МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Ігор ШЕЛЕХОВ
_____
(підпис)
червня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 «Комп'ютерні науки»
освітньо-професійної програми «Інформатика»
на тему: «Інформаційна система планування і організації навчальної діяльності»
здобувача групи ІН-01 Безрука Владислава Миколайовича

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

_____ Владислав БЕЗРУК
(підпис)

Керівник ст. викл.,
канд. фіз.-мат. наук, доц.
Оксана ШОВКОПЛЯС _____
(підпис)

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 «Комп'ютерні науки» освітньо-професійної програми «Інформатика»
здобувача групи ІН-01 Безрука Владислава Миколайовича

1. Тема роботи: «Інформаційна система планування і організації навчальної діяльності» затверджую наказом по СумДУ від *«00» червня 2024 р. №* _____
2. Термін здачі здобувачем кваліфікаційної роботи *до 28 травня 2024 року* _____
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для розробки інформаційної системи. 3) Розробка інформаційної системи планування і організації навчальної діяльності . 4) Аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проєкту (роботи), із значенням розділів проєкту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «06» травня 2024 р.

Завдання прийняв до виконання _____ (підпис) Керівник _____ (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*		
2	*Огляд технологій, що використовуються для розробки телеграм-ботів*		
3	*Розробка інформаційної системи планування і організації навчальної діяльності*		
4	*Аналіз отриманих результатів*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ (підпис) Керівник _____ (підпис)

АНОТАЦІЯ

Записка: 63 стр., 25 рис., 2 табл., 2 додатки, 15 використаних джерел.

Обґрунтування актуальності теми роботи – тема кваліфікаційної роботи є актуальною, оскільки присвячена розв'язанню важливої практичної задачі розробки телеграм-бота як надійного помічника для спрощення доступу до навчальних ресурсів та вдосконалення комунікації між здобувачами освіти та викладачами.

Об'єкт дослідження — процес розробки та впровадження інформаційної системи для планування і організації навчальної діяльності здобувачів освіти.

Мета роботи — розробка та впровадження інформаційної системи у вигляді телеграм-бота, спрямованого на полегшення навчальної діяльності здобувачів освіти.

Методи дослідження — алгоритми розробки та впровадження інформаційних систем у вигляді телеграм-ботів.

Результати — розроблено інформаційну систему у вигляді телеграм-бота. Система надає змогу здобувачам освіти отримувати розклад занять, автоматичні сповіщення про наближення занять, та допомогу під час дистанційного навчання. Проведено тестування розробленого рішення та розроблено план подальшого вдосконалення інформаційної системи.

ІНФОРМАЦІЙНА СИСТЕМА, ТЕЛЕГРАМ-БОТ, РОЗКЛАД ЗАНЯТЬ, АВТОМАТИЧНІ СПОВІЩЕННЯ, ДИСТАНЦІЙНЕ НАВЧАННЯ, PYTHON, AIOGRAM, JSON, ASYNCIO, PYCHARM

ЗМІСТ

ВСТУП

Актуальність. Сучасне навчання в університеті стає все більш залежним від інформаційних технологій, що значно полегшують доступ до навчальних ресурсів та покращують організацію навчальної діяльності здобувачів освіти. Однак, із зростанням обсягу доступної інформації і зростанням вимог до ефективної організації навчального процесу, виникає необхідність в інструментах, які спрощували б цей процес для здобувачів освіти. Саме тут і виникає актуальність розробки телеграм бота, який буде вірним помічником у цьому завданні. Тема кваліфікаційної роботи є актуальною, оскільки присвячена розв'язанню важливої практичної задачі розробки телеграм-бота як надійного помічника для спрощення доступу до навчальних ресурсів та вдосконалення комунікації між здобувачами освіти та викладачами.

Об'єкт дослідження. Процес розробки та впровадження інформаційної системи для планування і організації навчальної діяльності здобувачів освіти.

Предмет дослідження. Методи та технології розробки інформаційних систем у вигляді телеграм-ботів для планування і організації навчальної діяльності.

Гіпотеза. Використання телеграм-бота для організації та планування навчальної діяльності значно спростить доступ до навчальних ресурсів, покращить комунікацію між здобувачами освіти та викладачами, а також підвищить ефективність навчального процесу в університеті.

Наукова новизна. Розробка телеграм-бота для підтримки навчальної діяльності здобувачів освіти є інноваційним підходом у сучасному освітньому середовищі. Новизна дослідження полягає в застосуванні передових методів та технологій, таких як Python та Aiogram, для створення програмного засобу, який забезпечує високу продуктивність та надійність. Розроблений бот має потенціал для розширення функціоналу, включаючи інтерактивні навчальні завдання та спеціалізовану підтримку під час екзаменів, що додасть ще

більшої користі здобувачам освіти.

Апробація матеріалів роботи. Основні результати роботи оприлюднені та обговорені на міжнародній науково-технічній конференції здобувачів освіти та молодих вчених «Інформатика, математика, автоматика» (ІМА – 2024). Тема доповіді: «Телеграм-бот для планування та організації навчальної діяльності студента»

Зв'язок роботи з науковою темою. Кваліфікаційна робота виконана на кафедрі комп'ютерних наук та пов'язана з виконанням науково-дослідної роботи № 0118U006971 «Методи, математичні моделі та інформаційні технології аналізу і синтезу інфокомунікаційних систем» (2018-2023).

Структура. Дана робота складається зі вступу, аналітичного огляду, постановки задачі, вибір методу розв'язання поставленої задачі, опису програмного забезпечення інформаційної системи, висновків, списку використаних джерел та додатків.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Аналіз предметної області

В сучасному освітньому середовищі інформаційні технології здійснюють значний вплив на навчальний процес. Онлайн навчання стало необхідною складовою сучасної освіти, особливо в умовах зростаючого використання електронних ресурсів, доступних здобувачам освіти і викладачам.

Телеграм боти - це програмні засоби, розроблені для використання в месенджері Telegram (рис. 1.1). Вони можуть виконувати різні завдання, включаючи надання інформації, виконання функцій інтерактивного спілкування та автоматизацію процесів [8].

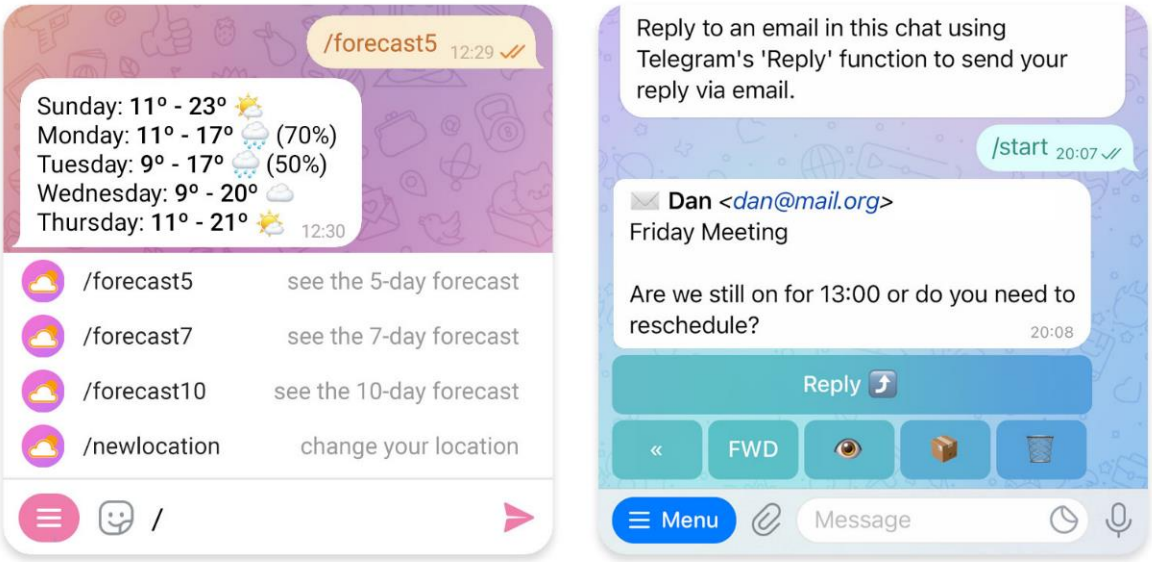

Рисунок 1.1 – Приклад роботи телеграм бота

Телеграм боти стали особливо популярними в освітній галузі завдяки їхній здатності полегшити доступ до навчальних ресурсів та покращити комунікацію між здобувачами освіти та викладачами.

Використання телеграм ботів у сфері освіти стало особливо актуальним в останні роки. Ці інтелектуальні агенти можуть виконувати різні завдання, спрощуючи та полегшуючи навчальний процес:

Телеграм боти можуть надавати здобувачам освіти доступ до лекційних матеріалів, навчальних посібників, відеолекцій, тестових завдань і інших навчальних ресурсів. Це полегшує процес самостійного вивчення матеріалу та робить його більш доступним [5, 9].

Телеграм боти можуть надавати інформацію про розклад занять (рис. 1.2), дати та час проведення лекцій, практичних занять, семінарів та інших навчальних подій [4].

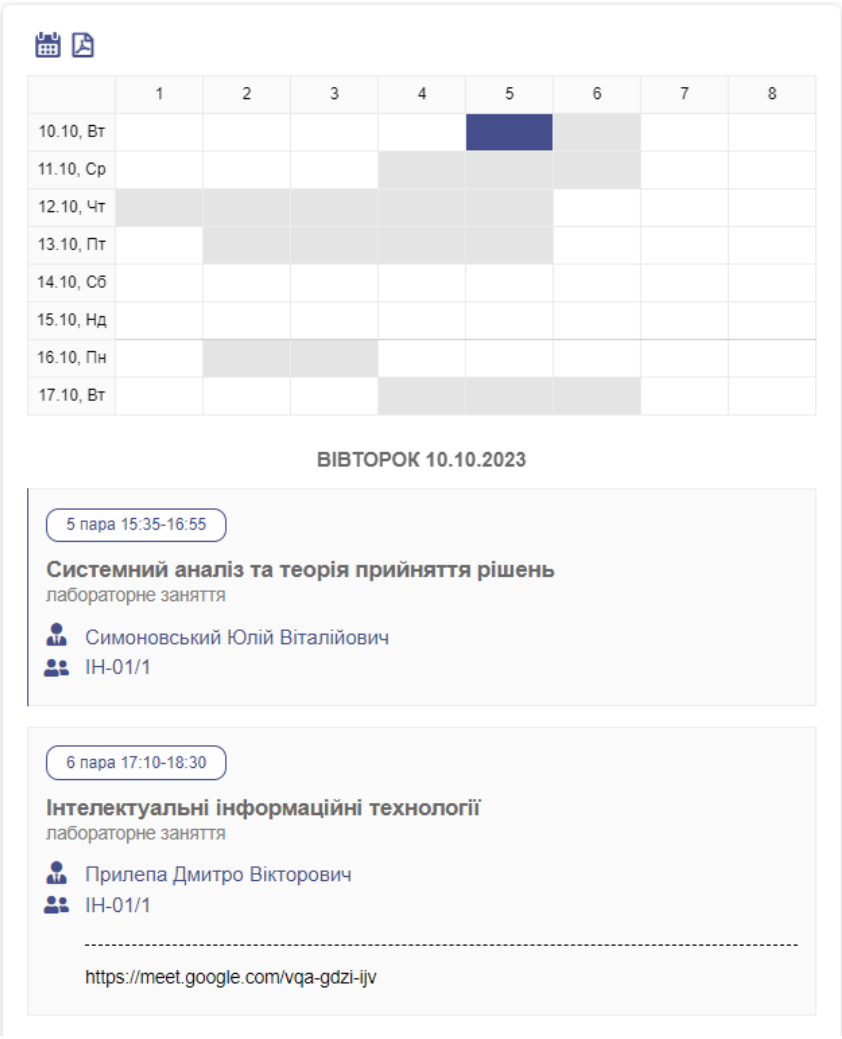

Рисунок 1.2 – Розклад занять в університеті

Це допомагає здобувачам освіти ефективно планувати свій навчальний графік [2].

Телеграм боти можуть включати інтерактивні завдання, тести та вправи, які допомагають здобувачам освіти закріплювати знання та вдосконалювати свої навички.

Боти можуть надсилати здобувачам освіти оповіщення та нагадування про важливі події, такі як дедлайни здачі робіт, дати екзаменів та інші важливі події.

З урахуванням росту популярності онлайн навчання та залежності від інформаційних технологій, розробка та впровадження телеграм бота для полегшення навчального процесу стає актуальним завданням. Важливо розробляти інноваційні рішення, що сприяють підвищенню якості освіти та забезпечують більш зручний та ефективний доступ до навчальних ресурсів.

1.2 Огляд існуючих рішень

Під час аналізу аналогічних проєктів було виявлено цікавий телеграм-бот @rozklad_bot.

Деякі функції цього бота:

- отримання розкладу занять в деяких університетах України (рис. 1.3): цей бот надає здобувачам освіти можливість зручного доступу до розкладу занять, що спрощує організацію навчального процесу;
- встановлення будильника для нагадувань (рис. 1.4): здобувачі освіти можуть встановити будильник за допомогою цього бота, щоб отримувати нагадування про найважливіші події та завдання.

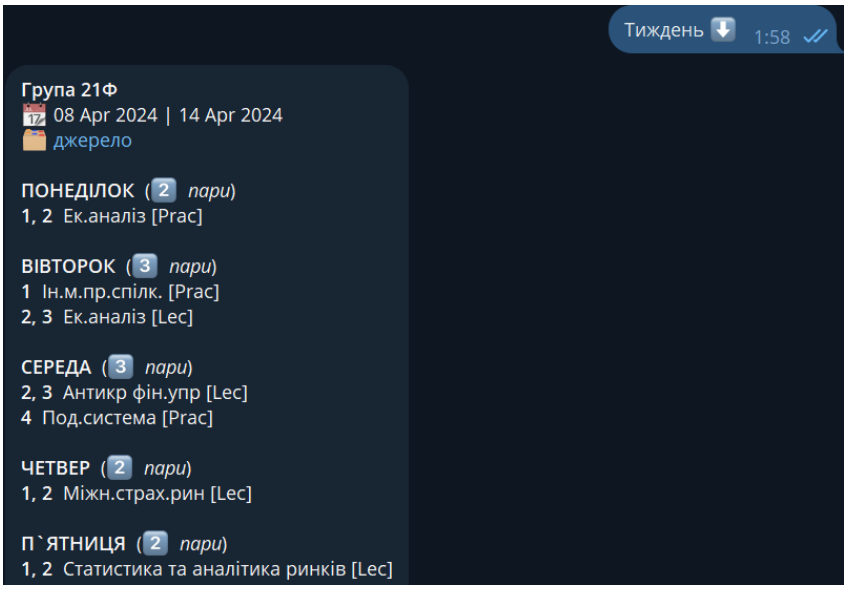

Рисунок 1.3 – Приклад роботи існуючого телеграм-бота: розклад на тиждень

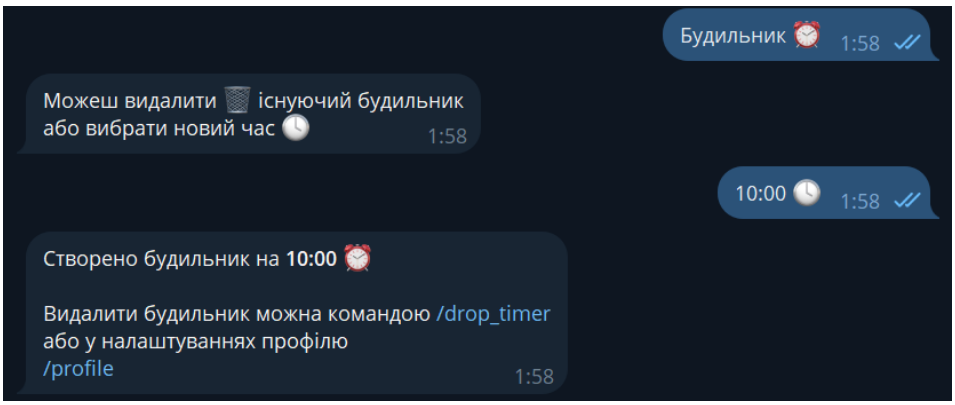

Рисунок 1.4 – Приклад роботи існуючого телеграм-бота: встановлення будильника для нагадувань

1.3 Постановка задачі

За результатами інформаційного огляду та аналізу аналогічних програмних продуктів, сформовано завдання роботи:

Головною метою дослідження є розробка та впровадження телеграм бота, спрямованого на полегшення навчальної діяльності здобувачів освіти університету в онлайн середовищі. Цей бот має надавати зручний та ефективний інструмент для отримання інформації про розклад занять, доступ до навчальних матеріалів, а також інші корисні функції, спрямовані на покращення якості навчання.

Для досягнення поставленої мети дослідження були сформульовані наступні завдання:

- аналіз потреб здобувачів освіти: провести аналіз потреб та вимог здобувачів освіти до функціоналу телеграм бота, спрямованого на полегшення навчального процесу;
- розробка архітектури бота: розробити архітектуру телеграм бота, визначити його основні функції та можливості;
- розробка інтерфейсу: створити зручний та інтуїтивно зрозумілий інтерфейс для взаємодії користувачів з ботом;
- інтеграція з навчальними системами: забезпечити інтеграцію телеграм бота з навчальними платформами та системами університету для отримання актуальної інформації про розклад занять та навчальні матеріали;

- розробка функціоналу для навчання: розробити функціонал для навчання, включаючи можливість перегляду лекцій, завдань, тестів та інших навчальних матеріалів;
- розробка системи нагадувань: впровадити систему нагадувань для здобувачів освіти про найважливіші події та дедлайни;
- тестування та вдосконалення: провести тестування розробленого бота, зібрати відгуки користувачів та внести необхідні виправлення та покращення.

Очікується, що результатом цього дослідження буде повнофункціональний телеграм бот, який забезпечить здобувачам освіти університету зручний доступ до навчальних ресурсів, допоможе в організації навчального процесу та сприятиме підвищенню якості навчання в онлайн середовищі.

2 ВИБІР МЕТОДУ РОЗВ'ЯЗАННЯ ЗАДАЧІ

2.1 Інформаційна модель

Розглянемо інформаційну модель: опис структури та взаємозв'язку основних компонентів системи.

Атрибути користувача: ПІПб, статус (здобувач освіти/викладач), група (якщо здобувач освіти), предмети (якщо викладач).

Операції користувача: авторизація, перегляд розкладу, отримання сповіщень, зміна налаштувань.

Атрибути розкладу: дата, час, група/викладач, предмет, аудиторія, посилання на додаткові матеріали.

Операції над розкладом: додавання нового заняття, редагування/видалення заняття, пошук занять.

Атрибути сповіщень: тип (для здобувачів освіти, для викладачів), текст повідомлення, час надсилання.

Операції над сповіщеннями: відправлення сповіщень до здобувачів освіти/викладачів, налаштування часу сповіщень.

Детальніше розглянемо інформаційну модель телеграм-бота (рис. 2.1):

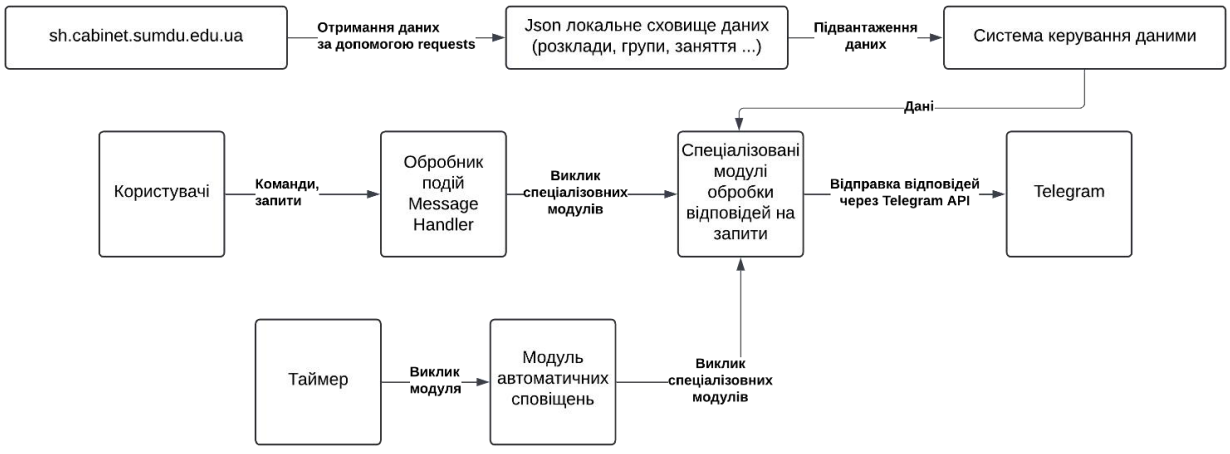

Рисунок 2.1 – Інформаційна модель телеграм-бота

Сумський державний університет зберігає розклад, дані про групи та предмети на сайті особистого кабінету. За допомогою простих запитів телеграм-бот отримує дані про розклад, групи та предмети з особистого

кабінету. Потім ці дані зберігаються у форматі JSON. Цей формат дозволяє легко зберігати, редагувати та отримувати дані.

Система керування даними підвантажує збережені дані у систему телеграм-бота, щоб бот міг оперувати ними. Запит або надіслана команда користувачем надходить до обробника подій, який працює у паралельному потоці. Потім обробник подій опрацьовує команду та викликає спеціалізований модуль.

У свою чергу, спеціалізований модуль готує результат та формує відповідь користувачеві на його введену команду. Потім ця сформована відповідь повертається користувачеві в телеграм.

Також паралельно до усіх модулів діє ще модуль автоматичних сповіщень, який спрацьовує за таймером. Цей модуль автоматично надсилає сповіщення в телеграм про найближчі заняття в групі.

2.2 Вибір засобів програмної реалізації

Для розробки телеграм бота була обрана мова програмування Python версії 3.9. Вибір Python обумовлений його простотою в освоєнні, широкою підтримкою спільноти розробників, а також наявністю численних бібліотек та фреймворків, що допомагають в розробці та взаємодії з іншими сервісами [7].

Для взаємодії з телеграмом та розробки функціоналу телеграм бота був використаний фреймворк aiogram. Aiogram - асинхронний фреймворк для роботи з Telegram ботами на Python [6, 10].

Цей фреймворк надає зручний інтерфейс для створення та керування телеграм ботами, а також дозволяє взаємодіяти з Telegram API [1, 8].

Основні переваги використання aiogram:

- асинхронність: фреймворк побудований з використанням asyncio, що дозволяє виконувати багато завдань паралельно та асинхронно [6];
- зручна обробка повідомлень: aiogram надає розробникам можливість легко обробляти повідомлення вхідних користувачів та відправляти їм відповіді;

- багатофункціональність: фреймворк підтримує велику кількість функцій Telegram API, що дозволяє розробляти різноманітний функціонал для бота [6].

Для забезпечення асинхронності та паралельності виконання завдань був використаний фреймворк asyncio. Це дозволило розробляти бота таким чином, щоб він міг одночасно відповідати на повідомлення користувачів, виконувати запити до зовнішніх сервісів та оновлювати інформацію.

Асинхронність є важливим аспектом при роботі з багатьма користувачами, оскільки дозволяє ефективно виконувати багато операцій паралельно без блокування виконання інших завдань [9].

Для реалізації можливості виконання математичних обчислень прямо в телеграмі був використаний фреймворк wolframalpha для Python [7]. Він дозволяє здійснювати запити до веб-сервісу Wolfram Alpha і отримувати обчислені результати.

Завдяки цьому фреймворку користувачі бота можуть легко виконувати різні математичні обчислення, отримувати відповіді на питання та використовувати бота як корисний інструмент для навчання та розв'язання задач.

Також було обрано модульну парадигму програмування, яка дозволяє розділити програму на окремі компоненти [12]. Це забезпечує ряд переваг, зокрема:

- простота розробки: модульна структура дозволяє розбити складні задачі на менші, більш керовані частини [12]. Кожен модуль відповідає за конкретну функціональність, що полегшує процес розробки та дозволяє розподілити завдання між розробниками;
- покращене тестування: модулі можуть бути протестовані окремо, що дозволяє виявляти та виправляти помилки на ранніх етапах розробки [12]. Це сприяє підвищенню якості програмного забезпечення та зниженню кількості помилок у фінальному продукті;
- підтримка та оновлення: модульний підхід спрощує підтримку та

оновлення програмного забезпечення. Окремі модулі можуть бути замінені або оновлені без впливу на інші частини програми, що робить систему більш гнучкою та стійкою до змін;

- повторне використання коду: модулі можуть бути повторно використані в інших проєктах, що знижує витрати часу та ресурсів на розробку нових функціональностей [12]. Це сприяє підвищенню ефективності роботи команди розробників;
- зрозуміла структура коду: модульна парадигма допомагає створити зрозумілу та організовану структуру коду [12]. Це полегшує розуміння і супроводження проєкту як для поточних, так і для нових членів команди.

Для розробки було обрано середовище програмування PyCharm (рис. 2.2), яке забезпечує потужні інструменти для ефективної розробки на Python. Використання PyCharm надає ряд переваг:

- інтелектуальний кодовий редактор: pyCharm пропонує автодоповнення, інспекцію коду та інтегровані підказки, що допомагають писати чистий та безпомилковий код [13]. Це значно підвищує продуктивність розробників;
- підтримка фреймворків: pyCharm має вбудовану підтримку для популярних Python-фреймворків, таких як Django, Flask, Pyramid та багатьох інших [13]. Це спрощує розробку веб-додатків та забезпечує інтеграцію з різними інструментами та бібліотеками;
- зручна робота з тестуванням: pyCharm підтримує різні фреймворки для тестування, включаючи unittest, pytest і nose. Це полегшує процес написання, запуску та аналізу тестів, забезпечуючи якість програмного забезпечення [13];
- інтеграція з системами контролю версій: pyCharm підтримує інтеграцію з Git, SVN, Mercurial та іншими системами контролю версій [13]. Це забезпечує зручне управління версіями коду, спрощує співпрацю та контроль змін у проєкті;

- рефакторинг коду: pyCharm надає потужні інструменти для рефакторингу, включаючи перейменування змінних, методів і класів, а також можливість безпечно переміщувати та змінювати структуру коду [13]. Це допомагає підтримувати чистоту та організованість коду в проєкті;
- інтегрована середовище розробки (IDE): pyCharm об'єднує всі необхідні інструменти в одному середовищі – редактор коду, інтегрований термінал, дебагер, профайлер, інструменти для роботи з базами даних та інше [13]. Це підвищує зручність і ефективність роботи розробників;
- підтримка різних мов програмування: окрім Python, PyCharm підтримує роботу з HTML, CSS, JavaScript, SQL та багатьма іншими мовами, що робить його універсальним інструментом для повноцінної розробки веб-додатків [13];
- велика кількість плагінів: pyCharm підтримує велику кількість плагінів, що дозволяє розширювати функціональність IDE відповідно до потреб проєкту та індивідуальних вимог розробників [13].

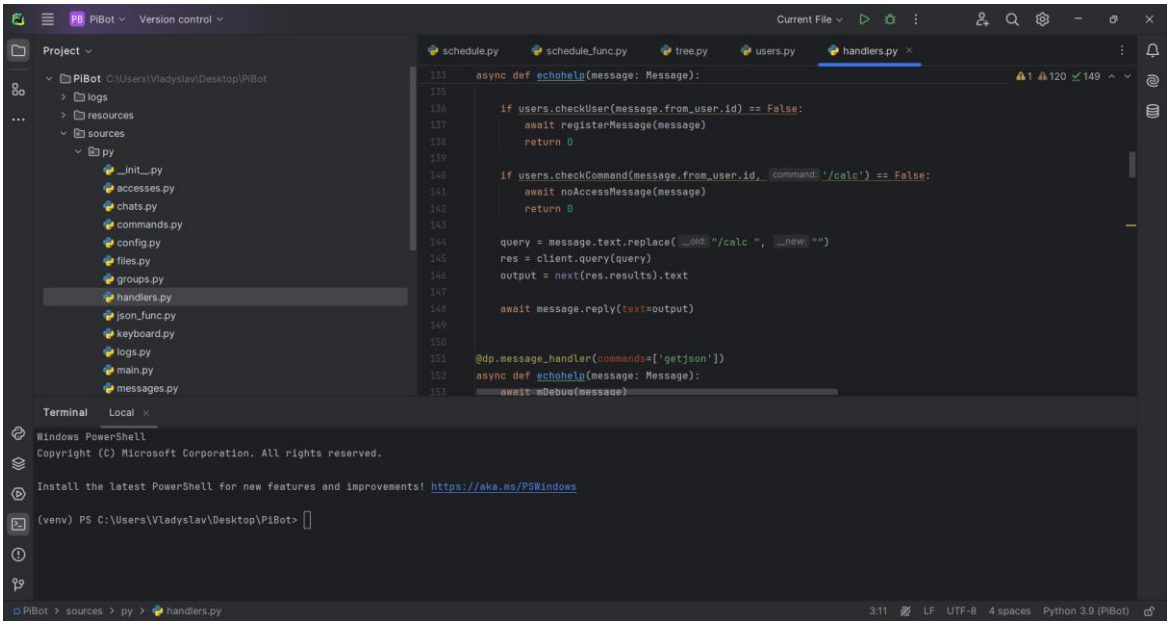

Рисунок 2.2 – Середовище розробки PyCharm

У проєкті для зберігання даних користувачів у телеграм-боті обрано формат JSON. JSON (JavaScript Object Notation) - це текстовий формат обміну даними, який легко читається людьми та зручно парсується комп'ютерами. Він є популярним вибором для зберігання та передачі структурованих даних

завдяки своїй простоті та гнучкості.

Основні переваги використання JSON:

- легкість у використанні: JSON має просту та зрозумілу структуру, що полегшує його використання для зберігання та обміну даними між різними системами [14]. Це особливо корисно при роботі з API, де JSON часто використовується як формат для передачі даних;
- сумісність: JSON підтримується практично всіма мовами програмування, що робить його універсальним інструментом для обміну даними [14]. Це дозволяє легко інтегрувати дані з різних джерел і використовувати їх у телеграм-боті;
- легкість у читанні та налагодженні: завдяки своїй структурі, JSON-файли легко читаються як розробниками, так і автоматичними парсерами [14]. Це спрощує процес налагодження та забезпечує зрозумілість збережених даних.

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Формування вхідних даних

Розглянемо детальніше вхідні дані, необхідні для роботи інформаційної системи (табл. 3.1).

Таблиця 3.1 – Формування вхідних даних

Назва даних	Мета	Методи	Вхідні дані
Збір інформації про розклад занять.	Отримати доступ до актуального розкладу занять для подальшого використання ботом.	Інтеграція з університетськими інформаційними системами, такими як системи управління навчанням (LMS), використання API (якщо доступне) для автоматичного.	Файли з розкладом у форматах XML, CSV, JSON або інших, зручних для обробки.
Інтеграція з навчальними платформами.	Забезпечити доступ до навчальних матеріалів через телеграм-бот.	Використання API навчальних платформ для отримання інформації про доступні матеріали, налаштування доступу до файлів, розміщених на хмарних сервісах (Google Drive, OneDrive).	Дані з API навчальних платформ, посилання на хмарні сховища з навчальними матеріалами.
Розробка системи нагадувань.	Впровадити систему автоматичних сповіщень для здобувачів освіти.	Збір інформації про важливі події та дедлайни від викладачів та адміністраторів, інтеграція з календарями та планувальниками.	Дані про розклад занять та події, інформація про дедлайни та важливі дати з університетських систем, календарні події та нагадування.

3.2 Опис програмної реалізації

Для роботи телеграм бота була обрана операційна система Linux, а саме, вона запускалася на одноплатному комп'ютері на базі архітектури ARM. Використання Linux дозволило забезпечити стабільну та ефективну роботу бота. На рис. 3.1 продемонстровано процес підключення до віддаленої системи.

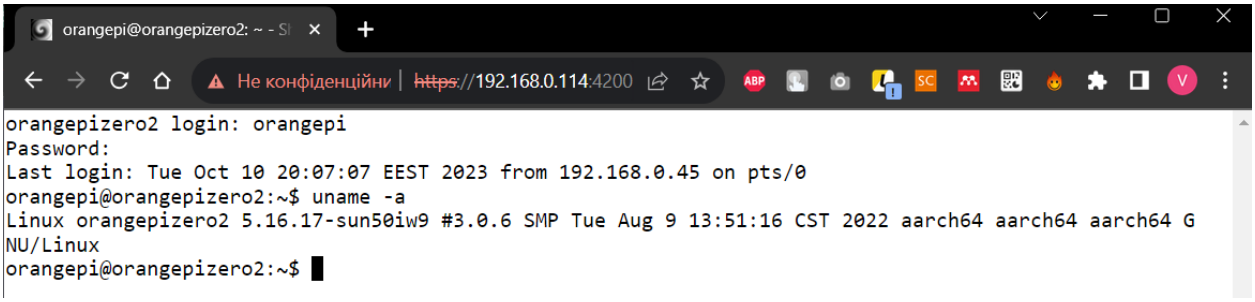

Рисунок 3.1 Підключення до віддаленої системи Linux

Для розробки програмного забезпечення було використано середовище розробки PyCharm для мови програмування Python [5, 7]. Це середовище надало зручний інтерфейс для розробки, відлагодження та тестування телеграм бота. В якості бібліотеки для роботи з Telegram API було обрано aiogram (рис. 3.2).

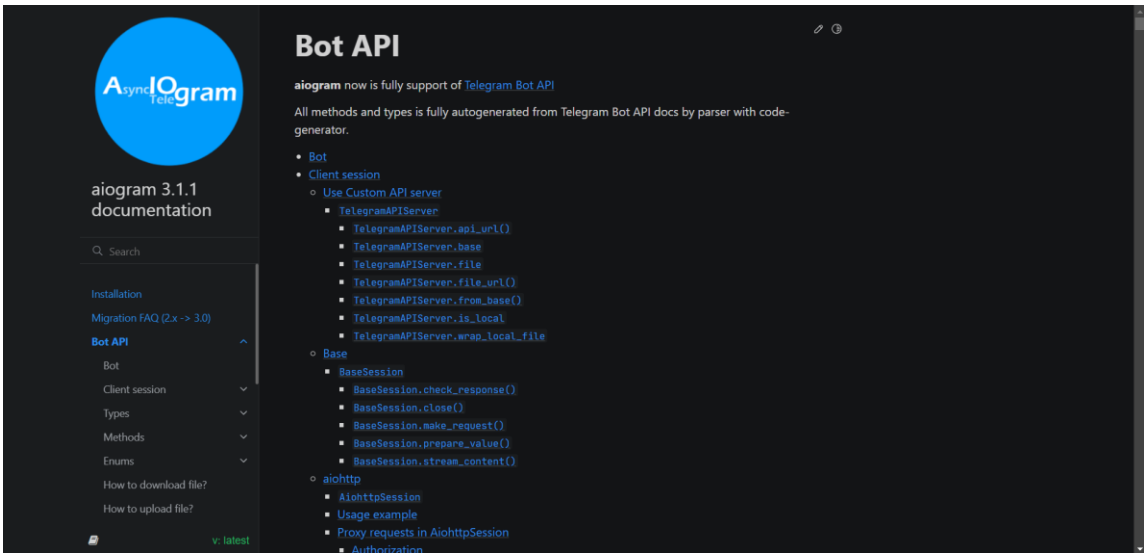

Рисунок 3.2 – Aiogram Bot API для розробки телеграм бота [10]

Сервіс для математичних обчислень WolframAlpha було обрано як інструмент для обробки математичних задач у телеграм-боті (рис. 3.3).

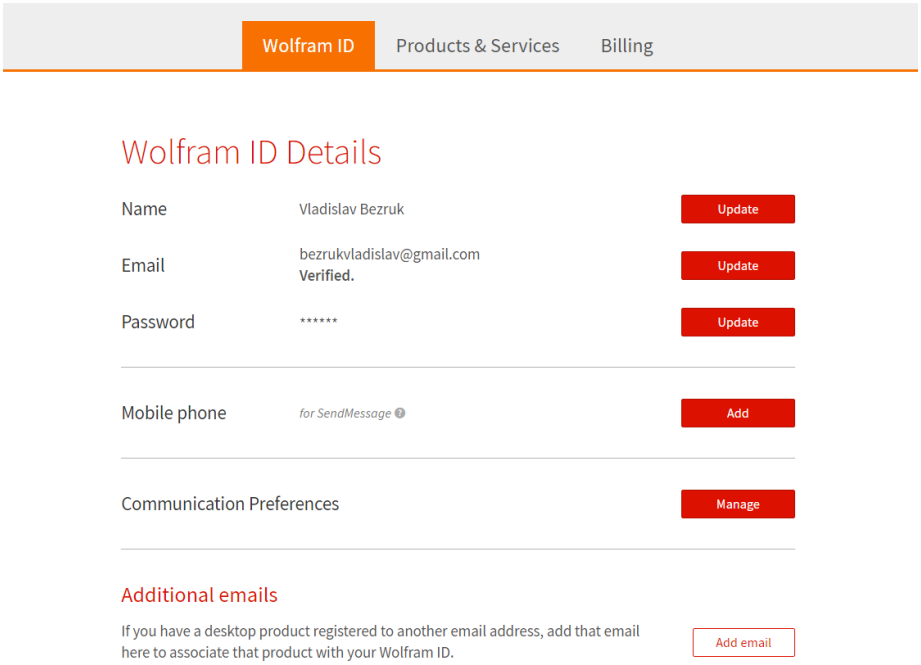

Рисунок 3.3 – Сервіс wolfram alpha для впровадження його API в різні системи

Телеграм має вбудований інструмент для створення інших телеграм-ботів користувачами (рис. 3.4).

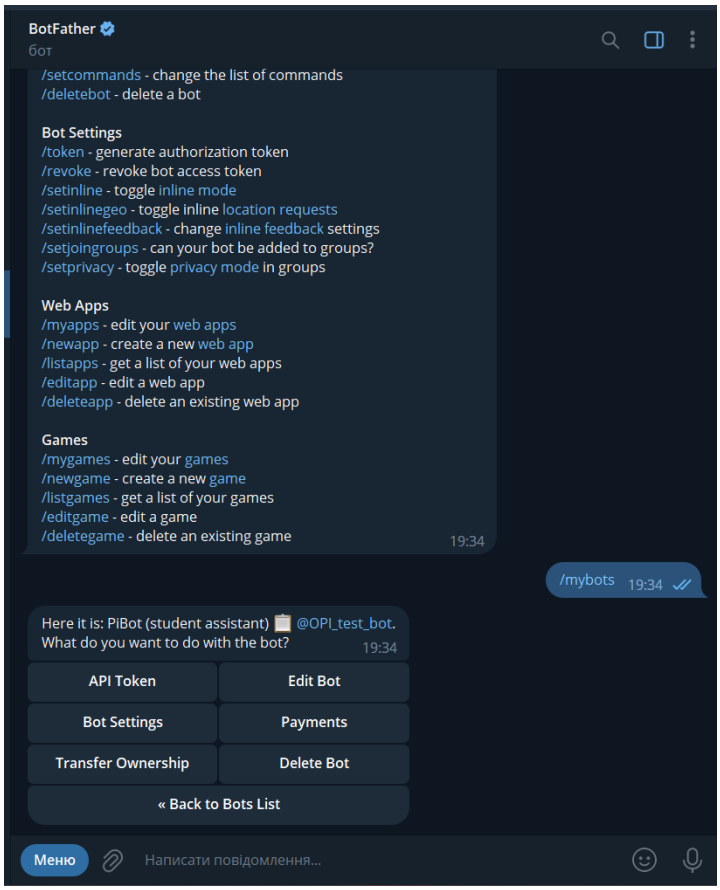

Рисунок 3.4 – Головний бот BotFather потрібнрий для створення ботів в телеграмі [2, 8]

Бот має різноманітний функціонал та команди, які надають користувачам зручний інструмент для отримання необхідної інформації та взаємодії з ботом.

До найважливіших команд бота включаються:

- /about: інформація про бота та розробників;
- /send: відправити анонімне повідомлення;
- /addGroup: встановити групу для автоматичного сповіщення занять;
- /removeGroup: видалити групу для автоматичного сповіщення занять;
- /date: розклад по даті;
- /calc: обрахувати приклад, рівняння, задати питання;
- /now: посилання на наступну пару;
- /today: пари сьогодні;
- /tomorrow: пари завтра;
- /week: розклад на тиждень;
- /setgroup: встановити групу;
- /setLink: встановити посилання на заняття.

Усі ці команди та функції бота спрямовані на полегшення навчального процесу здобувачів освіти та надання їм зручного інструменту для організації навчання.

Програмний код телеграм бота був структурований та організований з урахуванням принципів чистого та модульного програмування [12]. Це дозволило підтримувати та розширювати функціонал бота з легкістю.

Програмний код представлений у додатку лістинг програми (див. додаток А Лістинг програмного коду).

На рис. 3.5 представлена структурна діаграма всіх модулів розробленого програмного забезпечення.

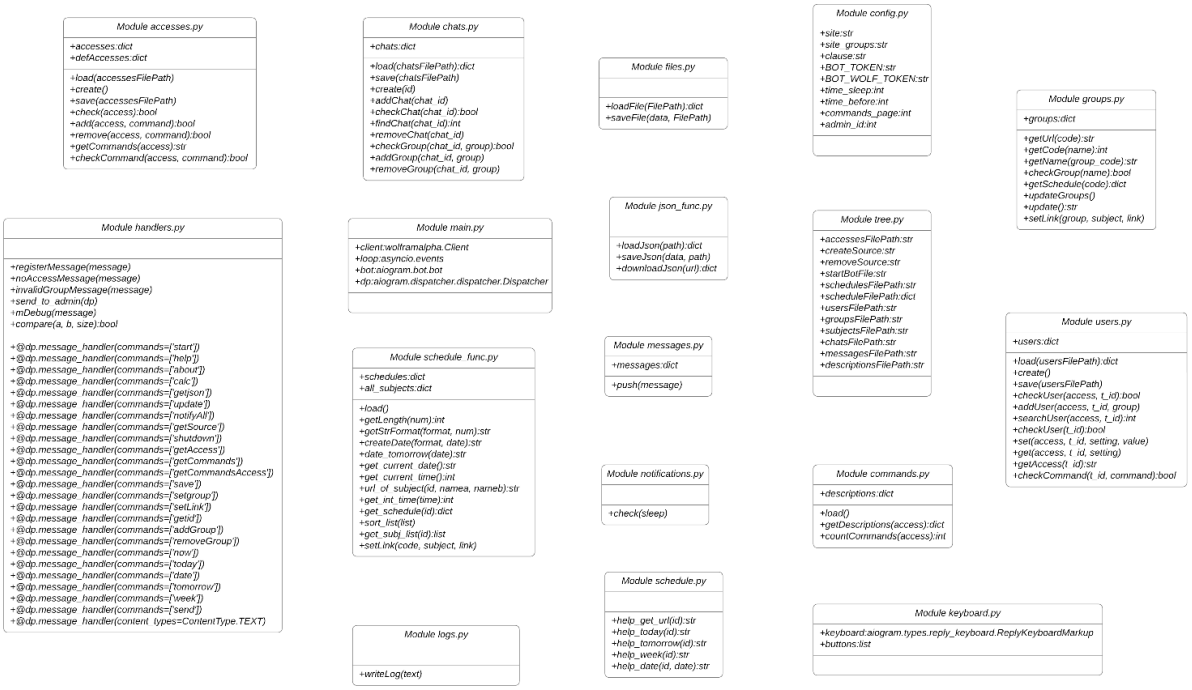

Рисунок 3.5 – Структурна діаграма модулів в UML анотації

Розглянемо основні модулі телеграм-бота (табл. 3.2).

Таблиця 3.2 – Опис модулів телеграм-бота

Назва модуля	Призначення
accesses.py	Відповідає за зберігання інформації про доступні команди, доступ до команд, рівні користувачів.
chats.py	Відповідає за зберігання інформації про чати в яких активовано телеграм-бот.
commands.py	Відповідає за зберігання опису команд, їхній функцій.
config.py	Налаштування телеграм-бота.
files.py	Відповідає за зберігання файлів, баз даних.
groups.py	Відповідає за зберігання, завантаження наявних груп в університеті. Завантаження розкладу за групою.
handlers.py	Це обробник подій, який

	активується, коли користувач надсилає команду боту.
json_func.py	Містить функції для роботи з json файлами.
keyboard.py	Містить налаштування віртуальної клавіатури для керування командами в телеграмі.
logs.py	Відповідає за запис та зберігання логів. Логування системних подій.
main.py	Є основним модулем. Він відповідає за ініціалізацію телеграм-бота, запуск обробника подій та основних систем телеграм-бота.
messages.py	Відповідає за зберігання та обробку повідомлень від користувачів до розбробника телеграм-бота.
notifications.py	Відповідає за надсилання автоматичних
schedule.py	Відповідає за обробку розкладу.
schedule_func.py	Містить функції для оброблення розкладу занять.
tree.py	Відповідає за зберігання інформації про шляхи до основих файлів та баз даних системи.
users.py	Відповідає за керування даними про користувачів.

Розглянемо програмний код обробника подій команди /today (рис. 3.6).

```python
@dp.message_handler(commands=['today'])
async def echohelp(message: Message):
    await mDebug(message)

    if users.checkUser(message.from_user.id) == False:
        await registerMessage(message)
        return 0

    if users.checkCommand(message.from_user.id, command: '/today') == False:
        await noAccessMessage(message)
        return 0

    group = users.get(users.getAccess(message.from_user.id), message.from_user.id, setting: 'group')

    if group == 'None':
        await invalidGroupMessage(message)
        return 0

    group = groups.getCode(str(group))

    result = schedule.help_today(str(group))
    await message.reply(text=result)
```

Рисунок 3.6 – Обробник команди /today – отримання розкладу занять на сьогодні

Програмний код працює за таким принципом: спочатку перевіряється, чи користувач зареєстрований в системі. Якщо ні, то бот надсилає повідомлення про необхідність зареєструватись.

Потім відбувається перевірка, чи користувач має доступ до команди. Після цього система отримує назву групи користувача. І лише після цього система формує розклад на сьогодні за групою.

В результаті система надсилає відповідь користувачеві у вигляді повідомлення.

3.3 Розгортання програмного забезпечення

Amazon Lightsail - це простий у використанні сервіс хмарного хостингу, який надається Amazon Web Services. Він призначений для розгортання та управління веб-додатками та веб-сайтами на віртуальних приватних серверах (VPS) [11].

Основні етапи розгортання:

1. Придбання сервера Amazon Lightsail: обрати план, що відповідає потребам, обрати образ операційної системи (Ubuntu 22.04) (рис. 3.7) [11].

2. Підключення: за допомогою SSH-клієнта відбувається підключення до сервера, щоб мати доступ до командного рядка (рис. 3.8) [11].

3. Розгортання: встановлення необхідних пакетів для роботи телеграм бота. Перенесення ПЗ на сервер та його запуск (рис. 3.9 та рис. 3.10) [11].

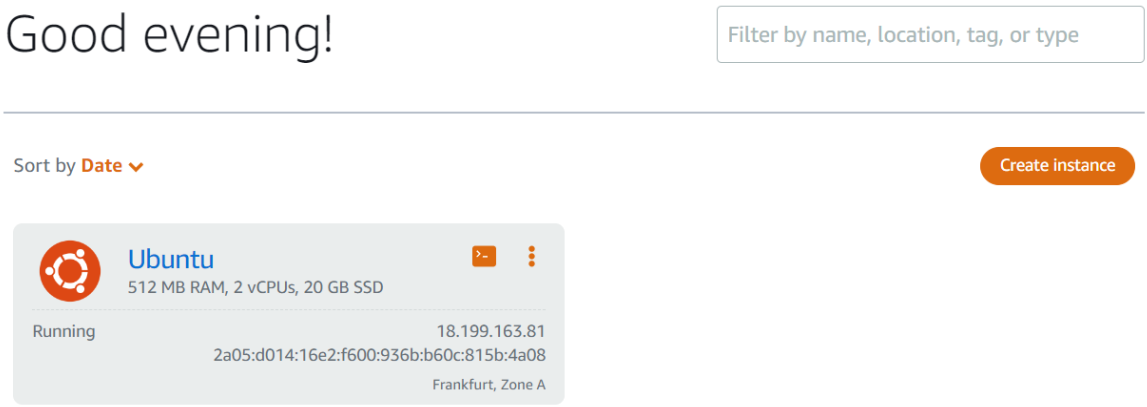

Рисунок 3.7 – Створення сервера Lightsail

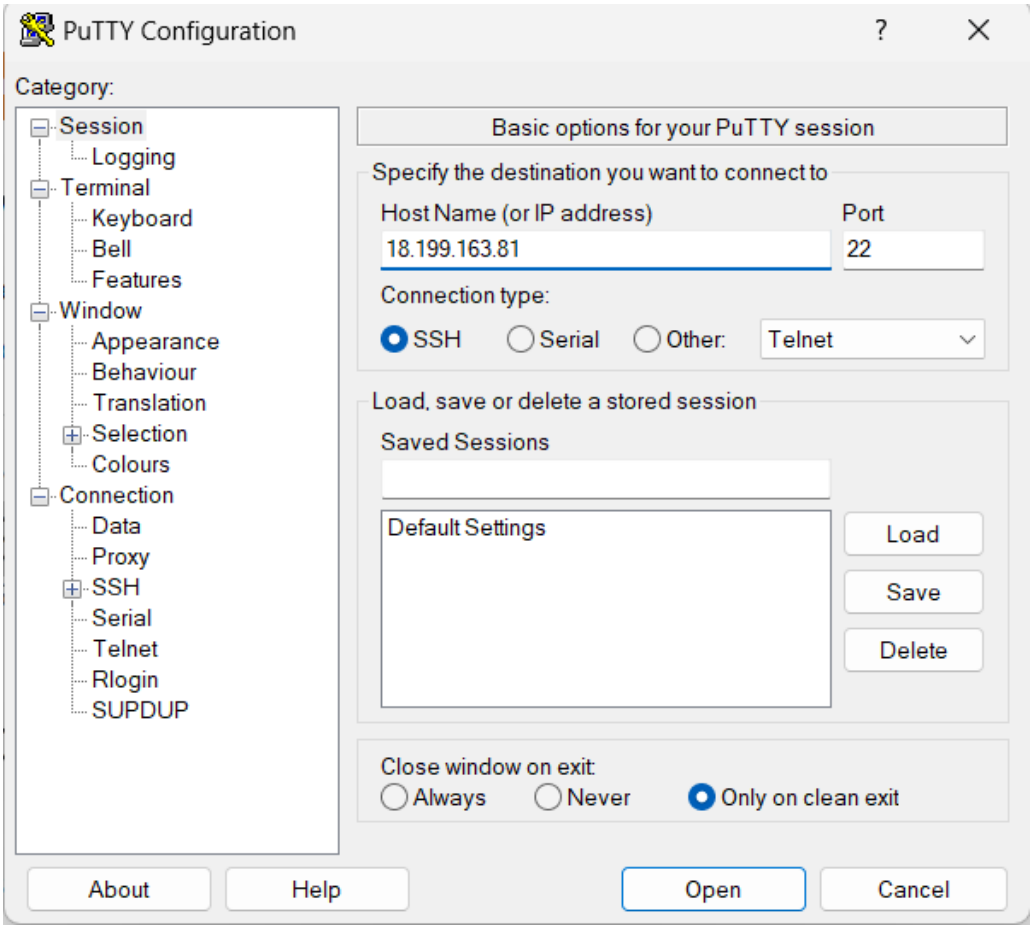

Рисунок 3.8 – Вхід на сервер через SSH клієнт

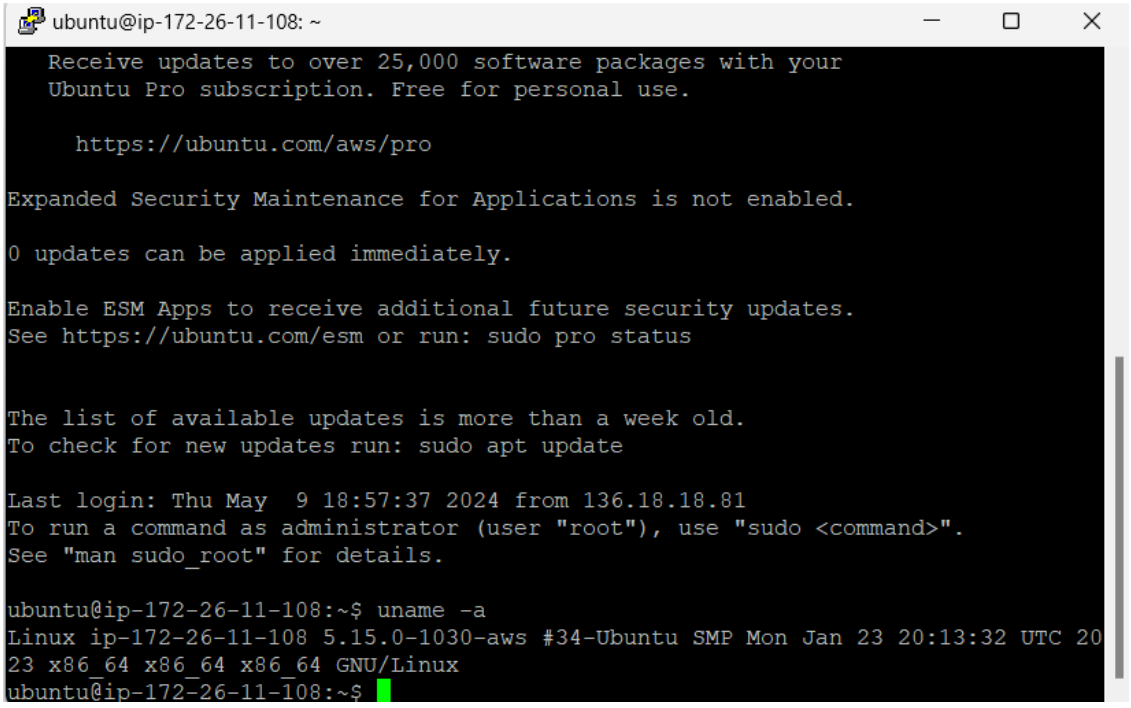

Рисунок 3.9 – Командний рядок сервера

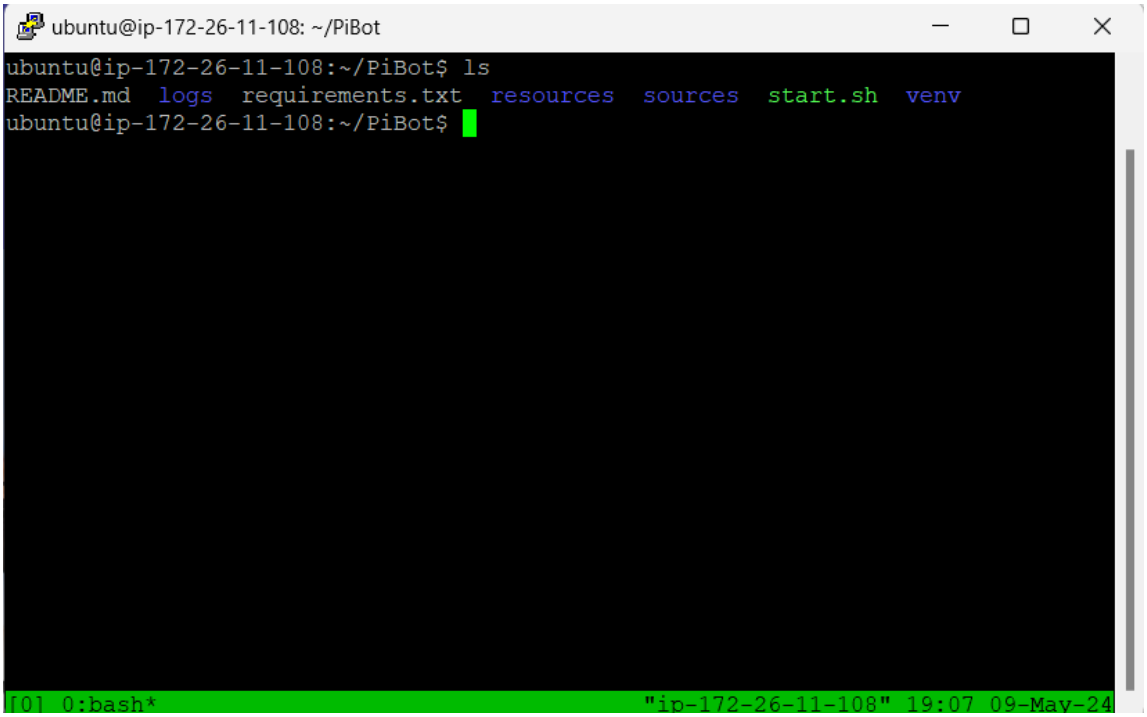

Рисунок 3.10 – Завантаження телеграм-бота на сервер

3.4 Тестування програмного продукту

В результаті система надсилає відповідь користувачеві у вигляді повідомлення.

Результати роботи розробленого телеграм-бота:

1. Початкова команда start та команда help, що відповідає за виведення

переліку команд користувачеві (рис. 3.11).

2. Встановлення групи користувачем (рис. 3.12).
3. Розклад занять групи на наступний день (рис. 3.13) та розклад занять групи на сьогодні (рис 3.14).
4. Найближче заняття для групи (рис 3.15).
5. Розклад занять групи на тиждень (рис 3.16).
6. Встановлення групи для автоматичних сповіщень про заняття в чаті (рис. 3.17).
7. Автоматичні сповіщення про найближчі заняття в університеті (рис. 3.18).
8. Математичні розрахунки в телеграм-боті за допомогою вбудованого сервісу WolframAlpha (рис. 3.19).

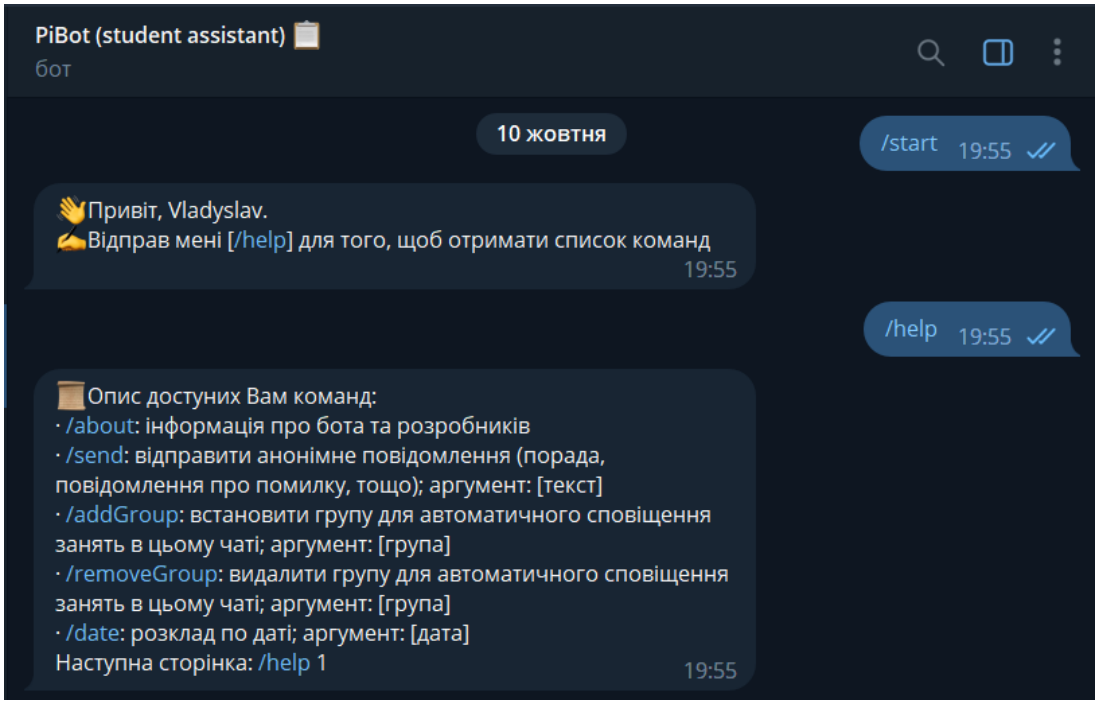

Рисунок 3.11 – Початок взаємодії з ботом PiBot в телеграмі. Демонстрацій команди help

Рисунок 3.12 – Демонстрація команди setgroup

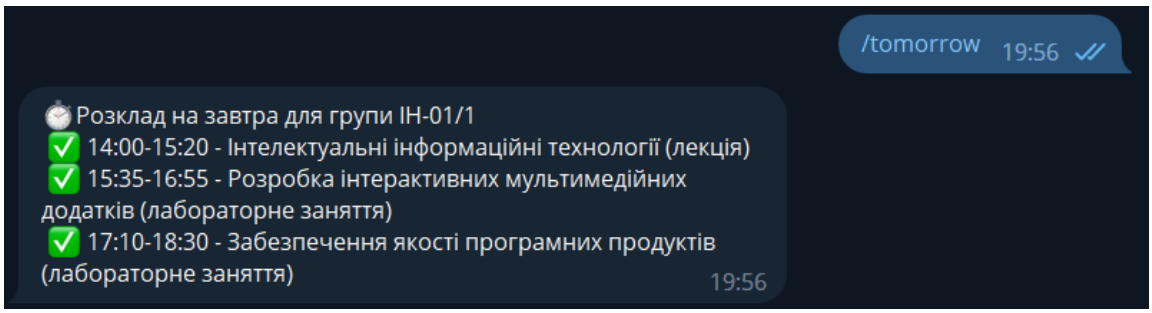

Рисунок 3.13 – Демонстрація команди tomorrow

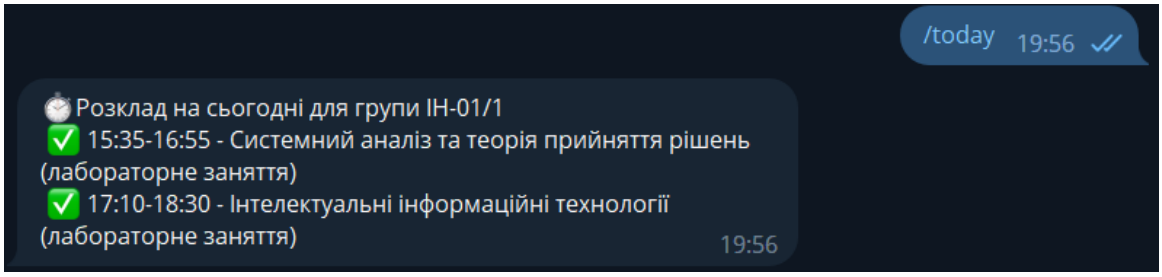

Рисунок 3.14 – Демонстрація команди today

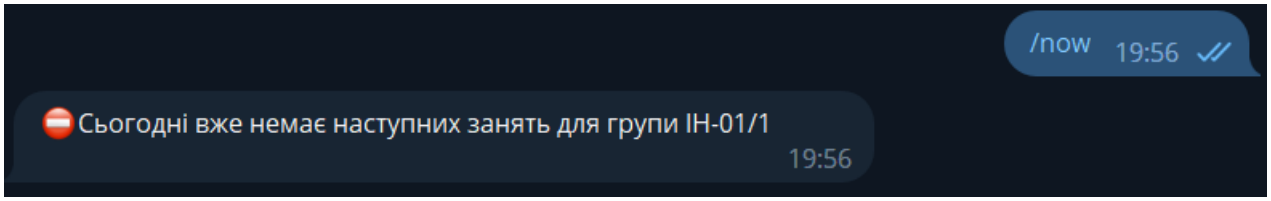

Рисунок 3.15 – Демонстрація команди now

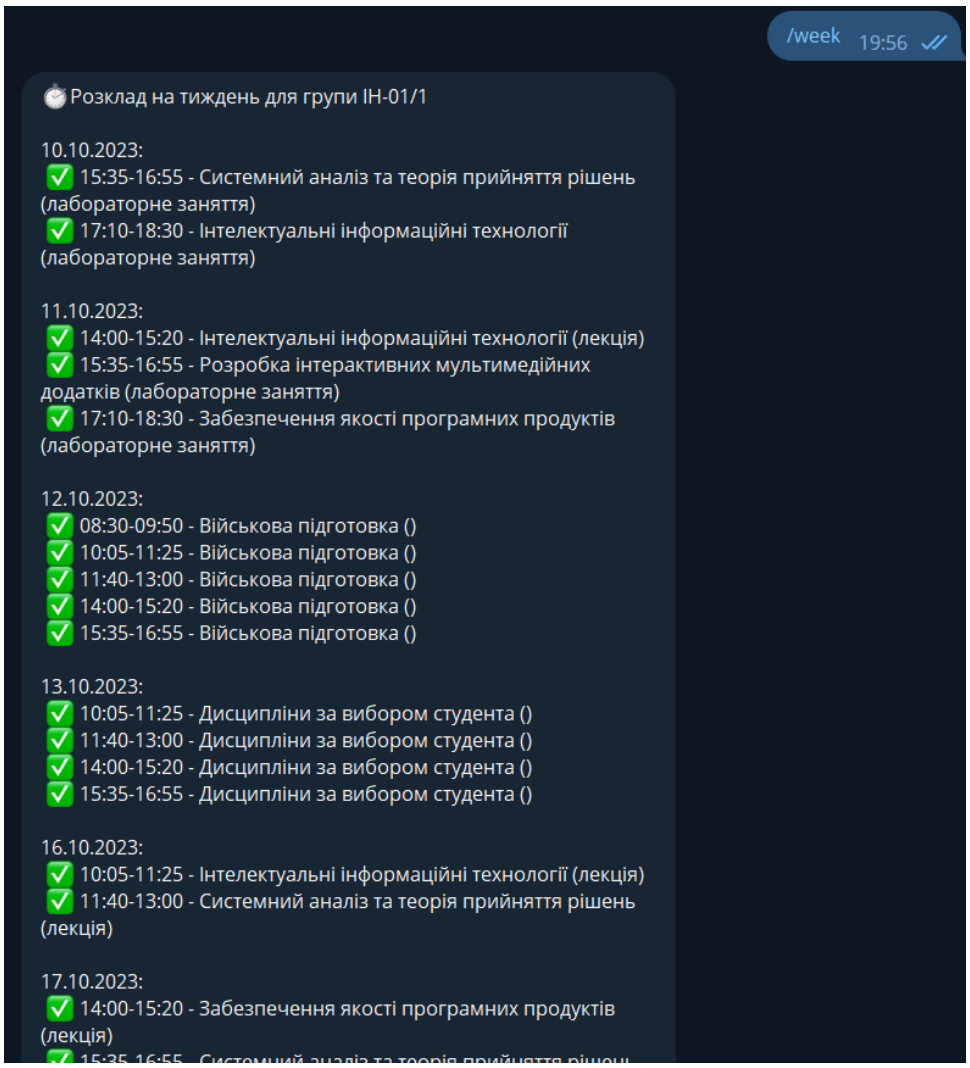

Рисунок 3.16 – Демонстрація команди week

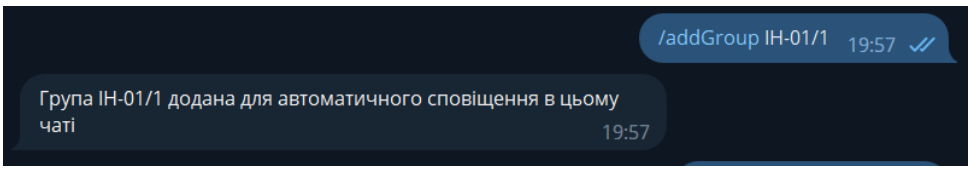

Рисунок 3.17 – Демонстрація команди addGroup

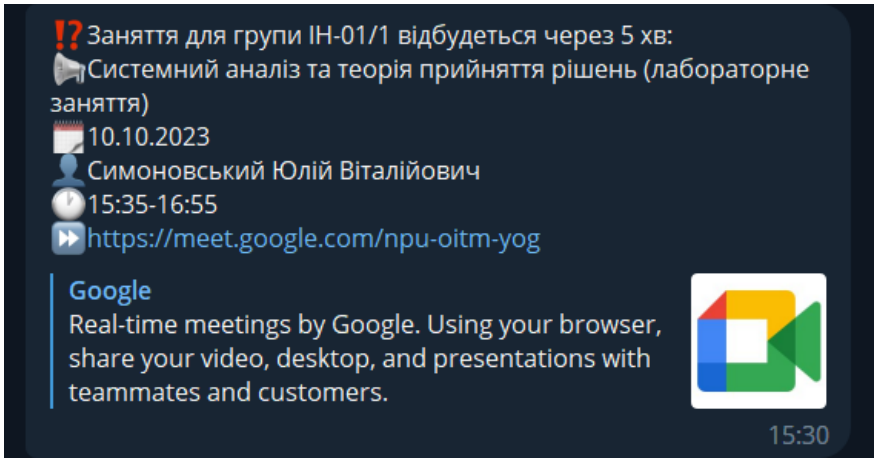

Рисунок 3.18 – Автоматичне сповіщення здобувачів освіти перед початком пари

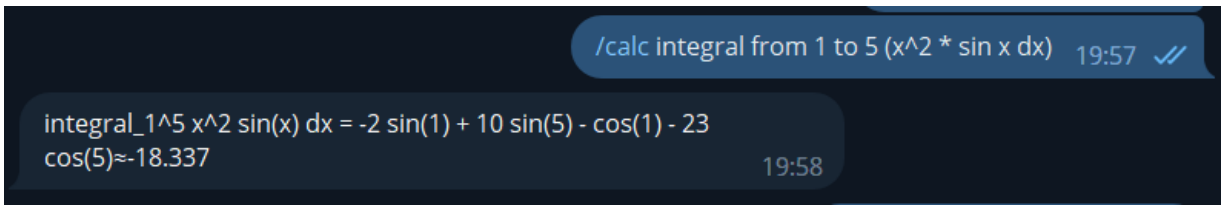

Рисунок 3.19 – Демонстрація математичних розрахунків із використанням wolfram alpha API

Отже, було успішно імлементовано функціонал бота який відповідає вимогам специфікації, що підтверджує успішне завершення етапу тестування.

4 ПЛАНУВАННЯ РОЗШИРЕННЯ ФУНКЦІОНАЛУ СИСТЕМИ

Майбутнє розширення функціоналу телеграм-бота спрямоване на забезпечення студентської спільноти додатковими зручностями:

- інтеграція нових функцій: управління особистим часом здобувачів освіти та системи сповіщень для викладачів дозволить краще організувати навчальний процес і покращити комунікацію між здобувачами освіти та викладачами;
- надсилання домашніх завдань через бота: додавання можливості здобувачам освіти надсилати та отримувати нагадування про терміни здачі домашніх завдань спростить процес організації навчальної діяльності;
- відгуки та оцінювання: здобувачам освіти буде надано можливість залишати відгуки та оцінювати заняття, викладачів та курси, що сприятиме покращенню якості навчання та забезпечить зворотний зв'язок з аудиторією;

Встановлений план добре відображає стратегічні цілі щодо подальшого розвитку бота та визначає напрямки його подальшої еволюції для забезпечення кращого досвіду користування ботом здобувачів освіти та викладачів.

ВИСНОВКИ

У ході дослідження за темою "Інформаційна система планування і організації навчальної діяльності" була розроблена та впроваджена інноваційна інформаційна система у вигляді телеграм-бота. Ця система призначена для полегшення навчального процесу здобувачів освіти у онлайн-середовищі та надання їм зручного інструменту для отримання інформації про розклад занять, посилання на навчальні матеріали та виконання математичних обчислень.

У роботі були використані сучасні технології та інструменти для розробки телеграм-бота. Зокрема, фреймворк aiogram дозволив створити бота з великим функціоналом і зручним інтерфейсом для користувачів. Мова програмування Python обрана за її простоту та ефективність у розробці.

Для забезпечення асинхронності та паралельності виконання завдань був використаний фреймворк asyncio, що дозволило боту ефективно взаємодіяти з багатьма користувачами одночасно. Фреймворк wolframalpha для Python був використаний для виконання математичних обчислень прямо в телеграмі, що робить бота корисним інструментом для здобувачів освіти під час навчання.

Розробка та впровадження телеграм-бота для полегшення навчального процесу здобувачів освіти є актуальною та перспективною ініціативою. Цей проєкт створює основу для подальшого розширення функціоналу та вдосконалення інструментів планування та організації онлайн-навчання.

Реалізовані завдання та досягнення мети роботи:

- аналіз потреб здобувачів освіти: проведено аналіз потреб та вимог здобувачів освіти до функціоналу телеграм-бота;
- розробка архітектури бота: розроблено архітектуру телеграм-бота, визначено його основні функції та можливості;
- розробка інтерфейсу: створено зручний та інтуїтивно зрозумілий інтерфейс для взаємодії користувачів з ботом;

- інтеграція з навчальними системами: забезпечено інтеграцію телеграм-бота з навчальними платформами та системами університету для отримання актуальної інформації про розклад занять;
- розробка функціоналу для навчання: розроблено функціонал для перегляду лекцій, завдань, тестів та інших навчальних матеріалів;
- розробка системи нагадувань: впроваджено систему нагадувань для здобувачів освіти про важливі події та дедлайни;
- тестування та вдосконалення: проведено тестування розробленого бота, зібрано відгуки користувачів та внесено необхідні виправлення та покращення.

Отже, мета роботи — розробка та впровадження інформаційної системи у вигляді телеграм-бота, спрямованого на полегшення навчальної діяльності здобувачів освіти була успішно досягнута. Створений бот забезпечує здобувачам освіти зручний доступ до навчальних ресурсів, допомагає в організації навчального процесу та сприяє підвищенню якості навчання в онлайн-середовищі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Любін І.М. Телеграм-бот для взаємодії викладачів та студентів. Київ, 2019. С. 7–16.

2. Чайка Б.В. Telegram-бот для забезпечення якісного навчання студентів. Сумський державний університет, 2022. С. 12–21.

3. Комарський О.С. Телеграм-бот для інформаційної підтримки навчального процесу. Київ, 2020. С. 7–15.

4. Мигасюк Р., Смотр О.О. РОЗРОБКА TELEGRAM БОТУ ДЛЯ АВТОМАТИЗАЦІЇ ПРОЦЕСУ ОТРИМАННЯ РОЗКЛАДУ В НАВЧАЛЬНОМУ ЗАКЛАДІ. ЛДУ БЖД, 2021. С. 3–5.

5. Матусевич М.М. Інформаційна система онлайн допомоги студенту. Сумський державний університет, 2020. С. 10–15.

6. Telegram-боти на Python: огляд п’яти найкращих фреймворків/бібліотек | Друкарня [Electronic resource]. URL: https://drukarnia.com.ua/articles/telegram-boti-na-python-oglyad-p-yati-naikrashikh-freimvorkiv-bibliotek-L7UA7 (дата звернення: 24.05.2024).

7. The Python Tutorial — Python 3.11.4 documentation [Electronic resource]. URL: https://docs.python.org/3.11/tutorial/index.html (: 24.05.2024).

8. Telegram Bot API [Electronic resource]. URL: https://core.telegram.org/bots/api (дата звернення: 24.05.2024).

9. Мельник Р.К. Telegram-бот для організації навчального часу студента. Київ, 2022. С. 23–39.

10. Welcome to aiogram’s documentation! — aiogram 2.25.1 documentation [Electronic resource]. URL: https://docs.aiogram.dev/en/latest/ (дата звернення: 24.05.2024).

11. AWS. Launch a Linux Virtual Machine with Amazon EC2 [Electronic resource] // Aws.Amazon.Com. 2019. URL: https://aws.amazon.com/getting-started/tutorials/launch-a-virtual-machine/ (дата звернення: 24.05.2024).

12. Розділ 10. Модульне програмування (2017 р.) [Electronic resource].

URL: https://victana.lviv.ua/knyhy/konspekty-lektsii/138-alhorytmizatsiia-ta-prohramuvannia-chastyna-1/665-rozdil-10-vydy-prohramuvannia-2017-r (дата звернення: 24.05.2024).

13. Основи Pycharm. Уроки для початківців. W3Schools українською [Electronic resource]. URL: https://w3schoolsua.github.io/hyperskill/pycharm-basics.html#gsc.tab=0 (дата звернення: 24.05.2024).

14. JSON [Electronic resource]. URL: https://www.json.org/json-uk.html (дата звернення: 24.05.2024).

15. Безрук В. М., Шовкопляс О. А. Телеграм-бот для планування та організації навчальної діяльності студента // Інформатика, математика, автоматика (IMA – 2024) : матеріали та програма міжнародної науково-технічної конференції, м. Суми, 22–26 квітня 2024 р. Суми : СумДУ, 2024. С. 61.

ДОДАТОК А ЛІСТИНГ ПРОГРАМНОГО КОДУ

Модуль accesses.py

```
import sources.py.files as files

from sources.py.tree import *

accesses = {}
defAccesses = {}


def load(accessesFilePath):
    global accesses

    accesses = files.loadFile(accessesFilePath)


def create():
    global accesses

    accesses = defAccesses


def save(accessesFilePath):
    files.saveFile(accesses, accessesFilePath)


def check(access):
    if accesses.get(access) == None:
        return False
    return True


def add(access, command):
    if check(access) and not accesses[access].count(command):
        accesses[access].append(command)
        return True
    return False


def remove(access, command):
    if check(access) and accesses[access].count(command) > 0:
        accesses[access].remove(command)
        return True
    else:
        return False


def getCommands(access):
    if check(access):
        result = f'У вас рівень {access}. Ви маєте такі команди:'

        for command in accesses[access]:
            result += f'\n\t{command}'

    else:
        result = f'Немає команд для рівня {access}\n'

    return result


def checkCommand(access, command):
```

```
    if check(access) and accesses[access].count(command) > 0:
        return True
    else:
        return False


load(accessesFilePath)
```

Модуль chats.py

```
import sources.py.files as files
import sources.py.groups as groups
import sources.py.logs as logs
from sources.py.tree import *

chats = {}


def load(chatsFilePath):
    global chats

    chats = files.loadFile(chatsFilePath)

    return chats


def save(chatsFilePath):
    files.saveFile(chats, chatsFilePath)


def create(id):
    chat = {}

    chat[id] = []


def addChat(chat_id):
    logs.writeLog(f'Aded new chat with id {chat_id}')

    chats[chat_id] = []

    save(chatsFilePath)


def checkChat(chat_id):
    for key in chats.keys():
        if str(chat_id) == str(key):
            return True
    return False


def findChat(chat_id):
    i = 0

    for key in chats.keys():
        if str(chat_id) == str(key):
            return i
        i = i + 1
    return -1


def removeChat(chat_id):
    i = findChat(chat_id)
```

```
    if i != -1:
        chats.pop(i)

    save(chatsFilePath)


def checkGroup(chat_id, group):
    if checkChat(chat_id) == True:
        if group in chats[chat_id]:
            return True
    return False


def addGroup(chat_id, group):
    if checkChat(chat_id) == False:
        addChat(chat_id)

    if checkGroup(chat_id, group) == False:
        chats[chat_id].append(group)

        groups.getSchedule(groups.getCode(group))

    save(chatsFilePath)


def removeGroup(chat_id, group):
    if checkChat(chat_id) == False:
        addChat(chat_id)

    if checkGroup(chat_id, group) == True:
        chats[chat_id].remove(group)

    save(chatsFilePath)


load(chatsFilePath)
```

Модуль commands.py

```
import sources.py.accesses as accesses
import sources.py.files as files
from sources.py.config import *
from sources.py.tree import *

descriptions = {}


def load():
    global descriptions

    descriptions = files.loadFile(descriptionsFilePath)


def getDescriptions(access):
    answer = {}

    i = 0

    for command in accesses.accesses[access]:
        answer[i] = clause + f' {command}: {descriptions[command]}\n'

        i = i + 1

    return answer
```

```
def countCommands(access):
    return len(accesses.accesses[access])
```

Модуль config.py

```
site =
"https://sh.cabinet.sumdu.edu.ua/index/json?method=getSchedules&id_grp="
site_groups = "https://sh.cabinet.sumdu.edu.ua/index/json?method=getGroups"

clause = '·'

BOT_TOKEN = "YOUR_BOT_TOKEN"
BOT_WOLF_TOKEN = "YOUR_WOLFRAM_ALPHA_TOKEN"

time_sleep = 60

time_before = 5

commands_page = 5

admin_id = YOUR_ADMIN_ID
```

Модуль files.py

```
import codecs
import json


def loadFile(FilePath):
    with codecs.open(FilePath, encoding='utf-8') as file:
        data = json.loads(file.read())

    return data


def saveFile(data, FilePath):
    with codecs.open(FilePath, "w", encoding='utf-8') as file:
        json.dump(data, file)
```

Модуль groups.py

```
import sources.py.files as files
import sources.py.json_func as json_func
import sources.py.schedule_func as schedule_func
from sources.py.config import *
from sources.py.tree import *

groups = {}


def getUrl(code):
    return site + code


def getCode(name):
    for code in groups.keys():
        if groups[code].lower() == name.lower():
            return code
    return -1


def getName(group_code):
    for code in groups.keys():
```

```
        if code == group_code:
            return groups[code]


def checkGroup(name):
    for code in groups.keys():
        if groups[code].lower() == name.lower():
            return True
    return False


def getSchedule(code):
    url = getUrl(code)

    try:
        schedule = json_func.downloadJson(url)

        filename = 'schedule-' + code + '.json'

        path = schedulesFilePath

        path += '/' + filename

        files.saveFile(schedule, path)

        return schedule

    except:
        print('Error while downloading schedule ' + code)


def updateGroups():
    try:
        global groups

        path = groupsFilePath

        local_groups = json_func.downloadJson(site_groups)

        formated_groups = {}

        for i in local_groups:
            formated_groups[i['KOD_GROUP']] = i['NAME_GROUP']

        files.saveFile(formated_groups, path)

        groups = files.loadFile(groupsFilePath)

        print('Оновлено групи')
    except:
        print('Не було оновлено групи')


def update():
    updateGroups()

    flag = 0
    result = 'Оновлено розклад для: '

    for code in groups.keys():
        group = groups[code]
        filename = 'schedule-' + code + '.json'
        path = schedulesFilePath + '/' + filename
```

```
        if os.path.exists(path) == True:
            flag = 1
            getSchedule(code)

            result += f'{group} '
            print(f'Updated {code}')
    if flag:
        return result
    else:
        return 'Не було оновлено розклад'


def setLink(group, subject, link):
    groupCode = getCode(group)

    schedule_func.setLink(groupCode, subject, link)


updateGroups()
```

Модуль handlers.py

```
import datetime
import subprocess
import sys
import time
from math import ceil

from aiogram.types import *

import sources.py.accesses as accesses
import sources.py.chats as chats
import sources.py.commands as commands
import sources.py.keyboard as keyboard
import sources.py.logs as logs
import sources.py.messages as messages
import sources.py.schedule_func as schedule_func
import sources.py.users as users
from main import bot, dp, client
from sources.py.config import *
from sources.py.schedule import *


async def registerMessage(message: Message):
    await message.answer(text=f"🆕{message.from_user.first_name}, Ви новий
користувач, ✍ напишіть команду /start")


async def noAccessMessage(message: Message):
    await message.answer(text=f"⛔{message.from_user.first_name}, Ви не маєте
доступу до цієї команди")


async def invalidGroupMessage(message: Message):
    await message.answer(
        text=f"🔑{message.from_user.first_name}, у Вас не встановлена група,
✍ напишіть команду /setgroup (ваша група)")


async def send_to_admin(dp):
    logs.writeLog('Bot started')

    await bot.send_message(chat_id=admin_id, text="Bot started!")
```

```
async def mDebug(message: Message):
    log_text = f"message = {message.text} username =
@{message.from_user.username} name = {message.from_user.first_name} chat_id =
{message.chat.id} user_id = {message.from_user.id}"

    logs.writeLog(log_text)

    await bot.send_message(chat_id=admin_id,
                           text=f"Debug[{message.date}]:\n \tmessage =
{message.text}\n \tusername = @{message.from_user.username}\n \tname =
{message.from_user.first_name}")


@dp.message_handler(commands=['start'])
async def echo(message: Message):
    await mDebug(message)

    if users.checkUser(message.from_user.id) == False:
        users.addUser('user', message.from_user.id, 'None')

    await message.answer(
        text=f"👋Привіт, {message.from_user.first_name}.\n✍Відправ мені
[/help] для того, щоб отримати список команд")


@dp.message_handler(commands=['help'])
async def echohelp(message: Message):
    await mDebug(message)

    if users.checkUser(message.from_user.id) == False:
        await registerMessage(message)
        return 0

    if users.checkCommand(message.from_user.id, '/help') == False:
        await noAccessMessage(message)
        return 0

    if len(message.text) > len("/help "):
        page = int(message.text.replace("/help ", ""))
    else:
        page = 0

    access = users.getAccess(message.from_user.id)

    descriptions = commands.getDescriptions(access)

    count = commands.countCommands(access)

    max_page = ceil(float(count) / commands_page)

    if page + 1 > max_page:
        await message.answer(text='Ви не можете це зробити!')
        return

    i = page * commands_page

    answer = '📜Опис доступних Вам команд:\n'

    while i < (page + 1) * commands_page and i < count:
        answer += descriptions[i]
        i = i + 1
```

```
    if (page > 0):
        if page - 1 != 0:
            answer += f'Попередня сторінка: /help {page - 1}\n'
        else:
            answer += 'Попередня сторінка: /help\n'
    if not page + 2 > max_page:
        answer += f'Наступна сторінка: /help {page + 1}\n'

    await message.reply(text=answer)


@dp.message_handler(commands=['about'])
async def echohelp(message: Message):
    await mDebug(message)

    if users.checkUser(message.from_user.id) == False:
        await registerMessage(message)
        return 0

    if users.checkCommand(message.from_user.id, '/about') == False:
        await noAccessMessage(message)
        return 0

    await message.reply(
        text='Цей бот був створений студентом спеціальності інформатика
Сумського державного університету. ' +
             'Бот допомагає студентам СумДУ отримувати розклад занять у
телеграм-чаті. ' +
             'Бот може надсилати автоматичні посилання на заняття до їх
початку. ' +
             'Він також вміє розв’язувати математичні приклади і знає
відповіді на деякі запитання. ' +
             'Надалі функціонал буде розширюватися.\n' +
             'Автор: @vladyslavbezruk\n'
             'GitHub: https://github.com/vladyslavbezruk'
    )


@dp.message_handler(commands=['calc'])
async def echohelp(message: Message):
    await mDebug(message)

    if users.checkUser(message.from_user.id) == False:
        await registerMessage(message)
        return 0

    if users.checkCommand(message.from_user.id, '/calc') == False:
        await noAccessMessage(message)
        return 0

    query = message.text.replace("/calc ", "")
    res = client.query(query)
    output = next(res.results).text

    await message.reply(text=output)


@dp.message_handler(commands=['getjson'])
async def echohelp(message: Message):
    await mDebug(message)

    if users.checkUser(message.from_user.id) == False:
        await registerMessage(message)
```

```
        return 0

    if users.checkCommand(message.from_user.id, '/getjson') == False:
        await noAccessMessage(message)
        return 0

    for schedule in scheduleFilePath:
        logs.writeLog('Sended jsons')

        await bot.send_document(chat_id=message.chat.id,
document=open(scheduleFilePath[schedule], 'rb'))

        time.sleep(1)


@dp.message_handler(commands=['update'])
async def echohelp(message: Message):
    await mDebug(message)

    if users.checkUser(message.from_user.id) == False:
        await registerMessage(message)
        return 0

    if users.checkCommand(message.from_user.id, '/update') == False:
        await noAccessMessage(message)
        return 0

    result = groups.update()

    logs.writeLog(result)

    schedule_func.load()

    await message.reply(text=result)


@dp.message_handler(commands=['notifyAll'])
async def echohelp(message: Message):
    await mDebug(message)

    if users.checkUser(message.from_user.id) == False:
        await registerMessage(message)
        return 0

    if users.checkCommand(message.from_user.id, '/notifyAll') == False:
        await noAccessMessage(message)
        return 0

    admin_message = message.text.replace("/notifyAll ", "")

    all_chats = [key for key in chats.chats.keys()]

    for access in users.users.keys():
        for user in users.users[access]:
            all_chats.append(user['id'])

    all_chats_uniq = list(set(all_chats))

    await message.reply(text='Запит надіслано!')

    for chat_id in all_chats_uniq:
        try:
            await bot.send_message(chat_id=chat_id, text=admin_message)
```

```
        except:
            print("Не вдалося відправити повідомлення в чат " + str(chat_id))


@dp.message_handler(commands=['getSource'])
async def echohelp(message: Message):
    await mDebug(message)

    if users.checkUser(message.from_user.id) == False:
        await registerMessage(message)
        return 0

    if users.checkCommand(message.from_user.id, '/getSource') == False:
        await noAccessMessage(message)
        return 0

    logs.writeLog('Sended sources')

    subprocess.call([f'./{createSource}'])
    await bot.send_document(chat_id=message.chat.id,
document=open('../../PiBot.zip', 'rb'))
    subprocess.call([f'./{removeSource}'])


def compare(a, b, size):
    for i in range(size):
        if a[i] != b[i]:
            return False

    return True


@dp.message_handler(commands=['shutdown'])
async def echohelp(message: Message):
    await mDebug(message)

    time = datetime.now()

    if users.checkUser(message.from_user.id) == False:
        await registerMessage(message)
        return 0

    if users.checkCommand(message.from_user.id, '/shutdown') == False:
        await noAccessMessage(message)
        return 0

    if compare(str(time), str(message.date), 18):
        await message.answer(text="Goodbye ...")

        accesses.save(accesses.accessesFilePath)
        users.save(users.usersFilePath)

        logs.writeLog('Shutdown ...')

        sys.exit(0)
        return


@dp.message_handler(commands=['getAccess'])
async def echohelp(message: Message):
    await mDebug(message)

    if users.checkUser(message.from_user.id) == False:
        await registerMessage(message)
```

```
        return 0

    if users.checkCommand(message.from_user.id, '/getAccess') == False:
        await noAccessMessage(message)
        return 0

    access = users.getAccess(message.from_user.id)
    await message.reply(text=f"Your access level - {access}")


@dp.message_handler(commands=['getCommands'])
async def echohelp(message: Message):
    await mDebug(message)

    if users.checkUser(message.from_user.id) == False:
        await registerMessage(message)
        return 0

    if users.checkCommand(message.from_user.id, '/getCommands') == False:
        await noAccessMessage(message)
        return 0

    commands = accesses.getCommands(users.getAccess(message.from_user.id))

    await message.reply(text=commands)


@dp.message_handler(commands=['getCommandsAccess'])
async def echohelp(message: Message):
    await mDebug(message)

    if users.checkUser(message.from_user.id) == False:
        await registerMessage(message)
        return 0

    if users.checkCommand(message.from_user.id, '/getCommandsAccess') ==
False:
        await noAccessMessage(message)
        return 0

    commands = accesses.getCommands(message.text.replace("/getCommandsAccess
", ""))

    await message.reply(text=commands)


@dp.message_handler(commands=['save'])
async def echohelp(message: Message):
    await mDebug(message)

    if users.checkUser(message.from_user.id) == False:
        await registerMessage(message)
        return 0

    if users.checkCommand(message.from_user.id, '/save') == False:
        await noAccessMessage(message)
        return 0

    logs.writeLog('Saving data ...')

    accesses.save(accesses.accessesFilePath)
    users.save(users.usersFilePath)

    await message.answer(text="All saved!")
```

```
@dp.message_handler(commands=['setgroup'])
async def echohelp(message: Message):
    await mDebug(message)

    if users.checkUser(message.from_user.id) == False:
        await registerMessage(message)
        return 0

    if users.checkCommand(message.from_user.id, '/setgroup') == False:
        await noAccessMessage(message)
        return 0

    group = message.text.replace("/setgroup ", "")

    if (not groups.checkGroup(group)):
        await message.answer(text=f"⁉{message.from_user.first_name}, немає
такої групи")
        return 0

    users.set(users.getAccess(message.from_user.id), message.from_user.id,
'group', str(group))

    await message.reply(text=f"‼Зараз Ваша група - {group}")


@dp.message_handler(commands=['setLink'])
async def echohelp(message: Message):
    await mDebug(message)

    if users.checkUser(message.from_user.id) == False:
        await registerMessage(message)
        return 0

    if users.checkCommand(message.from_user.id, '/setLink') == False:
        await noAccessMessage(message)
        return 0

    args = message.text.replace("/setLink ", "").split(", ")
    group = args[0]
    subject = args[1]
    link = args[2]

    if (not groups.checkGroup(group)):
        await message.reply(text=f"⁉{message.from_user.first_name}, немає
такої групи")
        return 0

    groups.setLink(group, subject, link)

    await message.reply(text=f"‼Посилання встановлено")


@dp.message_handler(commands=['getid'])
async def echohelp(message: Message):
    await mDebug(message)

    if users.checkUser(message.from_user.id) == False:
        await registerMessage(message)
        return 0

    if users.checkCommand(message.from_user.id, '/getid') == False:
```

```
        await noAccessMessage(message)
        return 0

    await message.answer(text=f"Your id - {str(message.from_user.id)}")


@dp.message_handler(commands=['addGroup'])
async def echohelp(message: Message):
    await mDebug(message)

    if users.checkUser(message.from_user.id) == False:
        await registerMessage(message)
        return 0

    if users.checkCommand(message.from_user.id, '/addGroup') == False:
        await noAccessMessage(message)
        return 0

    group = str(message.text.replace("/addGroup ", ""))
    chat_id = str(message.chat.id)

    if (not groups.checkGroup(group)):
        await message.reply(text=f"Немає такої групи як {group}")
        return
    if chats.checkGroup(chat_id, group) == False:
        chats.addGroup(chat_id, group)
        await message.reply(text=f"Група {group} додана для автоматичного
сповіщення в цьому чаті")
    else:
        await message.reply(text=f"Група {group} вже була додана для
автоматичного сповіщення в цьому чаті")


@dp.message_handler(commands=['removeGroup'])
async def echohelp(message: Message):
    await mDebug(message)

    if users.checkUser(message.from_user.id) == False:
        await registerMessage(message)
        return 0

    if users.checkCommand(message.from_user.id, '/removeGroup') == False:
        await noAccessMessage(message)
        return 0

    group = str(message.text.replace("/removeGroup ", ""))
    chat_id = str(message.chat.id)

    if chats.checkGroup(chat_id, group) == True:
        chats.removeGroup(chat_id, group)
        await message.reply(text=f"Група {group} видалена для автоматичного
сповіщення в цьому чаті")
    else:
        await message.reply(text=f"Група {group} вже була видалена для
автоматичного сповіщення в цьому чаті")


@dp.message_handler(commands=['now'])
async def echohelp(message: Message):
    await mDebug(message)

    if users.checkUser(message.from_user.id) == False:
        await registerMessage(message)
        return 0
```

```
    if users.checkCommand(message.from_user.id, '/now') == False:
        await noAccessMessage(message)
        return 0

    group = users.get(users.getAccess(message.from_user.id),
message.from_user.id, 'group')

    if group == 'None':
        await invalidGroupMessage(message)
        return 0

    group = groups.getCode(str(group))

    result = help_get_url(str(group))

    await message.reply(result)


@dp.message_handler(commands=['today'])
async def echohelp(message: Message):
    await mDebug(message)

    if users.checkUser(message.from_user.id) == False:
        await registerMessage(message)
        return 0

    if users.checkCommand(message.from_user.id, '/today') == False:
        await noAccessMessage(message)
        return 0

    group = users.get(users.getAccess(message.from_user.id),
message.from_user.id, 'group')

    if group == 'None':
        await invalidGroupMessage(message)
        return 0

    group = groups.getCode(str(group))

    result = help_today(str(group))
    await message.reply(text=result)


@dp.message_handler(commands=['date'])
async def echohelp(message: Message):
    await mDebug(message)

    if users.checkUser(message.from_user.id) == False:
        await registerMessage(message)
        return 0

    if users.checkCommand(message.from_user.id, '/date') == False:
        await noAccessMessage(message)
        return 0

    group = users.get(users.getAccess(message.from_user.id),
message.from_user.id, 'group')

    if group == 'None':
        await invalidGroupMessage(message)
        return 0

    group = groups.getCode(str(group))
```

```
    result = help_date(str(group), message.text.replace("/date ", ""))
    await message.reply(text=result)


@dp.message_handler(commands=['tomorrow'])
async def echohelp(message: Message):
    await mDebug(message)

    if users.checkUser(message.from_user.id) == False:
        await registerMessage(message)
        return 0

    if users.checkCommand(message.from_user.id, '/tomorrow') == False:
        await noAccessMessage(message)
        return 0

    group = users.get(users.getAccess(message.from_user.id),
message.from_user.id, 'group')

    if group == 'None':
        await invalidGroupMessage(message)
        return 0

    group = groups.getCode(str(group))

    result = help_tomorrow(str(group))
    await message.reply(text=result)


@dp.message_handler(commands=['week'])
async def echohelp(message: Message):
    await mDebug(message)

    if users.checkUser(message.from_user.id) == False:
        await registerMessage(message)
        return 0

    if users.checkCommand(message.from_user.id, '/week') == False:
        await noAccessMessage(message)
        return 0

    group = users.get(users.getAccess(message.from_user.id),
message.from_user.id, 'group')

    if group == 'None':
        await invalidGroupMessage(message)
        return 0

    group = groups.getCode(str(group))

    result = help_week(str(group))
    await message.reply(text=result)


@dp.message_handler(commands=['send'])
async def echohelp(message: Message):
    await mDebug(message)

    if users.checkUser(message.from_user.id) == False:
        await registerMessage(message)
        return 0

    if users.checkCommand(message.from_user.id, '/send') == False:
```

```
        await noAccessMessage(message)
        return 0

    messages.push(message)

    text_message = message.text.replace("/send ", "")

    text = f"⁉Message:\nusername = @{message.from_user.username} name =
{message.from_user.first_name} chat_id = {message.chat.id} user_id =
{message.from_user.id}\nMessage: {text_message}"
    await bot.send_message(chat_id=admin_id, text=text)

    await message.reply(text='Анонімне повідомлення відправлено!')


@dp.message_handler(content_types=ContentType.TEXT)
async def echoMessage(message: Message):
    await mDebug(message)

    if chats.checkChat(message.chat.id) == False:
        chats.addChat(message.chat.id)

        await bot.send_message(chat_id=message.chat.id, text="Кнопки:",
reply_markup=keyboard.keyboard)
```

Модуль json_func.py

```
import requests

import sources.py.files as files

requests.packages.urllib3.disable_warnings()
requests.packages.urllib3.util.ssl_.DEFAULT_CIPHERS += ':HIGH:!DH:!aNULL'
try:
    requests.packages.urllib3.contrib.pyopenssl.util.ssl_.DEFAULT_CIPHERS +=
':HIGH:!DH:!aNULL'
except AttributeError:
    pass


def loadJson(path):
    data = files.loadFile(path)

    return data


def saveJson(data, path):
    files.saveFile(data, path)


def downloadJson(url):
    data = requests.get(url, verify=False)

    return data.json()['result']
```

Модуль keyboard.py

```
from aiogram.types import *

keyboard = ReplyKeyboardMarkup(row_width=1, resize_keyboard=True)

buttons = []
buttons.append(KeyboardButton(text="/help"))
```

```
buttons.append(KeyboardButton(text="/now"))
buttons.append(KeyboardButton(text="/tomorrow"))
buttons.append(KeyboardButton(text="/week"))

for button in buttons:
    keyboard.add(button)
```

Модуль logs.py

```
from datetime import datetime

import sources.py.files as files
import sources.py.groups as groups
from sources.py.tree import *


def gitPush():
    os.system("./../../git-push.sh")


def writeLog(text):
    filename = 'log-' + str(datetime.now().date()) + '.log'
    time = str(datetime.now().time())

    logsFilePath = os.path.join("../..", "logs", filename)

    if os.path.exists(logsFilePath) == True:
        logs = files.loadFile(logsFilePath)
    else:
        groups.update()

        logs = {}

    logs[time] = text

    files.saveFile(logs, logsFilePath)
```

Модуль main.py

```
import asyncio

import wolframalpha
from aiogram import Bot, Dispatcher, executor

import sources.py.commands as commands
import sources.py.groups as groups
import sources.py.notifications as notifications
import sources.py.schedule_func as schedule_func
from sources.py.config import *

groups.update()

commands.load()

schedule_func.load()

client = wolframalpha.Client(BOT_WOLF_TOKEN)
loop = asyncio.get_event_loop()
bot = Bot(BOT_TOKEN, parse_mode="HTML")
dp = Dispatcher(bot, loop=loop)

if __name__ == "__main__":
    print('Bot started!')
```

```
    dp.loop.create_task(notifications.check(time_sleep))
    from sources.py.handlers import dp, send_to_admin

    executor.start_polling(dp, on_startup=send_to_admin)
```

Модуль messages.py

```
from aiogram.types import Message

import sources.py.json_func as json_func
from sources.py.tree import *


def push(message: Message):
    global messages

    messages = json_func.loadJson(messagesFilePath)

    text_message = message.text.replace("/send ", "")

    text = f"username = @{message.from_user.username} name =
{message.from_user.first_name} chat_id = {message.chat.id} user_id =
{message.from_user.id}\nMessage: {text_message}"

    messages.append(text)

    json_func.saveJson(messages, messagesFilePath)
```

Модуль notifications.py

```
import asyncio
from datetime import datetime as datef

import sources.py.chats as chats
import sources.py.groups as groups
import sources.py.logs as logs
import sources.py.schedule_func as schedule_func
from main import bot
from sources.py.config import *


async def check(sleep):
    subject_black_list = ['Військова підготовка ()', 'Дисципліни за вибором
студента ()']

    while True:
        await asyncio.sleep(sleep)

        print('Notifications check')

        datetime_str = datef.today()
        datetime_obj = datetime_str.strftime("%H%M")

        if datetime_obj == "0000":
            for chat_id in chats.chats.keys():
                try:
                    await bot.send_message(chat_id=chat_id, text='Слава
Україні!' + u'\U0001F634')
                except:
                    print("Something went wrong when bot trying send message
to user")
            result = groups.update()
            logs.writeLog(result)
```

```
            schedule_func.load()

        if datef.today().strftime("%A") not in ['Wednesday', 'Thursday']:

            date = schedule_func.get_current_date()
            time = schedule_func.get_current_time()

            for code in schedule_func.schedules.keys():
                schedule = schedule_func.get_subj_list(code)
                name = groups.getName(code)

                max = 0

                for subject in schedule:
                    if subject['name'] not in subject_black_list and subject[
                        'date'] == date and schedule_func.get_int_time(
                            subject['time_begin']) - time == time_before:
                        answer = f"⁉Заняття для групи {name} відбудеться через
{time_before}
хв:\n📣{subject['name']}\nℹ{subject['date']}\n👩‍🏫{subject['teacher']}\n🕐{sub
ject['time_begin']}-{subject['time_end']}\n⏭{subject['url']}"
                        for chat_id in chats.chats.keys():
                            if name in chats.chats[chat_id]:
                                try:
                                    await bot.send_message(chat_id=chat_id,
text=answer)
                                except:
                                    print("Something went wrong when bot
trying send message to user")

                    if subject['date'] == date and
schedule_func.get_int_time(subject['time_end']) > max:
                        max = schedule_func.get_int_time(subject['time_end'])

                if max == time and max > 0:
                    for chat_id in chats.chats.keys():
                        if name in chats.chats[chat_id]:
                            try:
                                await bot.send_message(chat_id=chat_id,
text=f"⁉Заняття для групи {name} закінчились!")
                            except:
                                print("Something went wrong when bot trying
send message to user")
                elif max == time and max == 0:
                    for chat_id in chats.chats.keys():
                        if name in chats.chats[chat_id]:
                            try:
                                await bot.send_message(chat_id=chat_id,
                                                       text=f"⁉Сьогодні немає
занять для групи {name}!")
                            except:
                                print("Something went wrong when bot trying
send message to user")
```

Модуль schedule.py

```
from sources.py.schedule_func import *


def help_get_url(id):
    name = groups.getName(id)

    schedule = get_subj_list(id)
```

```
    classes_today = False

    date = get_current_date()
    time = get_current_time()

    for subject in schedule:
        if subject['date'] == date and time <
get_int_time(subject['time_end']):
            classes_today = True
            break
    if classes_today:
        return f"🔜Найближче заняття для групи
{name}\n📢{subject['name']}\nℹ{subject['date']}\n👩‍🏫{subject['teacher']}\n🕐{
subject['time_begin']}-{subject['time_end']}\n⏭{subject['url']}"
    else:
        return f'⛔Сьогодні вже немає наступних занять для групи {name}'


def help_today(id):
    name = groups.getName(id)

    schedule = get_subj_list(id)

    classes_today = False

    date = get_current_date()

    result = f'📋Розклад на сьогодні для групи {name}\n'
    for subject in schedule:
        if (subject['date'] == date):
            classes_today = True
            result += ' ✅ ' + subject['time_begin'] + '-' +
subject['time_end'] + ' - ' + subject['name'] + '\n'

    if classes_today == True:
        return result
    else:
        return f'⛔Сьогодні немає занять для групи {name}'


def help_tomorrow(id):
    name = groups.getName(id)

    schedule = get_subj_list(id)

    classes_tomorrow = False

    date = get_current_date()
    date = date_tomorrow(date)

    result = f'📋Розклад на завтра для групи {name}\n'
    for subject in schedule:
        if (subject['date'] == date):
            classes_tomorrow = True
            result += ' ✅ ' + subject['time_begin'] + '-' +
subject['time_end'] + ' - ' + subject['name'] + '\n'
    if result != f'📋Розклад на завтра для групи {name}\n':
        return result
    else:
        return f'⛔Завтра немає занять для групи {name}'
```

```
def help_week(id):
    name = groups.getName(id)

    schedule = get_subj_list(id)

    date = schedule[0]['date']
    flag = False
    result = f'Розклад на тиждень для групи {name}\n'

    for subject in schedule:
        if (subject['date'] == date):
            if flag == False:
                result += '\n' + date + ':\n'
                flag = True
            result += ' ✅ ' + subject['time_begin'] + '-' +
subject['time_end'] + ' - ' + subject['name'] + '\n'
        else:
            flag = False

            date = subject['date']

            if (subject['date'] == date):
                if flag == False:
                    result += '\n' + date + ':\n'
                    flag = True
                result += ' ✅ ' + subject['time_begin'] + '-' +
subject['time_end'] + ' - ' + subject['name'] + '\n'

    if result != f'Розклад на тиждень для групи {name}\n':
        return result
    else:
        return f'⛔На цей тиждень немає занять для групи {name}'


def help_date(id, date):
    name = groups.getName(id)

    schedule = get_subj_list(id)

    flag = False

    result = f'Розклад на {date} для групи {name}\n'

    for subject in schedule:
        if (subject['date'] == date):
            flag = True
            result += ' ✅ ' + subject['time_begin'] + '-' +
subject['time_end'] + ' - ' + subject['name'] + '\n'

    if flag == True:
        return result
    else:
        return f'⛔На {date} немає занять для групи {name}'
```

Модуль schedule_func.py

```
from datetime import datetime, timedelta

import sources.py.files as files
import sources.py.groups as groups
from sources.py.tree import *
```

```
schedules = {}

all_subjects = {}


def load():
    global schedules
    global all_subjects

    schedules = {}

    for code in groups.groups.keys():
        group = groups.groups[code]

        filename = 'schedule-' + code + '.json'
        path = schedulesFilePath + '/' + filename

        if os.path.exists(path) == True:
            schedule = files.loadFile(path)

            schedules[code] = schedule

    all_subjects = files.loadFile(subjectsFilePath)


def getLength(num):
    length = 1
    num = int(num)

    while num >= 10:
        length = length + 1
        num = num / 10

    return length


def getStrFormat(format, num):
    length = getLength(int(num))
    bkp = str(int(num))
    cNum = ''

    if length < format:
        for i in range(format - length):
            cNum += '0'

    cNum += bkp

    return cNum


def createDate(format, date):
    cDate = getStrFormat(format, date[0])

    flag = 0

    for num in date:
        if flag == 1:
            cDate += '.' + getStrFormat(format, num)
        flag = 1

    return cDate
```

```
def date_tomorrow(date):
    date = datetime.strptime(date, '%d.%m.%Y').date()
    date = date + timedelta(days=1)
    date = date.strftime('%d.%m.%Y')
    date_arr = date.split('.')

    return createDate(2, date_arr)


def get_current_date():
    date = str(datetime.now().date()).split('-')
    date.reverse()
    date = str(date[0]) + '.' + str(date[1]) + '.' + str(date[2])

    return date


def get_current_time():
    time = str(datetime.now().time()).split(':')
    time = int(time[0]) * 60 + int(time[1])
    return time


def url_of_subject(id, namea, nameb):
    name = namea + ' (' + nameb + ')'
    if id not in all_subjects.keys():
        return 'Немає посилання на заняття'

    if name in all_subjects[id].keys():
        return all_subjects[id][name]

    return 'Немає посилання на заняття'


def get_int_time(time):
    time_int = time.split(':')
    time_int = int(time_int[0]) * 60 + int(time_int[1])
    return time_int


def set_schedule(id):
    code = id
    filename = 'schedule-' + code + '.json'
    path = schedulesFilePath + '/' + filename

    if os.path.exists(path) == False:
        groups.getSchedule(code)
        load()

    return schedules[code]


def sort_list(list):
    for i in range(0, len(list) - 1):
        for j in range(i + 1, len(list)):
            if list[i]['date'] == list[j]['date'] and
get_int_time(list[i]['time_begin']) > get_int_time(
                    list[j]['time_begin']):
                tmp = list[i]
                list[i] = list[j]
                list[j] = tmp


def get_subj_list(id):
```

```
    list_of_subjects = []

    schedule = set_schedule(id)

    for subject in schedule:
        if subject['NAME_STUD'] == None:
            subject['NAME_STUD'] = ""

        name = subject['NAME_DISC'] + ' (' + subject['NAME_STUD'] + ')'
        time = subject['TIME_PAIR']

        dict_of_subject = {}
        dict_of_subject['date'] = subject['DATE_REG']

        if subject['NAME_DISC'] == '':
            dict_of_subject['name'] = subject['NAME_STUD']
        else:
            dict_of_subject['name'] = subject['NAME_DISC'] + ' (' +
subject['NAME_STUD'] + ')'

        dict_of_subject['name'] += ' ' + subject['REASON'] + ' ' +
subject['NAME_AUD']

        dict_of_subject['time_begin'] = time[0:5]
        dict_of_subject['time_end'] = time[6:]

        dict_of_subject['url'] = url_of_subject(id, subject['NAME_DISC'],
subject['NAME_STUD'])

        dict_of_subject['teacher'] = subject['NAME_FIO']

        list_of_subjects.append(dict_of_subject)

    sort_list(list_of_subjects)

    return list_of_subjects


def setLink(code, subject, link):
    if all_subjects.get(code) == None:
        all_subjects[code] = {}

    all_subjects[code][subject] = link

    files.saveFile(all_subjects, subjectsFilePath)
```

Модуль tree.py

```
import os

accessesFilePath = os.path.join("../..", "resources", "json", "commands",
"accesses.json")

createSource = "../sh/createSource.sh"

removeSource = "../sh/removeSource.sh"

startBotFile = "../../start.sh"

schedulesFilePath = os.path.join("../..", "resources", "json", "schedules")

scheduleFilePath = {}
```

```
usersFilePath = os.path.join("../..", "resources", "json", "users",
"users.json")

groupsFilePath = os.path.join("../..", "resources", "json", "schedules",
"groups.json")

subjectsFilePath = os.path.join("../..", "resources", "json", "schedules",
"subjects.json")

chatsFilePath = os.path.join("../..", "resources", "json", "users",
"chats.json")

messagesFilePath = os.path.join("../..", "resources", "json", "users",
"messages.json")

descriptionsFilePath = os.path.join("../..", "resources", "json", "commands",
"descriptions.json")
```

Модуль users.py

```
import sources.py.accesses as accesses
import sources.py.files as files
import sources.py.logs as logs
from sources.py.tree import *

users = {}


def load(usersFilePath):
    global users

    users = files.loadFile(usersFilePath)

    return users


def create():
    global users

    users = {}

    users['admin'] = []
    users['user'] = []


def save(usersFilePath):
    files.saveFile(users, usersFilePath)


def checkUser(access, t_id):
    if users.get(access) == None:
        return False

    for user in users[access]:
        if user['id'] == t_id:
            return True
    return False


def addUser(access, t_id, group):
    user = {}
    user['id'] = t_id
    user['group'] = group
    users[access].append(user)
```

```
    save(usersFilePath)

    logs.writeLog(f'Aded new user with id {t_id}')


def searchUser(access, t_id):
    i = 0
    for user in users[access]:
        if user['id'] == t_id:
            return i
        i = i + 1


def checkUser(t_id):
    access = getAccess(t_id)

    if access == None:
        return False
    else:
        return True


def set(access, t_id, setting, value):
    if checkUser(t_id):
        users[access][searchUser(access, t_id)][setting] = value


def get(access, t_id, setting):
    if checkUser(t_id):
        return users[access][searchUser(access, t_id)][setting]


def getAccess(t_id):
    for access in users.keys():
        for user in users[access]:
            if user['id'] == t_id:
                return access
    return None


def checkCommand(t_id, command):
    return accesses.checkCommand(getAccess(t_id), command)


load(usersFilePath)
```

ДОДАТОК Б ЛІСТИНГ БІБЛІОТЕК

```
aiogram==2.13
aiohttp==3.8.5
aiosignal==1.3.1
async-timeout==4.0.3
attrs==23.1.0
Babel==2.12.1
certifi==2023.7.22
chardet==3.0.4
charset-normalizer==3.2.0
frozenlist==1.4.0
idna==2.8
jaraco.context==4.3.0
more-itertools==10.1.0
multidict==6.0.4
requests==2.22.0
urllib3==1.25.8
wolframalpha==5.0.0
xmltodict==0.13.0
yarl==1.9.2
```