МІЖНАРОДНИЙ ГУМАНІТАРНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки, програмної інженерії та комп'ютерних наук
Кафедра інформаційних технологій

Пояснювальна записка

до кваліфікаційної роботи
другого (магістерського) рівня

на тему Розробка інтелектуального помічника підтримки навчального процесу за допомогою штучного інтелекту

Виконав: студент 2 курсу, групи ІКК-2.1
спеціальності
121 Інженерія програмного забезпечення

Чернов І.О.

Керівник Соловська І.М.

Рецензент І.І. Йона

Одеса – 2023 р.

ДОВІДКА

кафедри ІТ про виконану магістерську роботу

студента 2 курсу ФКПІ та КН групи ІКК-2.1 маг

Чернова Ігоря Олеговича

на тему Розробка інтелектуального помічника підтримки навчального процесу за допомогою штучного інтелекту

Висновок нормоконтролера пояснювальна записка до кваліфікаційної роботи викон. з незначними порушеннями ДСТУ. Оформлено згідно вимог положення ЖТУ

Нормоконтролер викл. каф ІТ (науковий ступінь, вчене звання, посада) 15.12.2023 (підпис, дата) Клімішина І.В. (і. б. прізвище)

Висновок відповідального за наявність плагіату згідно з сертифікатом ID 1015712716 унікальність роботи підтверджено.

Відповідальна особа викл. каф ІТ (науковий ступінь, вчене звання, посада) 15.02.2023 (підпис, дата) Клімішина І.В (і. б. прізвище)

Попередня експертиза (захист) магістерської роботи

(бакалаврської роботи чи магістерської роботи)

студ. Чернова І.О. (прізвище і.б.) проведена "12" 12 2023 р.

Висновки Кваліфікаційна робота виконана з деякими недоліками. В роботі розроблено інноваційний інструмент для навчального процесу.

Магістерська робота рекомендується до захисту

Члени комісії (підпис) д.т.н., проф. Бременковська І.В (науковий ступінь, вчене звання, посада, прізвище і.б.)

(підпис) к.т.н, доц Григор'єва М.І. (науковий ступінь, вчене звання, посада, прізвище і.б.)

(підпис) к.т.н, доц Горбалов В.Є. (науковий ступінь, вчене звання, посада, прізвище і.б.)

МІЖНАРОДНИЙ ГУМАНІТАРНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, програмної інженерії та комп'ютерних наук
Кафедра інформаційних технологій
Освітній ступінь магістр
Галузь знань 12 Інформаційні технології
Спеціальність 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТ
к.т.н., доц.
_____ Т.І. Григор'єва
"05" 09 20__ року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ РОБОТУ

Чернову Ігорю Олеговичу

1. Тема роботи: Розробка інтелектуального помічника підтримки навчального процесу за допомогою штучного інтелекту
керівник роботи к.т.н., доцент каф. КН Соловська І.М.
затверджені наказом закладу вищої освіти від 25 вересня 2023 р. № 1957

2. Строк подання студентом роботи 2023 р.

3. Вихідні дані до роботи: 1) Аналіз предметної області. 2) Огляд існуючих рішень. 3) Інтелектуальний помічник. 4) Необхідність залучення штучного інтелекту при розробці інтелектуального помічника. 5) Сценарій реалізації 6) База даних.

4. Зміст розрахунково-пояснювальної записки
Розділ 1: Теоретична частина
Розділ 2: Розробка інтелектуального помічника підтримки навчального процесу за допомогою штучного інтелекту
Розділ 3: Опис технологічних аспектів програмного застосунку
Розділ 4: Розробка програмної частини проекту

5. Перелік графічного матеріалу (з зазначенням обов'язкових креслень)
Слайд 1 – Сценарій реалізації інтелектуального помічника

Слайд 2 – Обгрунтування залучення штучного інтелекту при розробці інтелектуального помічника

Слайд 3 – Блок схема алгоритму розробки інтелектуального помічника

Слайд 4 – База даних інтелектуального помічника

Слайд 5– Моделі таблиць user та schedule

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Завдання видав	Завдання прийняв

7. Дата видачі завдання 25.11.2022 р.

КАЛЕНДАРНИЙ ПЛАН

/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Вступ	25.09.2023-29.09.2023	
2	Аналіз програмних додатків та сервісів начального процесу	2.10.2023-23.10.2023	
3	Розробка інтелектуального помічника підтримки навчального процезу за допомогою штучного інтелекту	24.10.2023 – 10.11.2023	
4	Опис технологічних аспекітв програмного застосунку	13.11.2023-24.11.2024	
5	Розробка програмної частини проекту	13.11.2023-30.11.2024	
6	Висновки	30.11.2023-2.12.2023	
7	Перелік посилень	2.12.2023-5.12.2023	
8		6.12.2023-10.12.2023	

Студент _____ І. О. Чернов
(підпис)

Керівник роботи _____ І.М. Соловська
(підпис)

РЕФЕРАТ

Текстова частина магістерської роботи: 74 с., 10 рисунків, 5 таблиця, 1 додаток, 29 джерел.

ANDROID, IOS, TELEGRAM, TELEGRAM BOT, ШТУЧНИЙ ІНТЕЛЕКТ, ЗАСТОСУНОК, ФУНКЦІЇ, ДОДАТОК, ЗАВДАННЯ, ІНТЕРФЕЙС, РЕАЛІЗАЦІЯ, ПЕРЕВАГИ, НЕДОЛІКИ, ПЕРСОНАЛІЗАЦІЯ, БАЗА ДАНИХ, БЛОК СХЕМА

Об'єкт дослідження – інтелектуальний помічник для забезпечення навчального процесу за допомогою штучного інтелекту.

Мета роботи – для підтримки навчального процесу за допомогою штучного інтелекту полягає в створенні ефективного та інноваційного інструменту, який сприятиме покращенню навчального середовища та забезпечить оптимальну підтримку для учнів та викладачів.

Метод дослідження – аналітичний.

У магістерський роботі Проведене дослідження використання інтелектуального помічника у навчальному середовищі розкрило ряд ключових аспектів, що визначають його вплив та користь для учасників освітнього процесу. Однією з головних вагомих переваг є покращення ефективності навчання завдяки персоналізованій підтримці та доступу до необхідної інформації. Використання мови програмування Python, в порівнянні з іншими мовами, є оптимальним рішенням для створення інтелектуального помічника на базі штучного інтелекту.

ABSTRACT

The text part of the master's thesis: 74 p., 10 figures, 5 tables, 1 appendix, 29 sources.

ANDROID, IOS, TELEGRAM, TELEGRAM BOT, ARTIFICIAL INTELLIGENCE, APPLICATION, FUNCTIONS, APPLICATION, TASKS, INTERFACE, IMPLEMENTATION, ADVANTAGES, DISADVANTAGES, PERSONALISATION, DATABASE, FLOWCHART

Object of study - intelligent assistant to support the learning process using artificial intelligence.

The purpose of the study is to create an effective and innovative tool to support the learning process using artificial intelligence, which will improve the learning environment and provide optimal support for students and teachers.

The research method is analytical.

In the master's thesis, the study of the use of an intelligent assistant in the learning environment revealed a number of key aspects that determine its impact and benefits for participants in the educational process. One of the main significant benefits is the improvement of learning efficiency through personalised support and access to the necessary information. Compared to other programming languages, Python is the best solution for creating an AI-based intelligent assistant.

ВІДГУК КЕРІВНИКА

на кваліфікаційну роботу здобувача другого (магістерського) рівня
Чернова Ігоря Олеговича
на тему: «Розробка інтелектуального помічника підтримки навчального процесу за допомогою штучного інтелекту»

Сучасний стан розвитку технологій та методів навчального процесу все частіше оздоблюється різними сервісами та додатками, що зумовлює потребу у нових рішеннях. Таким рішенням може бути інтелектуальний помічник підтримки навчального процесу, який функціонує на базі штучного інтелекту.

В магістерській роботі розроблений інтелектуальний помічник, обладнаний штучним інтелектом, має потенціал у розумінні та аналізі великої кількості інформації.

Здобувач Чернов І.О. не повністю виконав завдання до кваліфікаційної роботи. В процесі роботи здобувач Чернов І.О. працював не організовано. Графік консультацій часто порушувався. Поставлене завдання не виконано у повному обсязі.

Під час виконання кваліфікаційної роботи здобувач Чернов І.О. не виконав усі поставлені завдання.

Кваліфікаційна робота відповідає вимогам до кваліфікаційних робіт другого (магістерського) рівня та заслуговує оцінки «задовільно».

Здобувач Чернов І.О. заслуговує присвоєння кваліфікації магістр з інженерії програмного забезпечення за заявленою спеціальністю 121 «Інженерія програмного забезпечення».

Керівник,
завідувач кафедри
комп'ютерних наук
к.т.н., доцент — І. М. Соловська

РЕЦЕНЗІЯ

на кваліфікаційну роботу здобувача другого (магістерського) рівня
Чернова Ігоря Олеговича
на тему: «Розробка інтелектуального помічника підтримки навчального процесу за допомогою штучного інтелекту»

Кваліфікаційна робота здобувача Чернова І.О. присвячена питанням розробки інтелектуального помічника підтримки навчального процесу за допомогою штучного інтелекту.

Розробка інтелектуального помічника, котрий включає в себе необхідні для навчання можливості із залученням штучного інтелекту, дозволяє значно спростити процес отримання інформації для здобувачів вищої освіти. Використання мови програмування Python дозволило реалізувати необхідний функціонал для створення інтелектуального помічника на базі штучного інтелекту.

Кваліфікаційна робота не відповідає завданню. Текст роботи зрозумілий але містить не всі пункти завдання, є зауваження до оформлення пояснювальної записки та демонстраційних слайдів.

До недоліків роботи слід віднести:

- недостатню увагу приділено в розробленому додатку питанням атестації здобувачів;
- доцільно було б розглянути можливість поєднання додатку з іншими ресурсами, зокрема групами та іншими додатками.

Проте, зазначені недоліки не знижують цінності виконаної роботи.

У цілому, кваліфікаційна робота здобувача Чернова І.О. відповідає вимогам до випускних кваліфікаційних робіт здобувачів другого (магістерського) рівня та заслуговує оцінки «задовільно».

Здобувач Чернов І.О. заслуговує присвоєння кваліфікації магістр з інженерії програмного забезпечення за заявленою спеціальністю 121 «Інженерія програмного забезпечення».

Рецензент,
завідувач кафедри комп'ютерної інженерії
та інноваційних технологій,
к.т.н., доцент Л.Г. Йона

UNICHECK
by Turnitin

Ім'я користувача:
Анна Серединко

ID перевірки:
1016024119

Дата перевірки:
20.12.2023 12:02:21 EET

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
20.12.2023 12:09:03 EET

ID користувача:
100001433

Назва документа: Головна. Звіт. Чернов

Кількість сторінок: 74 Кількість слів: 12525 Кількість символів: 99989 Розмір файлу: 934.44 KB ID файлу: 1015712716

2.4%
Схожість

Найбільша схожість: 0.41% з Інтернет-джерелом (https://codeclimate.com/github/leandrotoledo/python-telegram-bot/...

2.4% Джерела з Інтернету 402 ..... Сторінка 76

0.09% Джерела з Бібліотеки 1 ..... Сторінка 77

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи 10

ЗМІСТ

ВСТУП

Технологізація проникає в усі сфери сучасного суспільства, адаптація до нових технологічних взаємодій породжує різноманітні технології, які надають можливість спілкуватися з «віртуальними помічниками». Розвиток застосунків і чат-ботів дозволяє розширити можливості надання інформації учасникам навчального процесу, з одного боку, та спростити їм доступ до інформації — з іншого. Виникає нагальна потреба в імплементації нових форм навчання, а не лише в удосконаленні способів і методик викладання.

Доцільнішим є залучення до навчального процесу штучного інтелекту (ШІ). ШІ може стати в нагоді вчителям, які мають, зокрема, оцінювати роботи учнів та вести облік прогресу навчання. Завдяки використанню ШІ вчителі зможуть працювати ефективніше й до того ж економити свій час. Аналіз даних із допомогою ШІ допоможе їм удосконалювати навчальні програми та матеріали, щоб у процесі краще задовольняти потреби учнів, забезпечуючи їм при цьому більш пристойні результати. ШІ може також допомогти у навчанні й учням. Завдяки йому можна створювати індивідуальні навчальні програми, які враховують здібності кожного учня та його потреби. Такі програми зможуть сприяти учням ефективніше здобувати знання. ШІ також зможе забезпечити учням доступ до ширшого кола актуальних джерел інформації, що дозволить їм отримувати значно повнішу інформацію для навчання.

Чат-бот, інтегрований у систему навчання, написаний на мові програмування Python, з застосуванням бази даних та штучного інтелекту.

В цьому змісті й розрахований зміст магістерської роботи, який передбачає порівняння існуючих додатків, аналіз їхнього функціоналу та реалізацію чат-бота підтримки навчального процесу на базі штучного інтелекту.

1. АНАЛІЗ ПРОГРАМНИЙ ДОДАТКІВ ТА СЕРВІСІВ НАВЧАЛЬНОГО ПРОЦЕСУ

1.1. Аналіз функціоналу програмних додатків та сервісів

Обґрунтування функціоналу програмних додатків та сервісів, які використовують студенти та викладачі в навчальному процесі.

Огляд існуючих процесів, які відбуваються під час навчання, за умови формування наступних завдань, які виконують студенти в процесі навчання, серед яких можна виділити такі.

Збір та моніторинг даних.

Тобто введення та зберігання особистих та академічних даних студентів, таких як оцінки, розклад занять, інформація про викладачів. Можливість моніторингу прогресу та вивчення статистики.

Планування та розклад ранять.

Іншими словами, інтеграція системи з розкладом навчання, можливість створення персоналізованого розкладу для кожного студента, з урахуванням його виборів і обов'язкових курсів.

Онлайн-матеріали та завдання.

Сюди входять доступ до навчальних матеріалів, лекцій, завдань через систему CRM, можливість викладачів завантажувати матеріали, а студентам — здавати завдання онлайн.

Спільноти та зв’язок.

Створення внутрішньої соціальної мережі або форуму для спілкування між студентами, обговорення питань та для допомоги один одному.

Психологічна підтримка.

Включає в себе інтеграцію інструментів для моніторингу та надання психологічної підтримки, проведення анонімних консультацій, доступ до корисних ресурсів та інформації про здоров'я.

Кар'єрні можливості.

Інформування про кар’єрні можливості, стажування та вакансії, а також інструменти для підготовки резюме та співбесід.

Нагадування та повідомлення.

Система автоматичних нагадувань про важливі події, дедлайни, оновлення та інші повідомлення через електронну пошту, смс або застосунок.

Оцінка та звітність.

Тобто аналіз результатів студентів, автоматизована звітність для викладачів та адміністраторів щодо успішності та активності студентів.

Інтеграція з іншими системами.

Можливість інтеграції з іншими інформаційними системами університету, такими як фінансова звітність, бібліотека, система здоров'я тощо.

1.2. Огляд існуючих рішень

Використання різноманітних додатків, месенджерів та програмних середовищ у навчальному процесі стає необхідністю і знаходить все більше популярності серед освітян та студентів. Сучасні технології вже невід'ємно вплетені в тканину освіти, і їх використання має величезний вплив на ефективність навчання та організацію робочого процесу. Давайте розглянемо ряд існуючих рішень, що використовуються для організації часу, планування, керування завданнями, а також для формування заміток та систематизації інформації в навчальному середовищі.

В сучасному світі, де швидкість і легкість отримання інформації стали ключовими компонентами, додатки для організації часу та планування стають невід'ємною частиною студентського та викладацького досвіду. Однією з переваг є зручний та ефективний спосіб використання часу, що забезпечує оптимальне розподіл завдань та визначення пріоритетів. Популярність таких додатків, як Todoist,

Wunderlist та Microsoft To-Do, заснована на їхній простоті та можливості інтеграції із різними платформами.

Керування часом є критично важливим аспектом в навчальному середовищі, і тут входять у гру додатки, спрямовані на відстеження та оптимізацію продуктивності. RescueTime відстежує час, витрачений на різні завдання, допомагаючи аналізувати та покращувати ефективність. Forest, зі своєю нестандартною концепцією, створює ігровий момент, стимулюючи концентрацію та управління часом.

Ще однією ключовою сферою є робота із завданнями та завданнями. Програми, такі як Asana та Trello, стали невід'ємними інструментами для керування проектами та завданнями в освітній діяльності. Їхні гнучкі системи дошок, списків та завдань дозволяють організовувати роботу та взаємодіяти над завданнями у режимі реального часу.

Для зберігання та організації інформації студентів та викладачів широко використовують Evernote та OneNote. Можливість зберігати різні типи контенту, від тексту до аудіо та графіки, надає зручність у веденні нотаток та сприяє легкому доступу до потрібної інформації.

Особливу увагу варто звернути на платформи для навчання, такі як Khan Academy та Coursera. Вони не лише забезпечують безкоштовний доступ до величезного обсягу відмінних відео-уроків, але й створюють можливість для індивідуалізованого навчання, забезпечуючи доступ до різноманітних предметів та курсів.

Науковий аспект навчання підтримують сервіси для пошуку та організації наукових матеріалів, такі як Google Scholar та JSTOR. Вони надають доступ до наукових статей, книг та інших ресурсів, допомагаючи студентам та викладачам у проведенні досліджень та вивченні актуальної інформації у вибраній галузі.

Необхідно зазначити, що жоден із розглянутих додатків не надає можливості використання штучного інтелекту (ШІ) або інших передових технологій у своєму

функціоналі. Однак виникає суперечливий питання: чи варто штучний інтелект в системі освіти?

Штучний інтелект може вносити вагомий внесок у покращення навчання та роботу з інформацією. Можливості використання ШІ включають індивідуалізацію підходу до студентів, прогнозування їхніх потреб та надання персоналізованих рекомендацій щодо навчання. Штучний інтелект може допомагати в аналізі навчальних даних, виокремлюючи тренди та вказуючи на можливість оптимізації навчального процесу.

У підсумку, використання різноманітних додатків та програм у навчанні стає необхідністю, оскільки це забезпечує ефективне управління часом, планування та організацію інформації. Щоразу більше освітніх інституцій впроваджують ці технології для поліпшення навчання та забезпечення студентам та викладачам доступу до передових інструментів. Однак питання використання штучного інтелекту залишається відкритим, вимагаючи уважного аналізу та обговорення його можливих переваг та викликів в контексті освіти.

1.2.1. Організація та планування.

Wunderlist — це інтуїтивний додаток для створення списків завдань та спільної організації завдань між користувачами. Дозволяє легко створювати, редагувати та відстежувати завдання, встановлювати терміни та додавати коментарі. Забезпечує синхронізацію між різними пристроями та платформами, включаючи веб-версію, мобільні додатки і розширення для браузерів. Інтегрований зі сторонніми сервісами, Wunderlist надає зручність спільної роботи та подальшого вдосконалення організації завдань в робочому та особистому житті користувачів. [1]

Microsoft To-Do — це інтегрований додаток для створення списків завдань та організації розкладу від Microsoft. Забезпечує користувачів простим та інтуїтивно зрозумілим інтерфейсом, де вони можуть створювати завдання, встановлювати

терміни та додавати нотатки. Додаток синхронізується з іншими сервісами Microsoft, такими як Outlook, і дозволяє легко вести списки завдань на різних платформах. Включає функції нагадувань та підсумкового перегляду завдань для полегшення організації робочого та особистого життя користувачів.[2]

Todoist — це популярний додаток для створення списків завдань та планування робочого чи особистого часу. Забезпечує ефективний спосіб організації завдань за допомогою списків, підзадач та тегів. Todoist синхронізується на різних платформах, включаючи веб-версію, мобільні додатки та розширення для браузерів. Завдяки функціям спільної роботи та нагадуванням, додаток допомагає користувачам ефективно керувати своїм часом і завданнями, роблячи планування простим та зручним.[3]

Trello — це інноваційний додаток для управління завданнями та організації проєктів, що використовує візуальну концепцію дошок та карток. Кожна дошка являє собою проєкт, а карти — окремі завдання. Відмінно підходить для особистого та командного використання. Trello дозволяє легко створювати, переміщати та відстежувати завдання, використовуючи інтерфейс «перетягни і відпусти». Інтегрується з численними додатками та сервісами, такими як Google Drive та Dropbox, спрощуючи обмін інформацією. Зручна система коментування та прикріплення файлів полегшує спільну роботу. Trello надає велику гнучкість в управлінні проєктами та завданнями, забезпечуючи ефективність та взаємодію в командному середовищі.[4]

Усі чотири додатки — Wunderlist, Microsoft To-Do, Todoist та Trello — пропонують зручні інструменти для організації завдань та планування часу в робочому та особистому житті.

1.2.2. Управління часом.

RescueTime — це продуктивний інструмент для відстеження та аналізу використання часу на комп’ютері. Автоматично реєструє активність і надає користувачам детальну статистику, яка допомагає усвідомити, як час витрачається.

Зокрема, визначає час, проведений на різних веб-сайтах і програмах, щоб допомогти ефективно керувати робочим часом та уникати відхилень уваги. RescueTime також надає можливість встановлення цілей та отримання звітів, сприяючи покращенню продуктивності та збалансованості робочого дня. Цей інструмент є корисним для тих, хто бажає більш ефективно управляти своїм часом та звільнити його для важливих завдань.[5]

Forest — це мобільний додаток, що сприяє концентрації та управлінню часом. Користувачі можуть встановлювати таймер на конкретний період, під час якого «вирощується ліс» на їхньому екрані. Якщо користувач відволікається і залишає додаток, його «ліс» гине. При успішному завершенні сесії користувачі отримують віртуальні дерева, що додаються до їхнього лісу. Цей інтерактивний підхід до управління часом сприяє уникненню відхилень уваги та збереженню фокусу на роботі чи навчанні, роблячи процес більш захопливим та мотивуючим.[6]

RescueTime та Forest є відмінними інструментами для управління часом та підвищення продуктивності. RescueTime надає детальну статистику використання часу на комп'ютері, допомагаючи ефективно керувати робочим часом. З іншого боку, Forest використовує інтерактивний підхід, заохочуючи концентрацію та уникнення відволікань через вирощування віртуального лісу.

1.2.3. Замітки та організація інформації.

OneNote — це високофункціональний інструмент для створення та організації нотаток. Користувачі можуть створювати текстові, голосові та рукописні нотатки, прикріплювати файли та зображення. Інтегрований з іншими продуктами Microsoft, OneNote забезпечує синхронізацію на всіх пристроях та платформах. Зручність використання полягає у створенні блокнотів та секцій для кращого організованого зберігання інформації. Відмінність полягає в розширених можливостях маркування та обробки контенту, зробивши OneNote універсальним інструментом для навчання, роботи та власних потреб.[7]

Evernote — це потужний інструмент для зберігання та організації нотаток. Користувачі можуть створювати текстові, голосові та графічні нотатки, прикріплювати файли та вибирати теги для легкого пошуку. Забезпечує синхронізацію між різними пристроями та веб-версією. Evernote також дозволяє організувати нотатки в блокноти, спрощуючи структуру та доступ до інформації. Його функціонал включає розпізнавання тексту на зображеннях, що полегшує пошук та обробку інформації. Цей інструмент ідеально підходить для особистого та професійного використання.[8]

Notion — це унікальний інтегрований інструмент, що поєднує нотатки, завдання та бази даних в одному просторі. Користувачі можуть створювати структуровані сторінки, додавати текст, таблиці, графіку та вбудовувати файли. Зручна система блоків дозволяє вільно організовувати вміст. Notion інтегрується з численними сервісами, такими як Google Drive та Slack, спрощуючи спільну роботу. Його гнучкість та можливості роблять його ідеальним для особистого та командного використання, надаючи великий простір для творчості та співпраці.[9]

OneNote, Evernote та Notion — це потужні інструменти для створення та організації нотаток, кожен з яких має свої унікальні особливості.

1.2.4. Навчання та підтримка вивчення.

Quizlet — це популярна освітня платформа для вивчення та повторення матеріалів за допомогою флеш-карток та інших інтерактивних засобів. Користувачі можуть створювати власні набори карток або використовувати готові, ділитися ними та використовувати в режимі гри. Quizlet надає зручні інструменти для запам’ятовування та тестування знань, роблячи навчання ефективним та захоплюючим. Цей ресурс широко використовується студентами та вчителями для підготовки до іспитів та вивчення нового матеріалу в різних предметах. [10]

Khan Academy — це освітній онлайн-ресурс, який надає безкоштовний доступ до великої кількості навчальних матеріалів у формі відеолекцій, вправ та тестів.

Заснований Салманом Ханом у 2008 році, Khan Academy став популярним інструментом для самостійного вивчення та підготовки до шкільних іспитів. Сайт пропонує матеріали з різних предметів, включно з математикою, наукою, історією, мовами та багато інших. Кожен урок побудований на принципах пояснення концепції в доступній формі, після чого надається серія вправ для закріплення матеріалу. Особливою перевагою Khan Academy є індивідуалізовані рекомендації, які допомагають користувачам визначити свої слабкі та сильні сторони, а також вибрати відповідні завдання для оптимального навчання. Всі матеріали доступні безкоштовно, що робить освіту доступнішою для всіх прошарків суспільства. [11]

Coursera — це онлайн-платформа для навчання, яка надає доступ до курсів вищих навчальних закладів та компаній з усього світу. Заснована в 2012 році, Coursera стала однією з провідних платформ MOOC (Massive Open Online Courses). Платформа співпрацює з університетами та організаціями в наданні різноманітних курсів у різних галузях, включно з наукою, технікою, мистецтвом, бізнесом, гуманітарними науками тощо. Курси можуть бути безкоштовними, або користувачі можуть отримати сертифікат після успішного завершення за плату. Coursera пропонує інтерактивні лекції, домашні завдання, форуми для обговорення, та інші ресурси для поліпшення навчання. Із зростанням популярності онлайн-навчання, Coursera допомагає розширити доступ до високоякісної освіти та підвищити кваліфікацію в різних галузях для користувачів з усього світу. [12]

Quizlet, Khan Academy та Coursera є видатними освітніми ресурсами, кожен із яких вирізняється своїми унікальними можливостями. Кожен із цих ресурсів відкриває нові можливості для навчання та саморозвитку, зроблюючи освіту доступною для всіх.

1.2.5. Плагінти для браузера.

Grammarly — це інтелектуальний онлайн-редактор, спрямований на вдосконалення граматики, стилю та правопису в текстах. Забезпечує користувачів

коректором, який автоматично виявляє та виправляє помилки під час набору тексту у різних текстових середовищах, включаючи веб-сайти, електронні листи та текстові редактори. Grammarly також пропонує рекомендації щодо вдосконалення стилю, використання відповідного словникового запасу та виразності. Цей інструмент підходить для письмових завдань будь-якого типу, допомагаючи користувачам підняти якість своїх текстів та спілкування в цілому. [13]

Zotero — це безкоштовний менеджер джерел та дослідницький інструмент, створений для організації та зберігання наукових джерел. Забезпечує зручний збір та організацію літературних посилань, автоматично створює бібліографічні записи та дозволяє швидко вставляти їх у наукові роботи. Zotero інтегрується з браузерами, дозволяючи легко зберігати статті та веб-сторінки. Цей інструмент особливо корисний для студентів, вчених та дослідників, що ведуть дослідження та пишуть наукові роботи. [14]

Grammarly та Zotero є потужними інструментами для роботи з текстами та дослідженням.

1.2.6. Менеджери завдань та проектів.

Asana — це високоефективний інструмент для управління завданнями та проєктами, спрямований на поліпшення комунікації та співпраці в командах. Забезпечує створення завдань, призначення відповідальних осіб, встановлення термінів та прикріплення файлів. Має гнучку систему проектних дошок та списків, що дозволяє визначати пріоритети та відстежувати виконання завдань. Asana сприяє структурованому плануванню та взаємодії, надаючи засоби для обговорення та обміну інформацією. Інтегрується з численними додатками, такими як Slack та Google Drive, спрощуючи обмін даними та забезпечуючи потужний інструмент для ефективного управління завданнями та проєктами в різних галузях діяльності. [15]

Trello — це інноваційний інструмент для управління завданнями та організації проєктів, що визначається своєрідною візуальною концепцією дошок і карток. Кожна

дошка являє собою проєкт, а карти — окремі завдання. Trello дозволяє командам та користувачам особисто переглядати й керувати завданнями у формі карток, які можна перетягувати між різними списками на дошці. Цей простий інтерфейс полегшує відстеження прогресу та сприяє ефективному взаємодії. Додаток дозволяє прикріплювати файли, коментувати завдання, встановлювати терміни та визначати відповідальних. Зручний інтерфейс та можливість відкритого обміну інформацією роблять Trello популярним для управління проєктами, планування подій, а також особистих завдань та цілей. Із численними розширеннями та інтеграціями, такими як Google Drive та Slack, Trello стає потужним інструментом для ефективної комунікації та спільної роботи. [4]

Asana та Trello є високоефективними інструментами для управління завданнями та проєктами, кожен з яких володіє своїми унікальними особливостями.

1.2.7. Онлайн-бібліотеки та ресурси.

Google Scholar — це безкоштовний сервіс від Google, спеціально розроблений для пошуку наукових статей, дисертацій, конференцій та академічної літератури. Платформа надає доступ до великої кількості наукових джерел і публікацій у різних галузях знань. Google Scholar дозволяє вченим, студентам та дослідникам швидко знаходити актуальні та відповідні матеріали для своїх наукових робіт. Результати пошуку включають наукові статті, книги, патенти та інші академічні ресурси. Крім того, Google Scholar надає цитати та індексацію авторів, що полегшує вивчення наукової літератури та дозволяє встановлювати науковий вплив публікацій. Цей інструмент став необхідним для науковців та студентів, які шукають достовірні джерела для своїх наукових досліджень та публікацій, роблячи пошук наукової інформації більш зручним та доступним. [16]

JSTOR (Journal Storage) — це цифрова бібліотека та онлайн-ресурс, який надає доступ до обширної колекції академічних рецензованих журналів, книг та первинних джерел. JSTOR спрощує доступ до важливих ресурсів для студентів, вчителів та науковців. Платформа містить мільйони статей з різних галузей знань, таких як гуманітарні науки, соціальні науки, природничі науки та інші. JSTOR також пропонує функції пошуку, фільтрації та вивчення цитат, роблячи навчання та дослідження більш зручними. Один з основних переваг JSTOR — це збереження архівів, що дозволяє звертатися до старих видань і джерел, а не лише до найновіших. Висока якість та авторитетність матеріалів роблять JSTOR важливим інструментом для академічних досліджень та навчання в різних дисциплінах. [17]

Google Scholar та JSTOR є важливими ресурсами для науковців, студентів та дослідників у пошуку академічних джерел та наукових публікацій

1.2.8. Додатки для математичних розрахунків

MATLAB і Mathcad — це два потужних програмних продукти, які використовуються для розв’язання математичних завдань і аналізу даних. Обидва

інструменти дозволяють інженерам, науковцям та іншим фахівцям працювати з числовими даними, виконувати обчислення, визначати та моделювати складні математичні процеси. [18],[19]

MATLAB (Matrix Laboratory), розроблений MathWorks, є високорівневою мовою програмування, яка спеціально орієнтована на числові обчислення та обробку сигналів. Він включає широкий спектр інструментів для роботи з графікою, статистикою, алгоритмами машинного навчання та іншими галузями. MATLAB також має інтерактивне середовище для розробки, що полегшує створення та відлагодження складних програм. [18]

Mathcad, розроблений PTC, визначається як система технічних обчислень з великим акцентом на виразності та зручності введення математичних формул. Програма використовує технологію WYSIWYG (What You See Is What You Get), що дозволяє користувачам створювати документи, в яких обчислення і текст взаємодіють. Mathcad дозволяє створювати живі документи, в яких результати автоматично оновлюються при зміні вхідних даних. [19]

Обидва інструменти знаходять широке застосування у наукових дослідженнях, інженерії, фінансах та інших галузях завдяки своїм потужним можливостям у сфері обробки даних та моделювання математичних процесів.

1.2.9. Графічні рішення.

Crello — це інноваційний інструмент для створення графічного контенту, який дозволяє користувачам легко і швидко розробляти візуальний зміст для різних цілей. Цей інструмент став популярним серед маркетологів, дизайнерів, підприємців та інших фахівців, які шукають простий і ефективний спосіб створення якісного дизайну. Crello пропонує широкий вибір шаблонів для соціальних мереж, банерів, афіш, презентацій, обкладинок для YouTube та іншого графічного контенту. Користувачі можуть легко адаптувати ці шаблони до своїх потреб, змінюючи текст, графіку та інші елементи за допомогою інтуїтивно зрозумілого інтерфейсу. Однією з ключових

особливостей Crello є можливість завантажувати власні зображення та використовувати їх у дизайні. Також інструмент має бібліотеку готових елементів, таких як ікони, шрифти, фотографії, які можна використовувати для створення унікальних композицій. Інша важлива особливість — це можливість працювати в команді, спільно розробляти та редагувати проекти. Це робить Crello ефективним інструментом для колективної творчості та комунікації. Crello також надає можливість безпосередньо публікувати створений контент в соціальних мережах або завантажувати його для подальшого використання. В цілому, Crello вражає своєю простотою використання, розширеними можливостями налаштувань та доступністю для широкого кола користувачів, роблячи його потужним інструментом для графічного дизайну. [20]

Adobe Photoshop є найвизнанішим та найпоширенішим програмним продуктом для редагування та обробки графічного контенту. Завдяки своїй потужній функціональності, Photoshop використовується професіоналами у сферах дизайну, фотографії та графічного мистецтва. Програма дозволяє створювати та редагувати зображення, використовуючи шари, фільтри, інструменти малювання та інші просунуті можливості. Вона також підтримує роботу з різними форматами файлів і дозволяє зберігати готові роботи для використання в інших програмах. [21]

Canva є веб-сервісом для дизайну, який надає можливості створення графічних елементів, від презентацій та соціальних мереж до ілюстрацій та логотипів. Однією з його ключових особливостей є велика бібліотека шаблонів для різних відомостей, що дозволяє користувачам легко адаптувати їх до своїх потреб. Canva також вражає спрощеним інтерфейсом та можливістю працювати в реальному часі з іншими користувачами. [22]

Sketch — це програма для макетування та дизайну інтерфейсів, спеціально розроблена для роботи з веб-дизайном та мобільними додатками. Зручний інтерфейс та інтуїтивне управління роблять її популярним інструментом серед дизайнерів. Sketch надає можливість створення векторних зображень, використовуючи

різноманітні інструменти для швидкого та ефективного проектування. Крім того, програма має розширені функції для спільної роботи та інтеграції з іншими інструментами розробки. [23]

1.2.10. Платформи для навчання

Moodle — це відкрите програмне забезпечення для створення електронних освітніх платформ та віртуальних навчальних середовищ. Розроблено в 2002 році, Moodle став популярним інструментом для дистанційного навчання та ведення курсів онлайн. Платформа надає різноманітні можливості, такі як створення курсів, завдань, тестів, форумів для обговорення та співпраці. Moodle дозволяє вчителям персоналізувати курси, використовуючи різні ресурси, відео, аудіо та інтерактивні завдання. Заснований на принципах відкритості та гнучкості, Moodle відзначається можливістю розширення за допомогою плагінів та інтеграцією з іншими популярними інструментами. Moodle використовується в освітніх установах різного рівня, від шкіл до вищих навчальних закладів, для надання студентам доступу до навчальних ресурсів у будь-якому місці та в будь-який час, що робить його важливим інструментом у сучасному освітньому середовищі. [24]

Google Classroom — це інтегрована платформа для ведення електронного навчання та співпраці в середовищі Google Workspace for Education. Запущений у 2014 році, Classroom розроблений для полегшення взаємодії між вчителями та учнями в онлайн-середовищі. Платформа дозволяє вчителям створювати класи, розміщувати матеріали, створювати завдання та проводити тести. Учні можуть зручно доступатися до завдань, надсилати роботи та спілкуватися з вчителями та однокласниками. Google Classroom інтегрується з іншими сервісами Google, такими як Google Drive, Docs та Hangouts, забезпечуючи високий рівень спільної роботи та зручний обмін інформацією. Цей інструмент особливо актуальний для дистанційних та гібридних форм навчання, роблячи онлайн-освіту більш ефективною та доступною. Google Classroom є популярним серед шкіл, коледжів та інших освітніх установ для організації та ведення електронних курсів. [25]

В табл. 1.1 наведені короткий опис розглянутих рішень, а також їхні переваги, недоліки та характеристики.

Таблиця 1.1. — Коротка характеристика існуючих рішень

Назва продукту	**Короткий опис**	**Переваги**	**Недоліки**	**Характеристики**
Todoist	Засіб управління завданнями з простим інтерфейсом	Простий та зручний інтерфейс	Обмежені можливості у безкоштовній версії	Ефективне управління завданнями, можливість створення списків та підзадач.
Wunderlist	Попередня версія Todoist, припинена в 2020 році	Легкий у використанні	Зупинено підтримку, перейшовши до Microsoft To-Do	Зручний інтерфейс, колаборація над завданнями, простота використання.
Microsoft To-Do	Засіб для керування завданнями, інтегрований з Microsoft	Інтеграція з іншими продуктами Microsoft	Обмежені можливості порівняно з іншими завданнями	Інтеграція з Microsoft, спільний доступ до завдань.
Trello	Платформа для керування завданнями за допомогою дошок	Гнучка система дошок і списків	Потребує часу для освоєння	Організація завдань у вигляді дошок, спільна робота над проектами.
RescueTime	Відстеження часу для підвищення продуктивності	Відстеження часу для продуктивності	Деякі функції доступні лише у платній версії	Моніторинг витрат часу на різні завдання, аналіз продуктивності.
	Додаток для збереження концентрації та управління часом	Сприяє концентрації та управлінню часом	Деякі користувачі можуть вважати залишок відволіканням	Заохочення фокусу через вирощування віртуального лісу.
Evernote	Засіб для зберігання та організації нотаток та ідей	Можливість зберігання різних типів контенту	Обмежена безкоштовна версія	Збереження текстових, аудіо та візуальних нотаток.
OneNote	Блокнот Microsoft для створення та організації нотаток	Інтеграція з іншими продуктами Microsoft	Може виглядати складним для новачків	Записи, малюнки та організація нотаток у вигляді блокнота.
Notion	Засіб для заміток, планування та спільної роботи	Гнучкий та потужний інструмент для заміток	Деякі користувачі вважають за складним	Створення багатофункціональних сторінок для організації роботи.
Khan Academy	Безкоштовна платформа з навчальними відео та вправами	Безкоштовні онлайн-уроки з різних предметів	Обмежена можливість індивідуалізації навчання	Інтерактивні уроки та вправи для самостійного навчання.

Продовження таблиці 1.1

Coursera	Платформа для доступу до онлайн-курсів від університетів	Великий вибір онлайн-курсів від університетів	Платно для деяких спеціалізованих курсів	Онлайн-курси від провідних університетів та компаній.
Grammarly	Інструмент для перевірки граматики та стилю тексту	Перевірка граматики та стилю тексту	Деякі функції доступні лише у преміум-версії	Корекція граматичних та стилістичних помилок у тексті.
Zotero	Менеджер бібліографії для організації джерел та цитат	Управління бібліографією та джерелами	Вимагає часу для освоєння	Збереження та організація джерел для наукових робіт.
Asana	Засіб для управління проектами та завданнями	Потужний інструмент для управління проектами	Може здаватися складним для новачків	Створення, відстеження та оцінка завдань у групових проектах.
Google Scholar	Пошук наукових статей та публікацій	Пошук наукових статей та публікацій	Обмежена функціональність порівняно з іншими	Пошук наукової інформації у відкритому доступі.
JSTOR	Цифрова бібліотека з доступом до наукових статей та книг	Доступ до великої кількості наукових статей	Потрібна підписка для повного доступу	Збереження та використання наукових матеріалів.
Google Classroom	Платформа для управління класами та навчанням в середовищі Google	Інтеграція з Google Apps для освіти	Може бути обмеженим для деяких типів курсів	Управління класами та завданнями через сервіси Google.
Moodle	Відкритий інструмент для створення віртуальних навчальних середовищ	Відкритий інструмент для управління навчанням	Вимагає технічних знань для налаштування	Створення та управління віртуальними класами та матеріалами.

Подана таблиця ілюструє розмаїтість існуючих рішень для управління завданнями та навчанням. Очевидно, що на сьогоднішній день не існує єдиного застосунку, який повністю задовольнив би всі потреби здобувачів вищої освіти. Різноманітність додатків вказує на те, що студенти та викладачі шукають різні функціональність та можливості, але поки що не знаходять універсального інструменту, який задовольнив би всі їхні потреби.

Серед розглянутих додатків Google Classroom та Moodle виділяються найбільшим функціоналом та відзначаються як уніфіковані середовища для навчання.

Вони не обмежуються простою організацією завдань, але й надають інтегровані інструменти для спілкування, спільної роботи та управління класами. Це створює зручне та повне середовище для навчання, сприяючи ефективній комунікації та взаємодії.

Дивлячись на дані таблиці, важливо зауважити, що жоден з розглянутих додатків не пропонує можливості використання штучного інтелекту. Такі технології можуть інтегруватися для персоналізації навчання, аналізу продуктивності та надання індивідуальних рекомендацій, що є ключовим для підтримки здобувачів освіти. Відсутність цього аспекту усуває можливість автоматизації та розвитку навчального процесу в більш інтелектуальний спосіб.

У підсумку, результати проведеного аналізу підкреслюють потребу у створенні інтегрованих, універсальних платформ, які об'єднують усі аспекти навчання та використовують передові технології, зокрема штучний інтелект, для максимальної ефективності та підтримки користувачів.

2. РОЗРОБКА ІНТЕЛЕКТУАЛЬНОГО ПОМІЧНИКА ПІДТРИМКИ НАВЧАЛЬНОГО ПРОЦЕСУ ЗА ДОПОМОГОЮ ШТУЧНОГО ІНТЕЛЕКТУ

2.1. Інтелектуальний помічник

Інтелектуальний помічник є програмним або апаратним засобом, який використовує штучний інтелект для надання користувачам різноманітних послуг та вирішення завдань. Цей інноваційний інструмент стає все більш популярним в різних сферах, допомагаючи людям автоматизувати рутинні завдання, отримувати інформацію та взаємодіяти з технологією більш ефективно.

Інтелектуальні помічники можуть мати різноманітні форми, включаючи віртуальних асистентів на смартфонах, спеціалізовані програмні рішення для бізнесу, чат-боти на веб-сайтах та інші. Однією з ключових характеристик інтелектуальних помічників є їхній здатність взаємодіяти з користувачем природним мовленням, що робить комунікацію більш зручною та ефективною.

Перш за все, інтелектуальні помічники допомагають у вирішенні завдань, які вимагають обчислень, аналізу даних та обробки інформації. Вони можуть виконувати різноманітні завдання — від нагадування про події та замовлення товарів до виконання складних обчислень та розв’язання проблем. Завдяки розширеній обробці природної мови, вони можуть розуміти контекст і намагатися відповісти на питання користувачів адекватно.

Крім того, інтелектуальні помічники можуть виявити великий потенціал у покращенні продуктивності та організації робочого часу. Вони можуть надавати поради з управління часом, допомагати встановлювати пріоритети та навіть автоматизувати частину робочих процесів. Це особливо корисно для бізнесу та фрілансерів, які шукають шляхи оптимізації своєї діяльності.

Однією з важливих сфер використання інтелектуальних помічників є освіта. Вони можуть служити вчителями, які надають індивідуалізовану допомогу у навчанні,

створюючи персоналізовані програми для студентів та допомагаючи їм засвоювати матеріал більш ефективно. Інтелектуальні помічники також можуть стимулювати інтерактивне навчання та надавати доступ до різноманітних ресурсів для самостійного вивчення.

Насамкінець, важливо враховувати етичні та конфіденційність питання, пов'язані з використанням інтелектуальних помічників. Збирання та збереження даних користувачів повинні бути безпечними та дотримуватися законодавства про захист приватності.

У сучасному світі інформаційних технологій, розвиток педагогіки та навчальних підходів вимагає нових інструментів та рішень для ефективної організації навчального процесу. Одним із таких інноваційних рішень є чат-бот, який виступає у ролі надійного помічника для студентів, викладачів та адміністрації навчальних закладів.

2.1.1. Введення в Сутність Чат-бота:

Чат-бот – це програмне забезпечення, здатне автоматично взаємодіяти з користувачами через текстові повідомлення. У сфері освіти, чат-бот може виконувати широкий спектр завдань, спрощуючи комунікацію та забезпечуючи доступ до різноманітної інформації.

2.1.2. Миттєвий Зв'язок та Відповіді на Запитання:

Перша та основна функція чат-бота – надання миттєвих відповідей на запитання студентів та викладачів. Чат-бот може надати інформацію про розклад занять, аудиторії, викладачів, а також вирішувати практичні питання, такі як зміна пари або перенесення занять. Це сприяє швидкому та зручному отриманню необхідної інформації, виключаючи витрати часу на пошук.

2.1.3. Використання Штучного Інтелекту:

Чат-бот, заснований на технологіях штучного інтелекту (ШІ), може

вдосконалювати свої відповіді в залежності від контексту та історії взаємодії з користувачем. Він може навчатися розпізнавати індивідуальні особливості кожного користувача та надавати персоналізовані поради та рекомендації.

2.1.4. Ведення Розкладу та Виставлення Завдань:

Чат-бот може слугувати інструментом для ведення розкладу занять, а також виставлення завдань та дедлайнів. Студенти можуть отримувати повідомлення про найближчі події, завдання та оновлення в розкладі. Це полегшує організацію часу та планування, забезпечуючи систематизацію навчального процесу.

2.1.5. Керування Процесом Навчання:

Чат-бот може слугувати інструментом для керування процесом навчання. Він може стежити за активністю студентів, визначати прогалини в знаннях та надавати всю необхідну інформацію в короткий проміжок часу. Такий підхід сприяє індивідуалізації навчання та підвищенню ефективності освітнього процесу.

Чат-бот в сфері освіти – це не просто інструмент для надання інформації. Він стає справжнім помічником, що активно взаємодіє з користувачами, полегшуючи навчальний процес та сприяючи розвитку сучасного освітнього середовища. З використанням ШІ та персоналізованих підходів, чат-бот стає необхідним елементом для забезпечення якісної, ефективної та доступної освіти.

Узагальнюючи, можна сказати, що інтелектуальні помічники відкривають перед нами нові можливості в автоматизації та оптимізації різних сфер життя. Вони розвиваються разом із штучним інтелектом та технологічним прогресом, надаючи користувачам інноваційні та зручні інструменти для вирішення щоденних завдань.

2.2. Необхідність залучення штучного інтелекту при розробці інтелектуального помічника

Розробка інтелектуальних помічників і надання їм функціоналу штучного інтелекту стає необхідністю у світі, який швидко розвивається та динамічно

змінюється. Штучний інтелект (ШІ) вносить ключовий внесок у покращення функціоналу та ефективності інтелектуальних помічників, забезпечуючи їхню здатність аналізувати дані, розпізнавати зв’язки та навіть навчатися з часом.

Однією з ключових переваг використання ШІ у розробці інтелектуальних помічників є їхня здатність розуміти та обробляти природну мову. Діалоговий інтерфейс, який базується на штучному інтелекті, дозволяє користувачам спілкуватися з помічником більш природно та зручно. Спроможність розпізнавання контексту, інтонації та інших аспектів мовлення дозволяє інтелектуальному помічнику точніше розуміти потреби користувача та взаємодіяти з ним на більш високому рівні.

Інтелектуальні помічники, обладнані штучним інтелектом, мають великий потенціал у розумінні та аналізі великої кількості інформації. Завдяки алгоритмам машинного навчання, вони можуть швидко переглядати, класифікувати та аналізувати дані, що робить їх ефективними в розв’язанні завдань, пов’язаних з обробкою великих обсягів інформації. Це може бути корисним у різних галузях — від бізнесу та науки до освіти та медицини.

Ще однією важливою характеристикою, яку ШІ вносить у розробку інтелектуальних помічників, є їхня здатність навчатися. Алгоритми машинного навчання дозволяють помічникам покращувати свої навички та результативність з часом, використовуючи зібрані дані та отриманий досвід. Це означає, що інтелектуальні помічники можуть адаптуватися до змінних умов та вдосконалювати свою роботу на основі нових даних, що забезпечує постійне удосконалення їхнього функціоналу.

У розробці інтелектуальних помічників також важливою є здатність до персоналізації. Штучний інтелект може аналізувати попередні взаємодії з користувачем, враховуючи його вподобання, стилі комунікації та інші параметри. Це дозволяє створювати індивідуалізовані рекомендації, поради та відповіді, що відповідають конкретним потребам кожного користувача.

Ще однією галуззю, де штучний інтелект виявляється незамінним, є розпізнавання образів та голосу. Інтелектуальні помічники можуть використовувати технології комп'ютерного зору та обробки аудіоданих для розпізнавання облич, предметів та голосу користувача. Це розширює їхні можливості і дозволяє ефективніше взаємодіяти з фізичним світом через камери та мікрофони.

Застосування штучного інтелекту у розробці інтелектуальних помічників також сприяє автоматизації рутинних завдань. Помічники можуть виконувати роботу, пов'язану з організацією розкладів, нагадуванням про важливі події та автоматизацією інших завдань, звільняючи час користувача для більш важливих та творчих задач.

Однак важливо враховувати етичні аспекти та питання конфіденційності при використанні штучного інтелекту у розробці інтелектуальних помічників. Збір, зберігання та обробка особистих даних повинні відповідати найвищим стандартам безпеки та захисту приватності користувачів.

2.2.1. Огляд існуючих рішень

У сучасному цифровому світі, де технології активно використовуються у всіх сферах, важливим елементом стає взаємодія з інформацією та ресурсами шляхом інтелектуальних чат-ботів. Однією з передових платформ, яка спрощує цей процес та забезпечує зручність та ефективність, є Chatbase. Використовуючи Generative AI та алгоритми обробки природного мови (NLP), ця платформа дозволяє створювати інтелектуальних чат-ботів, які здатні розуміти та інтерпретувати запитання користувачів, забезпечуючи точні та оперативні відповіді.

Chatbase вирізняється не лише завдяки своїй легкості використання, але й можливістю інтеграції на веб-сайти через API або в якості віджета. Generative AI, що застосовується в платформі, надає можливість створювати ботів, які адаптуються до різних сценаріїв взаємодії з користувачами, що робить їх більш гнучкими та універсальними. Покращена функціональність включає в себе ведення розкладу та

створення завдань, допомагаючи у плануванні та організації навчального процесу.

Chatling використовує технологію Generative AI для створення чат-ботів, які можна навчати на вмісті вашого веб-сайту, документах та інших ресурсах автоматично. Ця платформа ставить за мету надати користувачам можливість миттєво спілкуватися з ботами, які завдяки навчанню на вмісті веб-сайту швидко та точно надають відповіді на запитання.

Tidio, використовуючи свою платформу Lyro, вирізняється від інших завдяки можливості збереження концентрації та підтримки управління часом користувачів. Цей інтелектуальний помічник швидко навчається поширеним запитанням та надає персоналізовану допомогу, сприяючи підвищенню продуктивності та концентрації.

Revechat відкриває нові можливості для користувачів, дозволяючи їм створювати власні чат-боти для підтримки клієнтів. Автоматична підтримка основних запитань та можливості аналітики і звітності роблять цю платформу ідеальним інструментом для покращення якості обслуговування та збільшення продажів.

Botsify є іншою цікавою платформою, яка дозволяє створювати чат-ботів для різних каналів, включаючи WordPress, WhatsApp, Telegram. Можливість передачі запитань між ботами та представниками служби підтримки допомагає забезпечити єдність комунікації та покращити взаємодію.

MobileMonkey визначається як одна з передових платформ чат-ботів, спрямованою на розширення маркетингових можливостей та покращення взаємодії з клієнтами. Однією з ключових особливостей цієї платформи є її багатоканальний підхід, що дозволяє компаніям долучатися до своєї аудиторії через різноманітні канали зв'язку, такі як Instagram, Facebook Messenger і SMS.

MobileMonkey забезпечує бізнесам потужні інструменти для взаємодії з клієнтами у режимі реального часу. Завдяки інтеграції з популярними платформами, ця платформа робить можливим створення і налаштування чат-ботів для ефективного ведення діалогу з аудиторією.

EBI.AI пропонує розширену розмовну платформу штучного інтелекту, яка

переводить взаємодію з чат-ботами на новий рівень. На відміну від стандартних рішень, ця платформа дозволяє створювати помічників, які здатні виконувати різноманітні завдання. Вона надає користувачам доступ до різноманітних інструментів, таких як живий чат та інтеграція з іншими системами, для повноцінної настройки розмовного штучного інтелекту.

Giosg - це ефективний інструмент для перетворення трафіку за допомогою ботів, спрямований на генерацію потенційних клієнтів. За допомогою розмовного маркетингу та продажів, ця платформа дозволяє залучити найцінніших потенційних клієнтів в Інтернеті. Вона перевершує статичні форми генерації потенційних клієнтів, що робить її чотириразово ефективнішою. Giosg вже успішно використовують понад 1,200 компаній, що свідчить про його високий ступінь ефективності та визнання в бізнес-середовищі.

Chatfuel - це інтуїтивно зрозуміла платформа для створення чат-ботів штучного інтелекту для Facebook Messenger та Instagram. Завдяки інтерфейсу перетягування, Chatfuel забезпечує легкість у впровадженні технології чат-ботів, що робить її ідеальним вибором для тих, хто хоче вперше впровадити цю технологію у своїй компанії. Без потреби в програмуванні, вона надає користувачам простоту та ефективність у створенні та управлінні чат-ботами для платформ Facebook та Instagram.

ProProfs Chat - це інструмент, спрямований на автоматизацію продажів та підтримки в режимі реального часу. Спеціально розроблений для компаній, що потребують ефективні рішення для спілкування з відвідувачами їхніх веб-сайтів, ProProfs Chat дозволяє створювати власні чат-боти. Це збільшує конверсію, покращує продажі та автоматизує комунікації з клієнтами, забезпечуючи комплексний підхід до покращення взаємодії з відвідувачами в режимі реального часу.

Використання інтелектуальних чат-ботів у сферах освіти та бізнесу відкриває нові можливості для зручної та ефективної комунікації. Ці платформи, застосовуючи передові технології штучного інтелекту та обробки природного мови, дозволяють

автоматизувати процеси, підвищити продуктивність та покращити якість обслуговування користувачів.

В табл. 2.1 наведені короткий опис розглянутих рішень, а також їхні переваги, недоліки та характеристики.

Таблиця 2.1. — Коротка характеристика існуючих рішень

Чат-бот	**Переваги**	**Недоліки**	**Характеристики**
Chatbase	Проста інтеграція, навчання за допомогою машини.	Відсутність продуктивності, обмежені можливості.	Аналітика чат-ботів, штучний інтелект.
Chatling	Вивчення вмісту веб-сайтів, динамічна адаптація.	Обмежені функції у безкоштовній версії.	Навчання чат-ботів на основі контенту, ШІ.
Lyro (Tidio)	Штучний інтелект для спілкування, персоналізована допомога.	Простий у застосуванні, без навчання.	Активація менше ніж за 3 хвилини, підтримка 24/7.
Revechat	Створення чат-ботів без програмування, цілодобова підтримка.	Можливості обробки повторюваних запитань.	Створення власних ботів, розширені звіти та аналітика.

Продовження таблиці 2.1

Botsify	Багатоканальна платформа, передача запитів між ботами та людьми.	Вимагає кодування для розширених відповідей.	Доступно понад 190 мов, 100+ інтеграцій.
MobileMonkey	Розширення маркетингу на Instagram, Facebook Messenger і SMS.	Залежність від каналів соціальних мереж.	Багатоканальна структура, розширені можливості маркетингу.
EBI.AI	Вдосконалена розмовна платформа ШІ, виконання різноманітних завдань.	Інтеграція з іншими системами.	Розмовний штучний інтелект, інтеграція, налаштування.
Giosg	Генерація потенційних клієнтів через розмовний маркетинг.	Обмежена ефективність для певних бізнесів.	4 рази ефективніший, залучення цільової аудиторії.
Chatfuel	Створення чат-ботів для Facebook Messenger та Instagram без кодування.	Обмежені функції у безкоштовній версії.	Простота впровадження, допомагає вперше використовувати технологію.
ProProfs Chat	Програмне забезпечення для чату в реальному часі, автоматизація продажів.	Вимагає активного участі для оптимального використання.	Чат-боти для продажів, підтримка, створення власних ботів.

2.2.2. Обгрунтування необхідності створення власного чат-боту

Сучасний освітній процес потребує інновацій та ефективних інструментів для навчання та взаємодії зі студентами. У цьому контексті створення власного чат-бота для освітніх потреб стає ключовим етапом у впровадженні передових технологій у сферу навчання.

Одні з основних обмежень готових платформ чат-ботів полягають у їхній загальній функціональності та неперсоналізованих можливостях. Комплексність освітнього процесу вимагає індивідуального підходу, створення унікальних сценаріїв та персоналізації взаємодії з учнями.

Створення власного чат-бота дозволяє адаптувати його під конкретні потреби освітнього закладу чи курсу. Персоналізовані сценарії, спеціальні завдання та інтерактивні можливості роблять чат-бота ефективним інструментом для залучення студентів у навчальний процес.

Окрім того, власний чат-бот може служити не лише інструментом для навчання, але й засобом для підтримки студентів. Він може відповідати на запитання, надавати рекомендації щодо матеріалів, допомагати в організації розкладу та навіть вести індивідуальний моніторинг успішності.

Також важливо відзначити, що чат-бот може сприяти розвитку навичок самостійності та відповідальності учнів. Взаємодія з інтелектуальним помічником, який завжди доступний для консультацій та надання інформації, може підвищити рівень самостійності студентів та підтримати їх у навчанні.

У підсумку, створення власного чат-бота для освітнього процесу виявляється не лише перевагою, але й необхідністю. Він відкриває нові можливості для інноваційного навчання, індивідуалізації та підтримки студентів, що робить його важливим інструментом для сучасних освітніх установ.

2.2.3. Відмінність та функціонал

Інноваційний чат-бот для навчання вирізняється великим спектром функціональності, яка не тільки полегшує, але й трансформує навчальний процес. Його унікальні особливості забезпечують ефективну інтерактивність та сприяють створенню умов для успішного засвоєння навчального матеріалу.

Однією з ключових переваг чат-бота є швидкий та миттєвий доступ до відповідей на запитання. Студенти та викладачі можуть отримати необхідну інформацію в реальному часі, що сприяє оперативній комунікації та підвищує продуктивність навчального процесу.

Додатково, чат-бот веде розклад занять та надсилає нагадування про важливі події, створюючи умови для ефективного планування часу студентів та викладачів.

Також він допомагає виставляти завдання та дедлайни, сприяючи систематизації навчального процесу.

Оптимізація комунікації - ще один ключовий аспект, який чат-бот вирішує. Єдиний канал спілкування студентів та викладачів робить взаємодію більш структурованою та зменшує ризик пропуску важливої інформації.

Враховуючи всі ці переваги, чат-бот стає необхідним інструментом для досягнення високої якості навчання. Його інтелектуальні можливості, швидкість реакції та адаптабельність роблять його ідеальним союзником як для студентів, так і для викладачів у сучасному освітньому процесі.

2.3. Сценарій реалізації

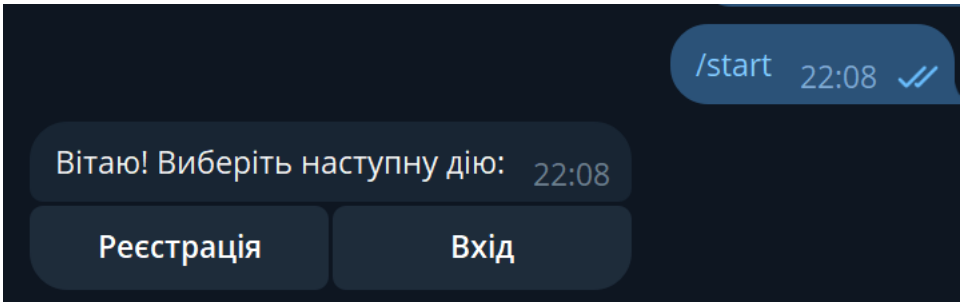
Крок 1: Запуск та підключення.

1.1. Користувач додає Помічника до свого списку контактів у Телеграмі.

1.2. Помічник вітає Користувача та пропонує реєстрацію або вхід для студентів і викладачів.

Крок 2: Реєстрація/Вхід.

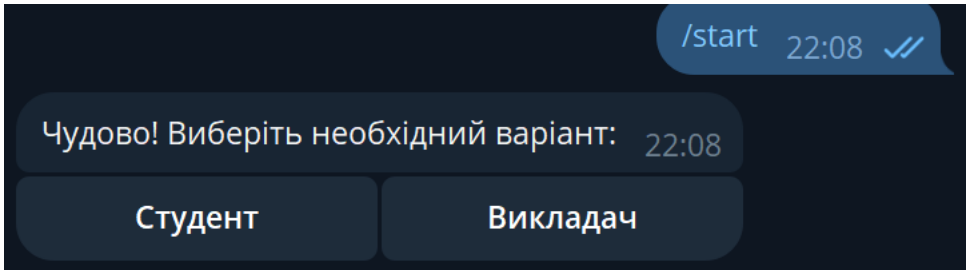
2.1. Користувач обирає «Реєстрація» та вводить необхідну інформацію (прізвище, ім'я, електронну пошту, студент/викладач).



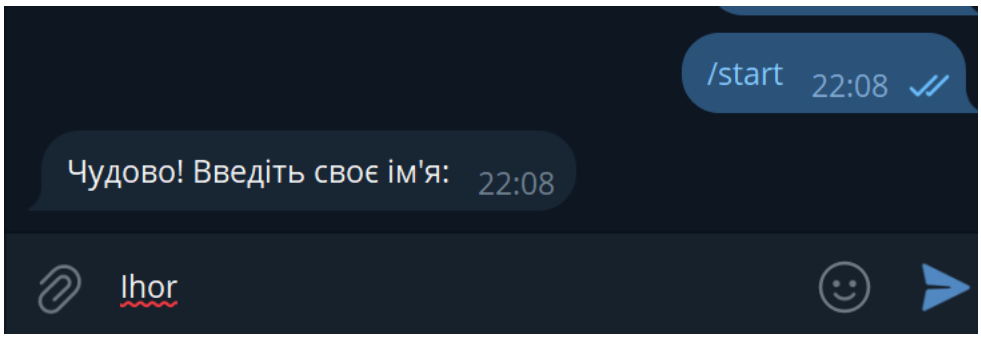
2.2. Якщо Користувач вже зареєстрований — «Вхід» з введенням логіну та пароля.

Крок 3: Обрання Ролі (Студент/Викладач).

3.1. Користувач обирає свою роль (студент чи викладач).



3.2. Відповідно до обраної ролі, Помічник надає відповідні функції.



Крок 4: Функції для Студентів.

4.1. Розклад занять.

Користувач може:

а) переглядати свій розклад занять;

б) перейти за посиланнями;

в) отримати інформацію про зміни у розкладі (сповіщення).

4.2. Нагадування про завдання.

Студент отримує сповіщення про невиконані завдання та дедлайни.

4.3. Підтримка при вивченні матеріалів.

Помічник може надати:

а) додаткову інформацію або відповісти на запитання;

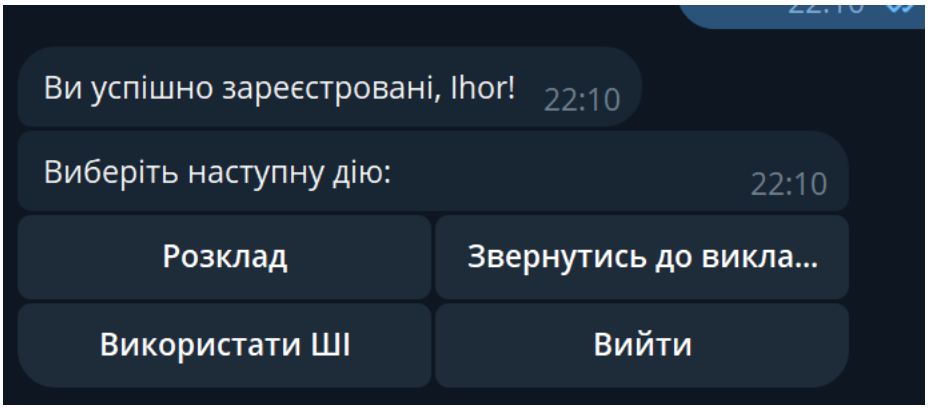
б) репозитарій, методички;

в) посилання на літературу, на moodle/classroom.

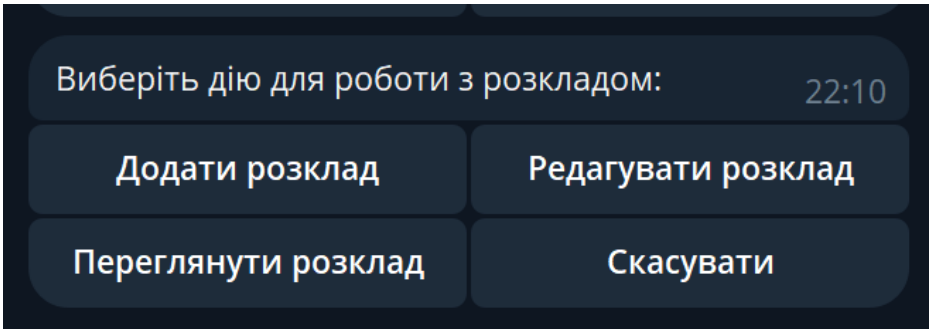
Крок 5: Функції для Викладачів.

5.1. Розклад занять.

Викладач може переглядати та оновлювати розклад занять.



5.2. Створення завдань і дедлайнів.



Викладач може створювати завдання для студентів та встановлювати дедлайни.

5.3. Взаємодія зі студентами

Викладач може:

а) отримувати питання від студентів та надавати відповіді;

б) отримувати запити на навчально-методичні матеріали (за потребою студента).

Крок 6: Зворотний зв’язок та підтримка.

6.1. Відгуки та оцінки.

Користувач може залишати відгуки та оцінки певних функцій.

6.2. Питання та взаємодія.

Користувач може ставити питання помічнику та отримувати індивідуальну підтримку.

Крок 7: Вихід.

7.1. Користувач може вибрати «Вихід», щоб завершити сеанс роботи з

Помічником.

7.2 Помічник дякує Користувачеві за використання та пропонує повернутися в будь-який час.

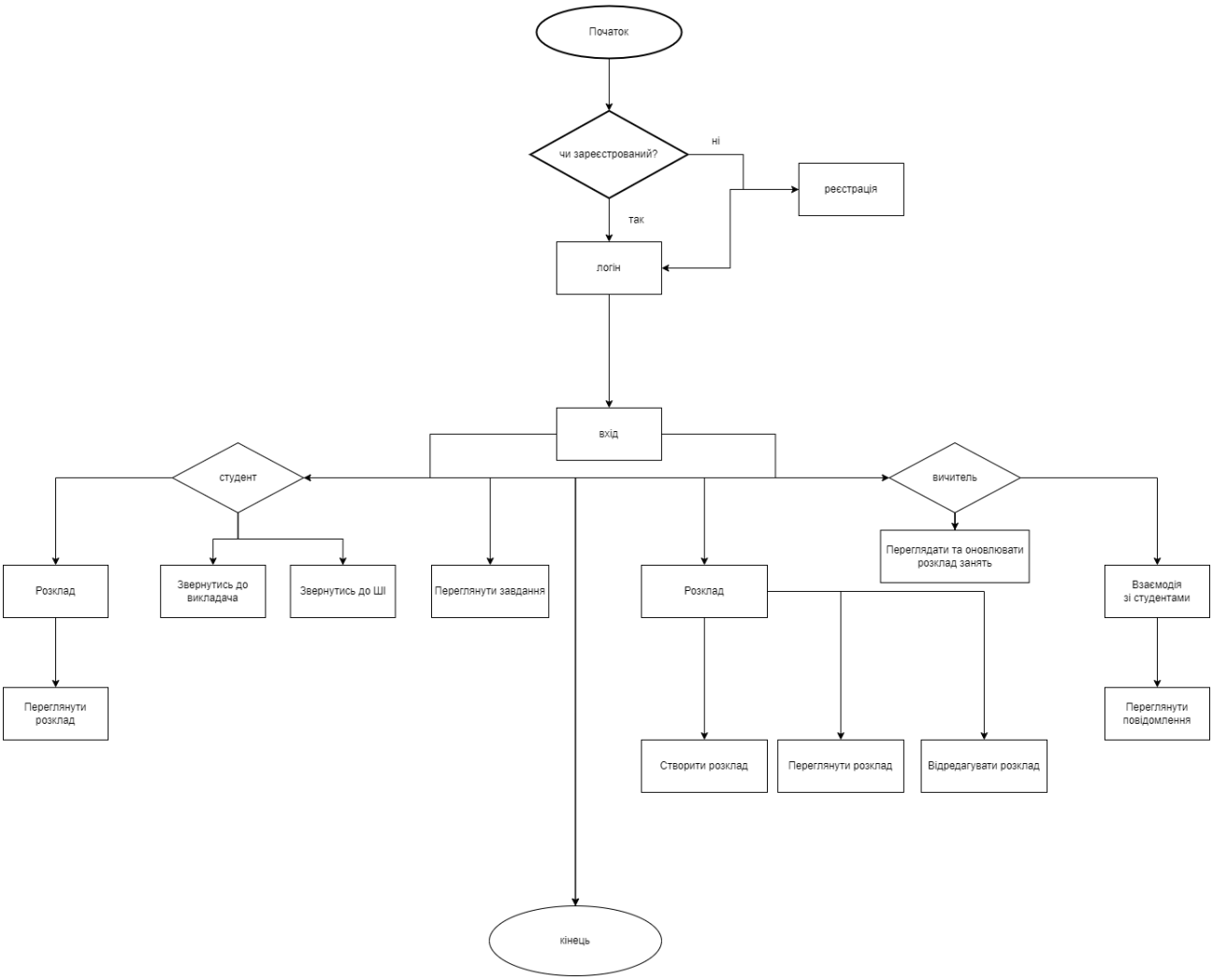


Рисунок 1.1. Блок схема програмного застосунку

Блок-схеми є потужним інструментом для візуалізації та аналізу складних процесів чи систем. В даному випадку, блок-схема інтеракції інтелектуального помічника у Телеграмі відіграє важливу роль у розумінні послідовності подій та взаємодій між користувачем і системою.

Переваги використання блок-схем включають їхню здатність до візуалізації кроків та логіки процесу, сприяючи кращому розумінню взаємодії між елементами

системи. Це особливо корисно для ілюстрації складних функціональностей, таких як реєстрація, вхід, вибір ролі та інші, що виникають у взаємодії з інтелектуальним помічником (див. Рисунок 1.1).

Переваги цієї блок-схеми полягає в чіткому відображенні кроків взаємодії та послідовності дій, починаючи від функції “start” у Телеграмі та подолання етапів реєстрації чи входу, і аж до закінчення сеансу. Вона віддзеркалює логічний порядок дій як для студентів, так і для викладачів, що спрощує їхнє взаєморозуміння та використання системи.

Блок-схема не лише полегшує розуміння послідовності розробки програмного застосунку, але й служить ефективним інструментом для вдосконалення та оптимізації взаємодії користувача з інтелектуальним помічником. Ця візуалізація є важливим етапом у розробці та вдосконаленні технологічних інтерфейсів, роблячи їх доступнішими та зручнішими для різних користувачів.

На самій блок-схемі проілюстровано покрокове використання програмного застосунку. Починаючи з функції “start”, у користувача далі є вибір між логіном та реєстрацією. В залежності від студента або викладача, далі функціонал розділяється. Для студента доступно перегляд розкладу, звернення до викладача, використання ШІ та перегляд завдань. У цей час викладачу доступне створення та редагування розкладу, оновлення завдань та перегляд повідомлень.

3. ОПИС ТЕХНОЛОГІЧНИХ АСПЕКТІВ ПРОГРАМНОГО ЗАСТОСУНКУ

3.1. Обґрунтування вибору мови програмування. Порівняння з іншими мовами

Вибір мови програмування – це завдання, яке потребує уважної аналізу та розгляду різних аспектів. У випадку розробки телеграм-бота, мова програмування впливає на ряд ключових аспектів, таких як продуктивність, зручність розробки, підтримка спільноти та інші.

Python, обраний для цього завдання, демонструє свої переваги на кількох рівнях. Його простота та читабельність сприяють швидкій розробці коду, що особливо важливо для телеграм-ботів, де важливо оперативно реагувати на повідомлення та команди користувачів.

Багатий екосистема бібліотек Python забезпечує розробникам доступ до готових рішень для різних аспектів роботи бота. Наприклад, використання бібліотеки python-telegram-bot полегшує взаємодію з Telegram API та забезпечує необхідні засоби для реалізації функціональності бота.

При виборі мови програмування для телеграм-бота, розглядання різних аспектів може визначити успішність проекту. Python, завдяки своїй простоті та лаконічності коду, стає частим вибором для швидкого розгортання функціональності та легшого супроводження. Його активна спільнота та розгалужена екосистема бібліотек, зокрема python-telegram-bot, дозволяють розробникам швидко і ефективно реалізовувати функціональність бота.

У порівнянні з JavaScript (Node.js), Python може виглядати менш асинхронним, але його код є більш інтуїтивно зрозумілим для багатьох розробників. В той час як Node.js дозволяє ефективно обробляти багато асинхронних операцій, він може здаватися складнішим для новачків та тим, хто не використовує асинхронний код регулярно.

Ruby, зі своїм чистим синтаксисом і фреймворком Ruby on Rails, може бути привабливим вибором для розробників, які шукають елегантність та продуктивність. Але можливо, йому не вистарчить швидкості розробки, яку може забезпечити Python.

Java, яка відзначається своєю платформенною незалежністю та об'єктно-орієнтованим підходом, може бути важчою мовою для розробки телеграм-бота через обсяг коду та формальність. З іншого боку, Go (Golang) відома своєю високою швидкодією та простотою вивчення, але її екосистема може виявитися менш розвиненою.

У кінцевому підсумку, вибір мови програмування – це баланс між простотою розробки, ефективністю та особистими вподобаннями розробника. Важливо розглядати всі фактори та враховувати конкретні потреби проекту для забезпечення успіху в його реалізації.

В табл. 3.1 наведено порівняння розглянутих рішень, а також їхні переваги та недоліки.

Таблиця 3.1. — Порівняння існуючих рішень.

Мова програмування	**Переваги**	**Недоліки**
Python	Простота та зрозумілість коду	Швидкодія може бути меншою порівняно з Go
	Велика кількість бібліотек для ботів	Асинхронний код може бути менш зручним
	Активна спільнота та підтримка	Менша продуктивність порівняно з Go
JavaScript (Node.js)	Асинхронність, ідеально для обробки подій	Складність асинхронного програмування
	Велика кількість пакетів та модулів	Читабельність коду може страждати через колбеки
	Швидкодія завдяки V8 JavaScript engine	Відсутність глобального стану для збереження
Ruby	Чистий та елегантний синтаксис	Швидкодія може бути меншою порівняно з іншими

Продовження таблиці 3.1

	Ruby on Rails для швидкого розгортання	Потребує додаткових удосконалень у продуктивності
	Гарна спільнота та підтримка	Залежність від Ruby on Rails для повноцінного фреймворку
Java	Платформенна незалежність та об'єктно-орієнтований підхід	Більший обсяг коду порівняно з іншими
	Розвинена екосистема та багато інструментів	Час розробки може бути більшим
	Висока переносимість коду	Строгість типів може здаватися обтяжливою
Go (Golang)	Висока швидкодія та ефективність	Обмежена екосистема, менше готових бібліотек
	Простота вивчення та зрозумілий синтаксис	Відсутність деяких просунутих функцій, що є в інших мовах

3.2. Паттерни програмування задля розробки чат-боту

Паттерни програмування — це рекомендовані та вже визначені шаблони, які розробники можуть використовувати для вирішення типових завдань у програмуванні. Ці паттерни забезпечують структуру та стратегії для ефективного організації коду та вирішення специфічних проблем.

3.2.1. Event Handling

"Опрацювання подій" (Event Handling) - це паттерн програмування, який використовується для реалізації механізму відслідковування та обробки подій або сигналів в програмі. У контексті Python, це може включати в себе відслідковування подій користувача, системних подій, а також подій, які виникають у веб-додатках чи інших програмах.

Опрацювання подій важливо для створення реактивних та відгукових програм. Зазвичай, цей паттерн включає в себе такі елементи, як визначення типів подій, реєстрація слухачів (event listeners), та виклик обробників (event handlers) для відповіді на події.

Для реалізації опрацювання подій в Python, можуть використовуватися різні інструменти та бібліотеки, такі як Tkinter для графічних інтерфейсів, Flask/Django для веб-додатків, а також asyncio для асинхронних програм.

Основні кроки при роботі з опрацюванням подій включають в себе визначення подій, створення об'єктів подій, реєстрацію слухачів та визначення обробників для відповіді на події.

Паттерн опрацювання подій є ключовим для створення програм, які можуть реагувати на зміни в оточенні та взаємодіяти з користувачами чи іншими системами.

```
],
SELECTING_GROUP: [CallbackQueryHandler(select_group)],
SELECTING_GROUP_ACTION: [MessageHandler(filters.TEXT & ~filters.COMMAND, handle_selected_group)],
SELECTING_DAY: [MessageHandler(filters.TEXT & ~filters.COMMAND, waiting_for_day_of_week)],
ENTER_SUBJECT: [MessageHandler(filters.TEXT & ~filters.COMMAND, waiting_for_subject)],
ENTER_TIME: [MessageHandler(filters.TEXT & ~filters.COMMAND, waiting_for_time)],
ENTER_TEACHER: [MessageHandler(filters.TEXT & ~filters.COMMAND, waiting_for_subject)],
ENTER_ROOM: [MessageHandler(filters.TEXT & ~filters.COMMAND, waiting_for_room)],
AI.SELECTING_AI: [
                MessageHandler(filters.TEXT & ~filters.COMMAND, AI.CHATTING)],
```

Рисунок 3.1 Приклад використання паттерну Event Handling

3.2.2. State

Паттерн "Стан" (State) - це паттерн програмування, який дозволяє об'єкту змінювати свою поведінку при зміні внутрішнього стану. Головна ідея полягає в тому, щоб розгортати логіку обробки подій та переходів між станами в окремих класах, які представляють конкретні стани.

Коли об'єкт перебуває в різних станах, він може реагувати на одні й ті ж події по-різному залежно від поточного стану. Це робить код чистим та легко змінюваним, оскільки нові стани можуть бути додані без зміни логіки у існуючих станах.

Паттерн "Стан" корисний в ситуаціях, де об'єкт поводиться по-різному залежно від свого поточного стану, і коли кілька станів може впливати на його поведінку. Він сприяє підтримці великої кількості станів та забезпечує їх гнучке управління.

Основні елементи паттерну включають визначення класів для кожного стану, визначення інтерфейсу для всіх станів та реалізацію конкретних класів станів, які

взаємодіють з контекстом (об'єктом, який має внутрішній стан).

Цей паттерн часто використовується для моделювання складних систем, де об'єкти можуть переходити між різними станами та реагувати на події в залежності від свого поточного стану.

В реалізованому програмному додатку даний паттерн використовується для зміни поведінки об’єкту при зміні внутрішнього стану(див. Рисунок 3.2).

```python
async def main_menu_after_registration(update: Update, context: CallbackContext) -> str:
    query = update.callback_query
    await query.answer()

    option = query.data
    context.user_data['option'] = option

    logging.info(f"DEBUG: main_menu_after_registration - option: {option}")

    if not query:
        logging.error("Callback query is None.")
        return "MAIN_MENU"

    try:
        await query.answer()
    except Exception as e:
        logging.error(f"Error answering query: {e}")

    return "MAIN_MENU"
```

Рисунок 3.2 Приклад використання паттерну State

3.2.3. Command

Паттерн "Команда" (Command) - це паттерн програмування, який дозволяє інкапсулювати запитання або операцію у вигляді об'єкта. Він перетворює запитання в об'єкт, що дозволяє параметризувати клієнта з різними запитаннями, чергами або операціями та підтримує відмінність між викликом та виконанням запитань.

Головна ідея полягає в розширенні клієнта за допомогою об'єкта команди. Об'єкт команди містить саму команду та інформацію про її виконання. Це дозволяє легко розширювати та змінювати поведінку системи, а також підтримує відмінність між клієнтом та об'єктом, який обробляє запитання.

Паттерн "Команда" використовується для реалізації черги операцій, скасування та повторення операцій, логування запитань, а також реалізації транзакцій. Це сприяє розділенню класів, які ініціюють дії (клієнтів) від тих, які виконують дії (отримувачів).

Основні компоненти паттерну включають команди, клієнтів, отримувачів та інвокерів. Команда інкапсулює запитання, клієнт ініціює команду, отримувач виконує конкретну операцію, а інвокер ініціює команду та відповідає за її виконання.

В реалізованому програмному додатку даний паттерн використовується для параметризування об'єктів запиту, визначати, обробляти та ставити запити в чергу, а також підтримує відміну операцій (див. Рисунок 3.3).

```
3 usages  Ihor Chernov
async def waiting_for_subject(update: Update, context: CallbackContext) -> int:
    subject = update.message.text
    context.user_data['subject'] = subject

    await update.message.reply_text("Введіть номер кабінету (необов'язково):")
    return ENTER_ROOM


2 usages  Ihor Chernov
async def waiting_for_room(update: Update, context: CallbackContext) -> int:
    room = update.message.text
    context.user_data['room'] = room
    # спеціальний внутрішній імпорт
    from handlers import handle_data

    # Отримати ім'я користувача з контексту
    get_login_name = handle_data( data: "some_date", context)

    group_name = context.user_data['group_name']
    day_of_week = context.user_data['day_of_week']
    time_schedule = context.user_data['time']
    subject = context.user_data['subject']
    room = context.user_data['room']

    logging.info(f"DEBUG: waiting_for_room - teacher: {get_login_name}")

    teacher = session.query(User).filter_by(username=get_login_name).first()
    group = session.query(Group).filter_by(name=group_name).first()

    logging.info(f"DEBUG: waiting_for_room - group_name: {group_name}")
    logging.info(f"DEBUG: waiting_for_room - teacher: {teacher.username}")
```

Рисунок 3.3 Приклад використання паттерну Command

Ці паттерни сприяють створенню гнучких, легко розширюваних та підтримуваних програмних рішень, підвищуючи якість та розуміння коду.

В табл. 3.2 наведено порівняння паттернів проектування, а також їхній опис та застосування в телеграм-ботах.

Таблиця 3.2. — Порівняння паттернів проектування.

Паттерн проектування	Опис	Застосування в телеграм-ботах
Опрацювання подій	Обробка подій та реакція на них	Реакція на повідомлення та команди від користувачів
Становий	Визначення різних станів та їх зміна	Управління різними режимами взаємодії бота
Команда	Інкапсуляція запитань та дій	Реалізація команд та обробка користувацьких вказівок
Стратегія	Визначення сімейства алгоритмів	Динамічний вибір алгоритму обробки повідомлень

3.3. Середовище програмування

Середовище програмування на Python - це набір інструментів і ресурсів, які спрощують розробку програм на мові програмування Python. Python визначається своєю простотою та лаконічністю, і середовища розробки роблять процес створення програм ефективнішим та зручнішим для програмістів.

3.3.1. IDLE

Одним із популярних середовищ для розробки на Python є IDLE (Integrated Development and Learning Environment). IDLE (Integrated Development and Learning Environment) - це інтегроване середовище розробки, створене для мови програмування Python. Воно входить до стандартного комплекту поставки Python і надає зручне середовище для написання, тестування та відлагодження коду.

Основна мета IDLE - це надати програмістам простий та легкий інтерфейс для розробки на Python, який би відповідав як початківцям, так і досвідченим розробникам. У ньому ви знайдете текстовий редактор з можливістю виділення синтаксису, що полегшує написання коду. Також, IDLE включає в себе інтерактивну оболонку Python, яка дозволяє вам виконувати команди та тестувати код без необхідності створення окремих скриптів.

Важливою особливістю є можливість відлагодження, що включає встановлення точок зупинки, крок за кроком виконання коду та перегляд значень змінних. IDLE

також має інтерфейс для імпортування та використання різних модулів та бібліотек.

Додатково, IDLE дозволяє переглядати документацію та вбудовану довідку, що є корисним для роботи з різними функціями та модулями. Також, ви можете розширити функціональність IDLE, встановлюючи розширення (plugins) за вашими потребами.

Загалом, IDLE - це інструмент, який спрощує процес розробки на Python, забезпечуючи зручний та ефективний інтерфейс для програмістів на різному рівні навичок.

3.3.2. Jupyter Notebook

Jupyter Notebook є інтерактивним середовищем для виконання коду на мові програмування Python та інших мов. Він надає зручний спосіб об'єднати текст, код, графіки та інші елементи в одному документі, називаному "ноутбуком".

Однією з ключових особливостей Jupyter Notebook є можливість виконувати код по частинах, що дозволяє аналізувати та відлагоджувати його крок за кроком. Це робить середовище ідеальним для вивчення нових концепцій, виконання наукових обчислень та подальшого документування результатів.

У Jupyter Notebook можна використовувати різні типи клітинок, такі як клітинки з кодом, текстові клітинки, а також клітинки для вставки графіків, таблиць та інших візуальних елементів. Це робить його потужним інструментом для створення інтерактивних та візуально привабливих документів.

Jupyter Notebook також відомий своєю широкою підтримкою для наукових бібліотек, таких як NumPy, Pandas, Matplotlib та інших. Це робить його популярним в області науки про дані, машинного навчання та досліджень.

Завдяки можливості експорту в різноманітні формати (наприклад, HTML, PDF, або презентації слайдів), Jupyter Notebook дозволяє легко ділитися та представляти результати своєї роботи.

3.3.3. PyCharm

PyCharm - це інтегроване середовище розробки (IDE) від компанії JetBrains, спеціалізоване на роботі з мовою програмування Python. Це потужний інструмент, призначений для забезпечення зручності та ефективності розробки програмних продуктів.

Основні риси PyCharm включають в себе високий рівень інтеграції з мовою Python, що робить його ідеальним вибором для розробників, які працюють з цією мовою. Інтелектуальне автодоповнення коду, відлагодження в реальному часі та аналіз коду сприяють підвищенню продуктивності та якості розробки.

PyCharm також підтримує віртуальні середовища та системи керування версіями, сприяючи організації та управлінню проектами. Інтегрована система плагінів дозволяє розширити функціональність IDE залежно від потреб користувача.

Великою перевагою PyCharm є його підтримка великої кількості фреймворків, бібліотек та інструментів, що допомагає розробникам швидше і легше створювати різноманітні проекти. Інтерфейс користувача PyCharm є інтуїтивно зрозумілим, що робить його доступним як для початківців, так і для досвідчених програмістів.

PyCharm - це інструмент, який сприяє розвитку та підтримці проектів на Python, надаючи розробникам ефективне та комфортне середовище для їх творчості та інновацій.

3.3.4. Anaconda та Spyder

Anaconda - це платформа для наукового програмування та аналізу даних, яка включає в себе дистрибутив Python, низку наукових бібліотек та інструменти для управління пакетами. Вона розроблена для спрощення процесу налаштування середовища для роботи з даними та обчисленнями, зокрема в області машинного навчання та аналітики.

Spyder - це інтегроване середовище розробки, яке часто входить до складу Anaconda, спроектоване спеціально для роботи з науковими обчисленнями та

аналізом даних. Воно надає зручний інтерфейс для написання, відлагодження та тестування коду, а також підтримує відображення графіків та взаємодію з об'єктами в середовищі.

Разом, Anaconda та Spyder надають засоби для створення та управління віртуальними середовищами Python, що дозволяє ізолювати проекти та їхні залежності. Це особливо корисно в області аналізу даних, де часто використовуються різні бібліотеки та інструменти. Anaconda спрощує установку цих компонентів та дозволяє зосередитися на роботі з даними, а Spyder забезпечує потужне інтегроване середовище розробки для наукових обчислень.

3.3.5. Visual Studio Code

Visual Studio Code, або просто VSCode, є легким та потужним редактором коду від Microsoft, призначеним для роботи з різними мовами програмування, включаючи Python. Одна з його ключових переваг - це широкий спектр розширень, які дозволяють користувачам налаштовувати і розширювати функціоналість редактора згідно з їхніми потребами.

VSCode володіє інтуїтивно зрозумілим інтерфейсом та високою продуктивністю, що робить його популярним серед розробників на різних рівнях навичок. Інтеграція з системами керування версіями, автоматичне виведення типів, відлагодження та підтримка віртуальних середовищ - всі ці функції сприяють зручності та ефективності розробки.

VSCode підтримує роботу з різними мовами програмування завдяки розширюваному API, а його інтегрована консоль дозволяє виконувати команди та скрипти прямо в середовищі редактора. Це зроблює його відмінним вибором для розробки на Python, а також для інших завдань, таких як веб-розробка, робота з базами даних та інші.

Загальною відмінною рисою VSCode є його активна спільнота та постійне оновлення, що робить його сучасним та відповідає вимогам сучасного розробника.

Усі ці середовища розробки роблять програмування на Python приємним та продуктивним процесом, надаючи програмістам засоби для швидкого написання, тестування та оптимізації свого коду.

В табл. 3.3 наведено порівняння розглянутих рішень, а також їхні переваги та недоліки.

Таблиця 3.3. — Порівняння існуючих рішень.

Середовище	**Переваги**	**Недоліки**
IDLE	- Легко встановлюється, оскільки входить у стандарт Python.	- Основна функціональність, можливо, обмежена порівняно із іншими IDE.
	- Простий інтерфейс для початківців.	- Обмежені можливості відлагодження.
	- Інтерактивна оболонка для швидкого тестування.	
Jupyter Notebook	- Інтерактивне середовище для аналізу даних та наукових обчислень.	- Може викликати плутанину в коді при великих проектах.
	- Можливість використовувати текст, код та візуальні елементи в одному документі.	- Ускладнена організація коду на рівні файлів та структур проекту.
PyCharm	- Повноцінне інтегроване середовище розробки для Python.	- Великі вимоги до ресурсів системи.
	- Високий рівень інтеграції з Python та підтримка багатьох фреймворків.	- Не є безкоштовним (існує платна версія).
	- Широкий функціонал відлагодження та підтримка віртуальних середовищ.	
Anaconda	- Легка установка та управління пакетами.	- Займає більше місця на диску порівняно з іншими середовищами.
	- Вбудована підтримка наукових бібліотек.	- Може бути надмірною для тих, хто працює лише з Python.
Spyder	- Спеціально розроблене для наукового програмування.	- Інтерфейс не такий сучасний порівняно з іншими IDE.
	- Інтегрована консоль та підтримка IPython.	- Може бути менш зручним для великих проектів та загального використання.
VSCode	- Легкий та швидкий редактор з великою кількістю розширень.	- За замовчуванням потребує налаштування для роботи з Python.
	- Активна спільнота та постійні оновлення.	-Може бути менше спеціалізованим для наукових обчислень.

Вибір PyCharm для реалізації телеграм-бота обумовлений кількома ключовими аспектами. По-перше, це повноцінне інтегроване середовище розробки, спеціально

адаптоване для мови програмування Python, що надає ефективні та зручні інструменти для роботи з цією мовою.

По-друге, PyCharm володіє високою інтеграцією з різноманітними фреймворками, що може бути важливим у випадку, якщо ви плануєте використовувати який-небудь конкретний фреймворк для розробки телеграм-бота чи розширення його функціоналу.

Третій важливий момент - це зручна система відлагодження та підтримка віртуальних середовищ, що сприяє розробці та тестуванню коду відповідно до ваших потреб. Взагалі, PyCharm відомий своєю потужністю, інтуїтивно зрозумілим інтерфейсом та широким набором функцій, що робить його привабливим вибором для розробників Python.

4. РОЗРОБКА ПРОГРАМНОЇ ЧАСТИНИ ПРОЕКТУ

4.1 Розробка структури бази даних

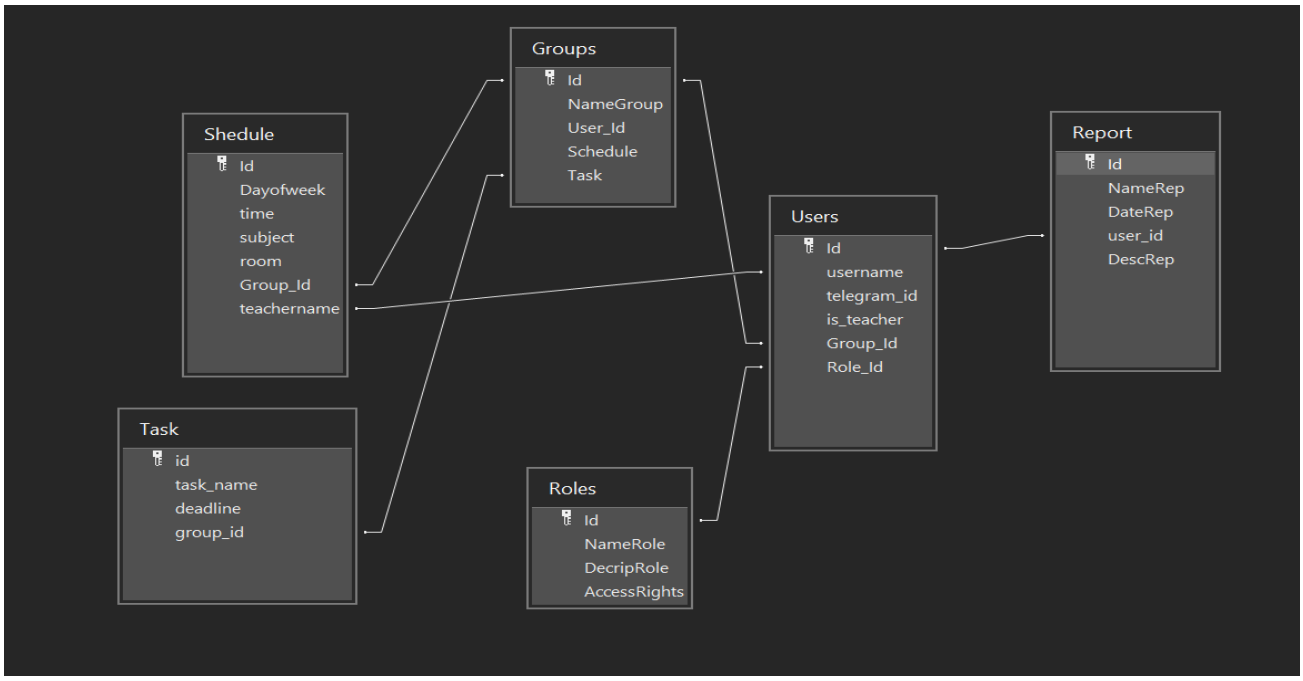


Рисунок 4.1. База даних програмного застосунку.

Бази даних (БД) для навчального закладу в табличному вигляді є ефективним засобом для зберігання, організації та керування різноманітною інформацією, пов'язаною з учнями, викладачами, розкладом, завданнями та іншими аспектами освітнього процесу (див. Рисунок 1.2).

Переваги використання бази даних включають в себе систематизацію даних та полегшення доступу до них. Така система дозволяє легко відстежувати та управляти інформацією щодо розкладу, успішності студентів, викладачів та інших аспектів освітнього процесу.

Перевага цієї бази даних виявляється у зручному табличному вигляді, що дозволяє швидко оцінювати та аналізувати дані. Кожна таблиця відображає конкретний аспект, такий як інформація про учнів, викладачів, розклад занять, завдання та інше. Взаємозв'язки між таблицями (наприклад, між розкладом та

списком студентів) сприяють збереженню консистентності даних та полегшують взаємодію між різними аспектами навчального процесу.

Важливо відзначити, що БД для навчального закладу є невід'ємною частиною сучасних інформаційних технологій у сфері освіти. Вона полегшує автоматизацію процесів, таких як генерація звітів, моніторинг успішності, планування розкладу та інше. Така система є необхідною для оптимізації управління навчальним процесом, забезпечуючи ефективну роботу навчального закладу та покращення якості освіти.

4.1.1. Опис таблиць Бази Даних

```python
class User(Base):
    __tablename__ = 'users'

    id = Column(Integer, primary_key=True)
    telegram_id = Column(Integer, nullable=False)
    username = Column(String(50), nullable=False)
    role_name = Column(String(20), ForeignKey('roles.name'))
    is_teacher = Column(Boolean, default=False)

    role = relationship( argument: 'Role', back_populates='users')

    group_name = Column(Integer, ForeignKey('groups.name'))
    group = relationship( argument: 'Group', back_populates='users')
```

Рисунок 2.1. Модель таблиці користувача

На рис. 2.1 зображено таблицю бази даних, котра має назву "users" і визначає сутність користувача. Кожен користувач характеризується унікальним ідентифікатором (id), який є первинним ключем. Також в таблиці є поля для зберігання ідентифікатора користувача в Telegram (telegram_id), користувальницького імені (username), ролі користувача (role_name), що є зовнішнім ключем, та прапорця, який вказує, чи є користувач вчителем (is_teacher).

Інші зовнішні ключі вказують на групу користувача (group_name) та його роль (role). Для забезпечення цих зв'язків використовуються відповідні властивості relationships.

```
class Schedule(Base):
    __tablename__ = 'schedules'
    id = Column(Integer, primary_key=True)
    day_of_week = Column(String, nullable=False)
    time = Column(Time, nullable=False)
    subject = Column(String, nullable=False)
    room = Column(String)
    group_id = Column(Integer, ForeignKey('groups.id'))
    group = relationship( argument: "Group", back_populates="schedule")
    teacher_name = Column(Integer, ForeignKey('users.username'))
```

Рисунок 2.2. Модель таблиці розкладу

На рис. 2.2 зображено таблицю бази даних, котра має назву "schedules" і визначає сутність розкладу занять. Кожний запис у цій таблиці ідентифікується унікальним ідентифікатором (id), що є первинним ключем. Також в таблиці присутні поля для визначення дня тижня (day_of_week), часу проведення заняття (time), назви предмету (subject), номеру кабінету (room), ідентифікатора групи (group_id), що є зовнішнім ключем, та ім'я вчителя (teacher_name), яке також є зовнішнім ключем, пов'язаним з іменем користувача з таблиці "users".

Таблиця встановлює зв'язок між розкладом і групою за допомогою полів "group_id" та "group", використовуючи відповідний об'єкт relationship. Також існує зв'язок з користувачем-вчителем за допомогою поля "teacher_name" та відповідного зовнішнього ключа.

Ця таблиця призначена для зберігання інформації про розклад занять для конкретної групи, зазначеного дня тижня та часу, предмету, а також кабінету, де відбувається заняття.

```python
class Group(Base):
    __tablename__ = 'groups'
    id = Column(Integer, primary_key=True)
    name = Column(String, unique=True)
    users = relationship( argument: "User", back_populates="group")
    schedule = relationship( argument: "Schedule", back_populates="group")
    tasks = relationship( argument: "Task", back_populates="group")
```

Рисунок 2.3. Модель таблиці групи

На рис. 2.3 зображено таблицю бази даних, котра має назву "groups" і представляє сутність групи. Кожна група ідентифікується унікальним ідентифікатором (id), який виступає у ролі первинного ключа. Також у таблиці присутнє поле для зберігання назви групи (name), яке визначено як унікальне.

Таблиця має зв'язки з іншими таблицями через відносини "users", "schedule" і "tasks", які вказані як об'єкти relationship. Зв'язок "users" вказує на зв'язок між групою та користувачами, які належать до цієї групи. Зв'язок "schedule" вказує на зв'язок з розкладом занять для даної групи. Зв'язок "tasks" вказує на зв'язок з завданнями, пов'язаними з даною групою.

Ця таблиця призначена для зберігання інформації про групи, які можуть мати зв'язки з користувачами, розкладом занять та завданнями.

```python
class Task(Base):
    __tablename__ = 'tasks'
    id = Column(Integer, primary_key=True)
    name = Column(String)
    deadline = Column(DateTime)
    group_id = Column(Integer, ForeignKey('groups.id'))
    group = relationship( argument: "Group", back_populates="tasks")
```

Рисунок 2.3. Модель таблиці завдань

На рис. 2.3 зображено таблицю бази даних, котра має назву "tasks" і визначає

сутність завдань. Кожне завдання ідентифікується унікальним ідентифікатором (id), що є первинним ключем. У таблиці також присутні поля для зберігання назви завдання (name) та строку його виконання (deadline), яке визначено у форматі DateTime.

Також у таблиці є зовнішній ключ (group_id), який посилається на ідентифікатор групи з таблиці "groups". Це поле встановлює зв'язок між завданням і конкретною групою.

Взаємодія між таблицею завдань і таблицею груп здійснюється через відношення "group" властивості relationship. Це дозволяє визначити, які завдання належать до певної групи в базі даних.

```
class Role(Base):
    __tablename__ = 'roles'

    id = Column(Integer, primary_key=True)
    name = Column(String(50), unique=True, nullable=False)

    users = relationship( argument: 'User', back_populates='role')
```

Рисунок 2.4. Модель таблиці ролі

На рис. 2.4 зображено таблицю бази даних, котра має назву "roles" і визначає сутність ролі користувача. Кожна роль ідентифікується унікальним ідентифікатором (id), який виступає у ролі первинного ключа. У таблиці також присутнє поле для зберігання назви ролі (name), яке має обмеження унікальності та не може бути порожнім.

Також в таблиці встановлений зв'язок з іншою таблицею "User" через поле "users". Це поле вказує на те, що кожна роль може мати багато користувачів, і взаємодія між цими таблицями реалізована через властивість relationship. Це дозволяє визначити, які користувачі мають певну роль в системі чи додатку.

У висновку слід відзначити, що використання ORM (Object-Relational Mapping) в проекті має значний позитивний вплив на розробку. Забезпечуючи абстракцію бази даних та використання об'єктно-орієнтованого підходу, ORM спрощує взаємодію з даними, роблячи код більш зрозумілим та менш обтяженим повторюваним SQL-кодом. Зменшення часу на розробку, полегшення тестування та зручність відслідковування змін роблять ORM необхідним інструментом для продуктивної та ефективної розробки програмного забезпечення.

4.2 Впровадження штучного інтелекту в застосунок

Використовуючи бібліотеку torch було реалізовано модель штучного інтелекту для роботою з базою даних та вхідним текстом користувача. Розписані всі необхідні змінні, перетворення та саме тренування, котре реалізовано асинхронно та працює тільки під час використання функції. Код реалізації наведений нижче:

```
import logging

import torch
import torch.nn as nn
import torch.optim as optim
from torch.multiprocessing import Process, Queue
from telegram import Update
from telegram.ext import CallbackContext
from models import session, Schedule
from keyboards import get_main_menu_keyboard
SELECTING_AI, CHATTING = range(2)
# Отримання даних з таблиці schedules та їх обробка
schedules_query = session.query(Schedule).all()
```

```
    schedule_inputs = [f"{schedule.day_of_week} {schedule.time} {schedule.subject}" for schedule in schedules_query]
    schedule_targets = [schedule.room for schedule in schedules_query]
    # Перетворення даних у числовий формат (замість текстового)
    schedule_input_mapping = {schedule_data: idx for idx, schedule_data in enumerate(set(schedule_inputs))}
    schedule_target_mapping = {room: idx for idx, room in enumerate(set(schedule_targets))}
    schedule_inputs = [[schedule_input_mapping[schedule_data]] for schedule_data in schedule_inputs]
    schedule_targets = [[schedule_target_mapping[room]] for room in schedule_targets]
    # Перетворення в тензори PyTorch
    schedule_inputs = torch.tensor(schedule_inputs, dtype=torch.float32)
    schedule_targets = torch.tensor(schedule_targets, dtype=torch.float32)
    # Enable logging
    logging.basicConfig(
        format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
    )
    logging.getLogger("httpx").setLevel(logging.WARNING)
    async def start_ai(update: Update, context: CallbackContext):
        return CHATTING
    async def start_chatting(update: Update, context: CallbackContext) -> str | None:
        user_text = update.message.text
        # Create a multiprocessing queue for passing the model between processes
        model_queue = Queue()
        # Create a separate process for training the model
        training_process = Process(target=train_schedule_model, args=(schedule_inputs,
```

```
schedule_targets,
                                                    len(schedule_input_mapping),
                                                    len(schedule_target_mapping),
                                                    model_queue))
            training_process.start()
            training_process.join()
            # Get the trained model from the queue
            trained_model = model_queue.get()
            logging.info(f"DEBUG: select_group - action: {trained_model}")
            # Use the trained model for making predictions
            with torch.no_grad():
                schedule_output                                                    =
trained_model(torch.tensor([[schedule_input_mapping[user_text]]], dtype=torch.float32))
                schedule_output = torch.argmax(schedule_output, dim=1)
                predicted_room = [key for key, value in schedule_target_mapping.items() if
value == schedule_output.item()][0]
            # Respond to the user
            await update.message.reply_text(f"Предмет: {user_text}, Аудиторія:
{predicted_room}",
                                            reply_markup=get_main_menu_keyboard())
            return "MAIN_MENU"
        # Визначення архітектури нейронної мережі для розкладу
        class ScheduleNN(nn.Module):
            def __init__(self, input_size, output_size):
                super(ScheduleNN, self).__init__()
                self.fc = nn.Linear(input_size, output_size
            def forward(self, x):
                x = self.fc(x)
```

```
        return x
    # Ініціалізація та навчання моделі
    def train_schedule_model(inputs, targets, input_size, output_size, queue):
        model = ScheduleNN(input_size, output_size)
        criterion = nn.CrossEntropyLoss()
        optimizer_schedule = optim.SGD(model.parameters(), lr=0.01)
        epochs = 1000
        for epoch in range(epochs):
            optimizer_schedule.zero_grad()
            schedule_outputs = model(inputs)
            schedule_loss = criterion(schedule_outputs, targets)
            schedule_loss.backward()
            optimizer_schedule.step()
            if (epoch + 1) % 100 == 0:
                print(f'Epoch [{epoch + 1}/{epochs}], Schedule Loss:
{schedule_loss.item():.4f}'
        # Put the trained model into the queue for later retrieval
        queue.put(model)
```

4.3 Робота з бібліотекою Telegram та приклад реалізації функцій

Для роботи з бібліотекою Telegram необхідно її інсталювати в проект. Для інсталяції даної бібліотеки необхідно в консолі прописати запит «pip install python-telegram-bot». Після цього в файлах роботи з телеграмом треба звертатись до бібліотеки для використання пакетних функцій, наприклад для виклику «MessageHandler» котрий може приймати в собі введення тексту користувачем, необхідно вписати наступне підключення: from telegram.ext import MessageHandler.

В головному файлі програми main.py ініціалізовано всі можливі обробки станів, котрі викликаються під час роботи застосунку для преходу між блоками меню. Також

в головному файлі імпортовані всі інші реалізації. Код головної програми наведено нижче:

```
# main.py
from telegram import Update
from telegram.ext import Application, ConversationHandler, CallbackQueryHandler, MessageHandler, filters, \
    CommandHandler
from handlers import start, choose_option, main_menu_after_registration, choose_main_menu_option, \
    choose_role_option, login, waiting_for_name, waiting_for_group
from schedule import SELECTING_ACTION, SELECTING_DAY, ENTER_SUBJECT, \
    ENTER_TIME, ENTER_TEACHER, ENTER_ROOM, waiting_for_day_of_week, waiting_for_time, waiting_for_subject, \
    waiting_for_room, select_group, SELECTING_GROUP, SELECTING_GROUP_ACTION, handle_selected_group, \
    select_option, VIEW_SCHEDULE, display_schedule, BACK_TO_MAIN_MENU, back_to_main_menu, BACK_SCHEDULE_ADD
import logging
import AI
# Enable logging
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)
logging.getLogger("httpx").setLevel(logging.WARNING)
# Константи для станів розмови
def main():
```

```
        application = Application.builder().token("6589876738:AAGVJITQlrpwhKZWELEKbinI0xMAG5Jeoso").build()
        conv_handler = ConversationHandler(
            entry_points=[CommandHandler("start", start)],
            states={
                "START_ROUTES": [
                    CallbackQueryHandler(choose_option),
                ],
                "REGISTER": [
                    CallbackQueryHandler(choose_role_option),
                ],
                "ENTER_NAME": [
                    CallbackQueryHandler(main_menu_after_registration),
                ],
                "WAITING_FOR_NAME": [
                    MessageHandler(filters.TEXT & ~filters.COMMAND, waiting_for_name),
                ],
                "WAITING_FOR_GROUP": [
                    MessageHandler(filters.TEXT & ~filters.COMMAND, waiting_for_group)
                ],
                "MAIN_MENU": [
                    CallbackQueryHandler(choose_main_menu_option),
                ],
                "WAITING_FOR_LOGIN": [
                    CallbackQueryHandler(choose_option),
```

```
            ],
            "LOGIN": [
                MessageHandler(filters.TEXT & ~filters.COMMAND, login),
            ],
            SELECTING_ACTION: [CallbackQueryHandler(select_option)],
            VIEW_SCHEDULE: [CallbackQueryHandler(display_schedule)],
            BACK_TO_MAIN_MENU: [
                CallbackQueryHandler(back_to_main_menu),
            ],
            SELECTING_GROUP: [CallbackQueryHandler(select_group)],
            SELECTING_GROUP_ACTION: [MessageHandler(filters.TEXT & ~filters.COMMAND, handle_selected_group)],
            SELECTING_DAY: [MessageHandler(filters.TEXT & ~filters.COMMAND, waiting_for_day_of_week)],
            ENTER_SUBJECT: [MessageHandler(filters.TEXT & ~filters.COMMAND, waiting_for_subject)],
            ENTER_TIME: [MessageHandler(filters.TEXT & ~filters.COMMAND, waiting_for_time)],
            ENTER_TEACHER: [MessageHandler(filters.TEXT & ~filters.COMMAND, waiting_for_subject)],
            ENTER_ROOM: [MessageHandler(filters.TEXT & ~filters.COMMAND, waiting_for_room)],
            AI.SELECTING_AI: [CallbackQueryHandler(AI.start_chatting)]
        },
        fallbacks=[],
    )

    application.add_handler(conv_handler)
```

```
application.run_polling(allowed_updates=Update.ALL_TYPES)


if __name__ == '__main__':
    main()
```

Для реєстрації користувача, див рис 4.2, та логіну, див рис 4.3, було створено файл handlers.py, де було реалізовано роботу з базою даних функцій реєстрації та логіну. Нижче наведено код роботи файлу handlers.py

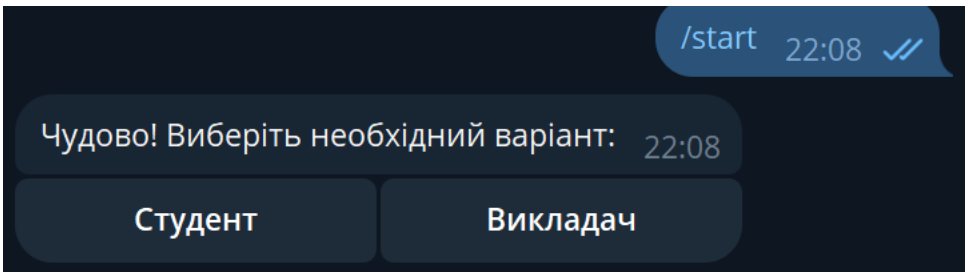


Рисунок 4.2 Реалізація функції реєстрації

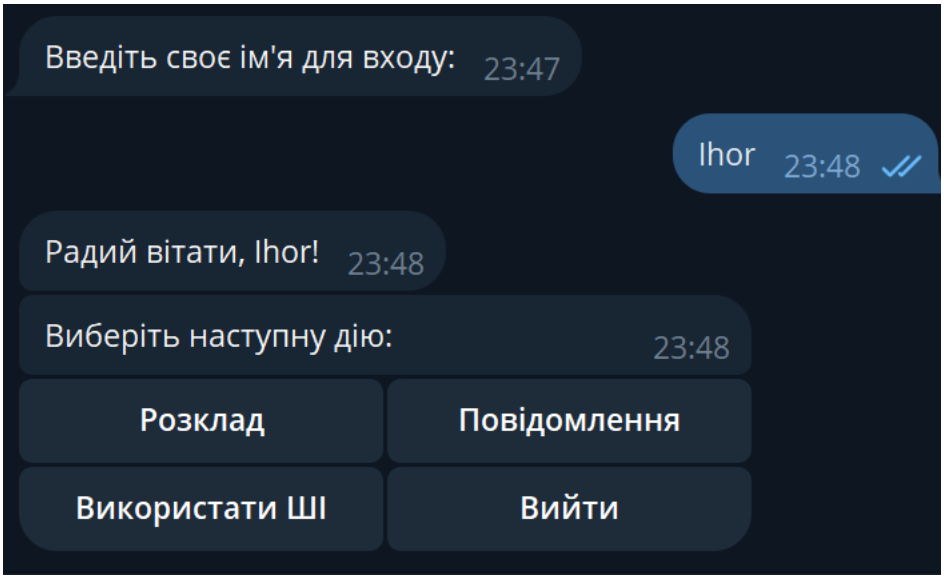


Рисунок 4.3 Реалізація функції логіну

```
# handlers.py
import logging
```

```
from telegram import Update
from telegram.ext import CallbackContext, ConversationHandler

from AI import SELECTING_AI
from models import Role, User, session, Group
from keyboards import get_start_menu_keyboard, get_registration_role_keyboard,
get_main_menu_keyboard, \
    get_back_keyboard, choose_schedule_keyboard
from schedule import SELECTING_ACTION
from utils import hash_password, verify_password, generate_reset_code
async def start(update: Update, context: CallbackContext) -> str:
    await update.message.reply_text("Вітаю! Виберіть наступну дію:",
reply_markup=get_start_menu_keyboard())
    return "START_ROUTES"
async def choose_option(update: Update, context: CallbackContext) -> str:
    query = update.callback_query
    await query.answer(
    option = query.data
    context.user_data['option'] = option
    logging.info(f"DEBUG: choose_role_option - option: {option}")
    if option == "REGISTER":
        await query.edit_message_text("Чудово! Виберіть необхідний варіант:",
                                      reply_markup=get_registration_role_keyboard())
        return "REGISTER"
    elif option == "LOGIN":
        await query.edit_message_text("Введіть своє ім'я для входу:")
        return "LOGIN"
```

```
    async def login(update: Update, context: CallbackContext) -> str:
        username = update.message.text
        user = session.query(User).filter_by(username=username).first()
        if not user:
            await update.message.reply_text("Нерпавильно введено ім'я. ",
                                            reply_markup=get_start_menu_keyboard())
            return "WAITING_FOR_LOGIN"
        # Отримати ім'я користувача з контексту
        context.user_data['logged_in_user_name'] = username
        logging.info(f"DEBUG: logged_in_user_name - teacher: {username}")
        await update.message.reply_text(f"Радий вітати, {username}!")
        await update.message.reply_text("Виберіть наступну дію:",
reply_markup=get_main_menu_keyboard())
        return "MAIN_MENU"
    async def choose_role_option(update: Update, context: CallbackContext) -> str:
        query = update.callback_query
        await query.answer()
        option = query.data
        context.user_data['option'] = option
        logging.info(f"DEBUG: choose_role_option - option: {option}")
        role = session.query(Role).filter_by(name=option).first()
        context.user_data['role'] = role
        if not role:
            role = Role(name=option)
            session.add(role)
            session.commit()
        else:
            role.name = option
```

```
            session.commit()
        context.user_data['role_name'] = role.name
        await query.edit_message_text("Чудово! Введіть свою групу:")
        return "WAITING_FOR_GROUP"
    async def waiting_for_group(update: Update, context: CallbackContext) -> int | str:
        group_name = update.message.text
        group = session.query(Group).filter_by(name=group_name).first()
        if not group:
            group = Group(name=group_name)
            session.add(group)
            session.commit()
        else:
            group.name = group_name
            session.commit()
        context.user_data['group'] = group_name
        await update.message.reply_text("Чудово! Введіть своє ім'я:")
        return "WAITING_FOR_NAME"
    async def waiting_for_name(update: Update, context: CallbackContext) -> int | str:
        user_name = update.message.text
        tg_id = update.message.from_user.id
        role_name = context.user_data.get('role_name')
        group_name = context.user_data.get('group')
        logging.info(f"DEBUG: waiting_for_name - user_name: {user_name}")
        logging.info(f"DEBUG: waiting_for_name - group: {group_name}"
        existing_user = session.query(User).filter_by(telegram_id=tg_id,
username=user_name, role_name=role_name,
                                                      group_name=group_name).first()
        if not existing_user and role_name == "teacher":
```

```
            new_user = User(telegram_id=tg_id, username=user_name,
role_name=role_name, is_teacher=True,
                            group_name=group_name)
            session.add(new_user)
            session.commit()
            logging.info(f"DEBUG: waiting_for_name - name for register: {user_name}
and it`s a teacher")
        elif not existing_user and role_name == "student":
            new_user = User(telegram_id=tg_id, username=user_name,
role_name=role_name, is_teacher=False,
                            group_name=group_name)
            session.add(new_user)
            session.commit()
            logging.info(f"DEBUG: waiting_for_name - name for register: {user_name}
and it`s a student")
        else:
            await update.message.reply_text(
                "Ви вже зареєстровані. Перезапустіть бота для реєстрації!")  # TODO:
на майбутнє, можна реалізувати цикл для повернення для логіну
            return ConversationHandler.END

        await update.message.reply_text(f"Ви успішно зареєстровані, {user_name}!")
        await update.message.reply_text("Виберіть наступну дію:",
reply_markup=get_main_menu_keyboard())

        return "ENTER_NAME"
```

```
    # функція для збереження даних залогіненого користувача та використання в інших файлах

    def handle_data(data, context: CallbackContext):
        data = context.user_data['logged_in_user_name']
        logging.info(f"DEBUG: handle_data - name: {data}")
        return data



    async def main_menu_after_registration(update: Update, context: CallbackContext) -> str:
        query = update.callback_query
        await query.answer()

        option = query.data
        context.user_data['option'] = option

        logging.info(f"DEBUG: main_menu_after_registration - option: {option}")

        if not query:
            logging.error("Callback query is None.")
            return "MAIN_MENU"

        try:
            await query.answer()
        except Exception as e:
            logging.error(f"Error answering query: {e}")
```

```
        return "MAIN_MENU"


    async def choose_main_menu_option(update: Update, context: CallbackContext) ->
str:
        query = update.callback_query
        await query.answer()

        option = query.data
        context.user_data['option'] = option

        if option == "manage_schedule":
            await query.edit_message_text("Виберіть дію для роботи з розкладом:",
reply_markup=choose_schedule_keyboard())
            return SELECTING_ACTION
        elif option == "ask_question":
            await handle_ask_question(update, context)
            return "MAIN_MENU"
        elif option == "use_AI":
            return SELECTING_AI
        elif option == "END_ROUTES":
            await choose_back_option(update, context)
            return "WAITING_FOR_LOGIN"


    async def handle_ask_question(update: Update, context: CallbackContext) -> str:
        await update.message.reply_text("Ask your question to the teacher:")
        return "MAIN_MENU"
```

```
async def choose_back_option(update: Update, context: CallbackContext) -> int:
    """End conversation from InlineKeyboardButton."""
    await update.callback_query.answer()

    text = "До зустрічі!"
    await update.callback_query.edit_message_text(text=text)

    return ConversationHandler.END
```

ВИСНОВКИ

У ході дослідження та аналізу різноманітних технологій для створення інтелектуального помічника, було звернуто увагу на важливість обрання оптимальних інструментів для реалізації функціоналу. Одним із ключових досягнень є створення високофункціонального інструменту, який забезпечує ефективну взаємодію з користувачами та дозволяє вирішувати різноманітні завдання у форматі чат-боту.

У контексті розробки чат-бота ключовим етапом виявилося впровадження реєстрації, створення та редагування розкладу занять та використання штучного інтелекту, спрямованого на спрощення рутинних завдань та оптимізацію взаємодії з користувачами. Основний акцент був зроблений на створенні інструментів, які дозволяють ефективно виконувати завдання, що часто виникають у взаємодії з чат-ботом.

Перевагою розробки стало впровадження функціоналу, який значно полегшує рутинні завдання та покращує взаємодію з чат-ботом. Також важливим стало створення інтуїтивно зрозумілого інтерфейсу для користувачів. Це включає в себе розробку різних наборів екранних клавіатур, які легко сприймається.

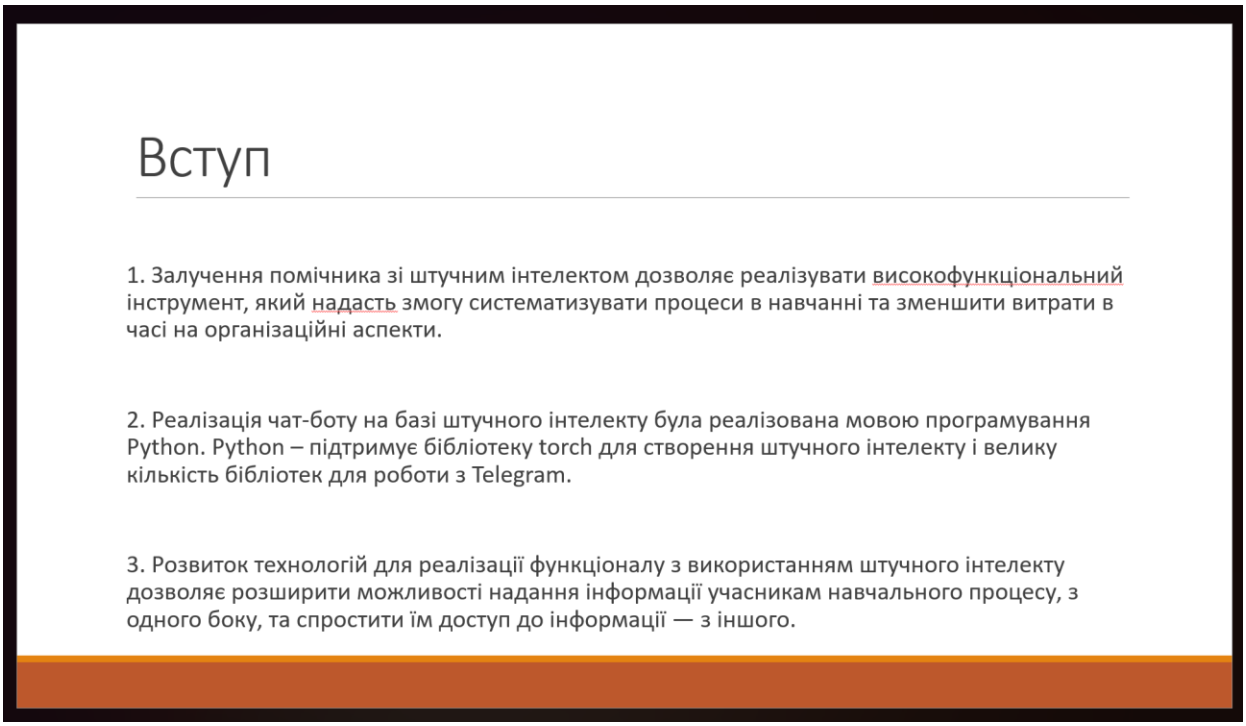
Щодо перспектив, розроблений чат-бот може знайти широке застосування в різних галузях, включаючи освіту, бізнес та медицину. Можливості подальшого розвитку включають в себе розширення функціоналу, удосконалення алгоритмів та адаптацію до різноманітних сценаріїв використання. Проект відкриває перспективи для подальших досліджень та вдосконалення інтелектуального помічника відповідно до зростаючих потреб користувачів та сучасних технологічних тенденцій.

ПЕРЕЛІК ДЖЕРЕЛ ТА ПОСИЛАНЬ

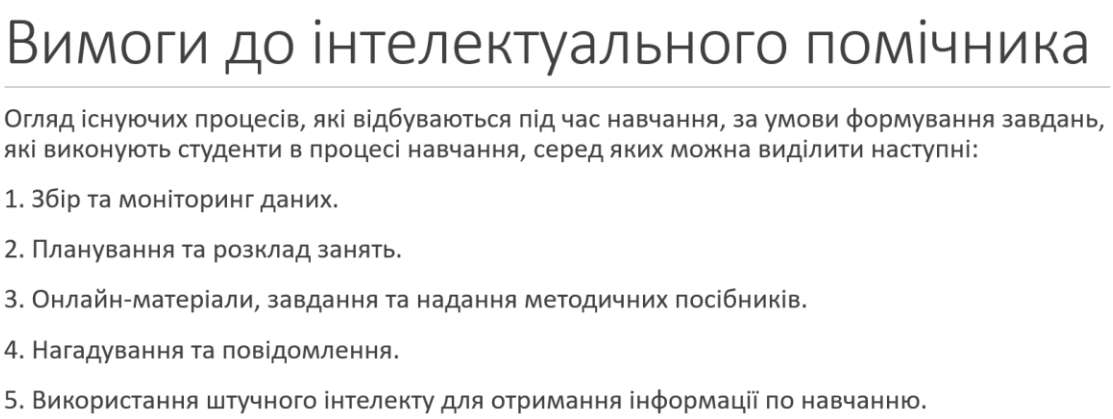
1. Head First Object-Oriented Analysis and Design – Brett D. McLaughlin. Розробка бізнес-об'єктів / Рокфорд Лотка. - М.: Вільямс, 2010. - 816 с.
2. Test Driven: TDD and Acceptance TDD for Python Developers – Lasse Koskela - M., 2013. - 480 с.
3. Алгоритмы на Python – Роберт Седжвик, Кевін Уэйн - М.: ДМК Пресс, 2011. - 456 с.
4. Head First Object-Oriented Analysis and Design – Brett D. McLaughlin Джеффри Рихтер. - М.: Київ, 2013. - 928 с.
5. Python. Бібліотека професіонала, Кей С. Хорстманн, Гарі Корнелл - М.: Вільямс, 2011. - 672
6. [Електронний ресурс] «Wunderlist». Режим доступу: https://en.wikipedia.org/wiki/Wunderlist
7. [Електронний ресурс] «Microsoft To-Do». Режим доступу: https://to-do.office.com/tasks/
8. [Електронний ресурс] «Todoist». Режим доступу: https://todoist.com/
9. [Електронний ресурс] «Trello». Режим доступу: https://trello.com/
10. [Електронний ресурс] «RescueTime». Режим доступу: https://www.rescuetime.com/
11. [Електронний ресурс] «Forest». Режим доступу: https://www.rescuetime.com/
12. [Електронний ресурс] «OneNote». Режим доступу: https://www.microsoft.com/uk-ua/microsoft-365/onenote/digital-note-taking-app
13. [Електронний ресурс] «Evernote». Режим доступу: https://evernote.com/
14. [Електронний ресурс] «Notion». Режим доступу: https://www.notion.so/
15. [Електронний ресурс] «Quizlet». Режим доступу: https://quizlet.com/ua
16. [Електронний ресурс] «Khan Academy». Режим доступу: https://uk.khanacademy.org/

17. [Електронний ресурс] «Coursera». Режим доступу: https://www.coursera.org/
18. [Електронний ресурс] «Grammarly». Режим доступу: https://www.grammarly.com/
19. [Електронний ресурс] «Zotero». Режим доступу: https://www.zotero.org/
20. [Електронний ресурс] «Asana». Режим доступу: https://asana.com/
21. [Електронний ресурс] «Google Scholar». Режим доступу: https://en.wikipedia.org/wiki/Google_Scholar
22. [Електронний ресурс] «JSTOR (Journal Storage)». Режим доступу: https://www.jstor.org/
23. [Електронний ресурс] «MATLAB». Режим доступу: https://www.mathworks.com/products/matlab.html
24. [Електронний ресурс] «Mathcad». Режим доступу: https://www.mathcad.com/en
25. [Електронний ресурс] «Crello». Режим доступу: https://vumonline.ua/partners/crello/
26. [Електронний ресурс] «Adobe Photoshop». Режим доступу: https://www.adobe.com/ua/products/photoshop.html
27. [Електронний ресурс] «Canva». Режим доступу: https://www.canva.com/
28. [Електронний ресурс] «Sketch». Режим доступу: https://www.sketch.com/
29. [Електронний ресурс] «Moodle». Режим доступу: https://moodle.org/?lang=uk
30. [Електронний ресурс] «Google Classroom». Режим доступу: https://uk.wikipedia.org/wiki/Google_Classroom
31. Баранов О.А. 2022. Трансформація: соціальна & цифрова & правова : монографія. Т. 1. Порятунок цивілізації: економіка. Одеса: Видавничий дім “Гельветика”, 2022. 272 с.
32. Спірін О.М. Деякі проблеми вивчення основ штучного інтелекту в курсі інформатики // Нові технології навчання. – К.: ІЗМН, 1997. – Вип. 21. – С. 47-54.

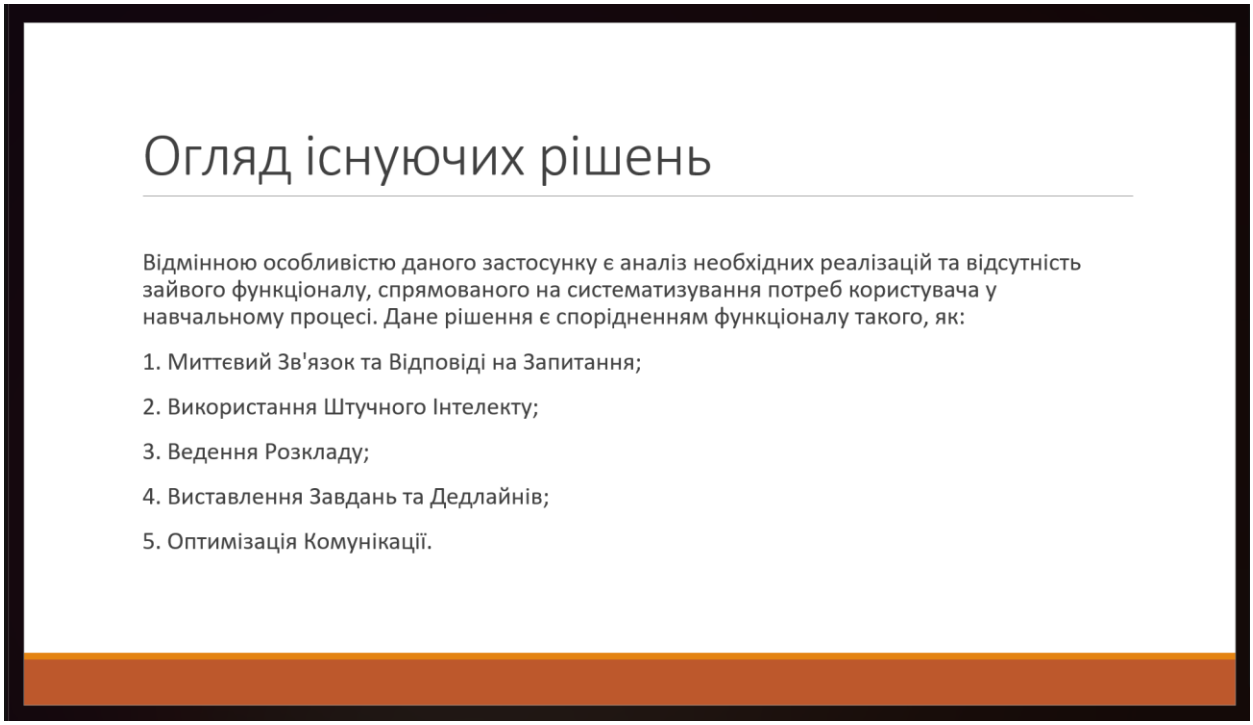
ДОДАТОК А



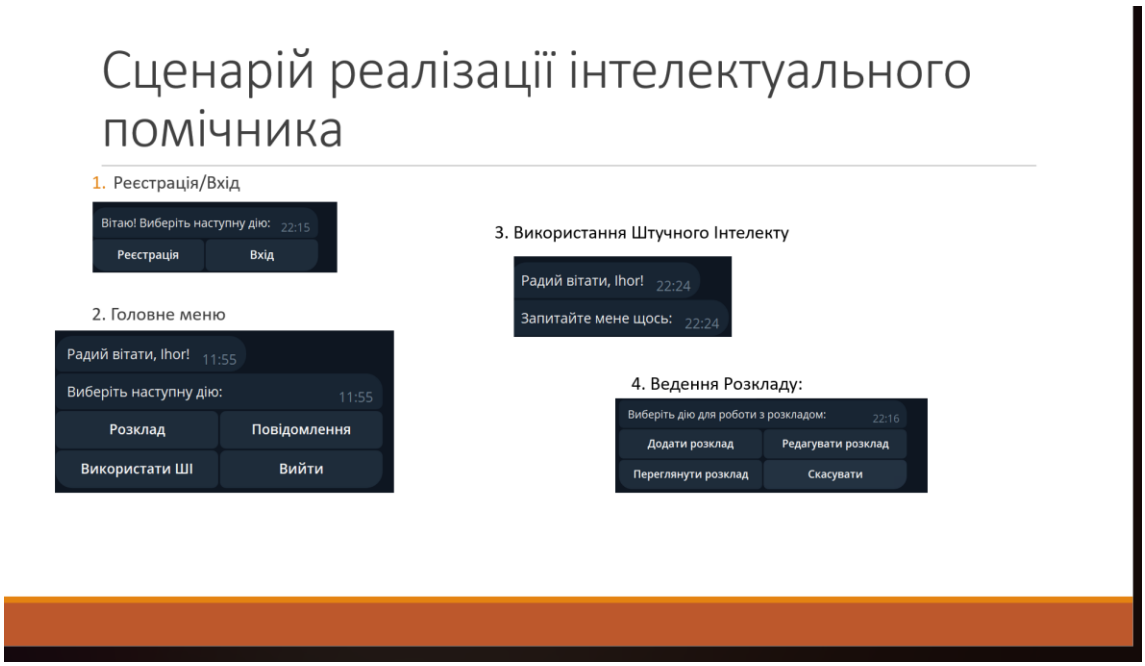
Слайд 1. Вступ



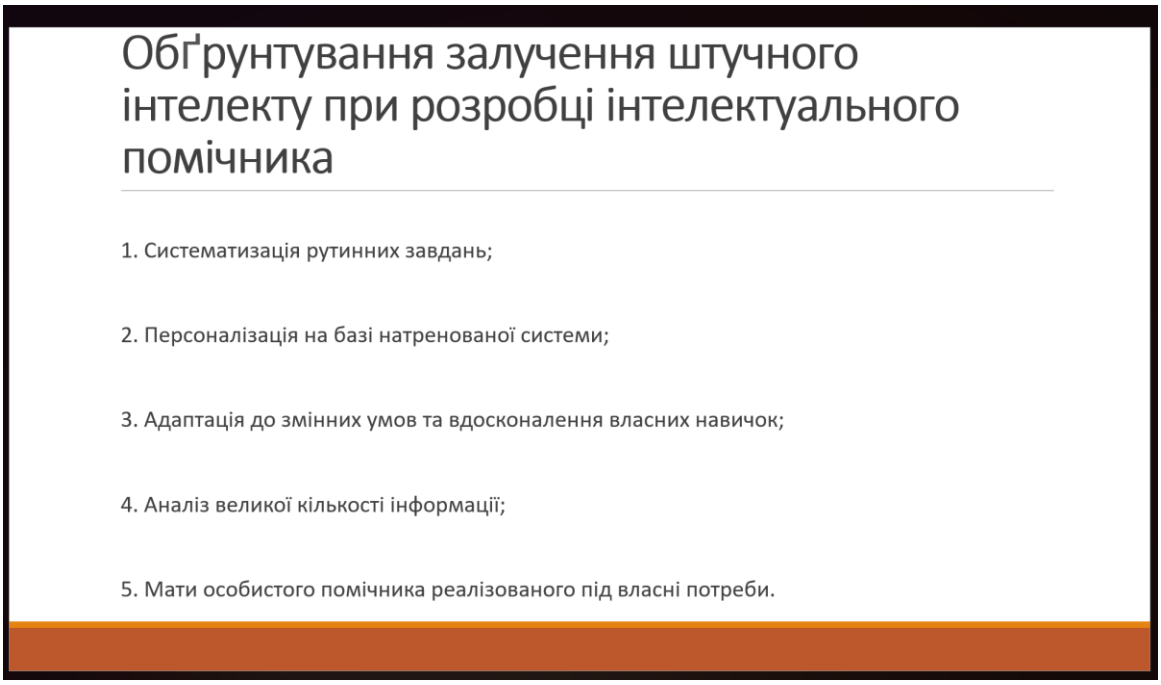
Слайд 2. Вимоги до інтелектуального помічника



Слайд 3. Оглял існуючих рішень



Слайд 4. Сценарій реалізації інтелектуального помічника



Слайд 5. Обгрунтування залучення штучного інтелекту при розробці інтелектуального помічника

ДОДАТОК Б

```
# handlers.py
import logging

from telegram import Update
from telegram.ext import CallbackContext, ConversationHandler

from AI import SELECTING_AI
from models import Role, User, session, Group
from keyboards import get_start_menu_keyboard, get_registration_role_keyboard, get_main_menu_keyboard, \
    get_back_keyboard, choose_schedule_keyboard
from schedule import SELECTING_ACTION
from utils import hash_password, verify_password, generate_reset_code


async def start(update: Update, context: CallbackContext) -> str:
    await update.message.reply_text("Вітаю! Виберіть наступну дію:", reply_markup=get_start_menu_keyboard())
    return "START_ROUTES"


async def choose_option(update: Update, context: CallbackContext) -> str:
    query = update.callback_query
    await query.answer()

    option = query.data
    context.user_data['option'] = option
```

```
    logging.info(f"DEBUG: choose_role_option - option: {option}")

    if option == "REGISTER":
        await query.edit_message_text("Чудово! Виберіть необхідний варіант:",
                                      reply_markup=get_registration_role_keyboard())
        return "REGISTER"
    elif option == "LOGIN":
        await query.edit_message_text("Введіть своє ім'я для входу:")
        return "LOGIN"



async def login(update: Update, context: CallbackContext) -> str:
    username = update.message.text

    user = session.query(User).filter_by(username=username).first()

    if not user:
        await update.message.reply_text("Нерпавильно введено ім'я. ",
                                        reply_markup=get_start_menu_keyboard())
        return "WAITING_FOR_LOGIN"

    # Отримати ім'я користувача з контексту
    context.user_data['logged_in_user_name'] = username
    logging.info(f"DEBUG: logged_in_user_name - teacher: {username}")

    await update.message.reply_text(f"Радий вітати, {username}!")
```

```
        await update.message.reply_text("Виберіть наступну дію:", reply_markup=get_main_menu_keyboard())
        return "MAIN_MENU"


    async def choose_role_option(update: Update, context: CallbackContext) -> str:
        query = update.callback_query
        await query.answer()

        option = query.data
        context.user_data['option'] = option

        logging.info(f"DEBUG: choose_role_option - option: {option}")

        role = session.query(Role).filter_by(name=option).first()
        context.user_data['role'] = role
        if not role:
            role = Role(name=option)
            session.add(role)
            session.commit()
        else:
            role.name = option
            session.commit()

        context.user_data['role_name'] = role.name
        await query.edit_message_text("Чудово! Введіть свою групу:")
        return "WAITING_FOR_GROUP"
```

```
async def waiting_for_group(update: Update, context: CallbackContext) -> int | str:
    group_name = update.message.text
    group = session.query(Group).filter_by(name=group_name).first()
    if not group:
        group = Group(name=group_name)
        session.add(group)
        session.commit()
    else:
        group.name = group_name
        session.commit()

    context.user_data['group'] = group_name

    await update.message.reply_text("Чудово! Введіть своє ім'я:")

    return "WAITING_FOR_NAME"


async def waiting_for_name(update: Update, context: CallbackContext) -> int | str:
    user_name = update.message.text
    tg_id = update.message.from_user.id
    role_name = context.user_data.get('role_name')
    group_name = context.user_data.get('group')

    logging.info(f"DEBUG: waiting_for_name - user_name: {user_name}")
    logging.info(f"DEBUG: waiting_for_name - group: {group_name}")
```

```
        existing_user = session.query(User).filter_by(telegram_id=tg_id, username=user_name, role_name=role_name,
                                                      group_name=group_name).first()
        if not existing_user and role_name == "teacher":
            new_user = User(telegram_id=tg_id, username=user_name, role_name=role_name, is_teacher=True,
                            group_name=group_name)
            session.add(new_user)
            session.commit()
            logging.info(f"DEBUG: waiting_for_name - name for register: {user_name} and it`s a teacher")
        elif not existing_user and role_name == "student":
            new_user = User(telegram_id=tg_id, username=user_name, role_name=role_name, is_teacher=False,
                            group_name=group_name)
            session.add(new_user)
            session.commit()
            logging.info(f"DEBUG: waiting_for_name - name for register: {user_name} and it`s a student")
        else:
            await update.message.reply_text(
                "Ви вже зареєстровані. Перезапустіть бота для реєстрації!")  # TODO: на майбутнє, можна реалізувати цикл для повернення для логіну
            return ConversationHandler.END

        await update.message.reply_text(f"Ви успішно зареєстровані, {user_name}!")
        await update.message.reply_text("Виберіть наступну дію:", reply_markup=get_main_menu_keyboard())
```

```
        return "ENTER_NAME"


    # функція для збереження даних залогіненого користувача та використання в інших файлах

    def handle_data(data, context: CallbackContext):
        data = context.user_data['logged_in_user_name']
        logging.info(f"DEBUG: handle_data - name: {data}")
        return data


    async def main_menu_after_registration(update: Update, context: CallbackContext) -> str:
        query = update.callback_query
        await query.answer()

        option = query.data
        context.user_data['option'] = option

        logging.info(f"DEBUG: main_menu_after_registration - option: {option}")

        if not query:
            logging.error("Callback query is None.")
            return "MAIN_MENU"

        try:
```

```
            await query.answer()
        except Exception as e:
            logging.error(f"Error answering query: {e}")

        return "MAIN_MENU"


    async def choose_main_menu_option(update: Update, context: CallbackContext) ->
str:
        query = update.callback_query
        await query.answer()

        option = query.data
        context.user_data['option'] = option

        if option == "manage_schedule":
            await query.edit_message_text("Виберіть дію для роботи з розкладом:",
reply_markup=choose_schedule_keyboard())
            return SELECTING_ACTION
        elif option == "ask_question":
            await handle_ask_question(update, context)
            return "MAIN_MENU"
        elif option == "use_AI":
            await query.edit_message_text("Хей, почнімо розмову")
            return SELECTING_AI
        elif option == "END_ROUTES":
            await choose_back_option(update, context)
            return "WAITING_FOR_LOGIN"
```

```
async def handle_ask_question(update: Update, context: CallbackContext) -> str:
    await update.message.reply_text("Ask your question to the teacher:")
    return "MAIN_MENU"


async def choose_back_option(update: Update, context: CallbackContext) -> int:
    """End conversation from InlineKeyboardButton."""
    await update.callback_query.answer()

    text = "До зустрічі!"
    await update.callback_query.edit_message_text(text=text)

    return ConversationHandler.END
import logging
from datetime import time
from typing import Callable, Any, Coroutine

from telegram import Update
from telegram.ext import CallbackContext

from keyboards import choose_schedule_keyboard, get_group_keyboard, \
    get_back_keyboard, get_main_menu_keyboard
from models import User, session, Schedule, Group

(SELECTING_ACTION, SELECTING_GROUP,
SELECTING_GROUP_ACTION, SELECTING_DAY,
```

```
    ENTER_SUBJECT, ENTER_TIME, ENTER_TEACHER, ENTER_ROOM, VIEW_SCHEDULE, BACK_TO_SCHEDULE_MENU,
    SELECTING_DAY_EDIT, ENTER_TIME_EDIT, SELECT_SUBJECT_TO_EDIT, ENTER_NEW_SCHEDULE_DATA) = range(
        14)


async def select_option(update: Update, context: CallbackContext) -> str | Any:
    query = update.callback_query

    option = query.data
    context.user_data['option'] = option

    logging.info(f"DEBUG: select_action - option: {option}")
    if option == 'add_lesson':
        context.user_data['schedule_action'] = 'add'
        await query.edit_message_text("Оберіть групу:", reply_markup=get_group_keyboard())
        return SELECTING_GROUP

    elif option == 'edit_lesson':
        context.user_data['schedule_action'] = 'edit'
        await query.edit_message_text("Введіть день тижня (наприклад, понеділок), на якому потрібно редагувати:")
        return SELECTING_DAY_EDIT
    elif option == 'show_schedule':
        await query.edit_message_text("Розклад для групи:", reply_markup=get_group_keyboard())
```

```
            return VIEW_SCHEDULE
        elif option == 'cancel':
            await query.edit_message_text("Вибір відмінено.", reply_markup=get_main_menu_keyboard())
            return "MAIN_MENU"
        logging.info(f"DEBUG: start_schedule_management - option: {option}")


    async def display_schedule(update: Update, context: CallbackContext) -> str | None:
        # Retrieve information from user_data
        query = update.callback_query
        option = query.data
        context.user_data['option'] = option
        logging.info(f"DEBUG: select_group - action: {option}")
        if option.startswith('select_group:'):
            selected_group_index = int(option[len('select_group:'):]) - 1
            groups = session.query(Group).all()
            selected_group_name = groups[selected_group_index].name
            context.user_data['group_name'] = selected_group_name
            logging.info(f"DEBUG: group_name - action: {selected_group_name}")
        # Check if the group_id is present

        get_group = context.user_data['group_name']

        if get_group is None:
            await update.message.reply_text("Group information not found. Please try again.")
            return
```

```
        # Retrieve the group from the database
        group = session.query(Group).filter_by(name=get_group).first()
        # Check if the group exists
        if group is None:
            await update.message.reply_text("Group not found. Please try again.")
            return
        # Retrieve the schedule entries for the group
        schedule_entries = session.query(Schedule).filter_by(group=group).all()
        # Check if there are schedule entries
        if not schedule_entries:
            await update.message.reply_text("No schedule entries found for the group.")
            return
        # Format and display the schedule
        schedule_text = "Розклад для групи: {}\n".format(group.name)
        for entry in schedule_entries:
            schedule_text += f"{entry.day_of_week}: {entry.time.strftime('%H:%M')} - {entry.subject} - аудиторія: {entry.room} - викладач: {entry.teacher_name}\n"

        await query.edit_message_text(schedule_text)

        await query.edit_message_text(schedule_text, reply_markup=get_back_keyboard())
        return BACK_TO_SCHEDULE_MENU


    async def back_to_schedule_menu(update: Update, context: CallbackContext) -> Callable[
        [Update, CallbackContext], Coroutine[Any, Any, int]]:
```

```
        await update.callback_query.answer()
        await update.callback_query.edit_message_text("Повертаємось до головного меню...",
                                                      reply_markup=choose_schedule_keyboard())
        return SELECTING_ACTION  # вихід до перегляду дій з розкладу


    async def select_group(update: Update, context: CallbackContext) -> int:
        query = update.callback_query
        option = query.data
        context.user_data['option'] = option

        logging.info(f"DEBUG: select_group - action: {option}")
        if option.startswith('select_group:'):
            selected_group_index = int(option[len('select_group:'):]) - 1
            groups = session.query(Group).all()
            selected_group_name = groups[selected_group_index].name

            context.user_data['group_name'] = selected_group_name
            logging.info(f"DEBUG: group_name - action: {selected_group_name}")
            await context.bot.send_message(chat_id=update.effective_chat.id, text=f"Ви обрали групу: {selected_group_name}")
            await query.edit_message_text("Введіть день тижня (наприклад, понеділок):")
            return SELECTING_DAY


    async def handle_selected_group(update, context):
```

```
        selected_group_name = context.user_data.get('selected_group')
        await update.message.reply_text(f"Ви обрали групу: {selected_group_name}")

        return SELECTING_DAY


    async def waiting_for_day_of_week(update: Update, context: CallbackContext) ->
int:
        day_of_week = update.message.text
        context.user_data['day_of_week'] = day_of_week

        await update.message.reply_text("Введіть час у форматі HH:MM:")
        return ENTER_TIME


    async def waiting_for_time(update: Update, context: CallbackContext) -> int:
        time_str = update.message.text

        try:
            # Розділіть години та хвилини та створіть об'єкт datetime.time
            hours, minutes = map(int, time_str.split(':'))
            time_object = time(hours, minutes)
        except ValueError:
            await update.message.reply_text("Невірний формат часу. Будь ласка,
використовуйте формат 'HH:MM'.")
            return ENTER_TIME

        context.user_data['time'] = time_object
```

```
    await update.message.reply_text("Введіть назву предмету:")
    return ENTER_SUBJECT


async def waiting_for_subject(update: Update, context: CallbackContext) -> int:
    subject = update.message.text
    context.user_data['subject'] = subject

    await update.message.reply_text("Введіть номер кабінету (необов'язково):")
    return ENTER_ROOM


async def waiting_for_room(update: Update, context: CallbackContext) -> int:
    room = update.message.text
    context.user_data['room'] = room
    # спеціальний внутрішній імпорт
    from handlers import handle_data

    # Отримати ім'я користувача з контексту
    get_login_name = handle_data("some_date", context)

    group_name = context.user_data['group_name']
    day_of_week = context.user_data['day_of_week']
    time_schedule = context.user_data['time']
    subject = context.user_data['subject']
    room = context.user_data['room']
```

```
        logging.info(f"DEBUG: waiting_for_room - teacher: {get_login_name}")

        teacher = session.query(User).filter_by(username=get_login_name).first()
        group = session.query(Group).filter_by(name=group_name).first()

        logging.info(f"DEBUG: waiting_for_room - group_name: {group_name}")
        logging.info(f"DEBUG: waiting_for_room - teacher: {teacher.username}")

        schedule = Schedule(day_of_week=day_of_week,
                            time=time_schedule,
                            subject=subject,
                            room=room,
                            group=group,
                            teacher_name=teacher.username)

        session.add(schedule)
        session.commit()

        await update.message.reply_text(f"Розклад для групи {group.name} додано.", reply_markup=get_back_keyboard())
        return BACK_TO_SCHEDULE_MENU


    # нижче реалізовано редагування розкладу. всі функції створені виключно для редагування

    # Приклад для функції отримання розкладу
    def get_schedule_entries(day_of_week, time):
```

```
        try:
            # Здійсніть запит до бази даних, отримуючи розклад для конкретного дня тижня та часу
            schedule_entries = session.query(Schedule).filter_by(day_of_week=day_of_week, time=time).all()
            return schedule_entries
        except Exception as e:
            print(f"Помилка при отриманні розкладу: {e}")
            return []


    # Приклад для функції редагування запису розкладу
    def edit_schedule_entry(schedule_id, new_data):
        try:
            # Здійсніть запит до бази даних, отримуючи потрібний запис розкладу
            schedule_entry = session.query(Schedule).get(schedule_id)

            if schedule_entry:
                # Змініть дані запису за допомогою нових даних
                schedule_entry.day_of_week = new_data['day_of_week']
                schedule_entry.time = new_data['time']
                schedule_entry.subject = new_data['subject']
                schedule_entry.room = new_data['room']
                schedule_entry.teacher_name = new_data['teacher_name']

                # Здійсніть збереження змін у базі даних
                session.commit()
```

```
            return True
        else:
            print(f"Запис розкладу із ID {schedule_id} не знайдено.")
            return False
    except Exception as e:
        print(f"Помилка при редагуванні запису розкладу: {e}")
        return False


async def edit_schedule(update: Update, context: CallbackContext) -> int:
    return SELECTING_DAY_EDIT


async def select_day_for_edit(update: Update, context: CallbackContext) -> int:
    day_of_week = update.message.text
    context.user_data['day_of_week'] = day_of_week

    await update.message.reply_text("Оберіть час у форматі HH:MM:")
    return ENTER_TIME_EDIT


async def enter_time_for_edit(update: Update, context: CallbackContext) -> int:
    time_str = update.message.text

    try:
        # Розділіть години та хвилини та створіть об'єкт datetime.time
        hours, minutes = map(int, time_str.split(':'))
        time_object = time(hours, minutes)
```

```
        except ValueError:
            await update.message.reply_text("Невірний формат часу. Будь ласка, використовуйте формат 'HH:MM'.")
            return ENTER_TIME_EDIT

        context.user_data['time'] = time_object

        # Отримати і відобразити список розкладу для вибраного дня і часу
        day_of_week = context.user_data['day_of_week']
        time_schedule = context.user_data['time']
        schedule_entries = get_schedule_entries(day_of_week, time_schedule)

        if not schedule_entries:
            await update.message.reply_text("Розклад не знайдено. Спробуйте інший день або час.", reply_markup=get_back_keyboard())
            return BACK_TO_SCHEDULE_MENU

        schedule_text = "Оберіть предмет для редагування:\n"
        for idx, entry in enumerate(schedule_entries, start=1):
            schedule_text += f"{idx}. {entry.subject} - аудиторія: {entry.room} - викладач: {entry.teacher_name}\n"

        await update.message.reply_text(schedule_text)
        return SELECT_SUBJECT_TO_EDIT


    async def select_subject_to_edit(update: Update, context: CallbackContext) -> int:
        selected_index = int(update.message.text) - 1
```

```
        schedule_entries = get_schedule_entries(context.user_data['day_of_week'], context.user_data['time'])

        if 0 <= selected_index < len(schedule_entries):
            selected_entry = schedule_entries[selected_index]
            context.user_data['selected_entry'] = selected_entry
            await update.message.reply_text(f"Обрано: {selected_entry.subject}. Введіть нові дані для редагування.")
            return ENTER_NEW_SCHEDULE_DATA

        await update.message.reply_text("Невірний вибір. Спробуйте ще раз.")
        return SELECT_SUBJECT_TO_EDIT


    async def enter_new_schedule_data(update: Update, context: CallbackContext) -> int:
        # Отримати нові дані і виконати редагування в базі даних
        new_subject = update.message.text
        # Отримати раніше вибраний запис
        selected_entry = context.user_data['selected_entry']

        # Редагувати дані запису в базі даних (це залежить від структури вашої бази)
        edit_schedule_entry(selected_entry, new_subject)

        await update.message.reply_text("Розклад відредаговано.", reply_markup=get_back_keyboard())
        return BACK_TO_SCHEDULE_MENU
    import logging
```

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.multiprocessing import Process, Queue
from telegram import Update
from telegram.ext import CallbackContext
from models import session, Schedule
from keyboards import get_main_menu_keyboard

SELECTING_AI, CHATTING = range(2)

# Отримання даних з таблиці schedules та їх обробка

schedules_query = session.query(Schedule).all()

schedule_inputs = [f"{schedule.day_of_week} {schedule.time} {schedule.subject}" for schedule in schedules_query]

schedule_targets = [schedule.room for schedule in schedules_query]

# Перетворення даних у числовий формат (замість текстового)

schedule_input_mapping = {schedule_data: idx for idx, schedule_data in enumerate(set(schedule_inputs))}

schedule_target_mapping = {room: idx for idx, room in enumerate(set(schedule_targets))}
```

```
    schedule_inputs = [[schedule_input_mapping[schedule_data]] for schedule_data in schedule_inputs]

    schedule_targets = [[schedule_target_mapping[room]] for room in schedule_targets]

    # Перетворення в тензори PyTorch

    schedule_inputs = torch.tensor(schedule_inputs, dtype=torch.float32)

    schedule_targets = torch.tensor(schedule_targets, dtype=torch.float32)

    # Enable logging
    logging.basicConfig(
        format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
    )
    logging.getLogger("httpx").setLevel(logging.WARNING)

    async def start_ai(update: Update, context: CallbackContext):
        return CHATTING



    async def start_chatting(update: Update, context: CallbackContext) -> str | None:
        user_text = update.message.text

        # Create a multiprocessing queue for passing the model between processes
        model_queue = Queue()
```

```
        # Create a separate process for training the model
        training_process = Process(target=train_schedule_model, args=(schedule_inputs, schedule_targets,
                                                                       len(schedule_input_mapping),
                                                                       len(schedule_target_mapping),
                                                                       model_queue))
        training_process.start()
        training_process.join()

        # Get the trained model from the queue
        trained_model = model_queue.get()
        logging.info(f"DEBUG: select_group - action: {trained_model}")

        # Use the trained model for making predictions
        with torch.no_grad():
            schedule_output = trained_model(torch.tensor([[schedule_input_mapping[user_text]]], dtype=torch.float32))
            schedule_output = torch.argmax(schedule_output, dim=1)
            predicted_room = [key for key, value in schedule_target_mapping.items() if value == schedule_output.item()][0]

        # Respond to the user
        await update.message.reply_text(f"Предмет: {user_text}, Аудиторія: {predicted_room}",
                                        reply_markup=get_main_menu_keyboard())
        return "MAIN_MENU"
```

```
# Визначення архітектури нейронної мережі для розкладу
class ScheduleNN(nn.Module):
    def __init__(self, input_size, output_size):
        super(ScheduleNN, self).__init__()
        self.fc = nn.Linear(input_size, output_size)

    def forward(self, x):
        x = self.fc(x)
        return x


# Ініціалізація та навчання моделі
def train_schedule_model(inputs, targets, input_size, output_size, queue):
    model = ScheduleNN(input_size, output_size)
    criterion = nn.CrossEntropyLoss()
    optimizer_schedule = optim.SGD(model.parameters(), lr=0.01)

    epochs = 1000
    for epoch in range(epochs):
        optimizer_schedule.zero_grad()
        schedule_outputs = model(inputs)
        schedule_loss = criterion(schedule_outputs, targets)
        schedule_loss.backward()
        optimizer_schedule.step()

        if (epoch + 1) % 100 == 0:
```

```
            print(f'Epoch [{epoch + 1}/{epochs}], Schedule Loss: {schedule_loss.item():.4f}')

            # Put the trained model into the queue for later retrieval
            queue.put(model)
        # models.py
        from sqlalchemy import create_engine, Column, Integer, String, ForeignKey, Table, DateTime, Boolean, Time
        from sqlalchemy.orm import declarative_base, Session, relationship, sessionmaker

        engine = create_engine('sqlite:///university.db', echo=True)
        Base = declarative_base()
        SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
        session = SessionLocal()


        class Role(Base):
            __tablename__ = 'roles'

            id = Column(Integer, primary_key=True)
            name = Column(String(50), unique=True, nullable=False)

            users = relationship('User', back_populates='role')


        class User(Base):
            __tablename__ = 'users'
```

```
    id = Column(Integer, primary_key=True)
    telegram_id = Column(Integer, nullable=False)
    username = Column(String(50), nullable=False)
    role_name = Column(String(20), ForeignKey('roles.name'))
    is_teacher = Column(Boolean, default=False)

    role = relationship('Role', back_populates='users')

    group_name = Column(Integer, ForeignKey('groups.name'))
    group = relationship('Group', back_populates='users')


class Schedule(Base):
    __tablename__ = 'schedules'
    id = Column(Integer, primary_key=True)
    day_of_week = Column(String, nullable=False)
    time = Column(Time, nullable=False)
    subject = Column(String, nullable=False)
    room = Column(String)
    group_id = Column(Integer, ForeignKey('groups.id'))
    group = relationship("Group", back_populates="schedule")
    teacher_name = Column(Integer, ForeignKey('users.username'))


class Group(Base):
    __tablename__ = 'groups'
    id = Column(Integer, primary_key=True)
    name = Column(String, unique=True)
```

```
    users = relationship("User", back_populates="group")
    schedule = relationship("Schedule", back_populates="group")
    tasks = relationship("Task", back_populates="group")


class Task(Base):
    __tablename__ = 'tasks'
    id = Column(Integer, primary_key=True)
    name = Column(String)
    deadline = Column(DateTime)
    group_id = Column(Integer, ForeignKey('groups.id'))
    group    = relationship("Group", back_populates="tasks")


Base.metadata.create_all(engine)
# keyboards.py
from telegram import InlineKeyboardButton, InlineKeyboardMarkup

from models import session, Group


def get_start_menu_keyboard():
    keyboard = [
        [InlineKeyboardButton("Реєстрація", callback_data="REGISTER"),
         InlineKeyboardButton("Вхід", callback_data="LOGIN")],
    ]
    return InlineKeyboardMarkup(keyboard)
```

```
def get_registration_role_keyboard():
    keyboard = [
        [InlineKeyboardButton("Студент", callback_data="student"),
         InlineKeyboardButton("Викладач", callback_data="teacher")],
    ]
    return InlineKeyboardMarkup(keyboard)


def get_main_menu_keyboard():
    keyboard = [
        [InlineKeyboardButton("Розклад", callback_data="manage_schedule"),
         InlineKeyboardButton("Повідомлення", callback_data="ask_question")],
        [InlineKeyboardButton("Використати ІШІ", callback_data="use_AI"),
         InlineKeyboardButton("Вийти", callback_data="END_ROUTES")],
    ]
    return InlineKeyboardMarkup(keyboard)


def get_back_keyboard():
    keyboard = [
        [InlineKeyboardButton("Вийти", callback_data="END_ROUTES")],
    ]
    return InlineKeyboardMarkup(keyboard)


def choose_schedule_keyboard():
```

```
        keyboard = [[InlineKeyboardButton("Додати розклад", callback_data='add_lesson'),
                     InlineKeyboardButton("Редагувати розклад", callback_data='edit_lesson')],
                    [InlineKeyboardButton("Переглянути розклад", callback_data='show_schedule'),
                     InlineKeyboardButton("Назад до меню", callback_data='cancel')],
                    ]
        return InlineKeyboardMarkup(keyboard)


    def get_group_keyboard():
        groups = session.query(Group).all()
        group_buttons = [
            [InlineKeyboardButton(group.name, callback_data=f"select_group:{i}")]
            for i, group in enumerate(groups, start=1)
        ]
        return InlineKeyboardMarkup(group_buttons)


    def get_main_menu():
        back_button = [InlineKeyboardButton("Назад", callback_data="get_main_menu")],
        return InlineKeyboardMarkup(back_button)


    def get_back_keyboard():
```

```
    back_to_main_menu_button = InlineKeyboardButton("Назад", callback_data='back_to_main_menu')
    return InlineKeyboardMarkup([[back_to_main_menu_button]])
```