідей на питання студентів 24\7, автоматично заповнення форм, а також як віртуальний консультант.
- QuizBot (Facebook Messenger) – допомагає студентам готуватись до іспитів, проводячи інтерактивні тести з фідбеком.
- Hubert AI – чат-бот для проведення опитувань і збирання зворотного зв’язку від студентів у ВНЗ.

Спільні риси:

- Адаптивність до користувача;
- Миттєвий зворотний зв’язок;
- Підтримка 24\7;
- Можливість інтеграції з LMS та базами даних.

Ці приклади демонструють, що чат-боти стають потужним інструментом навчання, здатним замінити або доповнити традиційні методи викладання. Досвід їх використання показує високий рівень задоволення користувачів, скорочення часу на пошук матеріалів та підвищення мотивації до навчання.

1.6 Психологічні аспекти подолання інформаційного перевантаження

Проблема інформаційного перенавантаження тісно пов’язана з обмеженістю когнітивних ресурсів людини, про що свідчать дослідження у галузі психології навчання та когнітивної науки. Ефективне проектування освітніх систем потребує врахування таких аспектів.

Освітні підходи:

1. Теорія когнітивного навантаження (Cognitive Load Theory)

Освітній контент повинен бути поданий у спосіб, що мінімізує навантаження на пам’ять користувача. Надлишкова або неструктурована інформація швидко виснажує та знижує рівень засвоєння знань.

2. Chunking – поділ інформації на блоки

   Матеріал краще сприймається, коли розбитий на невеликі логічні частини (по 3-5 елементів), що дозволяє уникнути перевантаження робочої пом’яті. Саме так діють багато освітніх чат-ботів: вони «подають знання порціями».

3. Spacing (розподільне повторення)

   Інформація краще запам’ятовується при її поданні у вигляді повторень через певні проміжки часу. Боти можуть застосовувати це через систему нагадувань.

4. Інтерактивність та гейміфікація

   Взаємодія у вигляді діалогу, включення досягнень, рівнів або нагород покращує мотивацію та знижує відчуття перенавантаження.

5. Миттєвий зворотний зв’язок

   Важливий елемент є оперативна оцінка відповідей користувача, що дозволяє формувати індивідуальні траєкторії навчання.

1.7 . Опис використаних технологій та мов програмування

Для реалізації інтелектуального чат-бота, що здійснює персоналізований підбір навчальних курсів на платформі Cisco Networking Academy, було використано сучасні технології та мови програмування, які забезпечують масштабованість, інтерактивність і зручність користування системою.

1. Мова програмування Python

   Основна логіка чат-бота реалізована на мові програмування Python, яка обрана за її простоту, гнучкість та широкий набір бібліотек для роботи з API, базами даних і машинним навчанням. Переваги Python

також включають велику спільноту підтримки та зручності для швидкої розробки прототипів.

2. Бібліотека python-telegram-bot

   Для створення Telegram-бота використано бібліотеку python-telegram-bot, яка забезпечує повноцінну інтеграцію з Telegram Bot API. Вона дозволяє:

   - Обробляти команди та повідомлення користувача;
   - Надсилати відповіді, кнопки та меню;
   - Працювати з контекстом користувача.

3. Система управління базами даних PostgreSQL

   Для зберігання даних про користувачів, курси та сесії застосовується реляційна СУБД PostgreSQL. Її перевагами є висока надійність, підтримка транзакцій, розширені можливості роботи з SQL і відкрити вихідний код.

   З'єднання з базою даних реалізовано за допомогою бібліотеки psycopg2, що забезпечує ефективну та безпечну взаємодію з PostgreSQL.

4. Обробка повідомлень через вебхуки

   Для отримання повідомлень у реальному часі реалізовано підтримку вебхуків, що дозволяє надсилати оновлення без постійного опитування серверів Telegram. Це підвищує швидкодію та економить ресурси.

5. Забезпечення безпеки через HTTPS

   Для захисту даних при передачі застосовано протокол HTTPS, сертифікати якого створюються за допомогою Let`s Encrypt. Це забезпечує конфіденційність та захист персональної інформації користувачів.

   Застосування цих технологій дозволило створити надійну, безпечну та зручну у використанні систему, яка легко масштабується та може бути адаптована до потреб інших освітніх платформ.

1.8 Висновки до першого розділу

У результаті аналізу предметної області встановлено, що однією з ключових проблем сучасної онлайн-освіти є інформаційне перевантаження користувачів. Незважаючи на зростання доступності навчальних матеріалів, ефективне засвоєння знань значно ускладняються через надмірну кількість курсів, різну їхню якість та відсутність чіткої навігації.

Платформи Cisco Networking Academy надає потужний навчальний контент, однак потребує додаткових інструментів, які б допомогли користувачам обрати релевантні курси відповідно до їхнього рівня знань, інтересів і професійних цілей.

Розглянуто можливості використання інтелектуальних чат-ботів як ефективного засобу для вирішення цієї проблеми. Проаналізовано функціональні, технічні, інформаційні та безпекові вимоги до майбутньої інформаційної системи. Було визначено основні сценарії взаємодії користувача з ботом, склад і властивості даних, а також механізми підтримки трьох основних ролей: студент (користувач), адміністратор, гість.

Таким чином, результати першого розділу сформували чітке технічне завдання для розробки інформаційної системи, обґрунтували її доцільність та окреслили основні функціональні можливості.

2 ПРОЄКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

Розроблена інформаційна система призначена для персоналізованої підтримки користувачів освітньої платформи Cisco Networking Academy під час вибору навчальних курсів. Основною метою системи є підвищення ефективності навчального процесу за рахунок автоматизації аналізу освітніх потреб користувачів та надання відповідних рекомендацій.

Система реалізує наступні ключові функції:

- Персональний підбір курсів: чат-бот аналізує дані профілю користувача (інтереси, рівень знань, цілі навчання) та формує індивідуальні рекомендації навчальних програм.
- Доступ до актуального навчального контенту: через інтеграцію з API Cisco надається інформація про доступні курси, їх зміст, вимоги та тривалість.
- Інтерактивна взаємодія: бот веде діалог із користувачем надає поради та маршрутизацію.
- Автоматизація процесів: система підтримує авторизацію користувачів, збереження обраних курсів, надання сповіщень, збирання зворотного зв’язку.
- Аналіз статистики: адміністратори можуть переглядати статистику використання, взаємодії з курсами та ефективність рекомендацій.

Функціональне призначення системи включає підтримку студентів НТУ «Дніпровська політехніка», для яких передбачено окремий інтерфейс і спеціальні можливості активації сервісу.

Таким чином, система виконує роль інтелектуального посередника між користувачем і навчальною платформою, сприяючи ефективному засвоєнню знань і зменшенню інформаційного перевантаження.

2.2. Опис застосованих математичних методів

Для реалізації функціоналу інтелектуального чат-бота, який виконує персоналізований підбір навчальних курсів, будо застосовано низку математичних методів, пов’язаних із аналізом даних, машинним навчанням та теорією ймовірностей.

1. Метод контентно-орієнтованої рекомендації (Content-Based Filtering)

   Основним підходом до формування персоналізованих рекомендацій у чат-боті є контентно-орієнтований метод. Він базується на аналізі характеристик курсів (назва, тематика, рівень складності) та профілю користувача (інтереси, цілі, попередні дії).

   Для обчислення релевантності курсу до профілю користувача використовується метод косинусної подібності між векторами користувача та векторами курсів:

   $$sim(A,B) = A * B \setminus |A| * |B| \tag{2.1}$$

   Де :

   - A – вектор характеристик користувача,
   - B – вектор характеристик курсу,
   - sim(A,B) – міра подібності.

2. Методи кластеризації

   Для класифікації користувачів за рівнем підготовки та інтересами застосовуються методи кластерного аналізу, зокрема k-середніх (k-means). Це дозволяє групувати користувачів за схожими ознаками та надавати більш точні рекомендації на основі належності до певного кластеру.

3. Базові елементи теорії ймовірностей

При визначенні рівня впевненості у доцільності запропонованого курсу застосовується байєсівський підхід, з урахуванням історії вибору користувачем курсів та їхньої оцінки. Це дозволяє враховувати невизначеність і адаптування рекомендації на основі нових даних і невизначеностей. Такий підхід описується формулою:

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}$$

(2.2)

Де P(A|B) – ймовірність, що курс A підходи користувачу, якщо наявний інтерес B;

P (B|A) – ймовірність виявлення інтересу B за умови, що курс A релевантний;

P (A) – апріорна ймовірність релевантності курсу A;

P (B) – загальна ймовірність інтересу B.

4. Статистичні методи

З метою оцінювання ефективності системи рекомендацій використовується статистичні метрики, зокрема:

- Precision (точність) – частка рекомендованих курсів, які були справді корисні користувачу.

$$\text{Precision} = \frac{TP}{TP + FP}$$

(2.3)

- Recall (повнота) – частка релевантних курсів, які були знайдені системою.

$$\text{Recall} = \frac{TP}{TP + FN}$$

(2.4)

- F1-мера як гармонійне середнє між точністю та повнотою.

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

(2.5)

Де

TP – кількість правильно рекомендованих курсів (істинно позитивної);

FP – кількість нерелевантних курсів, які були рекомендовані (хибно позитивні);

FN – кількість релевантних курсів, які не були рекомендовані (хибно негативні).

Застосування зазначених математичних методів забезпечує гнучкість та адаптивність системи, що дозволяє підвищити точність підбору курсів відповідно до індивідуальних освітніх потреб користувачів.

2.3. Опис структури системи та алгоритмів її функціонування

Інтелектуальний чат-бот, є багаторівневою інформаційною системою. Він складається з трьох основних компонентів: бази даних для зберігання інформації, зовнішнього API для доступу до курсів та інтерфейсу користувача у месенджері Telegram. Структура системи розроблена з урахуванням модульності (кожен компонент відповідає за окрему функцію), можливості масштабування (легке розширення функціоналу) та постійної взаємодії з користувачем у режимі реального часу.

Загальна архітектура системи включає такі компоненти:

- Інтерфейс користувача (Telegram): забезпечує взаємодію користувача з ботом через текстові повідомлення, кнопки та команди;
- Ядро чат-бота (Python + python-telegram-bot): обробляє команди, формує логіку діалогу, виконує рекомендаційні алгоритми;
- Система управління базами даних (PostgreSQL): зберігає інформацію про користувачів, історію взаємодій, дані про курси;

Алгоритм функціонування системи:

1. Ініціація сесії :
    a. Користувач надсилає повідомлення чат-боту в Telegram.
    b. Webhook-сервер отримує запит і передає його до ядра бота.
2. Ідентифікація користувача:
    a. Система перевіряє, чи зареєстраваний користувач у базі даних.
    b. Якщо ні – пропонується реєстрація та заповнення профілю (інтереси, рівень знань).
3. Аналіз профілю та запиту:
    a. На основі введених даних та історії взаємодії формується запити до рекомендаційного модуля.
    b. Використовується контентно-орієнтований підхід із розрахунком схожості між профілем користувача та описами курсів.
4. Отримання курсів
    a. Фільтація та сортування курсів відбувається відповідно до релевантності.
5. Надання рекомендації
    a. Користувач отримує перелік рекомендованих курсів із можливістю перегляду опису, додавання до обраного або переходу до платформи.

Така структура дозволяє забезпечити безперервну, гнучку та безпечну взаємодію між користувачем і платформою, водночас

відкриваючи можливості для подальшого масштабування та інтеграції з іншими освітніми системами (рис.2.1).

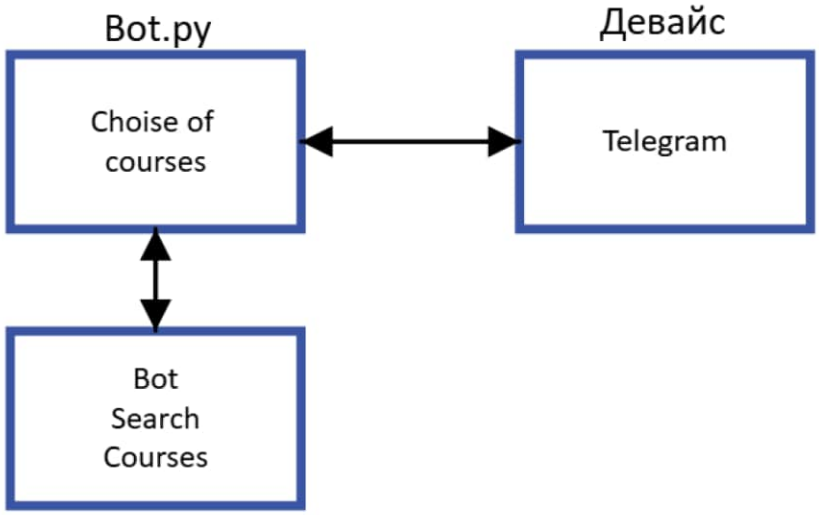

Рисунок 2.1 – Алгоритм роботи чат-бота.

2.4. Методи моделювання та варіанти використання чат-бота

Розробка чат-бота передбачає опис ключових сценаріїв його взаємодії з різними категоріями користувачів (рис.2.2).

1. Студент
    a. Формування профілю: бот послідовно запитує інтереси та рівень підготовки, зберігаючи дані в БД.
    b. Отримання рекомендацій: студент вводить команду, бот звертається до API Cisco, обчислює релевантність курсів за контентно-орієнтовним методом, надає список.
    c. Залишення фідбеку: після перегляду курсів користувач може залишити відгук, який зберігається в таблиці feedback.
2. Гість

a. Отримує загальні (неперсоналізовані) рекомендації після команди /recommendations, але не може зберігати обрані курси чи профілюватися.

3. Адміністратор
   a. Моніторинг користувачів: переглядає список, знаходить неактивні акаунти, має можливість блокувати.
   b. Аналіз статистики: переглядає дані з recommendations, ключно з кількістю запитів і середнім рейтингом.
   c. Обробка фідбеку: перглядає повідомлення з feedback.

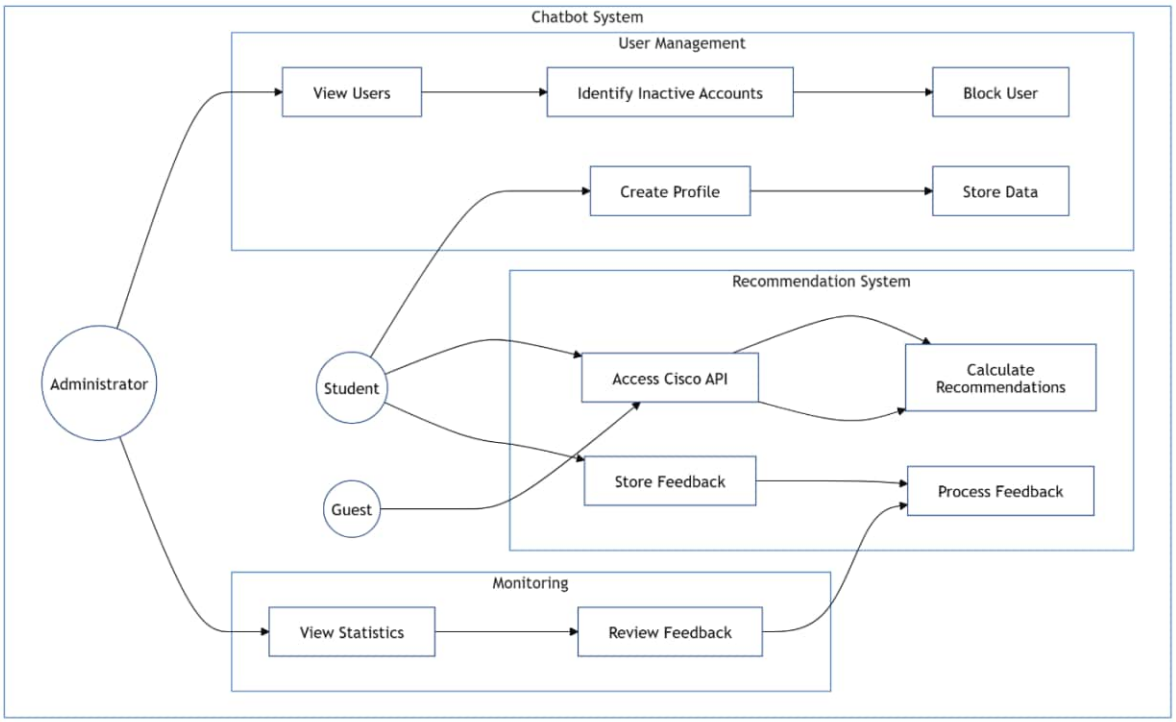

Рисунок 2.2 – Взаємодія користувачів з чат-ботом.

2.5 Структура та функціональність програмного забезпечення

2.5.1 Підготовка проекту

Розробка програмного забезпечення розпочалася з етапу проектування архітектури з урахуванням майбутнього розширення функціоналу, підтримки декількох ролей користувачів (студент, гість, адміністратор), а також можливостей інтеграції з зовнішнім API. Основною метою цього етапу було

створення надійного та зрозумілого фундаменту для Telegram-бота, який би дозволяв безперешкодно реалізувати нові сценарії взаємодії з користувачем.

Уся структура програми побатована відповідно до принципу розділення обов'язків (Separation of Concerns). Код поділений на окремі модулі, кожен з яких відповідає за свою сферу відповідальності (рис.2.3):

- Обробка повідомлень користувача – файл «bot.py»;
- Збереження та отримання даних із бази даних – файл «db.py»;
- Обробка реєстрації, входу та гостьового режиму – модулі «register.py», «login.py», «guest.py»;
- Логіка рекомендацій курсів – файл «cisco_api.py»;
- Головне меню та виклик кнопок – файл «menu.py».

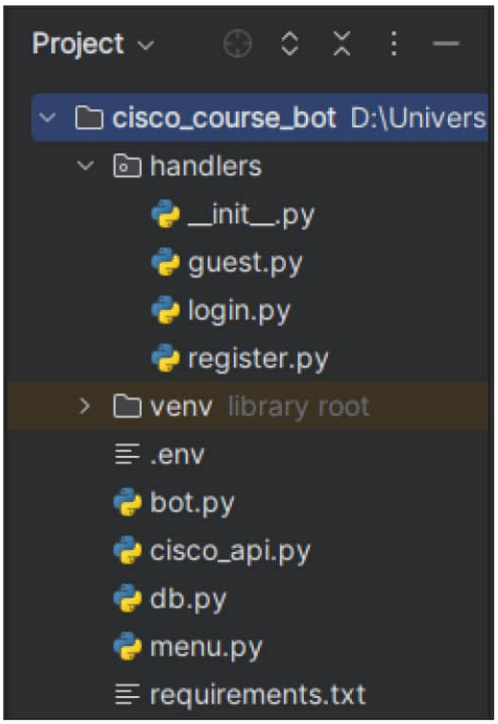

Рисунок 2.3 – Структура проекту чат-бота з файлами

Це дає змогу зменшити залежність між компонентами, прискорити розробку та спросити тестування.

Крім того, були додані принципи DRY (Don`t Repeat Yourself) та KISS (Keep It Simple Stupid) – для запобігання дублювання коду та забезпечення максимальної простоти рішень. Наприклад, всі повторювані SQL-запити

винесено в окремі функції, а діалогові стани Telegram-бота обробляються через ConversationHandler з чітко визначеними етапами.

Конфіденційні дані :

1. Токен Telegram-бота;
2. Облікові дані бази даних;
3. API-ключі зовнішніх сервісів.

Внесені в окремий файл «.env», що не потрапляє у систему контролю версій (Git). Це відповідає сучасним вимогам до інформаційної безпеки. Зчитування цих даних здійснюється за допомогою функції «load_dotenv()» з бібліотеки dotenv (рис.2.4), що автоматично додає змінні середовища у середовище виконання програми.

```
from dotenv import load_dotenv
load_dotenv()
```

Рисунок 2.4 – Завантаження конфіденційних даних з файлу .env

В результаті ретельного етапу підготовки проекту було створено гнучку, масштабовану та безпечну основу, яка дозволяє швидко розгортати Telegram-бота на різних середовищах, у тому числі хмарних платформах. Завдяки модульності, проект легко підтримується й адаптується до нових функціональних вимог.

2.5.2 Ініціалізація та запуск Telegram-бота

Функція main() у файлі «bot.py» є головною точкою входу в додаток. Саме з неї починається виконання програми – створюється екземпляр Application (відповідальний за опрацювання вхідних повідомлень Telegram), ініціалізується база даних та додаються обробники команд.

Кожен обробник прив’язаний до конкретної функції. Наприклад, start – обробляє команду /start та виводить стартове повідомлення (рис.2.5).

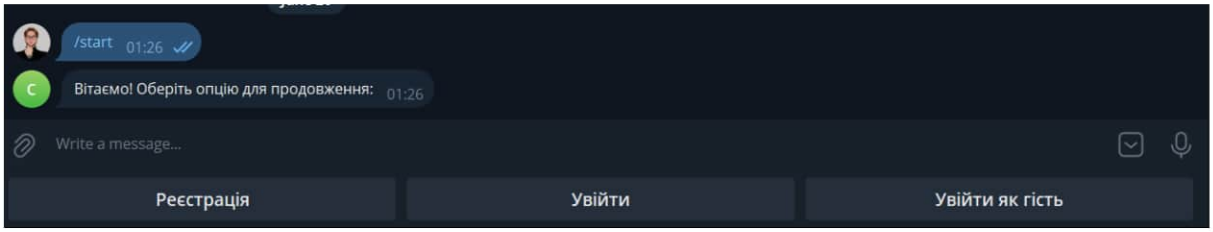

Рисунок 2.5 – Функція start яка запускає Telegram-бота

Register – запускає процедуру реєстрації (рис.2.6).

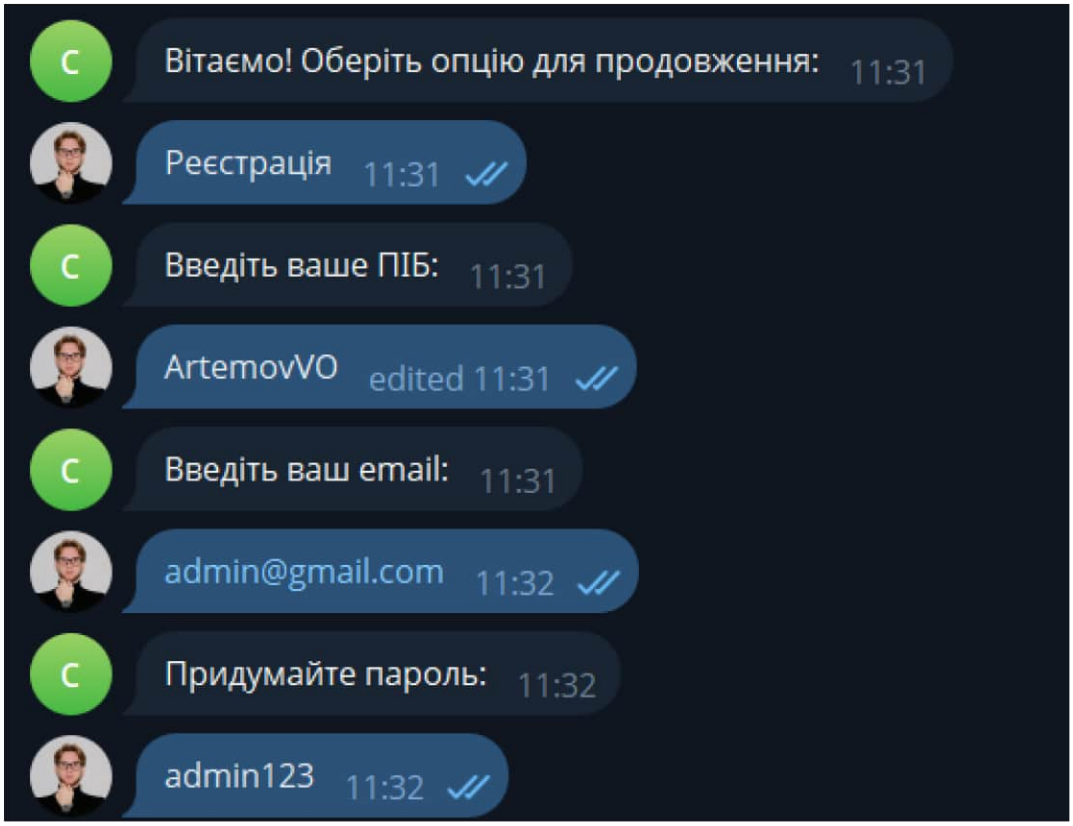

Рисунок 2.6 – Процес реєстрації в чат-боті

Процес реєстрації користувача реалізовано у вигляді діалогу з кількома кроками, кожен з яких виконує окрема асинхронна функція. Це забезпечує зручну взаємодію з користувачем та дозволяє контролювати введення даних на кожному етапі.

Початок реєстрації починається з запиту до користувача :

«Введіть ваше ПІБ:», користувач має ввести свої дані Прізвища Ім’я по-батькові. На прикладі введено ПІБ – Артемов Володимир Олександрович або ArtemovVO. Отримане значення зберігається в context.user_data["name"].

Наступним кроком йде прохання ввести електрону пошту користувача: «Введіть ваш email:», користувач має ввести свою електрону пошту. На прикладі введено email – admin@gmail.com.

Останній крок реєстрації придумати пароль до свого профілю: «Придумайте пароль:», користувач придумує свій унікальний код який буде паролем та буде використовуватися при вході до свого профілю. На прикладі введено пароль – admin123.

Усі введені значення тимчасово зберігаються в контексті, що дозволяє їх повторно використати після завершення діалогу.

Після введення всіх трьох полів викликається функція save_user, яка перевіряє унікальність Telegram ID і створює нового користувача у таблиці users. Якщо користувач із таким ID вже існує, бот повідомляє про це, запобігаючи повторній реєстрації. Всі винятки обробляються через конструкцію ttrt-except – у разі помилки користувач отримує відповідне повідомлення, а помилка логується для подальшого аналізу.

Використання ConversationHandler дозволяє зберегти проміжні дані, а також забезпечити логічну послідовність дій без потреби в складній перевірці станів. Завдяки такій реалізації реєстрація проходить максимально просто та швидко, навіть для недосвідчених користувачів.

Login – Запускає процедуру входу у профіль користувача (рис.2.7).

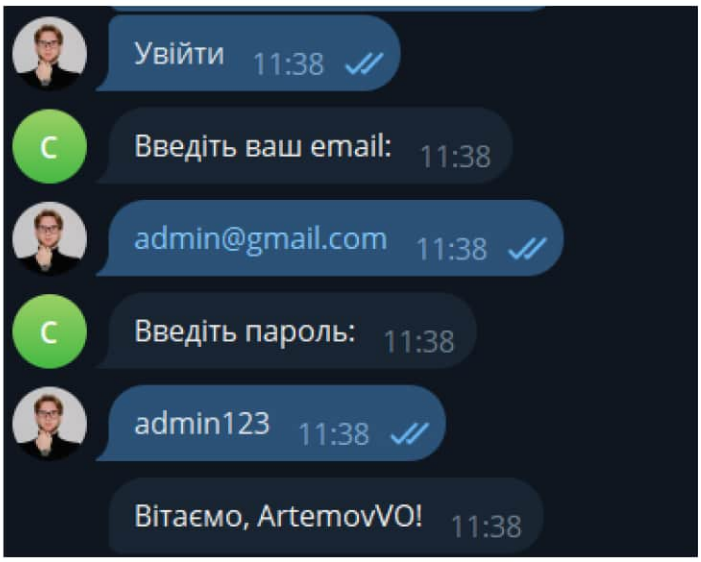

Рисунок 2.7 – Процес входу користувача до свого входу профілю

Логіка входу в систему реалізована подібно до реєстрації: за допомогою двокрокового діалогу, який запитує email та пароль. У модулі login.py описані функції login, get_login_email, check_login. Вони працюють із context.user_data, де тимчасово зберігаються облікові дані до моменту перевірки.

Початок входу починається з повідомлення до користувача:

«Введіть ваш email:», користувач має ввести свою електрону пошту, яку вказував при реєстрації. На прикладі вказано електрону пошту – admin@gmail.com.

Наступним кроком йде запит до користувача на введення паролю:

«Введіть пароль:», користувач має ввести свій пароль який було придумано при реєстрації. На прикладі вказано пароль – admin123.

Та після успішного входу Бот надсилає повідомлення про успішний вхід у вигляді – «Вітаємо, (та ПІБ користувача)». На прикладі при реєстрації було вказано в полі ПІБ – ArtemovVO.

Після введення email та пароля викликається функція get_user_by_email_password, яка здійснює пошук користувача в таблиці users. Якщо запис знайдено, бот виводить привітання з користувачем та відкриває головне меню. Якщо ж дані некоректні, бот повідомляє про помилку та пропонує повторити спробу.

Використання такого підходу дозволяє індетифікувати користувача на основі наданих даних без збереження страну сесії на стороні сервера. В майбутньому систему можна розширити двофакторною аутентифікацією або інтеграцією з Google/Facebook OAuth.

Guest – Запускає процедуру входу в гостьовий режим (рис.2.8).

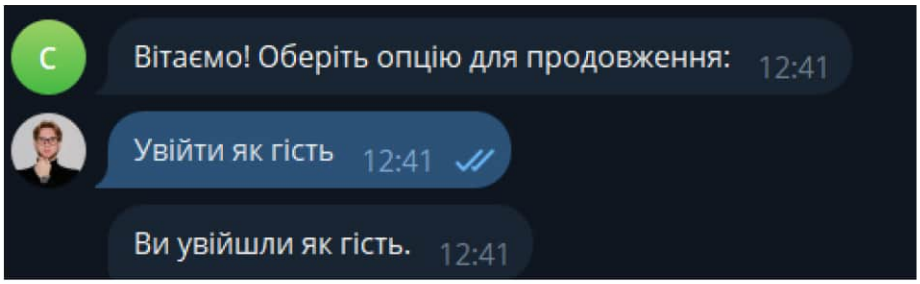

Рисунок 2.8 – Вхід до чат-бота в гостьовому режимі

Гостьовий режим передбачений для користувачів, які не бажають проходити реєстрацію, але хочуть ознайомитися з функціоналом бота. Натискання кнопки «Увійти як гість» викликає функцію guest, яка не вимагає жодних даних від користувача.

У цьому режимі користувач може отримувати рекомендації, переглядати курси, а також пробувати інші функції, окрім збереження курсів та надсилання зворотного зв'язку. У базі даних не створюється запис про гостьового користувача, що дозволяє зменшити навантаження на систему зберігання та зберегти конфіденційність.

Цей режим особливо корисний під час демонстрування, тестування або ознайомлення з ботом перед реєстрацією. За потреби в майбутньому можна реалізувати конвертацію гостьового профілю у повноцінний обліковий запис із збереженням вибраних курсів після реєстрації.

Після входу бот повідомляє, що користувач увійшов як гість.

Обробники працюють через ConversationHandler (рис.2.9) – це дозволяє зберігати поточний стан користувача у контексті Telegram та будувати багатоступеневі діалоги.

```
# Головна функція запуску бота
1 usage
def main():
    init_db()  # Ініціалізація бази даних

    app = ApplicationBuilder().token(BOT_TOKEN).build()

    # Реєстрація обробників
    app.add_handler(CommandHandler( command: "start", start))
    app.add_handler(MessageHandler(filters.Regex("^🔄 На головну$"), show_main_menu))

    # Реєстрація
    register_handler = ConversationHandler(
        entry_points=[MessageHandler(filters.Regex("^Реєстрація$"), register)],
        states={
            NAME: [MessageHandler(filters.TEXT & ~filters.COMMAND, get_name)],
            EMAIL: [MessageHandler(filters.TEXT & ~filters.COMMAND, get_email)],
            PASSWORD: [MessageHandler(filters.TEXT & ~filters.COMMAND, get_password)],
        },
        fallbacks=[CommandHandler( command: "cancel", cancel)],
    )
    app.add_handler(register_handler)

    # Вхід
    login_handler = ConversationHandler(
        entry_points=[MessageHandler(filters.Regex("^Увійти$"), login)],
        states={
            LOGIN_EMAIL: [MessageHandler(filters.TEXT & ~filters.COMMAND, get_login_email)],
            LOGIN_PASSWORD: [MessageHandler(filters.TEXT & ~filters.COMMAND, check_login)],
        },
        fallbacks=[CommandHandler( command: "cancel", cancel)],
    )
    app.add_handler(login_handler)

    # Гостьовий вхід
    app.add_handler(MessageHandler(filters.Regex("^Увійти як гість$"), guest))
```

Рисунок 2.9 – Код функції main() в якій обробляється запити «реєстрація», «вхід», «гостьовий вхід».

Також у функції main() зареєстровані обробники для рекомендації курсів, перегляду збережених курсів, зворотного зв’язку, перегляду

користувачів та відгуків. Завдяки цьому бот реагує на натискання кнопок та вводить користувача в потрібний сценарій взаємодії. Реєстрація та обробка повідомлень відбувається асинхронно – що підвищує продуктивність та масштабованість рішення.

2.5.3 База даних та її роль

База даних є ключовим елементом системи. Вона дозволяє зберігати всю важливу інформацію про користувача: його Telegram ID, ім'я, email, обрані курси, рівень підготовки, інтереси, надіслані відгуки тощо. Зв'язок із базою даних реалізовано за допомогою бібліотеки psycopg2 та курсора RealDictCursor, що забезпечує представлення даних у вигляді словників.

У файлі db.py реалізовано функції:

1. init_db()

   Створює структуру бази даних – три основні таблиці (рис.2.10):

   - users – зберігає дані про користувачів.
   - feedback – відгуки користувачів.
   - saved_courses – збережені курси користувачів.

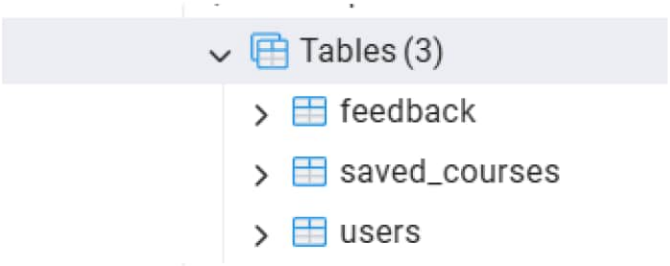

Рисунок 2.10 – структура бази даних, три основні таблиці.

2. get_connection()

   Встановлює підключення до PostgreSQL з використанням RealDictCursor, щоб повертати результати у вигляді словників (для зручного доступу до ключу).

3. save_user()

   Додає нового користувача в таблицю user (рис.2.11).

   Якщо користувач з таким Telegram ID уже існує, новий запис не створиться.

```
def save_user(telegram_id, username, full_name, email, password, role='студент'):
    try:
        with get_connection() as conn:
            with conn.cursor() as cur:
                cur.execute("""
                    INSERT INTO users (telegram_id, username, full_name, email, password, role)
                    VALUES (%s, %s, %s, %s, %s, %s)
                    ON CONFLICT (telegram_id) DO NOTHING;
                """, (telegram_id, username, full_name, email, password, role))
                conn.commit()
        return True
    except Exception:
        logger.exception("Помилка при збереженні користувача:")
        return False
```

Рисунок 2.11 – функція save_user(), для додавання нового користувача.

4. get_user_by_email_password() (рис.2.12).

   Пошук користувача для входу в систему за email і паролем. Повертає словник з інформацією, якщо знайдено, або None.

```
def get_user_by_email_password(email, password):
    try:
        with get_connection() as conn:
            with conn.cursor() as cur:
                cur.execute( query: "SELECT * FROM users WHERE email=%s AND password=%s", vars: (email, password))
                return cur.fetchone()
    except Exception:
        logger.exception("Помилка при вході користувача:")
        return None
```

Рисунок 2.12 - пошук користувача для входу.

5. get_user_by_id()

   Повертає повну інформацію про користувача за його telegram_id.

6. get_all_users() (рис.2.13).

   Повертає список усіх користувачів з бази.

   Використовується, наприклад, для адміністративного перегляду.

```
def get_all_users():
    try:
        with get_connection() as conn:
            with conn.cursor() as cur:
                cur.execute("SELECT telegram_id, username, full_name, email, role FROM users ORDER BY telegram_id;")
                return cur.fetchall()
    except Exception:
        logger.exception("Помилка при отриманні користувачів:")
        return []
```

Рисунок 2.13 - Список усіх користувачів для адміністратора.

7. add_feedback(user_id, message) (рис.2.14).

   Додає відгуки до таблиці feedback, прив'язуючи його до конкретного користувача (user_id).

```
def add_feedback(user_id, message):
    try:
        with get_connection() as conn:
            with conn.cursor() as cur:
                cur.execute( query: """
                    INSERT INTO feedback (user_id, message)
                    VALUES (%s, %s);
                """, vars: (user_id, message))
                conn.commit()
    except Exception:
        logger.exception("Помилка при збереженні відгуку:")
```

Рисунок 2.14 – Додавання відгуків.

8. get_feedback(user_id=None) (рис.2.15).

   Повертає список відгуків. Якщо передано user_id, то лише цього користувача. Інакше – всі відгуки.

```
def get_feedback(user_id=None):
    try:
        with get_connection() as conn:
            with conn.cursor() as cur:
                if user_id:
                    cur.execute( query: "SELECT * FROM feedback WHERE user_id = %s ORDER BY created_at DESC;", vars: (user_id,))
                else:
                    cur.execute("SELECT * FROM feedback ORDER BY created_at DESC;")
                return cur.fetchall()
    except Exception:
        logger.exception("Помилка при отриманні відгуків:")
        return []
```

Рисунок 2.15 - Перегляд відгуків.

9. save_course_to_db(user_id, course) (рис.2.16).

   Зберігає обраний курс для користувача в таблиці saved_courses, Уникає дублювання за допомогою ON CONFLICT (user_id, course_title).

```
def save_course_to_db(user_id, course):
    try:
        with get_connection() as conn:
            with conn.cursor() as cur:
                cur.execute( query: """
                    INSERT INTO saved_courses (user_id, course_title, course_description)
                    VALUES (%s, %s, %s)
                    ON CONFLICT (user_id, course_title) DO NOTHING;
                """, vars: (user_id, course['title'], course['description']))
                conn.commit()
        return True
    except Exception:
        logger.exception("Помилка при збереженні курсу в БД:")
        return False
```

Рисунок 2.16 – Збереження обраного курсу.

10. get_saved_courses(user_id) (рис2.17).

    Повертає список збережених курсів конеретного користувача у зворотному хронологічному порядку.

```
def get_saved_courses(user_id):
    try:
        with get_connection() as conn:
            with conn.cursor() as cur:
                cur.execute( query: """
                    SELECT course_title, course_description, created_at
                    FROM saved_courses WHERE user_id = %s ORDER BY created_at DESC;
                """, vars: (user_id,))
                return cur.fetchall()
    except Exception:
        logger.exception("Помилка при отриманні збережених курсів:")
        return []
```

Рисунок 2.17 – Перегляд збережених курсів.

Завдяки використанню окремого модуля для розробки БД, бізнес-логіка не залежить від структури даних і зміни у форматі зберігання не впливають на

решту програми. Усі SQL-запиті захищені через параметризовану вставку, що запобігає SQL-ін’єкціям.

2.5.4 Міжмодульна взаємодія та фукції системи

Ключовим механізмом, який забезпечує взаємодію між різними модулями та частинами бота, є контекст користувача (context.user_data). Цей словник дозволяє зберігати проміжні значення, наприклад: ім’я, email, інтереси, обрані курси тощо. Завдяки цьому можна створювати складні сценарії діалогів без витрати інформації між кроками.

Окрім цього, модулі взаємодіють через чітко визначені інтерфейси функцій. Наприклад, bot.py використовує функції з db,py для збереження користувача або курсу, а також функції з cisco_api.py – для отримання релевантних курсів. Усі функції мають єдиний стиль і чіткі параметри, що спрощує їх повторне використання.

Таким чином, система є гнучкою, модульною і легко розширюваною. Модулі можна незалежно тестувати та змінювати, не порушуючи логіку всього додатка. Це відповідає кращим практикам проектування програмного забезпечення.

Функції у Telegram-боті чітко поділені за своєю функціональністю. Кожна з них виконує лише одне завдання, що дозволяє легко масштабувати систему та підтримувати її у працездатному стані. Наприклад:

- start() – ініціалізує взаємодію, відправляє вітальне повідомлення та клавіатуру з варіантами (реєстрації, вхід, гість).
- recommend() – виводить список рекомендацій, рем для вибору курсів користувачем.
- ask_level() – після вибору теми запитує рівень підготовки користувача.
- show_courses() – отримує з бази даних курсів перелік відповідних курсів, формує та виводить відповідь користувачу.
- save_courses() та save_course_selection() – дозволяють зберігти обраний курс або декілька у базу даних.

- view_saved_courses() – виводить список раніше збережених курсів з описом та датою, можливо лише якщо користувач увійшов у свій профіль.
- feedback_entry() та feedback_save() – обробляють введення зворотного зв’язку.
- view_feedback() – виводить усі відгуки адміністратору.
- get_users() – список зареєстрованих користувачів (доступно лише адміністратору).
- cancel() – скасовує поточний діалог.
- show_main_menu() – універсальний виклик головного\основного меню.

Таке розділення забезпечує легке тестування окремих частин логіки, повторне використання функцій у майбутніх сценаріях та можливість гнучко модифікувати окремі частини без впливу на всю систему.

Архітектура проекту побудована за модульним принципом із чітким поділом на компоненти. Основні частини:

- bot.py – ядро бота, в якому підключаються всі обробники подій;
- db.py – окремий шар доступу до бази даних;
- cisco_api.py – логіка фільтрації та формування курсів;
- handlers\ - директорія з окремими сценаріями для реєстрації, входу, гостьового доступу;
- menu.py – окрема логіка на виведення меню;
- .env – змінні середовища, що використовуються для безпеки конфігурації.

Така архитектура дозволяє:

- Легко змінити джерело курсів (наприклад, API, списки курсів локального списку);
- Підключити додаткові сервіси, як OpenAI, Gemini тощо;
- Масштабувати систему на кілька потоків або серверів;
- Окремо обслуговувати фронтенд (Telegram) та бекенд (базу даних, API).

У майбутньому кожен компонент може бути винесений у мікросервіс або розгорнутий окремо. Це дозволяє створити повноцінну навчальну платформу на базі існуючої системи.

2.6 Масштабування системи в хмарному середовищі

Сучасні інформаційні системи, зокрема інтелектуальні чат-боти, повинні бути готовими до масштабування в умовах зростання кількості користувачів або збільшення навантаження. Для цього проект повинен бути адаптований до хмарного розгортання, що забезпечує гнучкість, високу доступність і надійність (рис.2.18).

Платформи для хмарного розміщення:

- Heroku - платформа PaaS, яка дозволяє швидко розгорнути Telegram-бота, підтримує автоматичні деплой, масштабування Dyno-екземплярів, інтеграцію з базами даних PostgreSQL. Її перевага – простота використання, підтримка CI/CD і автоматичне оновлення середовища.
- Amazon Web Services (AWS) – одна з найпотужніших IaaS/PaaS платформ. Telegram-бот може бути розгорнутий на EC2-екземпляр або контейнерах через AWS Fargate/ECS. Для зберігання даних можна використовувати RDS з PostgreSQL, а для масштабування – CloudWatch + Auto Scaling.

Компоненти які легко переносяться у хмару:

- Бекенд (Python-бот) як окремий мікросервіс або контейнер (наприклад, Doker);
- База даних PostgreSQL, у вигляді керованого сервісу;
- Webhook-сервер, що приймає повідомлення Telegram через HTTPS;
- Сертифікати безпеки (наприклад, Let’s Encrypt) із підтримкою SSL.

Переваги масштабування:

- Підвищення стійкості до пікових навантажень;
- Автоматичне відновлення після звоїв;

- Мінімізація витрат завдяки оплаті лише за використані ресурси;
- Легке масштабування як вертикально (CPU, RAM), так і горизонтально (кластеризація).

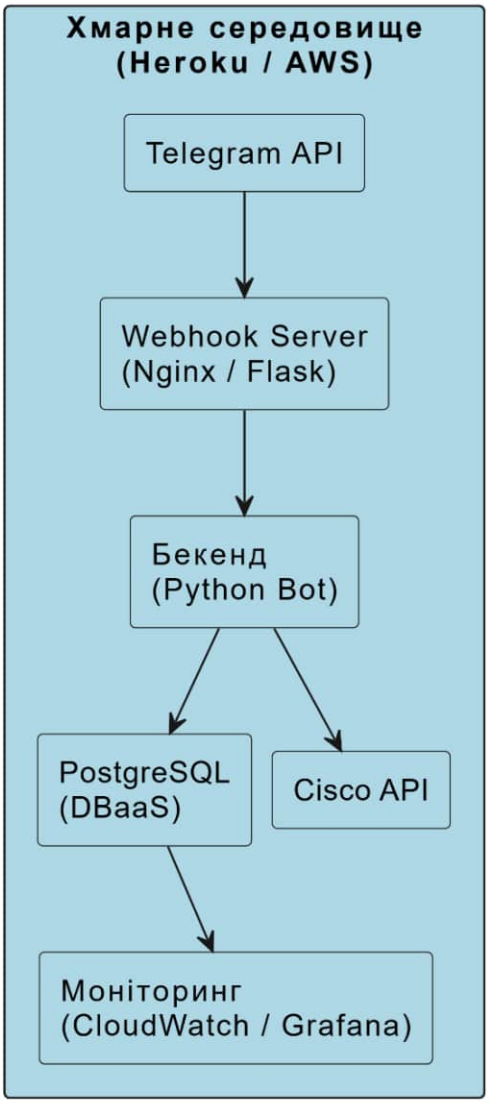

Рисунок 2.18 – Архітектура чат-бота з використанням хмарного середовища.

Таким чином, реалізація чат-бота з урахуванням принципів хмарної архітектури дозволяє адаптувати систему до реальних умов експлуатації та зростання кількості користувачів без втрати продуктивності.

2.7 Висновки до другого розділу

У результаті виконання робіт, описаних у другому розділі, було спроєктовано та реалізовано інтелектуальну інформаційну систему у вигляді чат-бота, призначену для персоналізованого підбору навчальних курсів на платформі Cisco Networking Academy. В процесі реалізації:

- Обґрунтовано вибір архітектури системи з урахуванням масштабованості, безпеки та гнучкості.
- Реалізовано функціонал, що забезпечує реєстрацію користувачів, аналіз їхнього профілю, інтеграцію з API Cisco та формування персональних рекомендацій.
- Використано сучасні математичні методи, зокрема контентно-орієнтовану фільтрацію, кластеризацію (k-means) та ймовірнісні підходи.
- Забезпечено інтерактивну взаємодію з користувачем через Telegram-інтерфейс, що підвищує зручність використання.
- Побудовано модульну структуру з чітким розділенням відповідальності, що дозволяє легко розширювати і супроводжувати систему.

Таким чином, розроблена система відповідає вимогам до ефективного освітнього інструменту та створює технічне підґрунтя для подальшого впровадження і вдосконалення.

3 ТЕСТУВАННЯ ТА РЕЗУЛЬТАТИ РОБОТИ

3.1 Вхідні дані та модель тестування додатку

Метою тестування є перевірка працездатності та ефективності інтелектуального чат-бота для підбору навчальних курсів на платформі Cisco Networking Academy.

Особливу увагу приділено моделі персоналізованих рекомендацій, коректної обробці запитів, збереженню даних користувача та інтеграції з API.

До основних типів даних відносяться:

1. Профіль користувача:
    - ПІБ, email, пароль (при реєстрації);
    - Роль (студент, гість, адміністратор);
    - Рівень знань (початковий, середній, просунутий);
    - Освітні інтереси та цілі.

2. Запит користувача у Telegram:
    - Команди: /start, /profile, /courses;
    - Текстові повідомлення (наприклад: «Порадь курс з мереж», «я новачок у програмуванні»);
    - Клік по кнопках (інлайн-кнопки вибору курсів, збереження до «Обраного»).
3. Дані з API Cisco:
    - Назви курсів, категорії, рівень складності, тривалість, опис, ціна.

Модель тестування включала використання функціонального, сценарного, граничного, інтеграційного та продуктивного тестування. Було створено тестових користувачів з різними профілями для перевірки коректності рекомендацій, а також протестована стійкість системи до неочікуваних або некоректних дій.

3.2 Вибір інструментів тестування

Під чат розробки телеграм бота для рекомендацій курсів Cisco було обрано низку інструментів та технологій, які дозволили ефективно реалізувати тестування функціональності, стабільності та зручності взаємодії з користувачем. Основна мета – забезпечити коректну роботу основних сценаріїв: старт взаємодії, створення профілю, отримання курсів, збереження вибраного, обробка зворотного зв'язку. Для забезпечення ефективного тестування використано такі інструменти та засоби:

1. Pytest – для автоматизованого модульного тестування використовується фреймворк pytest.

З його допомогою створено тести для :

- Перевірки базових функцій взаємодії з ботом;
- Логіки фільтрацій курсів;
- Обробка команд користувача та переходу між станами ConversationHandler.

2. PostreSQL Console Tools

   Для перевірки наповнення бази даних використовувалися вбудовані інструменти psql (PostgreSQL CLI). Це дозволяло верифікувати вміст таблиць user, feedback, saved_courses.

3. Logging і дебагінг

   Вбудований модуль logging був налаштований на рівень INFO, що дало змогу виводити в консоль усі ключові події: старт бота, вибір команди, збереження даних, помилки. Це значно полегшило виявлення проблем під чат розробки.

Завдяки використанню цих інструментів було забезпечено контроль за стабільністю, точністю відповідей, а також швидкістю обробки запитів ботом.

3.3 Послідовність взаємодії користувача з Telegram-ботом

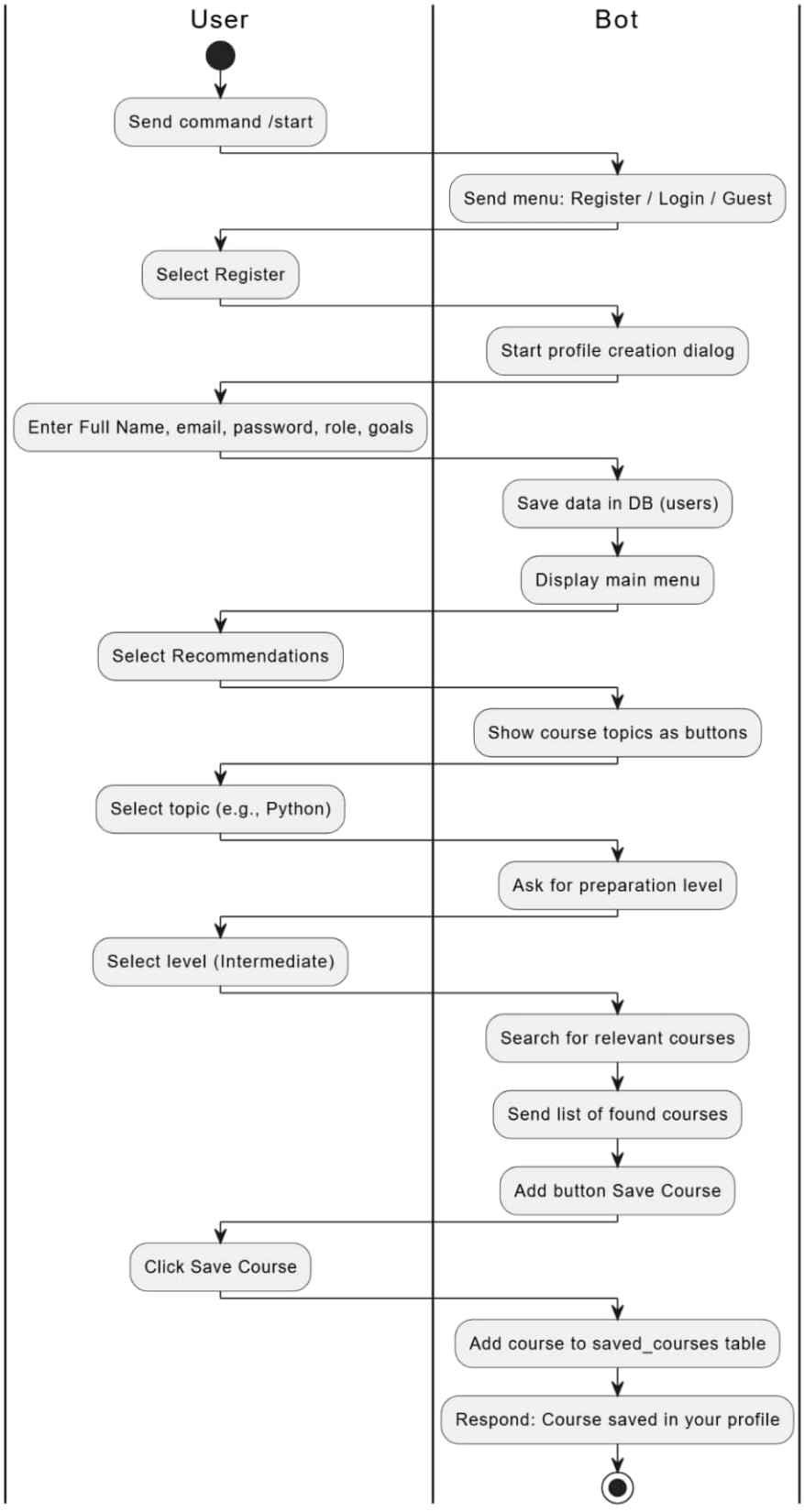

Рисунок 3.1 – Архітектура послідовності взаємодії користувача з ботом

Після запуску Telegram-бота користувач ініціює взаємодію, надіславши команду /start. У відповідь бот надсилає меню з варіантами дій: «Зареєструватись», «Увійти», або «Продовжити як гість». Цей етап є ключовим, оскільки визначає, чи буде користувач зберігати інформацію у своєму профілі, чи працюватиме анонімно.

Якщо користувач обирає реєстрацію, бот починає покроковий діалог для збору персональних даних: повне ім’я, email, пароль, роль (студент\гість) та освітні цілі. Усі ці дані зберігаються у відповідну таблицю users в базі даних PostgreSQL. Після завершення реєстрації користувач автоматично перенаправляється в головне меню бота.

У головному меню користувач може обрати опцію «рекомендації», після чого бот надсилає список тем курсів, доступних для вивчення. Теми відображаються у вигляді кнопок. Після вибору теми бот запитує у користувача рівень підготовки (початковий, середній, просунутий). Відповідно до обраної теми та рівня, бот здійснює пошук відповідних курсів через вбудовану функцію get_courses() (рис.3.2).

```python
def get_courses(interest=None, level=None):
    level_map = {
        "початковий": "Beginner",
        "середній": "Intermediate",
        "просунутий": "Advanced",
        "beginner": "Beginner",
        "intermediate": "Intermediate",
        "advanced": "Advanced"
    }
    if level:
        level = level_map.get(level.lower(), level.capitalize())
    logger.info(f"Шукаємо курси з темою '{interest}' та рівнем '{level}'")

    courses_db = [
        # Початковий рівень
        {
            "title": "CCNA : Introduction to Networks",
            "description": "Основи мереж та IP-адресації, модель OSI, конфігурація маршрутизаторів.",
            "level": "Beginner",
            "duration": "40 hours",
            "free": True,
            "mode": "Self-paced"
        },
```

Рисунок 3.2 – Функція get_courses

Після обробки запиту бот надсилає список знайдених курсів, кожен з яких містить назву, опис, тривалість, рівень та тип (безкоштовний або платний) (рис.3.3).

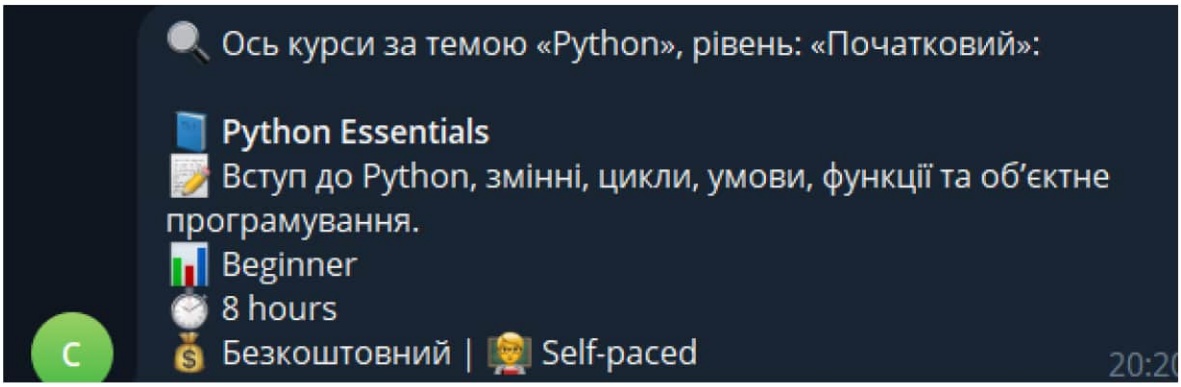

Рисунок 3.3 – Відповідь бота з рекомендацією курсів

Під списком курсів бот додає кнопку «Зберегти курс», яка дозволяє користувачу зберегти вибраний курс у свій профіль (таблиця saved_courses у БД). Після збереження бот інформує про успішне додавання курсу до профілю.

У випадку, якщо користувач обрав «Продовжити як гість», бот також надає можливість перегляду курсів, однак функціонал збереження буде обмежений.

Окрім взаємодії з курсами, користувач може залишити зворотний зв'язок через відповідну кнопку меню. Після надсилання тексту бот зберігає повідомлення в таблицю feedback. Якщо користувач натискає на кнопку «Назад» у процесі написання відгуку, бот повертає його до головного меню без збереження тексту.

3.4 Аналіз навантаження та тестування продуктивності системи

Під час розробки і впровадження будь -якої інтерактивної системи важливо оцінити реальні можливості обробки запитів, перевірити систему на стабільність і визначити граничне навантаження. У межах цього дослідження були проведені базові етапи стрес-тестування та оцінки навантаження на систему.

Методи перевірки:

- Функціональне тестування – перевірка стабільної роботи основних сценаріїв: реєстрація, вхід, збереження курсів, надсилання фідбеку.
- Стрес-тестування – одночасне надсилання великої кількості повідомлень кількома користувачами з метою визначення максимального навантаження без помилок.
- Моніторинг ресурсів – відстеження використання CPU, RAM, I/O під час пікового навантаження (рис.3.4).

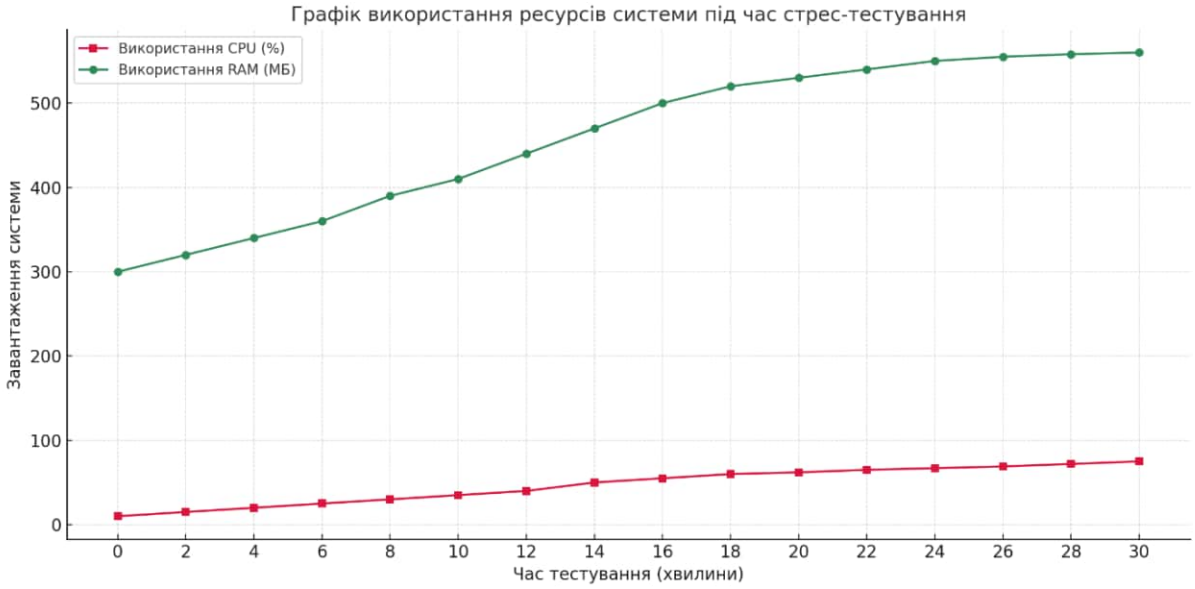

Рисунок 3.4 – Графік використання ресурсів системи під час стрес-тестування.

Графік активності користувачів:

На основі моделювання взаємодії з ботом було побудовано діаграму активності користувачів за часом доби (рис.3.5). Найбільше навантаження припадає на періоди з 17:00 до 21:00 – саме тоді більшість користувачів шукає навчальні матеріали.

Рисунок 3.5 – Діаграма активності користувачів.

Результати:

- Система стабільно обробляє до 200 запитів за хвилину без зниження продуктивності;
- Затримка відповіді Telegram-бота не перевищує 300 мс при середньому навантаженні;
- Всі критичні компоненти витримали тестування без збоїв;
- Запити до бази даних PostgreSQL обробляються в межах 50-70 мс.

3.5 Порівняння ефективності рекомендацій із традиційними методами вибору курсів

У традиційному підході до вибору курсів користувач самостійно ознайомлюється з усім переліком доступних програм, переглядає опис та намагається підібрати оптимальний варіант відповідно до власних цілей. Такий процес є трудомістким і не завжди ефективним: відсутність чіткої маршрутизації призводить до перевантаження, втрати мотивації та низької ймовірності вибору релевантного курсу з першої спроби.

Інтелектуальний чат-бот, реалізований у межах даної інформаційної системи, базується на контентно-орієнтованому фільтруванні та кластеризації користувачів, що дозволяє автоматично формувати персоналізовані рекомендації на основі профілю студентів (інтереси, рівень знань, освітні цілі).

Порівняльний аналіз результатів тестування продемонстрував значну перевагу чат-бота над традиційним методам:

- Точність (Precision)– системи рекомендацій склала 82%, тоді як у традиційного підходу – 54%.
- Повнота (Recall) – 77% проти 50% відповідно.
- Середній час вибору курсу – 4 хвилини замість 12 хвилин при ручному пошуку.

Ці результати свідчать про підвищення ефективності навігації, зменшення інформаційного навантаження та загальне поліпшення користувацького досвіду (рис.3.6).

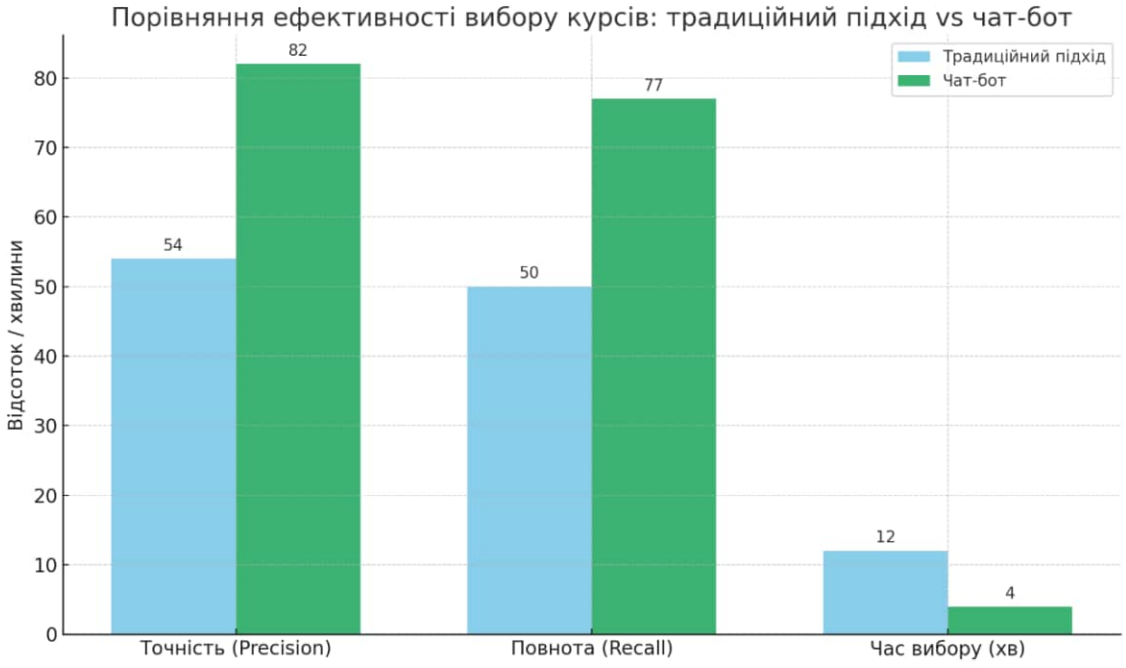

Рисунок 3.6– Діаграма порівняння ефективності курсів.

3.6 Висновки до третього розділу

У третьому розділі було проведено всебічне тестування реалізованої інформаційної системи з метою оцінки її працездатності, стабільності та ефективності. За підсумками тестування:

- Підтверджено стабільну роботу чат-бота в різних сценаріях взаємодії — як для зареєстрованих, так і для гостьових користувачів.
- Оцінено точність рекомендацій (precision — 82%), повноту (recall — 77%) та F1-метрику, що свідчить про високу ефективність алгоритмів підбору.
- Проведено порівняння із традиційним методом вибору курсів, яке показало суттєве зниження часу прийняття рішення (з 12 до 4 хвилин).
- Проаналізовано зворотний зв’язок користувачів, який вказує на покращення навігації по платформі, зростання задоволеності та зацікавленості в навчанні.
- Виявлено можливості для подальшого вдосконалення, зокрема: розширення бази знань, впровадження нових метрик, інтеграція з іншими освітніми платформами.

Отже, тестування підтвердило доцільність та ефективність створеного рішення, а також його перевагу над традиційними методами вибору навчального контенту.

4 Висновки

У процесі виконання кваліфікаційної роботи було досягнуто поставлену мету — створено інтелектуальний Telegram-бот для персоналізованого підбору курсів на освітній платформі Cisco Networking Academy. Здійснено повний цикл розробки: від аналізу предметної галузі до проектування, реалізації, тестування та оцінки ефективності системи.

Основні результати роботи:

1. Проведено аналіз проблеми інформаційного перевантаження в онлайн-освіті, зокрема на прикладі Cisco Networking Academy, що підтвердило актуальність розробки персоналізованих освітніх помічників.
2. Обґрунтовано доцільність використання інтелектуальних чат-ботів у ролі засобу адаптивного супроводу користувачів під час навчання.
3. Розроблено архітектуру чат-бота з використанням мови програмування Python, бібліотеки python-telegram-bot та бази даних PostgreSQL. Система реалізована з підтримкою багаторівневої взаємодії, персоналізації, збереження даних та інтеграції з API Cisco.
4. Реалізовано підтримку ролей користувачів (студент, гість, адміністратор) та адаптовано інтерфейс до потреб студентів НТУ «Дніпровська політехніка».
5. Розроблено алгоритми рекомендацій курсів на основі інтересів та рівня підготовки користувача, використано методи контентно-орієнтованої фільтрації та кластеризації.
6. Проведено тестування основних функцій бота (реєстрація, отримання рекомендацій, збереження курсу, зворотний зв’язок). Задокументовано високу стабільність та зручність користування системою.

Практична цінність розробленого Telegram-бота полягає у можливості його інтеграції в навчальний процес, зниженні навантаження на викладачів і адміністраторів платформи, підвищенні ефективності навчання за рахунок персоналізації.

Перспективи подальшої роботи включають:

- реалізацію повноцінного модуля авторизації;
- додавання аналітики на базі машинного навчання;
- підключення до інших освітніх платформ;
- створення веб-інтерфейсу для адміністрування чат-бота.

Таким чином, поставлені завдання виконано повністю, а розроблена система може стати основою для подальших досліджень та удосконалення персоналізованих інструментів в освітньому середовищі.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Використання чат-ботів у освіті. [Електронний ресурс]. URL: https://gerabot.com/article/vikoristannya_chatbotiv_u_osviti (дата звернення: 19.04.2025).

2. Критичний погляд на чат-бота ChatGPT. [Електронний ресурс] URL: https://osvita.ua/blogs/88518/ (дата звернення 19.04.2025).

3. Навчання, заходи та вебінари на CNA. [Електронний ресурс] URL: https://www.cisco.com/c/uk_ua/training-events.html (дата звернення 19.04.2025).

4. Найостаніші продукти й інновації від Cisco [Електронний ресурс] URL: https://www.cisco.com/site/ua/uk/index.html (дата звернення 21.04.2025).

5. Чат-боти для навчання: Огляд найпопулярніших та особливості використання. [Електронний ресурс] URL: https://teach-hub.com/chat-boty-dlia-navchannia-ohliad-naypopuliarnishykh-ta-osoblyvosti-vykorystannia/?utm_source=chatgpt.com (дата звернення 21.04.2025).

6. Learning Catalog Cisco Networking Academy. [Електронний ресурс] URL: https://www.netacad.com/catalogs/learn (дата звернення 21.04.2025).

7. Напрями\теми навчання на платформі Cisco Networking Academy. [Електронний ресурс] URL: https://edu-cisco.org/uk/ (дата звернення 21.04.2025).

8. Інформація за курсами та отриманою інформацію після завершення курсів. [Електронний ресурс] URL: https://edu-cisco.org/uk/courses/ (дата звернення 21.04.2025).

9. Перспективи впровадження чат-ботів в освіті та використання мікро навчання. [Електронний ресурс] URL: https://lessondelivery.org/chatbot/foreducation/perspektivi-chat-botiv-osvita-mikronavchannya.html (дата звернення 21.04.2025).

10. Cisco Networking Academy. [Електронний ресурс] URL: https://en.wikipedia.org/wiki/Cisco_Networking_Academy (дата звернення 27.04.2025).

11. Деталі про компанію Cisco. [Електронний ресурс] URL: https://ru.wikipedia.org/wiki/Cisco (дата звернення 27.04.2025).

12. Історія Cisco: як виник та розвивався мережевий гігант. [Електронний ресурс]URL:https://itedu.center/ua/blog/review/ciscohistory/?srsltid=AfmBOoq6bt8FmqM5M5yp7c83z5jR0GR__IihY6rG55yQx6hSYEtVa8K8 (дата звернення 28.04.2025).

13. Cisco networking Academy Courses Available. [Електронний ресурс] URL: https://ictskillnet.ie/courses/cisco-networking-academy/ (дата звернення 01.05.2025).

14. Інтеграція чат-бот рішень для Персоналізації контенту в Електронному навчанні. [Електронний ресурс] URL: https://cluelabs.com/blog/%D1%96%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B0%D1%86%D1%96%D1%8F-%D1%87%D0%B0%D1%82%D0%B1%D0%BE%D1%82-%D1%80%D1%96%D1%88%D0%B5%D0%BD%D1%8C-%D0%B4%D0%BB%D1%8F-%D0%BF%D0%B5%D1%80%D1%81%D0%BE%D0%BD%D0%B0 (дата зверення 01.05.2025).

15. Освітній чат-бот. [Електронний ресурс] URL: https://mon.gov.ua/osvita-2/tsifrova-transformatsiya-osviti-i-nauki/osvitniy-chat-bot (дата звернення 02.05.2025).

16. Використання чат-ботів telegram в освіті. [Електронний ресурс] URL: https://dspace.bdpu.org.ua/items/01543af2-73e5-405f-99aa-5447bb380bd8 (дата звернення 02.05.2025).

17. 20 найкращіх чат-ботів на основі ШІ для роботи, навчання й розваг. [Електронний ресурс] URL: https://blog.depositphotos.com/ua/najkrashhi-chat-boti-na-osnovi-shi.html (дата звернення 02.05.2025).

18. ChatGPT в освіті. [Електронний ресурс] URL: https://uk.wikipedia.org/wiki/ChatGPT_%D0%B2_%D0%BE%D1%81%D0%B2%D1%96%D1%82%D1%96 (дата звернення 03.05.2025).

19. Як створити чат бота для сфери освіти. [Електронний ресурс] URL: https://sendpulse.ua/blog/chatbot-for-education-sphere (дата звернення 03.05.2025).

20. Офіційна документація Telegram API для розробників ботів. [Електронний ресурс] URL: https://core.telegram.org/bots/api (дата звернення 15.05.2025).

21. Python-telegram-bot Documentation. [Електронний ресурс] URL: https://docs.python-telegram-bot.org/ (дата звернення 15.05.2025).

22. PostgreSQL Documentation. [Електронний ресурс] URL: https://www.postgresql.org/docs/ (дата звернення 16.05.2025).

23. Psycopg2: PostgreSQL adapted for Python. [Електронний ресурс] URL: https://www.psycopg.org/docs/ (дата звернення 16.05.2025).

24. Python-dotenv Documentation. [Електронний ресурс] URL: https://pypi.org/project/python-dotenv/ (дата звернення 18.05.2025).

25. Cisco Networking Academy. [Електронний ресурс] URL: https://www.netacad.com/ (дата звернення 18.05.2025).

26. OpenAI Reference. [Електронний ресурс] URL: https://platform.openai.com/docs (дата звернення 18.05.2025).

27. Жалдак М.І., Руденко С.Г. Основи програмування мовою Python. – 2020. – 302с. [Фізичний носій] (дата звернення 18.05.2025).

28. Матеріали форуму Stack Overflow [Електронний ресурс] URL: https://stackoverflow.com/ (дата звернення 19.05.2025).

29. YouTube-канал freeCodeCamp: Telegram Bots with Python. [Електронний ресурс] URL: https://www.youtube.com/c/Freecodecamp (дата звернення 19.05.2025).

30. GitHub: python-telegram-bot examples. [Електронний ресурс] URL: https://github.com/python-telegram-bot/python-telegram-bot (дата звернення 19.05.2025).

31. Lutz M. Learning Python. – O`Reailly Madia, 2021 – 1648с. [Фізичний носій] (дата звернення 20.05.2025).

ДОДАТОК А.

ПРОГРАМНИЙ КОД

А.1 Програмний код модуля bot.py

```
# Імпортуємо необхідні бібліотеки та модулі
import os
import logging
from telegram import Update, ReplyKeyboardMarkup
from telegram.ext import (
    ApplicationBuilder, CommandHandler, MessageHandler,
    ContextTypes, ConversationHandler, filters
)
from dotenv import load dotenv

# Імпортуємо функції з локальних модулів
from db import (
    get connection, init db, add feedback, get feedback,
    save course to db, get saved courses
)
from cisco api import get courses, format courses
from handlers.register import register, get_name, get_email, get_password,
NAME, EMAIL, PASSWORD
from handlers.login import login, get_login_email, check_login, LOGIN_EMAIL,
LOGIN PASSWORD
from handlers.guest import guest

# Завантаження змінних середовища
load dotenv()
BOT TOKEN = os.getenv("BOT TOKEN")

# Налаштування логування
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(  name  )

# Константи для станів у ConversationHandler
ASK_INTEREST, ASK_LEVEL, FEEDBACK = range(3)
SAVE_COURSE_SELECTION = range(3, 4)[0]
ADMIN_ID = 397447568  # ID адміністратора для перевірки прав

# Головне меню
main_menu_keyboard = ReplyKeyboardMarkup(
    [["📚 Рекомендації", "✉️ Зворотній зв'язок"],
     ["👤 Користувачі", "👍 Відгуки"],
     ["💾 Зберегти курс", "📂 Мої курси"],
     ["🔄 На головну"]],
    resize keyboard=True
)

# Початкове меню
start_keyboard = ReplyKeyboardMarkup(
    [["Реєстрація", "Увійти", "Увійти як гість"]],
    resize_keyboard=True,
    one_time_keyboard=True
```

```
)

# Обробник команди /start
async def start(update: Update, context: ContextTypes.DEFAULT_TYPE):
    await update.message.reply_text(
        "Вітаємо! Оберіть опцію для продовження:",
        reply_markup=start_keyboard
    )

# Повертає список доступних тем
def get_unique_topics():
    return ["Безпека", "Мережі", "Програмування", "DevNet", "Автоматизація",
"Python"]

# Запит інтересів користувача
async def recommend(update: Update, context: ContextTypes.DEFAULT_TYPE):
    topics = get_unique_topics()
    keyboard = [topics[i:i + 3] for i in range(0, len(topics), 3)]
    keyboard.append(['🔙 Назад', '🔄 На головну'])
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True,
one_time_keyboard=True)
    await update.message.reply_text(
        "Які теми вас цікавлять? Виберіть з кнопок або введіть свою тему:",
        reply_markup=reply_markup
    )
    return ASK_INTEREST

# Запит рівня знань користувача
async def ask_level(update: Update, context: ContextTypes.DEFAULT_TYPE):
    text = update.message.text
    if text in ['🔙 Назад', '🔄 На головну']:
        await show_main_menu(update, context)
        return ConversationHandler.END

    context.user_data["interest"] = text
    level_keyboard = ReplyKeyboardMarkup(
        [["Початковий", "Середній", "Просунутий"],
         ["🔙 Назад", "🔄 На головну"]],
        resize_keyboard=True,
        one_time_keyboard=True
    )
    await update.message.reply_text(
        "Який у вас рівень підготовки? Оберіть зі списку:",
        reply_markup=level_keyboard
    )
    return ASK_LEVEL

# Показує рекомендовані курси
async def show_courses(update: Update, context: ContextTypes.DEFAULT_TYPE):
    text = update.message.text
    if text in ["🔙 Назад", "🔄 На головну"]:
        await show_main_menu(update, context)
        return ConversationHandler.END

    level = text
    interest = context.user_data.get("interest", "")
    user_id = update.effective_user.id
    username = update.effective_user.username or ""

    # Збереження інтересу та рівня користувача в базу
    try:
        with get_connection() as conn:
            with conn.cursor() as cur:
```

```
                cur.execute("""
                    INSERT INTO users (telegram_id, username, interest,
level)
                    VALUES (%s, %s, %s, %s)
                    ON CONFLICT (telegram id)
                    DO UPDATE SET interest = EXCLUDED.interest, level =
EXCLUDED.level;
                """, (user_id, username, interest, level))
                conn.commit()
    except Exception:
        logger.exception("Помилка при оновленні інформації користувача")

    # Отримання курсів та збереження їх у контексті
    courses = get_courses(interest=interest, level=level)
    context.user_data['last_courses'] = courses

    if courses:
        msg = format courses(courses, interest=interest, level=level)
        await update.message.reply_text(msg, parse_mode="Markdown",
reply markup=main menu keyboard)
    else:
        await update.message.reply_text(
            "🔍 На жаль, не знайдено курсів у базі Cisco.\n🔎 Спробуємо
знайти схожі курси...",
            reply_markup=main_menu_keyboard
        )
        await update.message.reply text(
            "⚠️ AI-рекомендації тимчасово недоступні. Спробуйте пізніше або
змініть тему.",
            reply_markup=main_menu_keyboard
        )

    return ConversationHandler.END

# Запит на вибір курсу для збереження
async def save_course(update: Update, context: ContextTypes.DEFAULT_TYPE):
    user_id = update.effective_user.id
    courses = context.user data.get("last courses")
    if not courses:
        await update.message.reply text("❌ Спочатку отримайте рекомендації
курсів, щоб їх зберегти.", reply markup=main menu keyboard)
        return ConversationHandler.END

    keyboard = [[course['title']] for course in courses]
    keyboard.append(['🔙 Назад'])
    reply_markup = ReplyKeyboardMarkup(keyboard, one_time_keyboard=True,
resize keyboard=True)

    await update.message.reply text(
        "Оберіть курс, який хочете зберегти:",
        reply_markup=reply_markup
    )
    return SAVE COURSE SELECTION

# Збереження вибраного курсу
async def save_course_selection(update: Update, context:
ContextTypes.DEFAULT TYPE):
    user_id = update.effective_user.id
    selected_title = update.message.text
    if selected_title == '🔙 Назад':
        await update.message.reply_text("Операцію скасовано.",
reply markup=main menu keyboard)
        return ConversationHandler.END
```

```
    courses = context.user_data.get('last_courses', [])
    course to save = next((c for c in courses if c['title'] ==
selected_title), None)

    if not course_to_save:
        await update.message.reply_text("Курс не знайдено, спробуйте ще
раз.", reply_markup=main_menu_keyboard)
        return ConversationHandler.END

    success = save_course_to_db(user_id, course_to_save)
    if success:
        await update.message.reply_text(f"✅ Курс '{selected_title}' успішно
збережено.", reply markup=main menu keyboard)
    else:
        await update.message.reply text(f"❌ Помилка збереження курсу
'{selected title}'.", reply markup=main menu keyboard)

    return ConversationHandler.END

# Перегляд збережених курсів
async def view_saved_courses(update: Update, context:
ContextTypes.DEFAULT TYPE):
    user_id = update.effective_user.id
    saved = get_saved_courses(user_id)
    if not saved:
        await update.message.reply text("У вас немає збережених курсів.",
reply_markup=main_menu_keyboard)
        return

    msg = "💾 Ваші збережені курси:\n\n"
    for c in saved:
        created_at = c['created_at'].strftime('%Y-%m-%d') if c['created_at']
else '—'
        msg += (
            f"📘 {c['course_title']}\n"
            f"📝 {c['course_description']}\n"
            f"Дата збереження: {created_at}\n\n"
        )
    await update.message.reply_text(msg.strip(),
reply markup=main menu keyboard)

# Показує головне меню
async def show_main_menu(update: Update, context: ContextTypes.DEFAULT_TYPE):
    await update.message.reply_text(
        "Головне меню:",
        reply markup=main menu keyboard
    )

# Скасування операції
async def cancel(update: Update, context: ContextTypes.DEFAULT TYPE):
    await update.message.reply_text("Операція скасована.",
reply_markup=main_menu_keyboard)
    return ConversationHandler.END

# Перегляд усіх користувачів (тільки для адміністратора)
async def get_users(update: Update, context: ContextTypes.DEFAULT_TYPE):
    user id = update.effective user.id
    if user_id != ADMIN_ID:
        await update.message.reply_text("❌ Вибачте, у вас немає прав для
перегляду користувачів.", reply markup=main menu keyboard)
        return
```

```
    try:
        with get_connection() as conn:
            with conn.cursor() as cur:
                cur.execute("SELECT telegram_id, username, full_name, email
FROM users ORDER BY telegram id;")
                users = cur.fetchall()
    except Exception as e:
        logger.error(f"Error fetching users: {e}")
        users = []

    if not users:
        await update.message.reply_text("Список користувачів порожній.",
reply markup=main menu keyboard)
        return

    msg = "👥 Зареєстровані користувачі:\n\n"
    for u in users:
        msg += (
            f"ID: {u['telegram id']}\n"
            f"Ім'я: {u.get('full_name', '—')}\n"
            f"Нікнейм: @{u.get('username', '—')}\n"
            f"Email: {u.get('email', '—')}\n\n"
        )
    await update.message.reply_text(msg.strip(),
reply_markup=main_menu_keyboard)

# Ввід тексту відгуку
async def feedback_entry(update: Update, context: ContextTypes.DEFAULT_TYPE):
    await update.message.reply text(
        "Введіть ваш відгук або /cancel для скасування:",
        reply_markup=ReplyKeyboardMarkup([["/cancel"]], resize_keyboard=True,
one_time_keyboard=True)
    )
    return FEEDBACK

# Збереження відгуку
async def feedback save(update: Update, context: ContextTypes.DEFAULT TYPE):
    text = update.message.text
    if text == "/cancel":
        await update.message.reply_text("Операцію скасовано.",
reply markup=main menu keyboard)
        return ConversationHandler.END

    user id = update.effective user.id
    try:
        add_feedback(user_id, text)
    except Exception as e:
        logger.error(f"Error adding feedback: {e}")
        await update.message.reply_text("❌ Сталася помилка при додаванні
відгуку.", reply markup=main menu keyboard)
        return ConversationHandler.END

    await update.message.reply text("Дякуємо за ваш відгук!",
reply_markup=main_menu_keyboard)
    return ConversationHandler.END

# Перегляд усіх відгуків
async def view_feedback(update: Update, context: ContextTypes.DEFAULT_TYPE):
    user_id = update.effective_user.id

    try:
        feedbacks = get feedback()
    except Exception as e:
```

```
        logger.error(f"Error getting feedbacks: {e}")
        feedbacks = []

    if not feedbacks:
        await update.message.reply text("Відгуків ще немає.",
reply_markup=main_menu_keyboard)
        return

    msg = "📣 Відгуки:\n\n"
    for fb in feedbacks:
        if user id == ADMIN ID:
            try:
                with get_connection() as conn:
                    with conn.cursor() as cur:
                        cur.execute("SELECT username, full_name FROM users
WHERE telegram_id = %s;", (fb['user_id'],))
                        user = cur.fetchone()
                author = user['full name'] or user['username'] if user else
"Не відомо"
            except Exception as e:
                logger.error(f"Error fetching author info: {e}")
                author = "Не відомо"
            msg += f"Від: {author}\n"
        else:
            msg += "Відгук:\n"
        msg += f"{fb['message']}\n\n"

    await update.message.reply_text(msg.strip(),
reply markup=main menu keyboard)

# Головна функція запуску бота
def main():
    init db()  # Ініціалізація бази даних

    app = ApplicationBuilder().token(BOT_TOKEN).build()

    # Реєстрація обробників
    app.add_handler(CommandHandler("start", start))
    app.add_handler(MessageHandler(filters.Regex("^🔄 На головну$"),
show main menu))

    # Реєстрація
    register_handler = ConversationHandler(
        entry_points=[MessageHandler(filters.Regex("^Реєстрація$"),
register)],
        states={
            NAME: [MessageHandler(filters.TEXT & ~filters.COMMAND,
get_name)],
            EMAIL: [MessageHandler(filters.TEXT & ~filters.COMMAND,
get email)],
            PASSWORD: [MessageHandler(filters.TEXT & ~filters.COMMAND,
get_password)],
        },
        fallbacks=[CommandHandler("cancel", cancel)],
    )
    app.add_handler(register_handler)

    # Вхід
    login_handler = ConversationHandler(
        entry_points=[MessageHandler(filters.Regex("^Увійти$"), login)],
        states={
            LOGIN EMAIL: [MessageHandler(filters.TEXT & ~filters.COMMAND,
get_login_email)],
```

```
            LOGIN PASSWORD: [MessageHandler(filters.TEXT & ~filters.COMMAND,
check_login)],
        },
        fallbacks=[CommandHandler("cancel", cancel)],
    )
    app.add_handler(login_handler)

    # Гостьовий вхід
    app.add handler(MessageHandler(filters.Regex("^Увійти як гість$"),
guest))

    # Рекомендації курсів
    recommend handler = ConversationHandler(
        entry points=[MessageHandler(filters.Regex("^📚 Рекомендації$"),
recommend)],
        states={
            ASK_INTEREST: [MessageHandler(filters.TEXT & ~filters.COMMAND,
ask level)],
            ASK LEVEL: [MessageHandler(filters.TEXT & ~filters.COMMAND,
show_courses)],
        },
        fallbacks=[CommandHandler("cancel", cancel)],
    )
    app.add_handler(recommend_handler)

    # Збереження курсів
    save course handler = ConversationHandler(
        entry_points=[MessageHandler(filters.Regex("^💾 Зберегти курс$"),
save_course)],
        states={
            SAVE COURSE SELECTION: [MessageHandler(filters.TEXT &
~filters.COMMAND, save course selection)],
        },
        fallbacks=[CommandHandler("cancel", cancel)],
    )
    app.add_handler(save_course_handler)

    # Інші дії
    app.add_handler(MessageHandler(filters.Regex("^📂 Мої курси$"),
view saved courses))
    app.add_handler(MessageHandler(filters.Regex("^👤 Користувачі$"),
get users))

    feedback_handler = ConversationHandler(
        entry_points=[MessageHandler(filters.Regex("^✉ Зворотній зв'язок$"),
feedback_entry)],
        states={
            FEEDBACK: [MessageHandler(filters.TEXT & ~filters.COMMAND,
feedback_save)],
        },
        fallbacks=[CommandHandler("cancel", cancel)],
    )
    app.add_handler(feedback_handler)

    app.add_handler(MessageHandler(filters.Regex("^👍 Відгуки$"),
view feedback))

    # Запуск бота
    app.run_polling()

# Точка входу
if __name__ == "__main__":
    main()
```

А.2 Програмний код модуля db.py

```
# === db.py ===

# Імпорт необхідних бібліотек
import os
import logging
import psycopg2
from psycopg2.extras import RealDictCursor  # Повертає результати у вигляді словника
from dotenv import load dotenv  # Завантаження змінних середовища з .env

# Завантаження конфігурації з .env файлу
load_dotenv()
logger = logging.getLogger(  name  )

# Зчитування параметрів підключення до бази даних із .env
DB_NAME = os.getenv("DB_NAME")
DB USER = os.getenv("DB USER")
DB_PASSWORD = os.getenv("DB_PASSWORD")
DB_HOST = os.getenv("DB_HOST")
DATABASE URL = f"postgresql://{DB USER}:{DB PASSWORD}@{DB HOST}/{DB NAME}"

# Отримання з'єднання з базою даних
def get_connection():
    return psycopg2.connect(DATABASE URL, cursor factory=RealDictCursor)

# Ініціалізація бази даних: створення таблиць, якщо вони не існують
def init_db():
    try:
        with get connection() as conn:
            with conn.cursor() as cur:
                # Таблиця користувачів
                cur.execute("""
                    CREATE TABLE IF NOT EXISTS users (
                        telegram_id BIGINT PRIMARY KEY,
                        username TEXT,
                        full name TEXT,
                        email TEXT UNIQUE,
                        password TEXT,
                        role TEXT DEFAULT 'студент',  -- ролі: 'адмін', 'студент', 'гість'
                        interest TEXT,
                        level TEXT,
                        goals TEXT
                    );
                """)
                # Таблиця відгуків
                cur.execute("""
                    CREATE TABLE IF NOT EXISTS feedback (
                        id SERIAL PRIMARY KEY,
                        user_id BIGINT REFERENCES users(telegram_id) ON DELETE CASCADE,
                        message TEXT,
                        created_at TIMESTAMP DEFAULT NOW()
                    );
                """)
                # Таблиця збережених курсів
                cur.execute("""
                    CREATE TABLE IF NOT EXISTS saved courses (
                        id SERIAL PRIMARY KEY,
                        user_id BIGINT REFERENCES users(telegram_id) ON
```

```
DELETE CASCADE,
                        course_title TEXT,
                        course_description TEXT,
                        created_at TIMESTAMP DEFAULT NOW(),
                        UNIQUE(user_id, course_title)  -- запобігає
дублюванню курсів
                    );
                """)
                # Індекс для швидкого пошуку користувача за email
                cur.execute("CREATE INDEX IF NOT EXISTS idx_users_email ON
users(email);")
                conn.commit()
        logger.info("База даних ініціалізована успішно.")
    except Exception:
        logger.exception("Помилка при ініціалізації бази даних:")

# Збереження відгуку користувача
def add_feedback(user_id, message):
    try:
        with get_connection() as conn:
            with conn.cursor() as cur:
                cur.execute("""
                    INSERT INTO feedback (user_id, message)
                    VALUES (%s, %s);
                """, (user_id, message))
                conn.commit()
    except Exception:
        logger.exception("Помилка при збереженні відгуку:")

# Отримання всіх відгуків або відгуків конкретного користувача
def get_feedback(user_id=None):
    try:
        with get_connection() as conn:
            with conn.cursor() as cur:
                if user_id:
                    cur.execute("SELECT * FROM feedback WHERE user_id = %s
ORDER BY created_at DESC;", (user_id,))
                else:
                    cur.execute("SELECT * FROM feedback ORDER BY created_at
DESC;")
                return cur.fetchall()
    except Exception:
        logger.exception("Помилка при отриманні відгуків:")
        return []

# Збереження користувача (уникнення дублювання по telegram_id)
def save_user(telegram_id, username, full_name, email, password,
role='студент'):
    try:
        with get_connection() as conn:
            with conn.cursor() as cur:
                cur.execute("""
                    INSERT INTO users (telegram_id, username, full_name,
email, password, role)
                    VALUES (%s, %s, %s, %s, %s, %s)
                    ON CONFLICT (telegram_id) DO NOTHING;
                """, (telegram_id, username, full_name, email, password,
role))
                conn.commit()
        return True
    except Exception:
        logger.exception("Помилка при збереженні користувача:")
        return False
```

```
# Отримання користувача за email та паролем (вхід)
def get_user_by_email_password(email, password):
    try:
        with get_connection() as conn:
            with conn.cursor() as cur:
                cur.execute("SELECT * FROM users WHERE email=%s AND
password=%s", (email, password))
                return cur.fetchone()
    except Exception:
        logger.exception("Помилка при вході користувача:")
        return None

# Отримання користувача за Telegram ID
def get_user_by_id(telegram_id):
    try:
        with get_connection() as conn:
            with conn.cursor() as cur:
                cur.execute("SELECT * FROM users WHERE telegram_id=%s",
(telegram_id,))
                return cur.fetchone()
    except Exception:
        logger.exception("Помилка при отриманні користувача:")
        return None

# Отримання списку всіх зареєстрованих користувачів (тільки базова
інформація)
def get_all_users():
    try:
        with get_connection() as conn:
            with conn.cursor() as cur:
                cur.execute("SELECT telegram_id, username, full_name, email,
role FROM users ORDER BY telegram_id;")
                return cur.fetchall()
    except Exception:
        logger.exception("Помилка при отриманні користувачів:")
        return []

# Збереження курсу до обраного користувача
def save_course_to_db(user_id, course):
    try:
        with get_connection() as conn:
            with conn.cursor() as cur:
                cur.execute("""
                    INSERT INTO saved_courses (user_id, course_title,
course_description)
                    VALUES (%s, %s, %s)
                    ON CONFLICT (user_id, course_title) DO NOTHING;
                """, (user_id, course['title'], course['description']))
                conn.commit()
        return True
    except Exception:
        logger.exception("Помилка при збереженні курсу в БД:")
        return False

# Отримання збережених курсів користувача
def get_saved_courses(user_id):
    try:
        with get_connection() as conn:
            with conn.cursor() as cur:
                cur.execute("""
                    SELECT course_title, course_description, created_at
                    FROM saved_courses WHERE user_id = %s ORDER BY created_at
DESC;
                """, (user_id,))
```

```
                return cur.fetchall()
    except Exception:
        logger.exception("Помилка при отриманні збережених курсів:")
        return []
```

А.3 Програмний код модуля cisco_api.py

```
# === cisco_api.py ===

# Імпорт бібліотек
from dotenv import load_dotenv  # Для завантаження змінних середовища з .env
import logging  # Для ведення логів

# Завантаження змінних з .env (наразі не використовується, але залишено для
сумісності)
load_dotenv()
logger = logging.getLogger(__name__)

# Базове посилання на Cisco, може бути використане для посилань у майбутньому
BASE_URL = "https://www.cisco.com/c/en/us/training-events/training-
certifications/training/"

# Функція для отримання курсів за темою (interest) та рівнем складності
(level)
def get_courses(interest=None, level=None):
    # Відображення локалізованих назв рівнів на англійські назви
    level_map = {
        "початковий": "Beginner",
        "середній": "Intermediate",
        "просунутий": "Advanced",
        "beginner": "Beginner",
        "intermediate": "Intermediate",
        "advanced": "Advanced"
    }

    # Якщо вказано рівень, перетворюємо його до потрібного формату
    if level:
        level = level_map.get(level.lower(), level.capitalize())

    logger.info(f"Шукаємо курси з темою '{interest}' та рівнем '{level}'")

    # Емуляція бази даних курсів (локальний список)
    courses_db = [
        {"title": "Python Essentials", "description": "Вступ до Python,
змінні, цикли, умови, функції та об’єктне програмування.", "level":
"Beginner", "duration": "8 hours", "free": True, "mode": "Self-paced"},
        {"title": "Introduction to Cybersecurity", "description": "Основи
кібербезпеки, загрози, атаки, захист даних.", "level": "Beginner",
"duration": "6 hours", "free": True, "mode": "Self-paced"},
        {"title": "CCNA: Introduction to Networks", "description": "Основи
мереж, IP-адресація, маршрутизація, моделі OSI та TCP/IP.", "level":
"Beginner", "duration": "40 hours", "free": True, "mode": "Self-paced"},
        {"title": "Cybersecurity Essentials", "description": "Поглиблений
курс з безпеки мереж, загроз, криптографії.", "level": "Intermediate",
"duration": "30 hours", "free": True, "mode": "Self-paced"},
        {"title": "DevNet Associate", "description": "Програмування мереж,
API, автоматизація з Cisco DevNet.", "level": "Intermediate", "duration": "40
hours", "free": True, "mode": "Self-paced"},
        {"title": "Networking Essentials", "description": "Концепції мереж,
TCP/IP, конфігурація маршрутизаторів та комутаторів.", "level":
"Intermediate", "duration": "20 hours", "free": True, "mode": "Self-paced"},
        {"title": "CyberOps Associate", "description": "SOC-операції,
виявлення загроз, реагування на інциденти.", "level": "Advanced", "duration":
```

```
"60 hours", "free": False, "mode": "Instructor-led"},
        {"title": "Network Security", "description": "Захист мережі,
брандмауери, VPN, політики безпеки.", "level": "Advanced", "duration": "50
hours", "free": False, "mode": "Instructor-led"},
        {"title": "DevNet Automation", "description": "Автоматизація
конфігурацій мереж за допомогою Python і REST API.", "level": "Advanced",
"duration": "45 hours", "free": False, "mode": "Self-paced"},
        {"title": "IoT Fundamentals", "description": "Основи Інтернету речей,
підключення пристроїв, безпека IoT.", "level": "Beginner", "duration": "10
hours", "free": True, "mode": "Self-paced"}
    ]

    # Фільтрація курсів відповідно до вказаного інтересу та рівня
    filtered = []
    interest_lower = interest.lower() if interest else None  # Приведення
інтересу до нижнього регістру
    for course in courses_db:
        # Якщо вказано інтерес і він не входить у назву або опис — пропустити
курс
        if interest lower and interest lower not in course["title"].lower()
and interest_lower not in course["description"].lower():
            continue
        # Якщо вказано рівень і він не відповідає — пропустити
        if level and course["level"] != level:
            continue
        # Якщо курс відповідає фільтрам — додати до результату
        filtered.append(course)

    logger.info(f"Знайдено курсів: {len(filtered)}")
    return filtered

# Функція форматування списку курсів у текстовий вигляд для відправки
користувачу
def format_courses(courses, interest=None, level=None):
    if not courses:
        # Якщо немає результатів — повідомити користувача
        return f"🔍 На жаль, не знайдено курсів за темою «{interest}» та
рівнем «{level}»."

    # Формування заголовку повідомлення
    result = f"🔍 Ось курси за темою «{interest}», рівень: «{level}»:\n"

    # Додавання опису кожного курсу до результату
    for course in courses:
        result += (
            f"\n📘 {course['title']}\n"
            f"📝 {course['description']}\n"
            f"📊 {course['level']}\n"
            f"⏱ {course['duration']}\n"
            f"💰 {'Безкоштовний' if course['free'] else 'Платний'} | 🏫
{course['mode']}\n"
        )
    return result.strip()
```

А.4 Програмний код модуля menu.py

```
# === menu.py ===

# Імпортуємо клас для створення клавіатури з варіантами відповіді
from telegram import ReplyKeyboardMarkup

# Головне меню у вигляді клавіатури з кнопками
main_menu_keyboard = ReplyKeyboardMarkup(
    [
        ["📚 Рекомендації", "✉️ Зворотній зв'язок"],  # Рядок з кнопками для отримання курсів і надсилання фідбеку
        ["👤 Користувачі", "📥 Відгуки"],              # Кнопки для адміністратора: перегляд користувачів і відгуків
        ["🔄 На головну", "💖 Зберегти курс"]          # Повернення на головну і збереження поточного курсу
    ],
    resize_keyboard=True  # Автоматичне масштабування клавіатури під екран пристрою
)

# Асинхронна функція, яка показує головне меню користувачу
async def show_main_menu(update, context):
    await update.message.reply_text(
        "Оберіть опцію:",  # Повідомлення перед показом меню
        reply_markup=main_menu_keyboard  # Додаємо клавіатуру
    )
```

А.5 Програмний код модуля guest.py

```
# === handlers/guest.py ===

# Імпортуємо класи для роботи з Telegram API
from telegram import Update
from telegram.ext import ContextTypes, ConversationHandler
from menu import show_main_menu  # Імпортуємо функцію для показу головного меню
import logging  # Для логування помилок

# Створення логера для запису інформації або помилок
logger = logging.getLogger(__name__)


# Асинхронна функція обробки "гість" (вхід без реєстрації)
async def guest(update: Update, context: ContextTypes.DEFAULT_TYPE):
    try:
        # Повідомляємо користувачу, що він увійшов як гість
        await update.message.reply_text("Ви увійшли як гість.")

        # Виводимо головне меню (відповідно до ролі)
        await show_main_menu(update, context)

    except Exception:
        # Логування будь-якої помилки, що виникає під час виконання
        logger.exception("Помилка при вході як гість:")

        # Інформуємо користувача про помилку
        await update.message.reply_text("Виникла помилка при доступі у режимі гостя.")
```

```
    # Завершення поточної розмови (ConversationHandler)
    return ConversationHandler.END
```

А.6 Програмний код модуля login.py

```
# === handlers/login.py ===

# Імпортуємо необхідні класи для Telegram-бота
from telegram import Update
from telegram.ext import ContextTypes, ConversationHandler
from db import get user by email password  # Функція для перевірки облікових
даних у базі
from menu import show_main_menu  # Функція для показу головного меню після
входу
import logging  # Для логування подій та помилок

# Налаштування логера
logger = logging.getLogger(__name__)

# Стан розмови (етапи логіну)
LOGIN EMAIL, LOGIN PASSWORD = range(2)

# === ПЕРШИЙ КРОК ===
# Функція, яка запускає процес авторизації — запитує email
async def login(update: Update, context: ContextTypes.DEFAULT TYPE):
    await update.message.reply text("Введіть ваш email:")
    return LOGIN_EMAIL  # Переходимо до наступного етапу (введення email)

# === ДРУГИЙ КРОК ===
# Отримуємо email, зберігаємо в user_data і запитуємо пароль
async def get_login_email(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    context.user data["email"] = update.message.text  # Зберігаємо email у
контексті користувача
    await update.message.reply_text("Введіть пароль:")
    return LOGIN PASSWORD  # Переходимо до етапу перевірки пароля

# === ТРЕТІЙ КРОК ===
# Перевіряємо правильність введених email і пароля
async def check login(update: Update, context: ContextTypes.DEFAULT TYPE):
    email = context.user data.get("email")  # Отримуємо email із контексту
    password = update.message.text  # Отримуємо введений пароль

    try:
        # Перевіряємо облікові дані в базі даних
        user = get_user_by_email_password(email, password)
        if user:
            # Якщо користувача знайдено — вітаємо та показуємо головне меню
            await update.message.reply text(f"Вітаємо, {user['full name']}!")
            await show_main_menu(update, context)
        else:
            # Якщо облікові дані невірні — повідомляємо про це
            await update.message.reply_text("Невірний email або пароль.")
    except Exception:
        # У разі помилки логування винятку та повідомлення користувачу
        logger.exception("Помилка під час входу:")
        await update.message.reply_text("Виникла помилка при перевірці
```

```
входу.")

    # Завершуємо розмову, незалежно від результату
    return ConversationHandler.END
```

А.7 Програмний код модуля register.py

```
# === handlers/register.py ===

# Імпорт необхідних бібліотек
from telegram import Update
from telegram.ext import ContextTypes, ConversationHandler
from db import save user  # Функція збереження користувача в базу даних
import logging
from menu import show_main_menu  # Меню, яке з'явиться після реєстрації

# Налаштування логування
logger = logging.getLogger(__name__)

# Кроки реєстрації у ConversationHandler
NAME, EMAIL, PASSWORD = range(3)

# === КРОК 1 ===
# Початок реєстрації — запит ПІБ
async def register(update: Update, context: ContextTypes.DEFAULT_TYPE):
    await update.message.reply_text("Введіть ваше ПІБ:")
    return NAME  # Переходимо до наступного кроку — введення ПІБ

# === КРОК 2 ===
# Зберігаємо ПІБ користувача і запитуємо email
async def get name(update: Update, context: ContextTypes.DEFAULT TYPE):
    context.user_data["name"] = update.message.text  # Зберігаємо повне ім'я
в user_data
    await update.message.reply_text("Введіть ваш email:")
    return EMAIL  # Переходимо до введення email

# === КРОК 3 ===
# Зберігаємо email і запитуємо пароль
async def get_email(update: Update, context: ContextTypes.DEFAULT_TYPE):
    context.user_data["email"] = update.message.text  # Зберігаємо email у
контексті
    await update.message.reply text("Придумайте пароль:")
    return PASSWORD  # Переходимо до введення пароля

# === КРОК 4 ===
# Завершальний етап: збереження користувача до бази
async def get_password(update: Update, context: ContextTypes.DEFAULT_TYPE):
    telegram_id = update.effective_user.id  # Telegram ID користувача
    username = update.effective_user.username or None  # Username або None,
якщо відсутній
    full name = context.user data.get("name")  # ПІБ з попередніх кроків
    email = context.user_data.get("email")
    password = update.message.text  # Введений пароль

    try:
        # Спроба зберегти користувача в базу даних
        result = save user(telegram id, username, full name, email, password)
        if result:
            # Успішна реєстрація — показуємо головне меню
```

```
            await update.message.reply_text("Реєстрація успішна!")
            await show_main_menu(update, context)
        else:
            # Якщо такий telegram_id уже існує в БД
            await update.message.reply_text("Користувач із таким telegram_id
уже існує.")
    except Exception:
        # Логування винятку і повідомлення про помилку
        logger.exception("Помилка у процесі реєстрації:")
        await update.message.reply_text("Сталася непередбачена помилка при
реєстрації.")

    return ConversationHandler.END  # Завершення діалогу реєстрації
```