РЕФЕРАТ

Пояснювальна записка: ___ с., ___ рис., ___ табл., __ дод., ___ джерел.

Об'єкт розробки: чат-бот на базі месенджеру Telegram для моніторингу розкладу занять.

Мета кваліфікаційної роботи: розробка програмного забезпечення на базі месенджеру Telegram для зручного моніторингу розкладу занять в учбовому закладі.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточнюється постановка завдання.

У першому розділі проведено аналіз предметної галузі, визначено актуальність завдання та призначення розробки, розроблена постановка завдання, задані вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі виконано аналіз існуючих рішень, обрано вибір платформи для розробки, виконано проектування і розробка програми, наведено опис алгоритму і структури функціонування програми, визначені вхідні і вихідні дані, наведені характеристики складу параметрів технічних засобів, описаний виклик та завантаження програми, описана робота програми.

В економічному розділі визначено трудомісткість розробленої інформаційної системи, проведений підрахунок вартості роботи по створенню програми та розраховано час на його створення.

Практичне значення роботи полягає в можливості завдяки розробленому додатку всім учасникам освітнього процесу легко та швидко отримувати актуальну інформацію щодо розкладу занять, їх замін та графіку навчального процесу.

Актуальність теми кваліфікаційної роботи визначається великим попитом на подібні розробки, через те, що розроблене програмне забезпечення дозволить студентам та викладачам завжди залишатися в курсі останніх новин щодо освітнього процесу та вчасно реагувати на зміни в розкладі, а також не забувати відвідувати заняття.

Список ключових слів: ЧАТ-БОТ, МЕСЕНДЖЕР, РОЗКЛАД, РОЗСИЛКА, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, АЛГОРИТМ, ПРОЄКТУВАННЯ, ПРОГРАМА.

ABSTRACT

Explanatory note: ___ pp., ___ fig., ___ table, __ appendix, ___ sources.

Object of development: a chatbot based on the Telegram messenger to monitor the class schedule.

The purpose of the qualification work: development of software based on the Telegram messenger for convenient monitoring of the schedule of classes in the educational institution.

The introduction considers the analysis and the current state of the problem, specifies the purpose of the qualification work and the field of its application, provides a justification for the relevance of the topic and clarifies the problem.

In the first section the analysis of the subject area is carried out, the urgency of the task and purpose of development is defined, the statement of the task is developed, requirements to software realization, technologies and software are set.

The second section analyzes existing solutions, selects a platform for development, performs design and development of the program, describes the algorithm and structure of the program, defines input and output data, provides characteristics of the parameters of hardware, describes the call and download of the program, describes the program .

The economic section determines the complexity of the developed information system, calculates the cost of work to create a program and calculates the time for its creation.

The practical significance of the work lies in the possibility, thanks to the developed application, for all participants of the educational process to easily and quickly receive up-to-date information on the schedule of classes, their replacement and the schedule of the educational process.

The relevance of the topic of qualification work is determined by the great demand for such developments, due to the fact that the developed software will allow students and teachers to stay up to date with the latest news on the educational process and respond to changes in schedule, and do not forget to attend classes.

Keywords: CHAT-BOT, MESSENGER, SCHEDULE, NEWSLETTER, SOFTWARE, ALGORITHM, DESIGN, PROGRAM.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних;

ІС – інформаційна система;

ОС – операційна система;

ПЗ – програмне забезпечення;

ПК – персональний комп’ютер;

СКБД – система керування базами даних;

MS – Microsoft.

ЗМІСТ

ВСТУП

В даній кваліфікаційній роботі засобами середовища Microsoft Visual Studio Code розроблено чат-бота для месенджеру Telegram, що призначений для допомоги студентам та викладачам в учбовому процесі.

Під час навчання в закладі освіти будь-якого типу кожен студент постійно повинен слідкувати за інформацією щодо навчального процесу. Те ж саме стосується і викладачів. Особливо це актуально на початку навчального семестру, коли розклад ще може змінюватись і одразу запам’ятати його не вдається, вже не говорячи про заміни або переноси занять, які оновлюються майже кожен день.

Наразі це не дуже зручно, оскільки при собі потрібно постійно тримати графік освітнього процесу та розклад занять, потрібно слідкувати за новими замінами та відпрацюваннями. Виникає потреба в автоматизації цього процесу, щоб за декілька легких дій можна було отримати актуальну інформацію.

Задоволення цієї потреби буде виконано в форматі чат-бота на базі месенджеру Telegram. Чат-бот – це програмне забезпечення, що імітує діалог зі справжньою людиною. Це дуже зручно, оскільки месенджер Telegram зараз надзвичайно популярний в нашій країні, особливо серед студентів. Через це більшості користувачів навіть не доведеться встановлювати додаткове програмне забезпечення, а можна буде використовувати звичний для них месенджер, в якому вони спілкуються щодня.

Виконувати оновлення інформаційної бази зможе співробітник учбової частини, куратор або староста групи. Для цього їм потрібно буде лише завантажити в діалог з чат-ботом готовий Excel файл. Такі файли зазвичай формуються навчальною частиною і вносити жодні корективи в їх оформлення не доведеться, окрім тимчасових переносів, що корегуються старостою групи.

Також корисною буде система оголошень, яка дозволить проінформувати всіх користувачів чат-бота одночасно. Завдяки цьому зникне необхідність розсилати новини по різним чатам, оскільки вся інформація буде зібрана в

одному місці.

Отже, можна дійти висновку, що тема кваліфікаційної роботи є актуальною, оскільки таке програмне забезпечення дозволить студентам та викладачам завжди залишатися в курсі останніх новин щодо освітнього процесу.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Загальні відомості з предметної галузі

1.1.1. Можливості месенджеру Telegram

Telegram - клауд-месенджер, програмне забезпечення для смартфонів, планшетів та ПК, яке дозволяє обмінюватися текстовими повідомленнями, графічними та відеофайлами, а також безкоштовно телефонувати іншим користувачам програми. Створений російським розробником Павлом Дуровим.

Обліковий запис користувача прив'язується до номера мобільного телефону: щоб авторизуватися, потрібно ввести код авторизації з СМС. Такі коди мають обмежені терміни придатності. Таким чином, користувач позбавляється необхідності запам'ятовувати чи зберігати десь свій пароль.

Також є можливість увімкнути двофакторну автентифікацію. Тоді для входу додатково знадобиться пароль, заданий користувачем у налаштуваннях. Деякі можливості, такі як Telegram Passport, вимагають увімкненої двофакторної автентифікації.

Для месенджера був створений протокол MTProto, що передбачає використання декількох протоколів шифрування. Під час авторизації і аутентифікації використовуються алгоритми RSA-2048, DH-2048 для шифрування, під час передачі повідомлень протоколу в мережу вони шифруються AES з ключем, відомим клієнту і серверу. Також застосовуються криптографічні хеш-алгоритми SHA-1 і MD5.

Безпека від перехоплення повідомлень, що пересилаються з боку сервера Telegram, забезпечується лише в режимі «секретних» чатів (Secret Chats), доступному з 8 жовтня 2013 року. Цей режим реалізує шифрування, при якому відправник і одержувач мають спільний лише для них ключ (end-to-end шифрування), із застосуванням алгоритму AES-256 у режимі IGE (англ. Infinite Garble Extension) для повідомлень, що пересилаються. На відміну від

звичайного режиму, повідомлення в секретних чатах не розшифровуються сервером, історія листування зберігається лише на тих двох пристроях, на яких був створений чат.

Підтримується обмін фотографіями, відеозаписами та файлами будь-якого типу. Для фотографій доступна функція пошуку в інтернеті. Розмір файлів обмежений 2 Гб. Програма використовує систему докачування файлів після обриву зв'язку.

Є можливість організовувати мультичати до 200 учасників, починаючи з 14 березня 2016 - супергрупи до 5000 учасників. Станом на 17 червня 2018 року підтримуються чати до 100 тисяч учасників.

30 вересня 2020 року вийшло оновлення офіційних клієнтів Telegram, де було змінено інтерфейс обговорення в каналах. Він нагадує систему коментування публікацій у блозі. У каналах, прив'язаних до групи, під кожною публікацією є кнопка для відображення гілки відповідей на її автоматичну копію в цій групі - коментарів до публікації.

У березні 2017 року Telegram запустив можливість голосових викликів. Спочатку така функція стала доступною користувачам із Західної Європи, а згодом і для всього світу. Для активації послуги необхідно було отримати дзвінок від користувача із активованими дзвінками. Дзвінки, за аналогією з листуванням, також захищені шифруванням.

Якість голосового виклику автоматично налаштовується залежно від якості зв'язку користувача. Так, під час користування мобільним інтернетом стиснення буде дещо вищим, ніж при з'єднанні через Wi-Fi. Функція стала доступною для мобільних клієнтів у березні 2017, для десктопних застосунків така функція стала доступна у травні 2017.

У 2020 році було додано відеодзвінки. Відеодзвінки, як і голосові дзвінки, підтримують peer-to-peer.

У Telegram працює платформа чатботів. Боти можуть виконувати різноманітні завдання, такі як пошук в інтернеті чи держреєстрах, покупки, платежі, розваги, модерація груп тощо. У спілкуванні беруть участь користувач

Telegram та комп'ютерна програма від стороннього розробника.

Користувач може взаємодіяти з ботом за допомогою елементів інтерфейсу месенджера: надсилання повідомлень, натискання на команди та кнопки, використання інлайн-режиму. Telegram надає три способи взаємодії користувача з ботом: приватний чат (класичний спосіб), група й так званий інлайн-режим:

– найпоширеніший спосіб - приватний чат. Здебільшого бот не може ініціювати діалог із користувачем;

– деякі боти можуть бути учасниками груп. Наприклад, у групах бот може підтримувати розмову, модерувати повідомлення або бути ведучим гри.

– інлайн-режим нагадує інтерфейс пошукової системи. Користувач уводить у поле для введення повідомлень запит, що починається з короткого імені бота. Далі користувач може вибрати й надіслати один з результатів.

Деякі боти можуть бути учасниками каналів. У каналах ботів застосовують здебільшого для запланованого розміщення повідомлень. З такими ботами користувач взаємодіє через приватний чат або веб-інтерфейс.

Бот взаємодіє з користувачами за допомогою Bot API. Керування ботами на платформі доступне в боті BotFather. Зокрема, там можна створити бота та переглянути чи змінити присвоєний йому токен API. [20]

1.1.2. Огляд існуючих рішень

У зв’язку з розробкою кваліфікаційної роботи, було прийнято рішення знайти та проаналізувати вже існуючі схожі проєкти для кращого розуміння того, які можливості та функціональність повинні бути втілені в чат-боті даної розробки.

Для аналізу обрано чат-бота «Розклад для студентів СПбДУ». Вся комунікація з ботом здійснюється за допомогою меню. Користувачу не потрібно вводити жодну інформацію власноруч. Після запуску бота, необхідно

ввести повну інформацію про свою навчальну програму. На рис. 1.1 зображено введення інформації про навчальну програму.

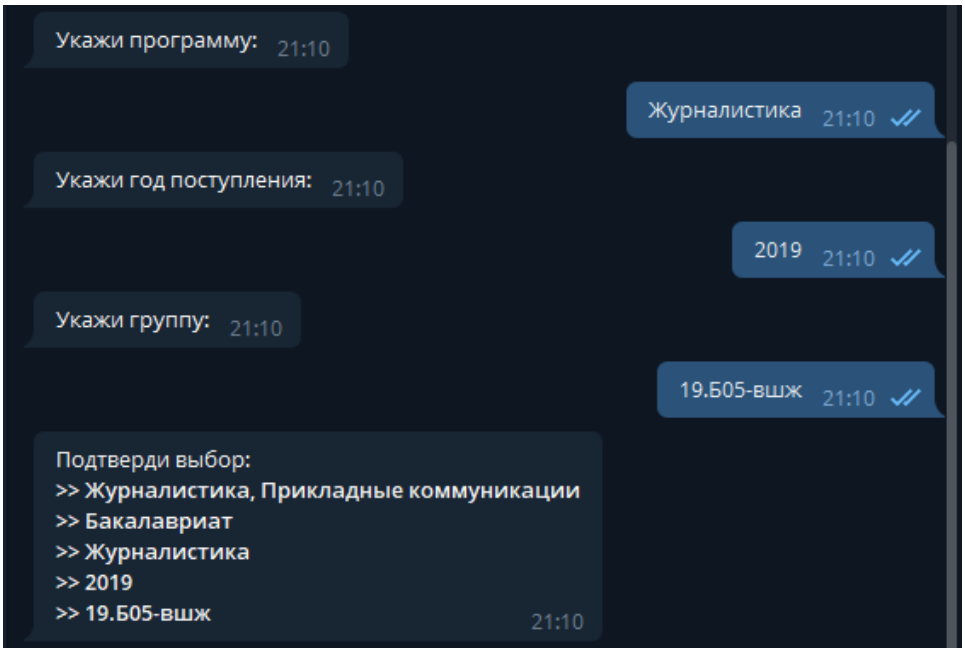


Рис. 1.1. Введення інформації про навчальну програму

Після встановлення групи, є можливість переглянути розклад занять. На рис. 1.2 зображено перегляд розкладу занять.

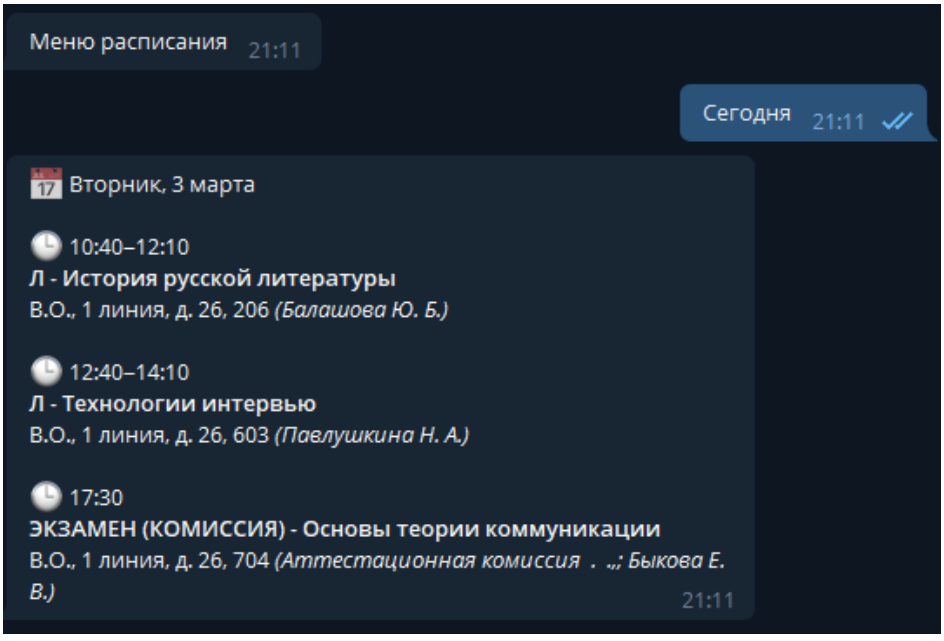


Рис. 1.2. Перегляд розкладу занять

Також чат-бот надає можливість перегляду розкладу міського електротранспорту. На рис. 1.3 зображено перегляд найближчого відправлення від університету.

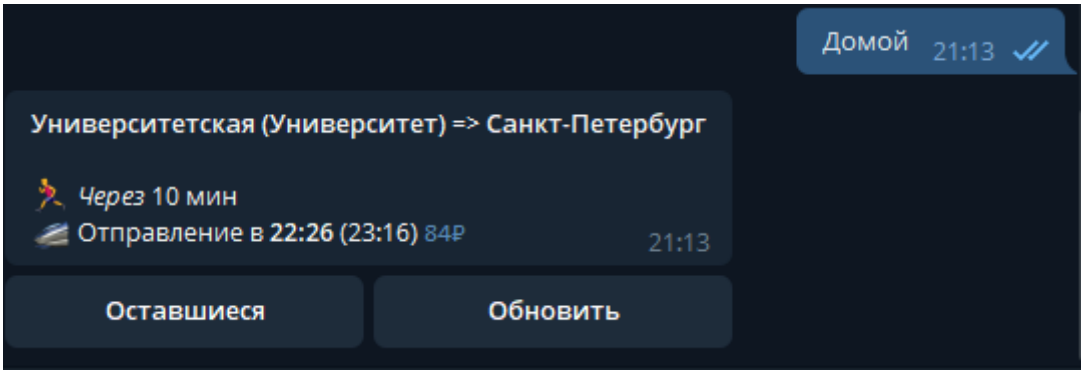


Рис. 1.3. Перегляд найближчого відправлення транспорту від університету

1.2. Призначення розробки та галузь застосування

Головне призначення чат-бота - надання допомоги студентам та викладачам щоб легко та швидко отримувати актуальну інформацію щодо розкладу занять, їх замін та графіку навчального процесу.

Розроблене програмне забезпечення дозволить студентам та викладачам завжди залишатися в курсі останніх новин щодо освітнього процесу та вчасно реагувати на зміни в розкладі, а також не забувати відвідувати заняття.

За допомогою чат-бота будь-який користувач зможе переглянути актуальну інформацію про графік освітнього процесу, про розклад занять та їх заміни. Для цього необхідно буде лише мати підключення до інтернету та створений обліковий запис в месенджері Telegram. Інформацію про розклад занять можна буде отримати як за шифром групи, так і за ПІБ викладача.

Співробітник учбової частини, куратор або староста групи зможе виконати оновлення інформації за допомогою інтерфейсу чат-бота. Для цього потрібно буде лише перенести відповідний файл в діалог з чат-ботом. Також є можливість робити оголошення, які отримають всі користувачі бота-помічника.

Користувач матиме можливість підписатись на щоденну розсилку

розкладу занять. Для цього необхідно буде обрати викладача або групу та зручний час.

Всі перелічені вище можливості призначені для того, щоб полегшити освітній процес та зробити його максимально комфортним для студентів та викладачів.

1.3. Підстава для розробки

Підставою для розробки кваліфікаційної роботи бакалавра на тему «Розробка чат-бота на базі месенджеру Telegram для моніторингу розкладу занять в закладі освіти засобами середовища Microsoft Visual Studio Code» є наказ по Національному технічному університету «Дніпровська політехніка» від __.__. 2021р. № ____-_.

1.4. Постановка завдання

В межах даної кваліфікаційної роботи необхідно розробити чат-бота на базі месенджеру Telegram, який дозволить зручно отримувати інформацію про розклад занять (графік навчального процесу, заміни), виконувати оновлення цієї інформації та допоможе користувачам залишатись в курсі останніх новин коледжу.

Для виконання даної роботи необхідно виконати наступні завдання:

– описати процес проєктування чат-бота;
– обґрунтувати вибір мови програмування
– виконати програмування додатку;
– перевірити результати його реалізації.

В програмному додатку необхідно реалізувати наступні функції:

– ролі користувачів: «Адміністратор» та «Користувач з правом перегляду»;
– для користувача з роллю «Адміністратор» надати можливість робити

оголошення;

– для користувача з роллю «Адміністратор» надати можливість завантажити файли з розкладом занять, графіком навчального процесу та замін занять в форматі Excel;

– для всіх користувачів можливість переглянути розклад за шифром групи або ПІБ викладача на обраний день;

– для всіх користувачів можливість налаштувати щоденну розсилку розкладу занять за шифром групи або ПІБ викладача.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

В межах даної кваліфікаційної роботи необхідно організувати базу даних, яка міститиме інформацію про графік навчального процесу, розклад занять та їх заміни. Також необхідно розробити чат-бота, який дозволить користувачу оперативно отримати інформацію з бази даних або оновити її.

Чат-бот повинен мати дружній до користувача інтерфейс, надавати користувачу підказки під час роботи та перевіряти всі вхідні дані на коректність.

Додаток повинен бути реалізований з використанням таких ролей користувачів: «Адміністратор» та «Користувач з правом перегляду».

Для ролі «Адміністратор» необхідно реалізувати функції оновлення графіка навчального процесу, розкладу занять та їх замін. Для цього користувач зможе перейти у відповідне меню та перенести готовий файл в діалог з чат-ботом. Також необхідно реалізувати функцію для надсилання оголошень, які отримуватимуть всі користувачі бота.

Для всіх користувачів необхідно реалізувати функції перегляду розкладу занять за вказаним шифром групи або ПІБ викладача. Інформація повинна бути надана з урахуванням графіка освітнього процесу та існуючих замін на обрану дату.

Також для всіх користувачів необхідно надати можливість підписки на щоденну розсилку розкладу занять. Для цього користувачеві буде необхідно обрати викладача або групу та зручний час для отримання повідомлень. У випадку, якщо для обраного викладача або групи не було знайдено занять за розкладом та замін на наступний день, чат-бот не повинен надсилати повідомлення.

1.5.2. Вимоги до інформаційної безпеки

Для надійної роботи системи необхідно:

1. Використовувати ліцензійне програмне забезпечення на сервері.
2. Здійснювати захист від вірусів на сервері.
3. Здійснювати захист від несанкціонованого доступу.
4. Застосовувати на сервері джерело безперебійного живлення для захисту від перепадів напруги або збоїв у живленні.
5. Здійснювати контроль даних, що вводяться користувачами.
6. Автоматично завершувати сеанс роботи з користувачем у разі тривалої перерви його активності.

1.5.3. Вимоги до складу та параметрів технічних засобів

Рекомендовані вимоги до апаратного забезпечення сервера для розташування чат-бота:

- операційна система: Windows 7/8/10/Server, Linux;
- двоядерний процесор з частотою не менше, ніж 3 ГГц;
- вільний об’єм пам’яті на жорсткому диску: 128 Мб;
- об’єм оперативної пам’яті: 2 Гб;
- доступ до інтернету.

Рекомендовані вимоги до апаратного забезпечення користувача:

– пристрій зі встановленим месенджером Telegram, наявність в ньому облікового запису;

– доступ до інтернету.

1.5.4. Вимоги до інформаційної та програмної сумісності

Рекомендовані вимоги до програмного забезпечення сервера та користувача для розташування чат-бота:

– встановлений месенджер Telegram;

– наявність інтерпретатора Python з версією 3.7 або вище;

– СУБД Redis.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

В межах даної кваліфікаційної роботи необхідно розробити чат-бота на базі месенджеру Telegram, який дозволить зручно отримувати інформацію про розклад занять (графік навчального процесу, заміни), виконувати оновлення цієї інформації та допоможе користувачам залишатись в курсі останніх новин коледжу.

Додаток реалізований з використанням таких ролей користувачів: «Адміністратор» та «Користувач з правом перегляду».

Для ролі «Адміністратор» реалізовано функції оновлення графіка навчального процесу, розкладу занять та їх замін. Для цього користувач може перейти у відповідне меню та перенести готовий файл в діалог з чат-ботом. Також реалізовано функцію для надсилання оголошень, які отримуватимуть всі користувачі бота.

Для всіх користувачів реалізовано функції перегляду розкладу занять за вказаним шифром групи або ПІБ викладача. Інформація надана з урахуванням графіка освітнього процесу та існуючих замін на обрану дату.

Також для всіх користувачів є можливість підписки на щоденну розсилку розкладу занять. Для цього користувачеві необхідно обрати викладача або групу та зручний час для отримання повідомлень. У випадку, якщо для обраного викладача або групи не було знайдено занять за розкладом та замін на наступний день, чат-бот не надсилає жодних повідомлень.

2.2. Опис застосованих математичних методів

Математичні методи під час проєктування та розробки програмного забезпечення не застосовувалися.

2.3. Опис використаних технологій та мов програмування

Для реалізації додатку кваліфікаційної роботи обрано мову програмування Python.

Python - інтерпретована об'єктно-орієнтована мова програмування високого рівня з суворою динамічною типізацією. Розроблена в 1990 році Гвідо ван Россумом. Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднування наявних компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій, так і у вихідній формі на всіх основних платформах. В мові програмування Python підтримується кілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована [14].

Серед основних переваг мови Python можна назвати такі:

– чистий синтаксис;

– стандартний дистрибутив має велику кількість корисних модулів (включно з модулем для розробки графічного інтерфейсу);

– можливість використання Python в діалоговому режимі;

– стандартний дистрибутив має просте, але разом із тим досить потужне середовище розробки, яке зветься IDLE і яке написане на мові Python;

– зручний для розв'язування математичних проблем (має засоби роботи з комплексними числами, може оперувати з цілими числами довільної величини, у діалоговому режимі може використовуватися як потужний калькулятор);

– відкритий код.

Елегантний синтаксис Python, динамічна обробка типів, а також те, що це інтерпретована мова, роблять її ідеальною для написання скриптів та швидкої розробки прикладних програм у багатьох галузях на більшості

платформ [7].

Інтерпретатор мови Python може бути розширений функціями та типами даних, розробленими на C чи C++. Python також зручна як мова розширення для прикладних програм, що потребують подальшого налагодження.

Python підтримує динамічну типізацію, тобто, тип змінної визначається лише під час виконання. Python має багату бібліотеку для роботи з рядками, зокрема, кодованими в юнікоді.

З колекцій Python підтримує кортежі, списки і множини. Система класів підтримує множинне успадкування. Будь-який тип, включаючи базовий, входить до системи класів, й за необхідності можливе успадкування навіть від базових типів.

При своїй роботі основний інтерпретатор Python постійно використовує велику кількість потоко-небезпечних даних. В основному це словники, в яких зберігаються атрибути об'єктів. Для уникнення руйнування цих даних при спільній модифікації з різних потоків перед початком виконання декількох інструкцій потік інтерпретатора захоплює GIL, а після закінчення звільняє. Внаслідок цієї особливості в кожен момент часу в одному процесі може виконуватися тільки один потік Python коду, навіть якщо на комп'ютері є кілька процесорів або процесорних ядер. Була зроблена спроба переходу до більш гранульованої синхронізації, проте через часті захоплення та звільнення блокувань ця реалізація виявилася занадто повільною.

Python, як і багато інших інтерпретованих мов, має загальний недолік - порівняно невисоку швидкість виконання програм. Однак, у випадку з Python цей недолік компенсується зменшенням часу розробки програми.

Безліч програм та бібліотек для інтеграції з іншими мовами програмування надають можливість використовувати іншу мову для написання критичних ділянок.

Незважаючи на всі недоліки, Python - стабільна і поширена мова програмування. Вона використовується в багатьох проєктах і в різних сферах: як основна мова програмування або для створення розширень та інтеграції

додатків. На Python вже реалізовано велику кількість проєктів, також він активно використовується для створення прототипів майбутніх програм. Python використовується в багатьох великих компаніях, таких як Google та Facebook.

Мова програмування Python обрана для даної роботи, оскільки вона надає широкі можливості для розробки чат-ботів на базі месенджеру Telegram та має безліч корисних бібліотек, які полегшують процес розробки програмного додатку.

В якості середовища розробки обрано Microsoft Visual Studio Code.

Visual Studio Code - це засіб для створення, редагування та відлагодження сучасних веб-додатків і програм. Visual Studio Code розповсюджується безкоштовно і доступний у версіях для платформ Windows, Linux та OS X.

Компанія Microsoft представила Visual Studio Code у квітні 2015 на конференції Build 2015. Це середовище розробки стало першим крос-платформовим продуктом у лінійці Visual Studio. За основу для Visual Studio Code використовуються напрацювання вільного проєкту Atom, що розвивається компанією GitHub. Зокрема, Visual Studio Code є надбудовою над Atom Shell, що використовують браузерний рушій Chromium і Node.js.

Visual Studio Code - це редактор вихідного коду. Він підтримує ряд мов програмування, підсвічування синтаксису, рефакторинг, відлагодження, навігацію по коду, підтримку Git та інші можливості. Багато можливостей Visual Studio Code не доступні через графічний інтерфейс, найчастіше вони використовуються через систему команд або JSON файли [12].

Visual Studio також дозволяє замінювати кодову сторінку при збереженні документа, символи перекладу рядка і мову програмування поточного документа.

На сьогодні за допомогою вбудованого в продукт призначеного для користувача інтерфейсу можна завантажити і встановити кілька тисяч розширень в категорії «мови програмування».

Саме це середовище обрано при розробці кваліфікаційної роботи, оскільки воно надає всю необхідну для розробника функціональність.

Для реалізації бази даних додатку було обрано СУБД MongoDB.

MongoDB - документоорієнтована система управління базами даних з відкритим вихідним кодом, яка не потребує опису схеми таблиць. Класифікована як NoSQL, використовує JSON-подібні документи і схему бази даних. Написана мовою C++.

MongoDB підтримує зберігання документів в JSON-подібному форматі, має досить гнучку мову для формування запитів, може створювати індекси для різних збережених атрибутів, ефективно забезпечує зберігання великих бінарних об'єктів, підтримує журналювання операцій змін і додавання даних в БД, може працювати відповідно до парадигми «Map/Reduce», підтримує реплікацію і побудову відмовостійких конфігурацій. У MongoDB підтримується автоматичне відновлення після збою і перенесення навантаження з вузла, який вийшов з ладу. Розширення кластера або перетворення одного сервера на кластер проводиться без зупинки роботи БД простим додаванням нових машин [4].

За рахунок мінімізації семантики для роботи з транзакціями з'являється можливість вирішення цілого ряду проблем, пов'язаних з нестачею продуктивності, причому горизонтальне масштабування стає простішим. Використовувана модель документів зберігання даних простіше кодується, простіше управляється. Можливість забезпечувати найкращу продуктивність повинна існувати паралельно з підтримкою більшої функціональності, ніж це дозволяє використання пар «ключ-значення».

MongoDB підходить для наступних застосувань:

– зберігання та реєстрація подій;
– системи управління документами і контентом;
– електронна комерція;
– ігри;
– дані моніторингу, датчиків;
– мобільні додатки;

– сховище операційних даних веб-сторінок (наприклад, зберігання коментарів, оцінок, профілів користувачів, сеанси користувачів).

З червня 2018 року в MongoDB існує підтримка транзакцій, що задовольняє вимогам ACID. У MongoDB є офіційні драйвери для основних мов програмування (C++, C#, Java, JavaScript, PHP та Python). Існує також велика кількість неофіційних або підтримуваних спільнотою драйверів для інших мов програмування і фреймворків.

Основним інтерфейсом до бази даних була командна оболонка «mongo». З версії MongoDB 3.2 в якості графічної оболонки поставляється «MongoDB Compass». Існують продукти і сторонні проєкти, які пропонують інструменти з GUI для адміністрування і перегляду даних [13].

При розробці кваліфікаційної роботи обрано саме СУБД MongoDB, оскільки вона надає всю необхідну функціональність, а також надає можливість безкоштовно розгорнути хмарну базу даних, що дозволяє не робити це локально на сервері.

2.4. Опис структури системи та алгоритмів її функціонування

2.4.1. UML проєктування

UML - мова графічного опису для об'єктного моделювання в області розробки програмного забезпечення, для моделювання бізнес-процесів, системного проєктування та відображення організаційних структур.

UML є мовою широкого профілю, це - відкритий стандарт, який використовує графічні позначення для створення абстрактної моделі системи, званої UML-моделлю. UML була створена для визначення, візуалізації, проєктування та документування, в основному, програмних систем.

Візуальне моделювання в UML можна уявити як певний процес порівневого спуску від найбільш загальної і абстрактної концептуальної моделі вихідної системи до логічної, а потім і до фізичної моделі відповідної програмної системи. Для досягнення цих цілей спочатку будується модель у

формі так званої діаграми варіантів використання, яка описує функціональне призначення системи.

2.4.1.1 Діаграма варіантів використання

Діаграма варіантів використання є вихідним концептуальним поданням або концептуальною моделлю системи в процесі її проєктування і розробки.

Розробка діаграми варіантів використання переслідує мети:

– визначити загальні межі і контекст модельованої предметної області на початкових етапах проєктування системи;

– сформулювати загальні вимоги до функціональної поведінки проєктованої системи;

– розробити вихідну концептуальну модель системи для її подальшої деталізації у формі логічних і фізичних моделей;

– підготувати вихідну документацію для взаємодії розробників системи з її замовниками і користувачами.

Суть даної діаграми полягає в наступному: проєктована система представляється у вигляді безлічі сутностей або акторів, що взаємодіють з системою за допомогою варіантів використання. При цьому актором називається будь-яка сутність, що взаємодіє з системою ззовні. Це може бути людина, технічний пристрій, програма або будь-яка інша система, яка може служити джерелом впливу на модельовану систему так, як визначить сам розробник. У свою чергу, варіант використання служить для опису сервісів, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, які керують системою при діалозі з актором.

В програмній системі кваліфікаційної роботи виділено дві ролі - «Адміністратор» та «Користувач з правом перегляду». Вони зв’язані відношенням «узагальнення», тобто актор «Адміністратор» має всі ті ж права, що і актор «Користувач з правом перегляду».

На рис. 2.1 наведено діаграму варіантів використання.

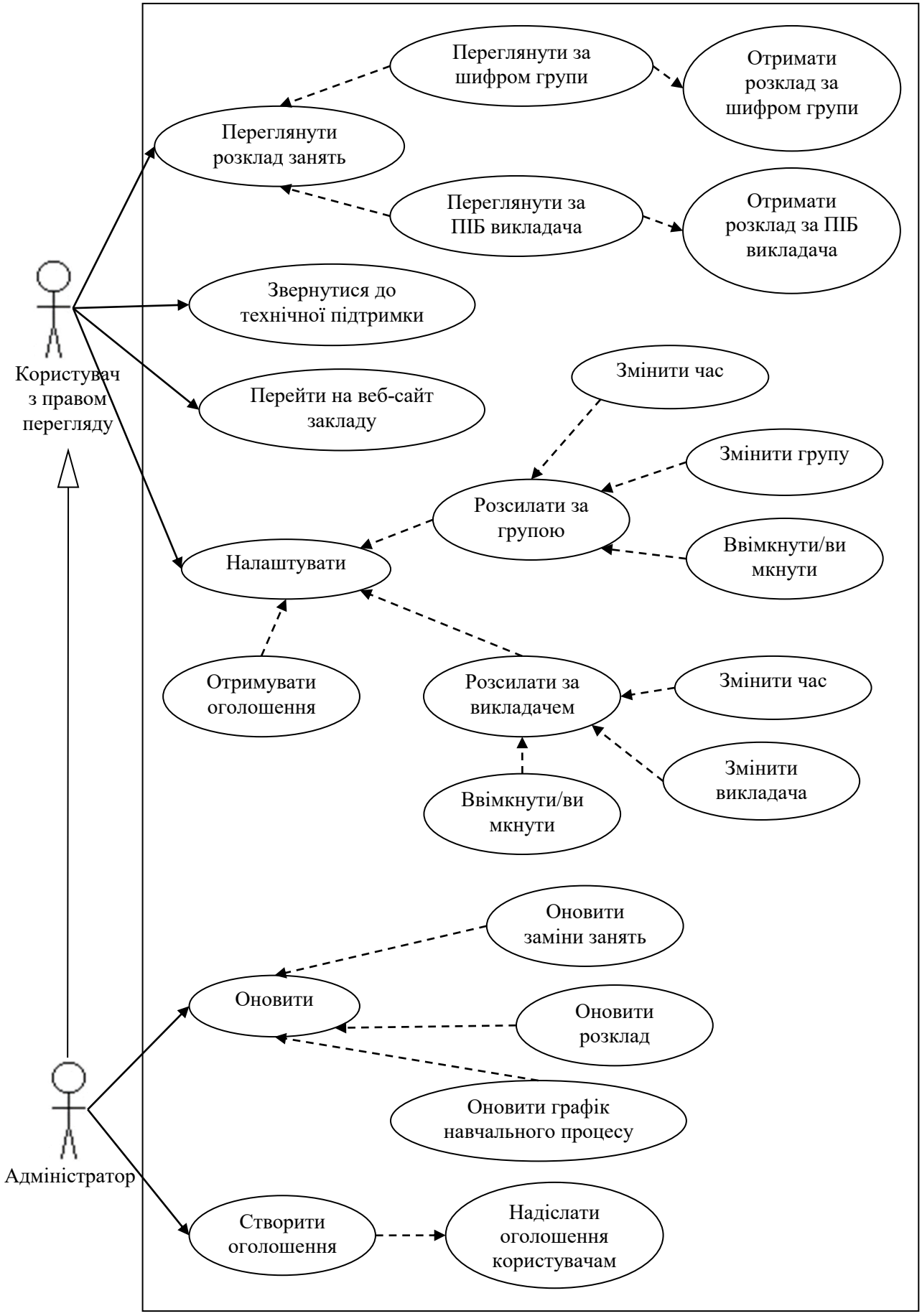


Рис. 2.1. Діаграма варіантів використання

Сценарій «Переглянути розклад занять» передбачає вибір користувачем одного з пунктів меню:

– «Переглянути за шифром групи»;

– «Переглянути за ПІБ викладача».

Сценарій «Переглянути за шифром групи» передбачає:

1 Введення користувачем шифру групи.

2 Пошук за введеним шифром існуючих груп. Є можливість часткового введення шифру. Якщо збігів не знайдено, користувач отримає відповідне повідомлення. Якщо знайдено декілька збігів, користувач може обрати, яку саме групу він мав на увазі.

3 Вибір дня, на який користувач бажає отримати розклад.

4 Виклик сценарія «Отримати розклад за шифром групи» для відповідної групи.

Сценарій «Отримати розклад за шифром групи» передбачає:

1 Пошук інформації про групу в графіку навчального процесу в обраний день. Якщо відомостей не знайдено, користувач отримає відповідне повідомлення.

2 Пошук інформації про групу в розкладі занять за обраний день тижня з урахуванням чисельника або знаменника. Якщо відомостей не знайдено, користувач отримає відповідне повідомлення.

3 Пошук інформації про групу в замінах занять за обраний день. Якщо відомостей не знайдено, користувач отримає відповідне повідомлення.

4 Отримання користувачем всієї зібраної інформації.

Сценарій «Переглянути за ПІБ викладача» передбачає:

1 Введення користувачем ПІБ викладача.

2 Пошук викладачів за введеним ПІБ. Є можливість часткового введення. Якщо збігів не знайдено, користувач отримає відповідне повідомлення. Якщо знайдено декілька збігів, користувач може обрати, якого саме викладача він мав на увазі.

3 Вибір дня, на який користувач бажає отримати розклад.

4 Виклик сценарію «Отримати розклад за ПІБ викладача» для відповідного викладача.

Сценарій «Отримати розклад за ПІБ викладача» передбачає:

1 Пошук інформації про розклад викладача за обраний день тижня з урахуванням чисельника або знаменника. Якщо відомостей не знайдено, користувач отримає відповідне повідомлення.

2 Пошук інформації про викладача в замінах занять за обраний день. Якщо відомостей не знайдено, користувач отримає відповідне повідомлення.

3 Отримання користувачем всієї зібраної інформації.

Сценарій «Звернутися до технічної підтримки» передбачає перенаправлення користувача в діалог з оператором технічної підтримки, якщо оператора вказано в файлі налаштувань.

Сценарій «Перейти на веб-сайт закладу» передбачає перенаправлення користувача на веб-сайт закладу, якщо його адреса вказана в файлі налаштувань.

Сценарій «Налаштувати» передбачає вибір користувачем одного з пунктів меню:

– «Розсилати за групою»;

– «Розсилати за викладачем»;

– «Отримувати оголошення».

Сценарій «Розсилати за групою» передбачає вибір користувачем одного з пунктів меню:

– «Змінити час»;

– «Змінити групу»;

– «Ввімкнути/вимкнути».

Сценарій «Змінити час» передбачає вибір користувачем часу для отримання щоденної розсилки.

Сценарій «Змінити групу» передбачає:

1 Введення користувачем шифру групи.

2 Пошук за введеним шифром існуючих груп. Є можливість часткового введення шифру. Якщо збігів не знайдено, користувач отримає відповідне повідомлення. Якщо знайдено декілька збігів, користувач може обрати, яку саме групу він мав на увазі.

3 Якщо групу було знайдено, користувач отримає повідомлення про встановлення нової групи.

Сценарій «Ввімкнути/вимкнути» передбачає відповідно додавання або видалення розсилки з планувальника, в залежності від поточного стану.

Сценарій «Розсилати за викладачем» передбачає вибір користувачем одного з пунктів меню:

– «Змінити час»;

– «Змінити викладача»;

– «Ввімкнути/вимкнути».

Сценарій «Змінити викладача» передбачає:

1 Введення користувачем ПІБ викладача.

2 Пошук викладачів за введеним ПІБ. Є можливість часткового введення. Якщо збігів не знайдено, користувач отримає відповідне повідомлення. Якщо знайдено декілька збігів, користувач може обрати, якого саме викладача він мав на увазі.

3 Якщо викладача було знайдено, користувач отримає повідомлення про встановлення нового викладача.

Роботу сценаріїв «Змінити час» та «Ввімкнути/вимкнути» зазначено раніше.

Сценарій «Отримувати оголошення» передбачає включення або виключення користувача зі списку користувачів, що отримують оголошення від адміністратора.

Сценарій «Оновити» передбачає вибір користувачем одного з пунктів меню:

– «Оновити заміни занять»;

– «Оновити розклад»;

– «Оновити графік навчального процесу».

Сценарій «Оновити заміни занять» передбачає:

1 Завантаження користувачем Excel файлу з замінами занять.

2 Підтвердження оновлення замін.

3 Завантаження всіх замін, дата яких не менша поточної.

Сценарій «Оновити розклад» передбачає:

1 Завантаження користувачем Excel файлу з розкладом занять.

2 Введення користувачем назви листа, на якому міститься розклад. Є можливість обрати запропонований варіант.

3 Перевірку, чи існує вказаний лист в завантаженому файлі. Якщо листа немає, то користувач отримає відповідне повідомлення.

4 Підтвердження оновлення розкладу занять.

5 Видалення старого розкладу та завантаження нового.

Сценарій «Оновити графік навчального процесу» передбачає:

1 Завантаження користувачем Excel файлу з графіком навчального процесу.

2 Введення користувачем назви листа, на якому міститься графік навчального процесу. Є можливість обрати запропонований варіант.

3 Перевірку, чи існує вказаний лист в завантаженому файлі. Якщо листа немає, то користувач отримає відповідне повідомлення.

4 Підтвердження оновлення графіку навчального процесу.

5 Видалення старого графіка та завантаження нового.

Сценарій «Створити оголошення» передбачає:

1 Введення користувачем тексту оголошення.

2 Підтвердження надсилання введеного оголошення. При позитивній відповіді викликається сценарій «Надіслати оголошення користувачам».

Сценарій «Надіслати оголошення користувачам» передбачає:

1 Отримання списку користувачів, які хоч раз користувались ботом.

2 Видалення зі списку користувачів, які відмовились від отримання оголошень.

3 Надсилання оголошення кожному користувачеві зі списку.

2.4.1.2 Діаграма станів

Діаграми станів використовуються для опису поведінки екземпляра сутності (класу, об'єкта, компонента, вузла або системи в цілому). Поведінка моделюється через опис можливих станів екземпляра сутності і переходів між ними протягом його життєвого циклу, починаючи від створення і закінчуючи знищенням. Діаграма станів - це зв'язний орієнтований граф, вершинами якого є стани, а дуги служать для позначення переходів зі стану в стан.

Під станом розуміється ситуація в ході життя екземпляра сутності, коли ця ситуація задовольняє деякій умові, екземпляр виконує деякі операції або чекає настання деякої події. Наприклад, для об'єкта його стан може бути задано у вигляді набору конкретних значень атрибутів, при цьому зміна цих значень буде приводити до зміни стану об'єкта, що моделюється.

В UML розрізняють два види операцій: дія і діяльність. Дія - це атомарна операція, виконання якої не може бути перервано, яка веде до зміни стану або повертає значення. Діяльність - це складова (неатомарна) операція, реалізована екземпляром в конкретному стані, виконання якої може бути перервано.

Перехід - відношення між двома станами, що показує можливий шлях зміни стану екземпляра сутності. Вважається, що в стані екземпляр сутності знаходиться тривалий час, а перехід виконується миттєво.

Перехід відображається у вигляді односпрямованої асоціації між двома станами. При зміні станів кажуть, що перехід спрацьовує. До спрацьовування переходу екземпляр сутності знаходиться в стані, званому вихідним, а після його спрацьовування - в цільовому.

Розрізняють два види переходів - нетригерний та тригерний. Перехід першого виду, званий також переходом по завершенні, спрацьовує неявно, коли всі основні операції в початковому стані успішно завершують свою роботу. Для настання тригерного переходу необхідний наступ деякої події.

На рис. 2.2 зображено діаграму станів розробленого додатку.

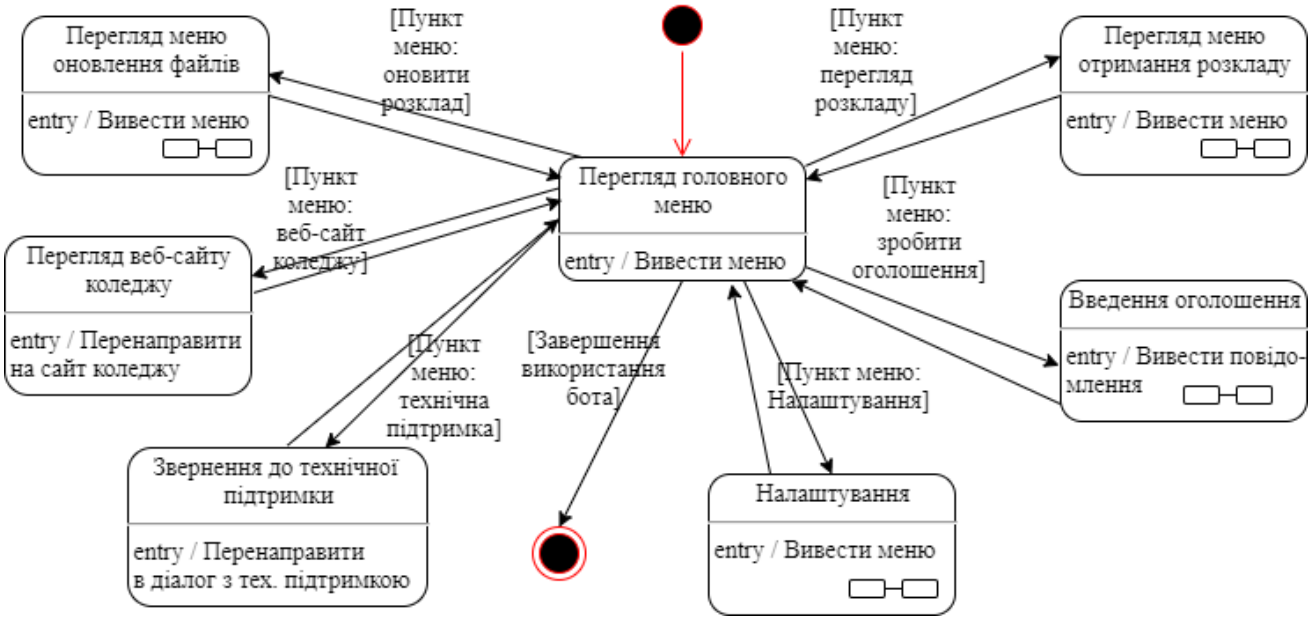


Рис. 2.2. Діаграма станів додатку

На рис. 2.3 зображено деталізацію стану «Перегляд меню отримання розкладу».

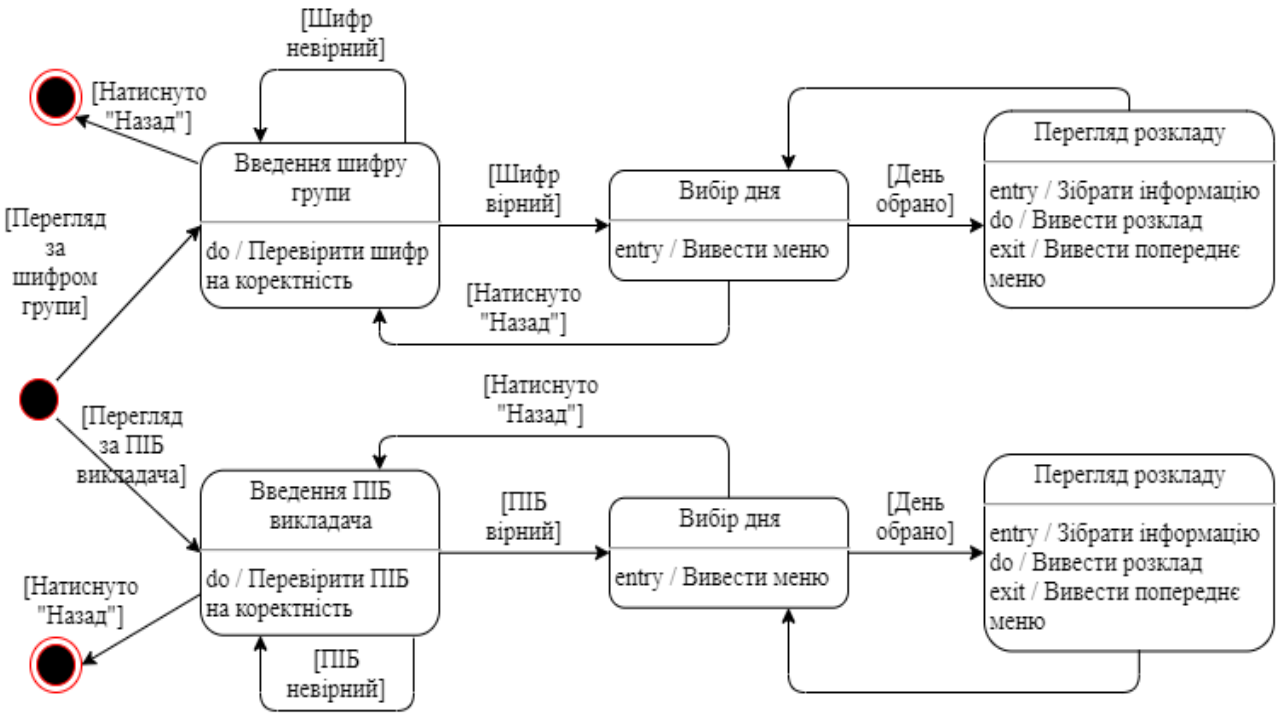


Рис. 2.3. Деталізація стану «Перегляд меню отримання розкладу»

На рис. 2.4 зображено деталізацію стану «Введення оголошення».

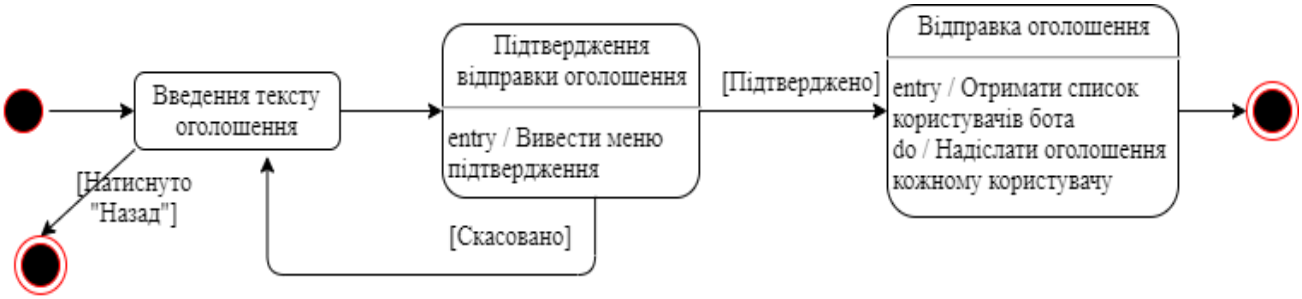


Рис. 2.4. Деталізація стану «Введення оголошення»

На рис. 2.5 зображено деталізацію стану «Перегляд меню оновлення файлів».

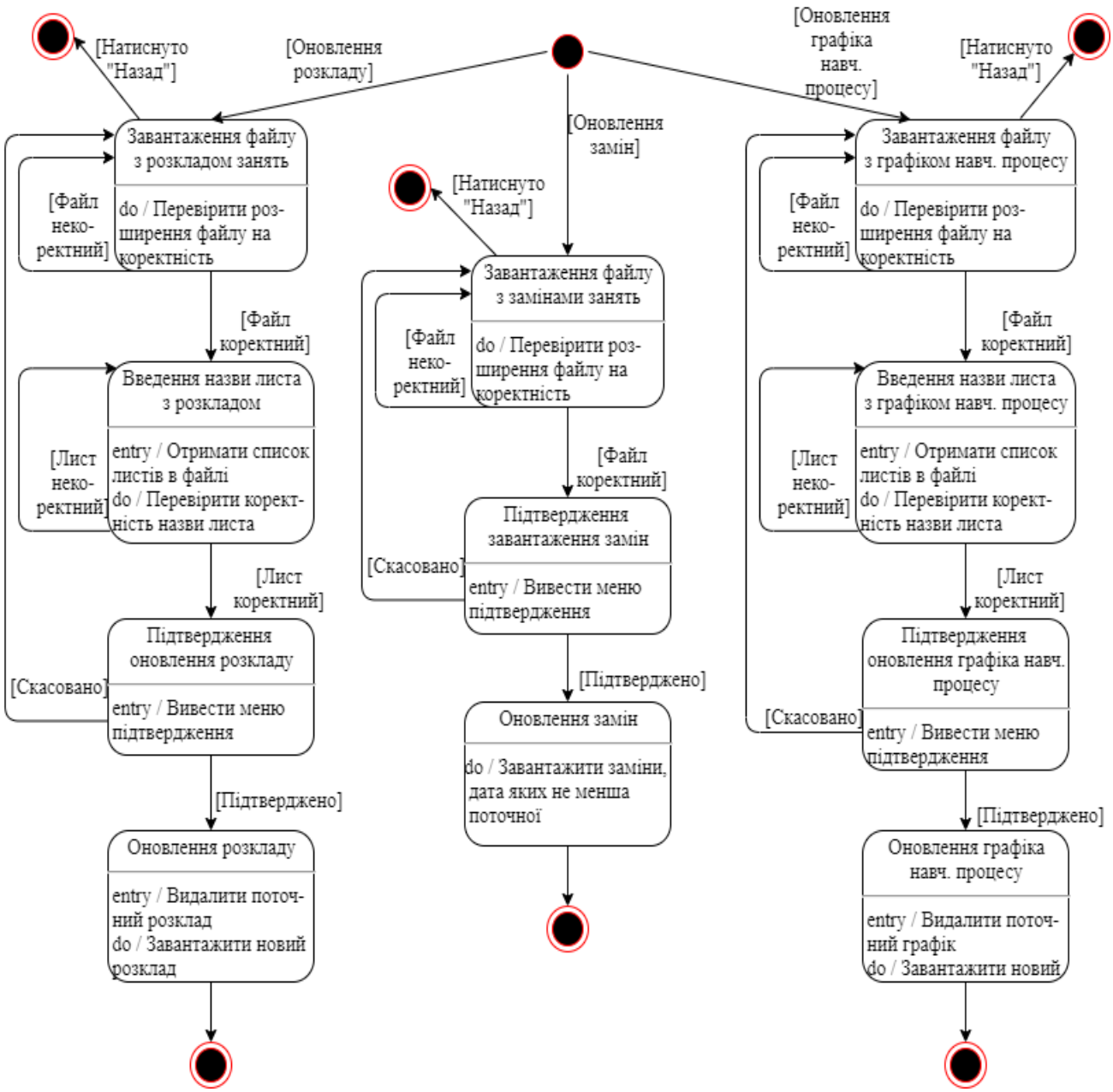


Рис. 2.5. Деталізація стану «Перегляд меню оновлення файлів»

На рис. 2.6 зображено деталізацію стану «Налаштування».

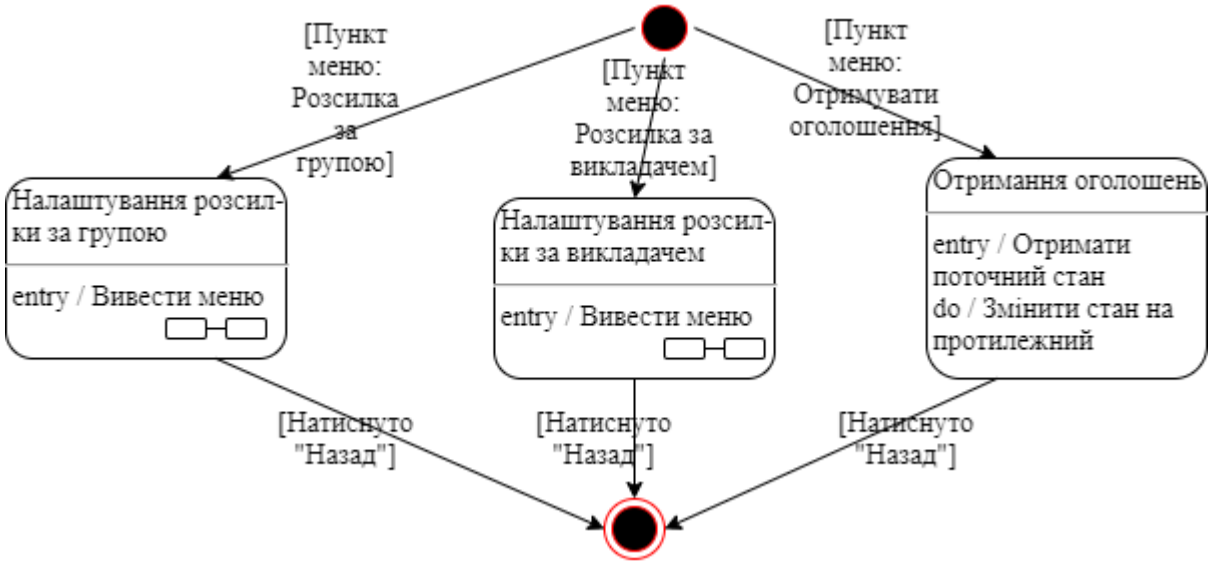


Рис. 2.6. Деталізація стану «Налаштування»

На рис. 2.7 зображено деталізацію стану «Налаштування розсилки за групою».

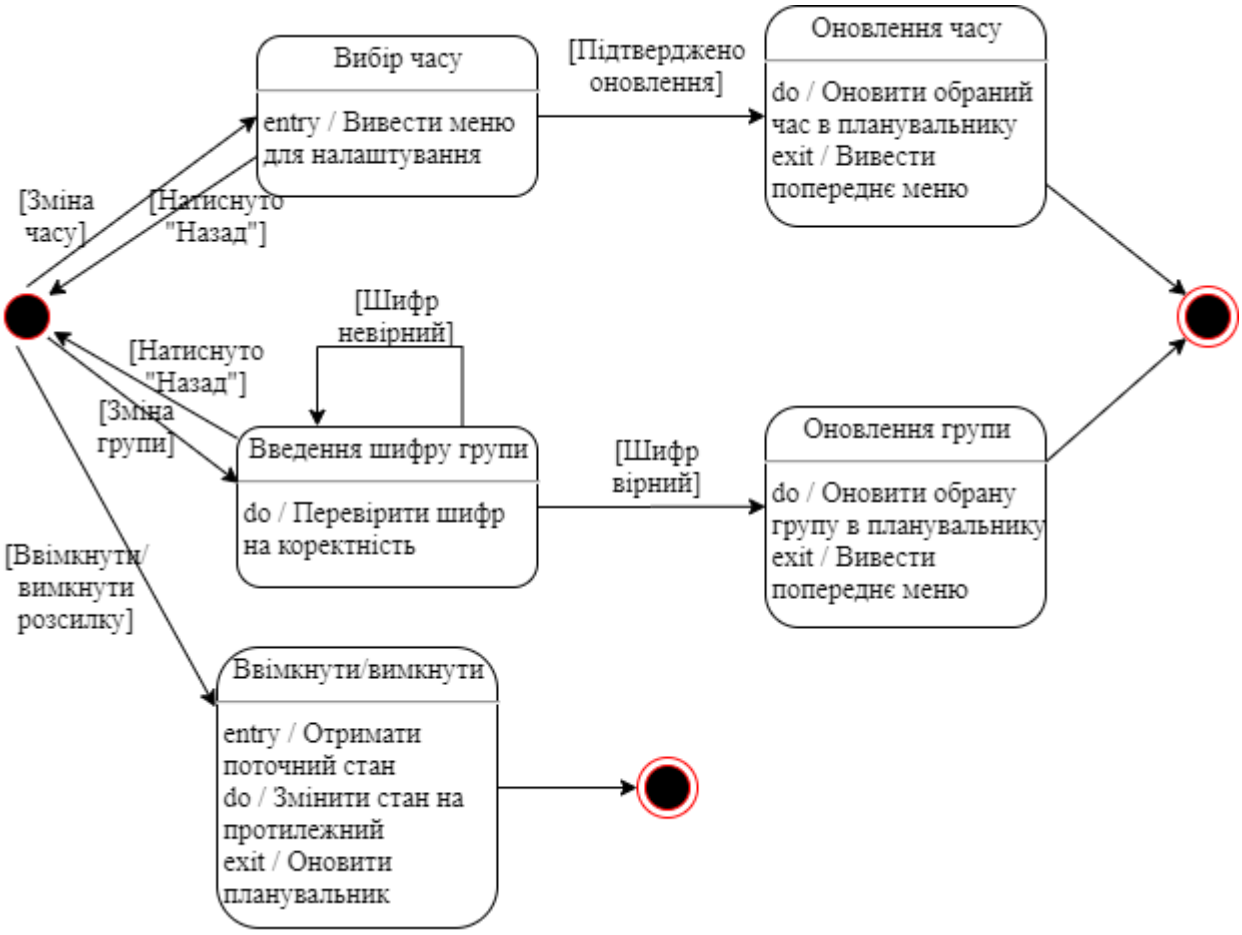


Рис. 2.7. Деталізація стану «Налаштування розсилки за групою»

На рис. 2.8 зображено деталізацію стану «Налаштування розсилки за викладачем».

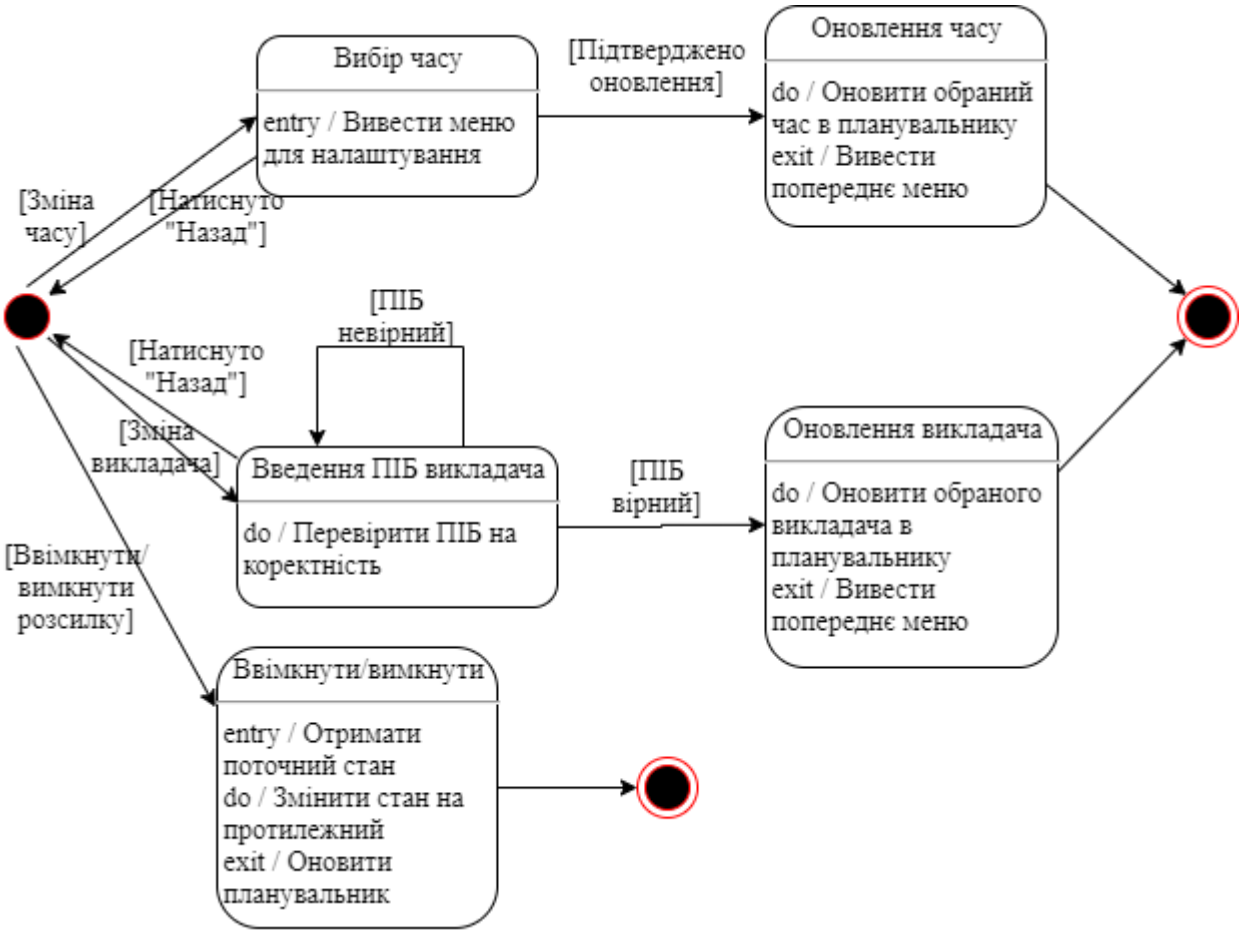


Рис. 2.8. Деталізація стану «Налаштування розсилки за викладачем»

2.4.2. Налаштування чат-бота

Для встановлення налаштувань, які необхідні для роботи чат-бота, використовується файл «settings.xml», розташований в кореневій директорії проєкту.

XML - стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками, зокрема, через інтернет. XML-документ складається з текстових знаків, і придатний до читання людиною.

Стандарт XML визначає набір базових лексичних та синтаксичних правил

для побудови мови описання інформації шляхом застосування простих тегів. Цей формат достатньо гнучкий для того, аби бути придатним для застосування в різних галузях.

XML розроблялася як мова з простим формальним синтаксисом, зручним для створення і обробки документів програмами і одночасно зручним для читання і створення документів людиною, з підкресленням націленості на використання в інтернеті. Мова називається розширюваною, оскільки нею не фіксується розмітка, яка використовується в документах - розробник вільний створити розмітку відповідно до потреб конкретної області, будучи обмеженим лише синтаксичними правилами мови.

Розширення XML - це конкретна граматика, створена на базі XML і представлена словником тегів і їх атрибутів, а також набором правил, що визначають які атрибути і елементи можуть входити до складу інших елементів. Поєднання простого формального синтаксису, зручності для людини, розширюваності, а також базування на кодуваннях «юнікод» для подання змісту документів привело до широкого використання XML.

Елемент є поняттям логічної структури документа. Кожен документ містить один або кілька елементів. Межі елементів представлені початковим і кінцевим тегами. Ім'я елемента в початковому і кінцевому тегах елемента має збігатися.

Тег - конструкція розмітки, яка містить ім'я елемента.

На рис. 2.9 наведено структуру XML-документа з налаштуванням бота.

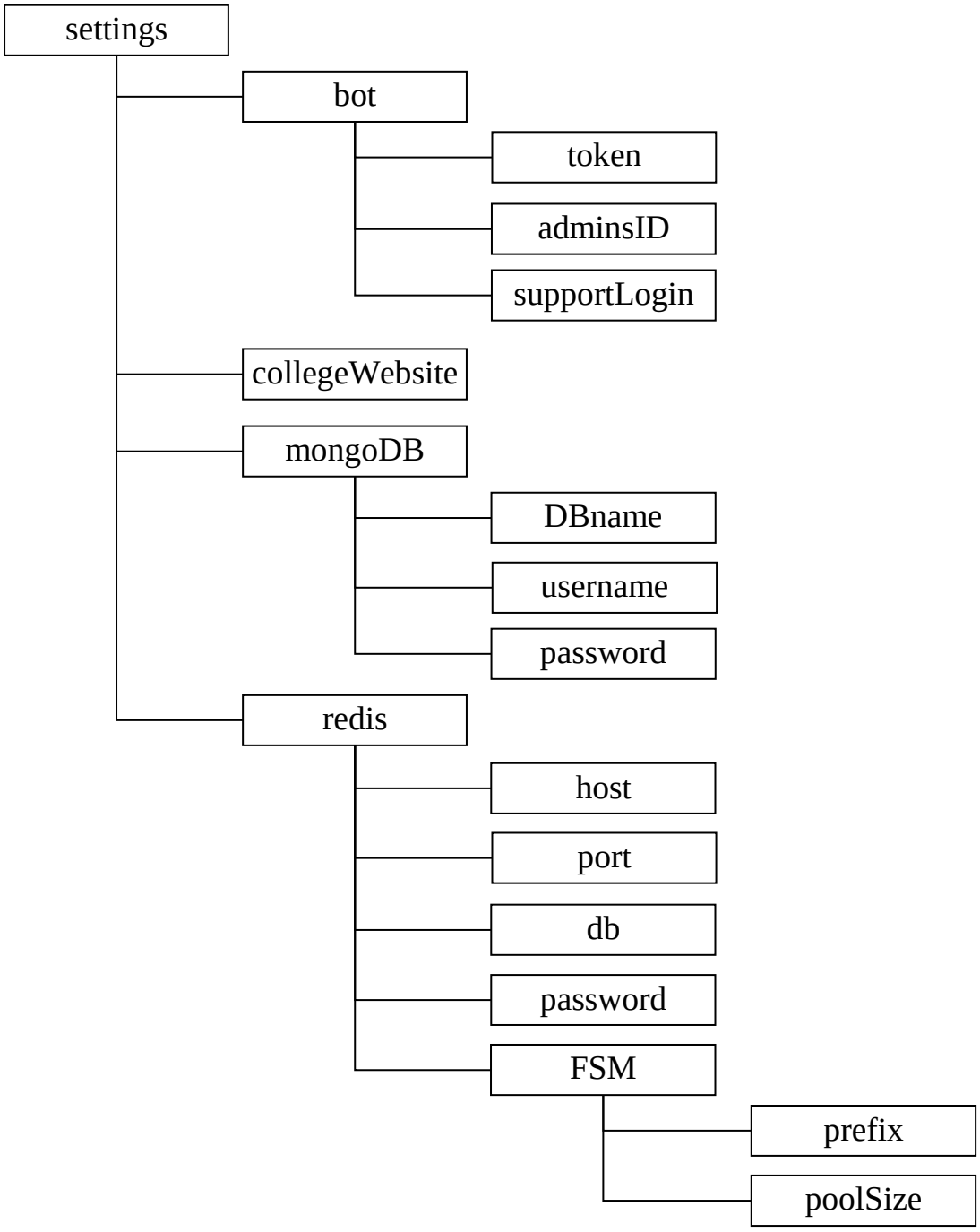


Рис. 2.9. Структура XML-документа з налаштуванням бота

В таблиці 2.1 наведено призначення тегів файлу налаштувань бота.

Таблиця 2.1

Призначення тегів файлу налаштувань бота

Тег	Призначення	Обов’язковий
bot.token	Ключ для доступу до чат-бота	+
bot.AdminsID	Список ID облікових записів Telegram, які будуть наділені правами адміністратора	-
bot.supportLogin	Ім’я користувача облікового запису Telegram, який виступатиме в якості технічної підтримки	-
collegeWebsite	Посилання на головну сторінку сайту закладу	-
mongoDB	Містить теги, призначені для налаштування СУБД MongoDB	+
mongoDB.DBname	Назва бази даних	+
mongoDB.username	Ім’я користувача для доступу до бази даних	+
mongoDB.password	Пароль для доступу до бази даних	+
redis	Містить теги, призначені для налаштування сервера Redis	+
redis.host	Адреса сервера	+
redis.port	Порт сервера	+
redis.password	Пароль сервера	-
redis.db	Номер бази даних на сервері	+
FSM	Містить теги, призначені для роботи кінцевих автоматів	+
FSM.poolsize	Розмір пулу з’єднань з базою даних	+
FSM.prefix	Префікс для даних, що зберігатимуться	+

2.4.3. Проєктування користувацького інтерфейсу

За допомогою графічного інтерфейсу користувач взаємодіє з чат-ботом, тому його проєктуванню було приділено значну увагу. Графічний інтерфейс було спроєктовано таким чином, щоб мінімізувати необхідність користувачеві вводити інформацію власноруч, тим самим зменшуючи можливість здійснення помилки.

Telegram Bot API надає можливість використовувати два види клавіатур з кнопками - ReplyKeyboardMarkup та InlineKeyboardMarkup.

ReplyKeyboardMarkup складається з кнопок, які представляють собою шаблони відповідей. Ця клавіатура може бути корисною, коли потрібно виключити необхідність вводити відповідь самостійно, надавши користувачеві декілька варіантів на вибір. Приклади застосування такого виду клавіатури зображено на рис. 2.10 та 2.11.

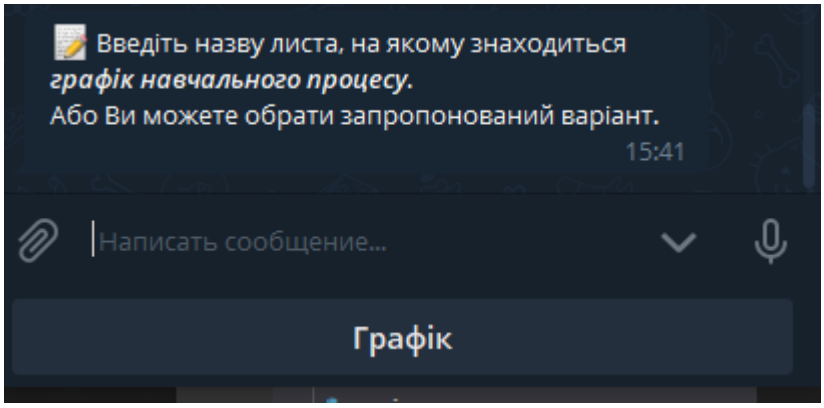


Рис. 2.10. Можливість обрати запропоновану назву листа Excel файлу

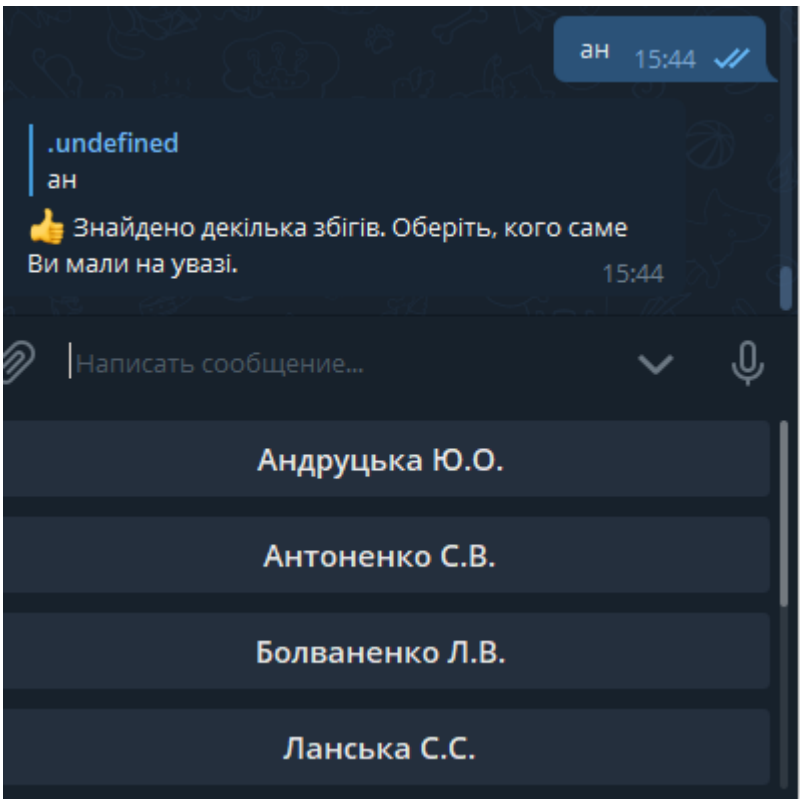


Рис. 2.11. Можливість обрати викладача за частково введеним ПІБ

Клавіатура InlineKeyboardMarkup складається з кнопок, які можуть виконувати більш складні дії. Кнопки такої клавіатури надають користувачам можливість:

– переходити за URL-адресою;
– отримувати меню, кожен пункт якого має відповідну задачу;
– отримувати відеогру в діалог;
– отримувати рахунок для сплати.

Приклад застосування такого виду клавіатури зображено на рис. 2.12.

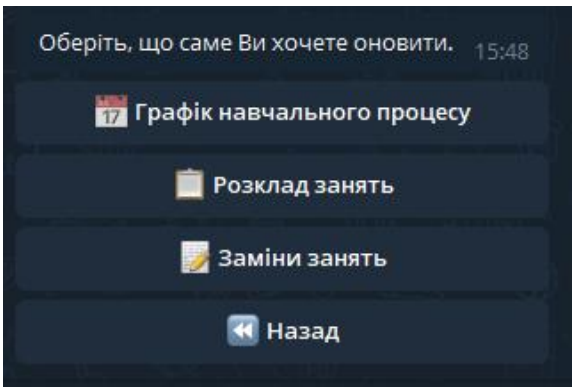


Рис. 2.12. Меню оновлення файлів

Для полегшення введення команд чат-бот пропонує доступні команди після введення символу «/». На рис. 2.13 зображено запропоновані команди.

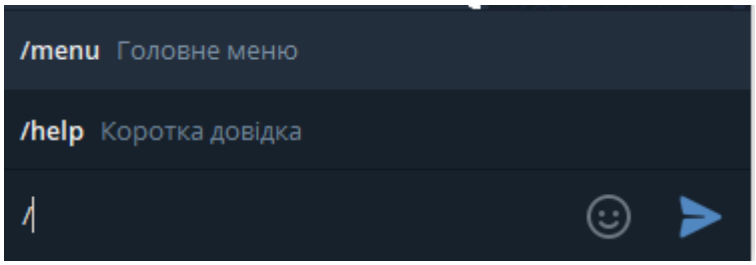


Рис. 2.13. Можливість вибору команди

2.4.4. Взаємозв’язок програмних модулів

Модулі та пакети значно спрощують роботу програміста. Класи, об'єкти, функції і константи, якими доводиться часто користуватися можна упакувати в модуль, і в подальшому завантажувати його в свої програми при необхідності. Пакети дозволяють формувати простір імен для роботи з модулями.

Під модулем в Python розуміється файл з розширенням «.py». Модулі призначені для того, щоб в них зберігати часто використовувані функції, класи, константи тощо.

Пакет в Python - це каталог, що включає в себе інші каталоги та модулі, але при цьому додатково містить файл «__init__.py». Пакети використовуються для формування простору імен, що дозволяє працювати з модулями через

вказівку рівня вкладеності.

В таблиці 2.2 наведено призначення пакетів програмного додатку.

Таблиця 2.2

Призначення пакетів програмного додатку

Назва	Призначення
data_parsers	Пакет містить програмні модулі, призначені для обробки даних, що подаються в Excel документах, а саме: розклад занять, графік навчального процесу та заміни занять.
database	В пакеті знаходяться модулі, які реалізують роботу з базою даних.
handlers	Пакет містить програмні модулі, в яких реалізовано обробники подій, а саме: вибір користувачем пунктів меню, введення інформації тощо.
keyboards	В пакеті знаходяться клавіатури, що виводяться користувачеві в якості меню та підказок.

На рис. 2.14 зображено схему взаємозв’язку пакетів програмного додатку.

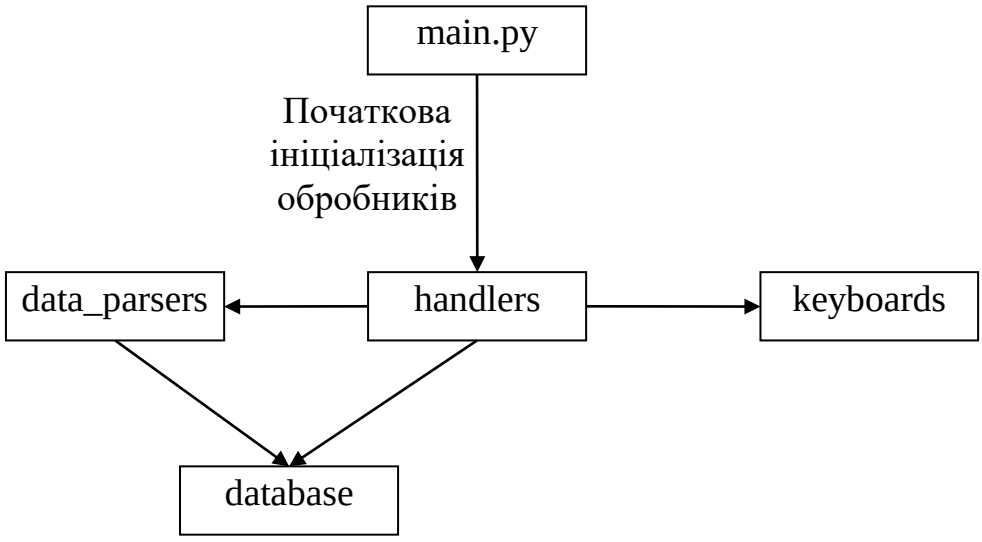


Рис. 2.14. Схема взаємозв’язку пакетів програмного додатку

2.4.5. Ключові алгоритми програмного додатку

До ключових алгоритмів програмного додатку належать:

1 Алгоритм завантаження графіка навчального процесу. Алгоритм реалізовано в програмному модулі «graph_file_handler.py», він працює наступним чином:

– здійснюється пошук шапки таблиці з графіком навчального процесу. В разі, якщо її не знайдено, виведеться повідомлення про помилку;

– виконується зчитування інформації про навчальний процес (теоретичне навчання, практика тощо) та відповідного оформлення клітинок Excel документу;

– отримується список груп;

– отримується список дат в UNIX форматі;

– видаляється старий графік навчального процесу;

– для кожної групи на відповідну дату встановлюється відомість про навчальний процес;

– після збору інформації по одній групі, дані завантажуються в базу даних.

2 Алгоритм завантаження розкладу занять. Алгоритм реалізовано в програмному модулі «schedule_file_handler.py», він працює наступним чином:

– здійснюється пошук шапки таблиці з розкладом занять. В разі, якщо її не знайдено, виведеться повідомлення про помилку;

– отримується список груп;

– знаходяться номери колонок «дні» та «пара». Якщо їх немає, виведеться повідомлення про помилку;

– для кожної групи зчитуються дані про розклад за знаменником та чисельником окремо.

3 Алгоритм завантаження замін занять. Алгоритм реалізовано в програмному модулі «changes_file_handler.py», він працює наступним чином:

– виконується пошук листів Excel файлу, в назвах яких міститься дата, не менша поточної;

– для кожного листа знаходиться шапка таблиці;

– виконується завантаження кожної заміни в базу даних. Якщо якесь значення пусте, запис не завантажиться. Якщо запис з такою датою, групою та номером пари вже існує в базі даних, він оновлюється.

Реалізацію вищезазначених алгоритмів наведено в додатку А.

2.5. Обґрунтування та організація вхідних та вихідних даних програми

Вхідними даними програми виступає інформація щодо назви групи, ПІБ викладача, дата для перегляду розкладу та текст повідомлення для розсилки.

Вихідними даними роботи є:

– розклад занять за шифром групи

– розклад занять за ПІБ викладача.

2.6. Опис роботи розробленої системи

2.6.1. Використані технічні засоби

Для розробки даного програмного забезпечення використано ПК з наступними основними характеристиками:

– операційна система: Windows 10;

– двоядерний процесор з частотою 3 ГГц;

– вільний об’єм пам’яті на жорсткому диску: 128 Мб;

– об’єм оперативної пам’яті: 2 Гб;

– доступ до інтернету.

2.6.2. Використані програмні засоби

Для реалізації додатку кваліфікаційної роботи обрано мову програмування Python. Для реалізації бази даних додатку було обрано СУБД MongoDB. В якості середовища розробки обрано Microsoft Visual Studio Code.

2.6.3. Виклик та завантаження програми

Для того, щоб розпочати використання чат-боту, необхідно мати встановлений клієнт месенджеру Telegram на своєму пристрої та створити в ньому обліковий запис, якщо його немає. Потім необхідно в рядку пошуку набрати «ntu_122_17_1_bot» та натиснути кнопку «Старт».

На рис. 2.15 зображено інформацію про бота.

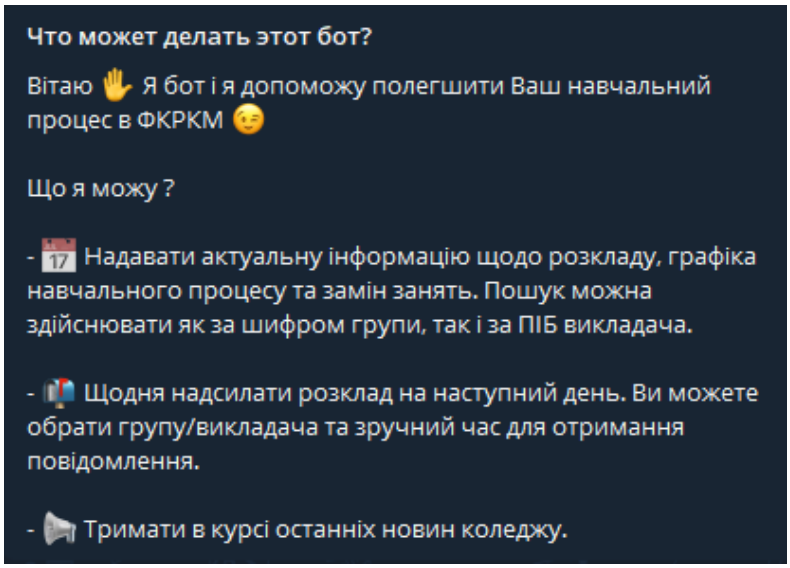


Рис. 2.15. Інформація про бота

2.6.4. Опис інтерфейсу користувача

Одразу після запуску виводиться головне меню. На рис. 2.16 зображено головне меню, призначене для користувача з правами адміністратора.

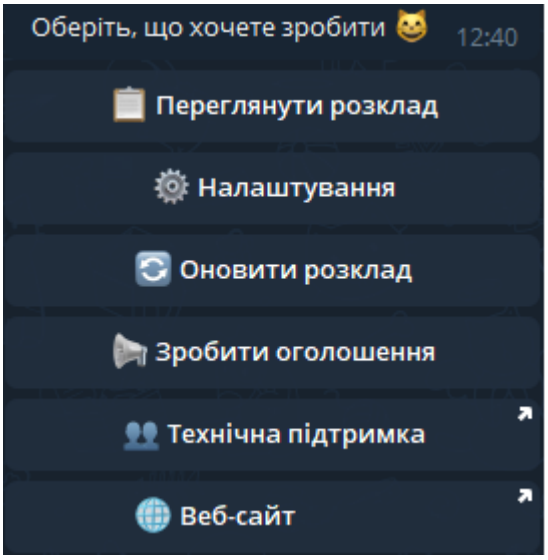


Рис. 2.16. Головне меню бота для адміністратора

На рис. 2.17 зображено головне меню, призначене для користувача з правом перегляду.

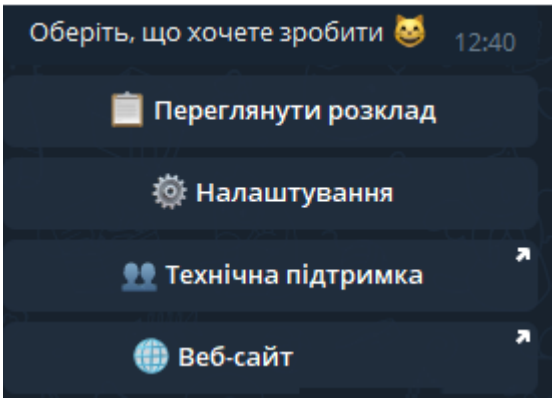


Рис. 2.17. Головне меню бота для користувача з правом перегляду

Якщо обрати пункт «Переглянути розклад», виведеться меню, зображене на рис. 2.18. В ньому необхідно обрати параметр, за яким буде виведено розклад.

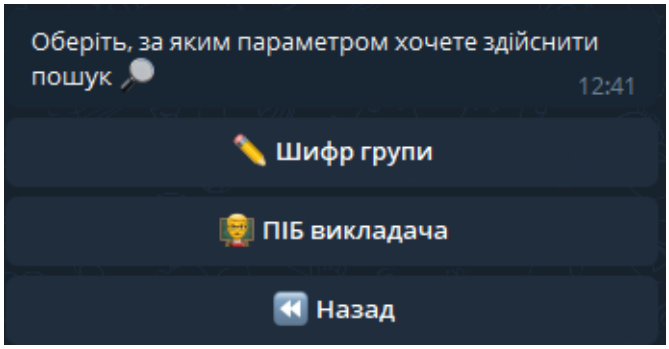


Рис. 2.18. Меню вибору параметра для перегляду розкладу

Обравши «ПІБ викладача», можна обрати викладача, розклад якого користувач переглядав в останній раз, а можна ввести знову. Підтримується пошук за частково введеним ПІБ, приклад наведено на рис. 2.19.

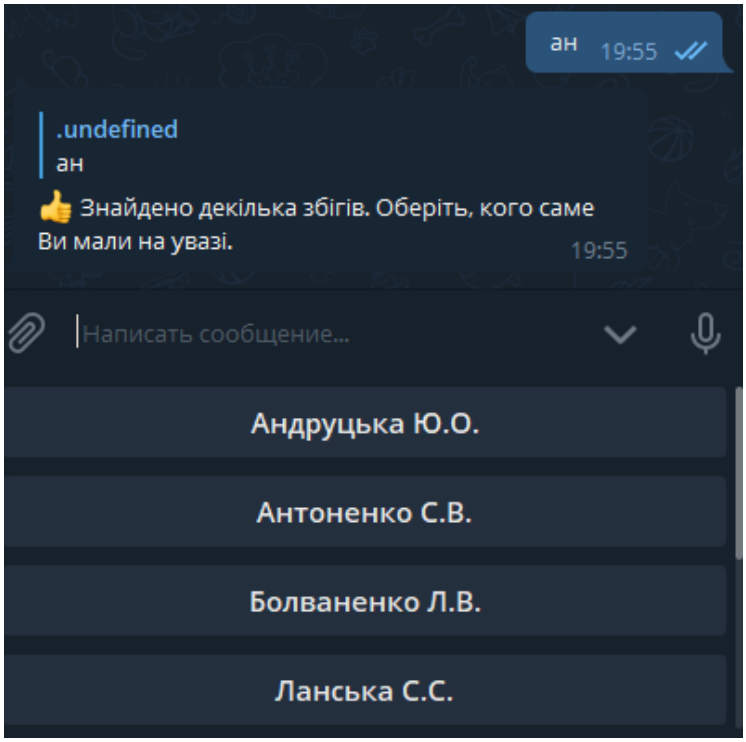


Рис. 2.19. Пошук за частково введеним ПІБ

Після введення коректного ПІБ викладача, виведеться меню, зображене на рис. 2.20.

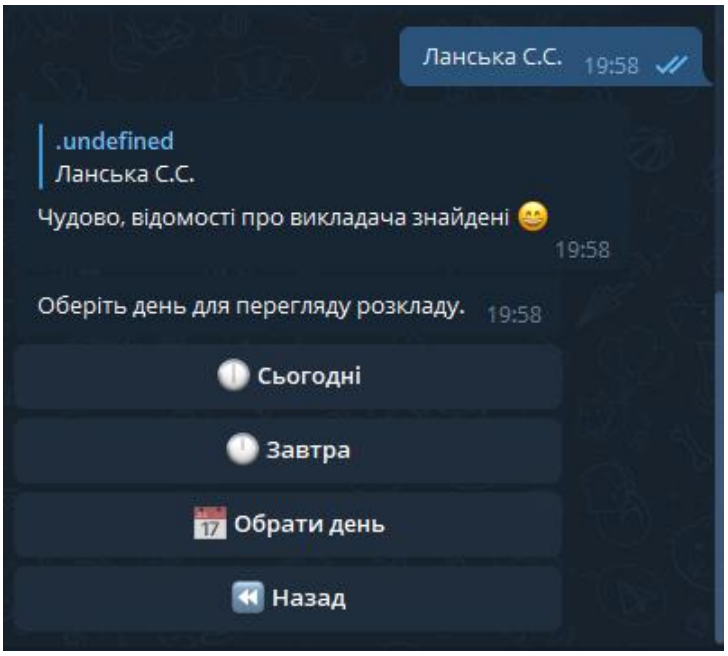


Рис. 2.20. Меню вибору дня для перегляду розкладу

Натиснувши пункт «Обрати день», користувач отримає меню, зображене на рис. 2.21. Необхідно обрати дату за допомогою стрілок та натиснути кнопку «Підтвердити».

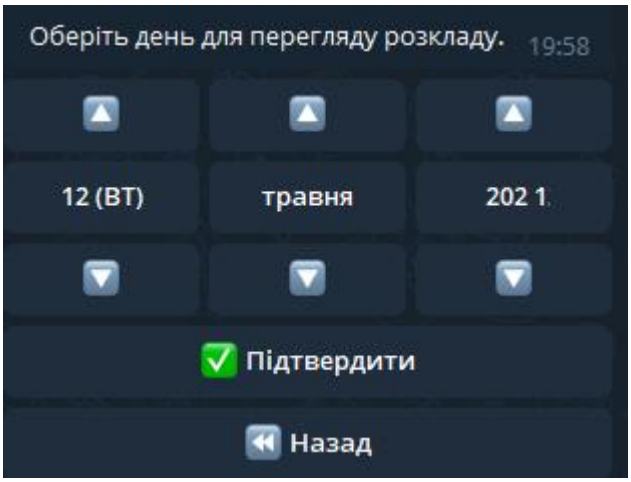


Рис. 2.21. Меню вибору дати

На рис. 2.22 зображено отриману інформацію про розклад викладача.

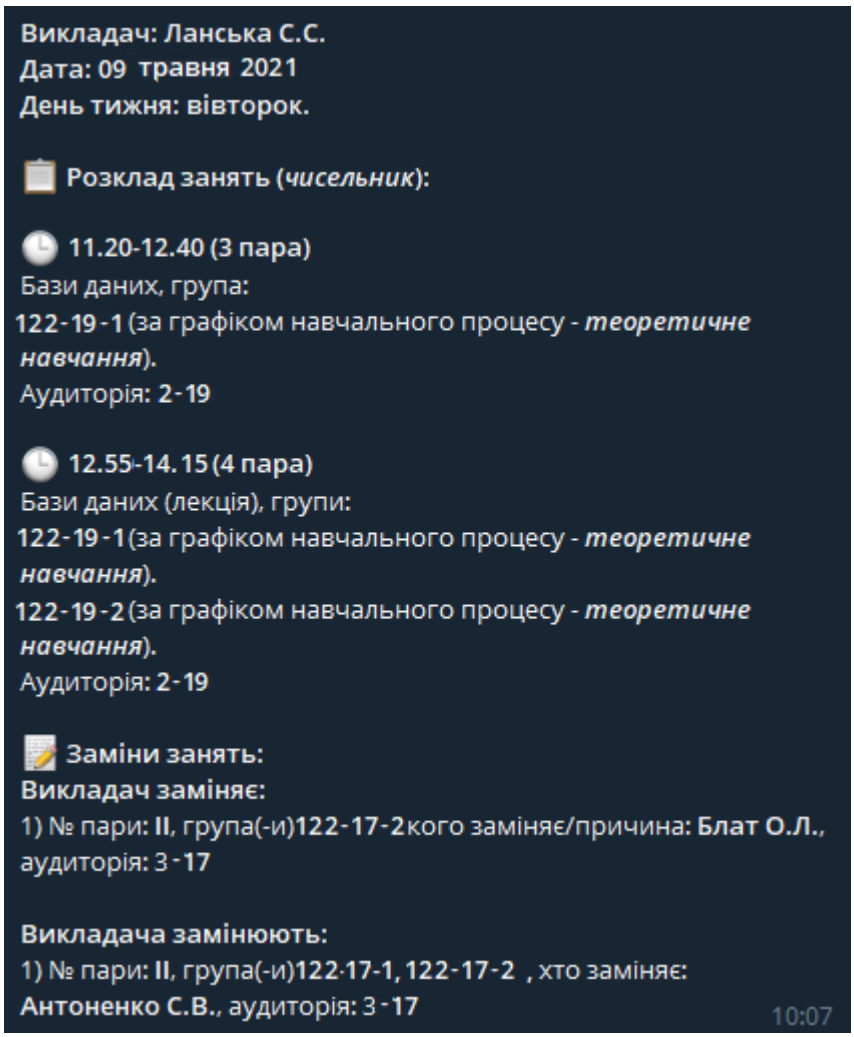


Рис. 2.22. Отримання розкладу занять за ПІБ викладача

Перегляд розкладу за шифром групи відбувається схожим чином. На рис. 2.23 зображено отримання розкладу за шифром групи.

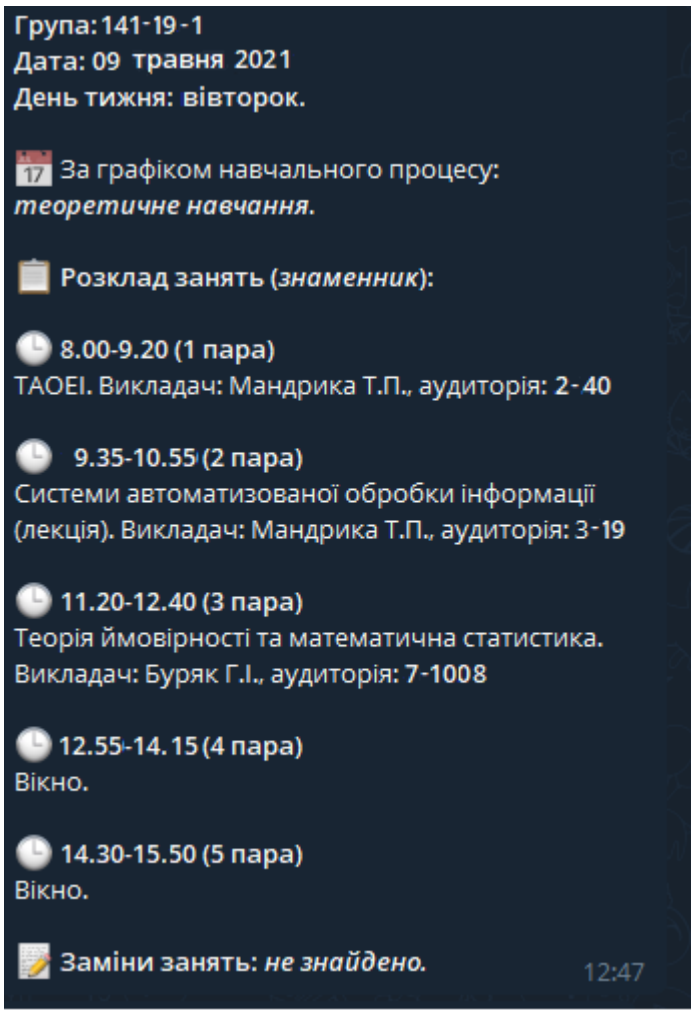


Рис. 2.23. Отримання розкладу за шифром групи

При виборі адміністратором пункту головного меню «Оновити розклад» виведеться меню, зображене на рис. 2.24.

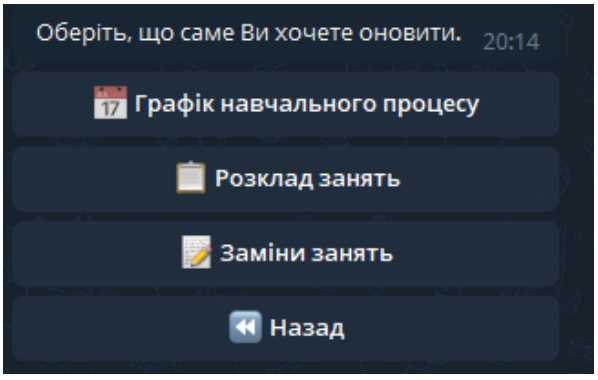


Рис. 2.24. Меню оновлення файлів

При виборі пункту меню «Графік навчального процесу» користувач отримає повідомлення, зображене на рис. 2.25.

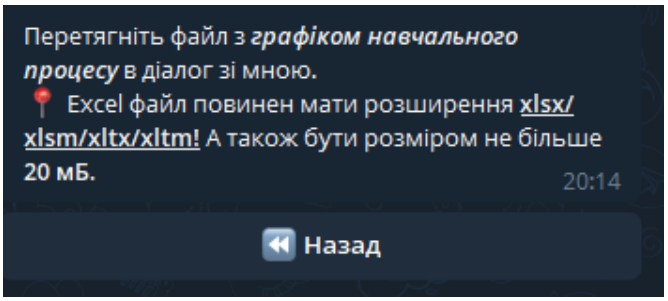


Рис. 2.25. Повідомлення про завантаження графіка навчального процесу

Після завантаження Excel файлу буде виведено повідомлення, зображене на рис. 2.26.

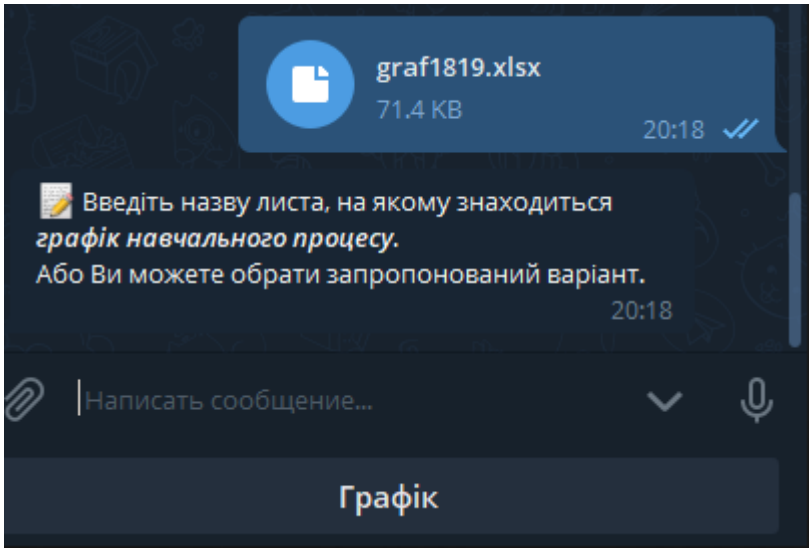


Рис. 2. 26. Повідомлення про введення назви листа

Можна обрати запропонований варіант, а можна ввести власний. Якщо в файлі було знайдено введений лист, то виведеться меню з підтвердженням оновлення, зображене на рис. 2. 27.

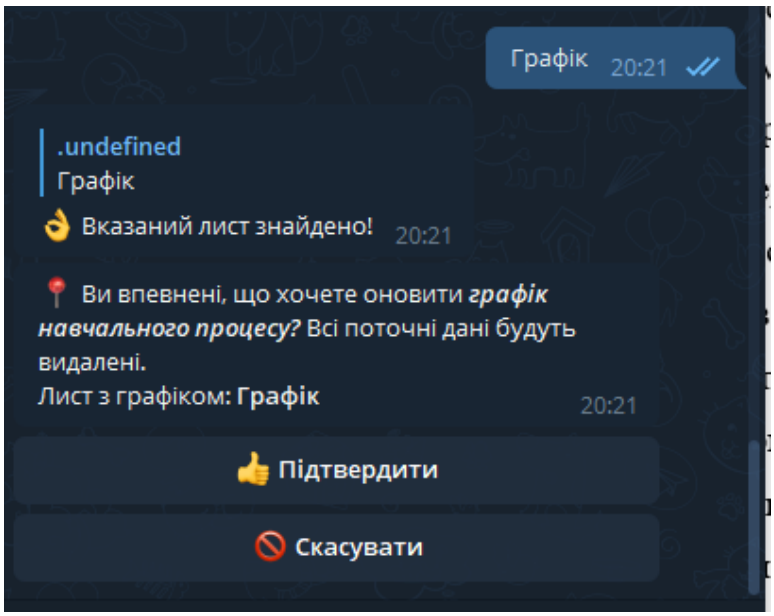
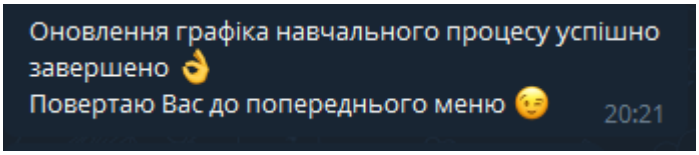


Рис. 2.27. Підтвердження оновлення графіка навчального процесу

Якщо натиснути кнопку «Скасувати», то оновлення буде скасовано, а користувач повернеться до попереднього меню. Якщо натиснути кнопку «Підтвердити», то видалиться попередній графік та почне завантажуватись новий. На рис. 2.28 зображено повідомлення після успішного оновлення графіка навчального процесу.



Рису. 2.28. Повідомлення про успішне оновлення графіка

Оновлення розкладу занять та замін занять відбувається схожим чином.

Для того, щоб зробити оголошення для всіх користувачів бота, адміністратор повинен обрати пункт головного меню «Зробити оголошення». Він отримає повідомлення, зображене на рис. 2.29.

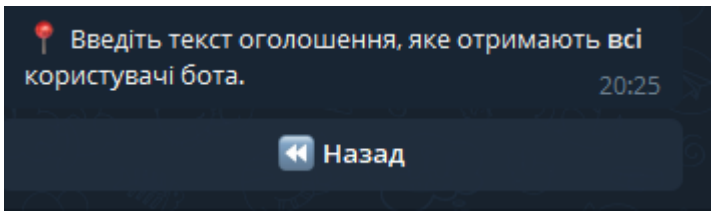


Рис. 2.29. Повідомлення про введення тексту оголошення

Після введення тексту оголошення, користувач отримає меню з підтвердженням відправки, зображене на рис. 2.30.

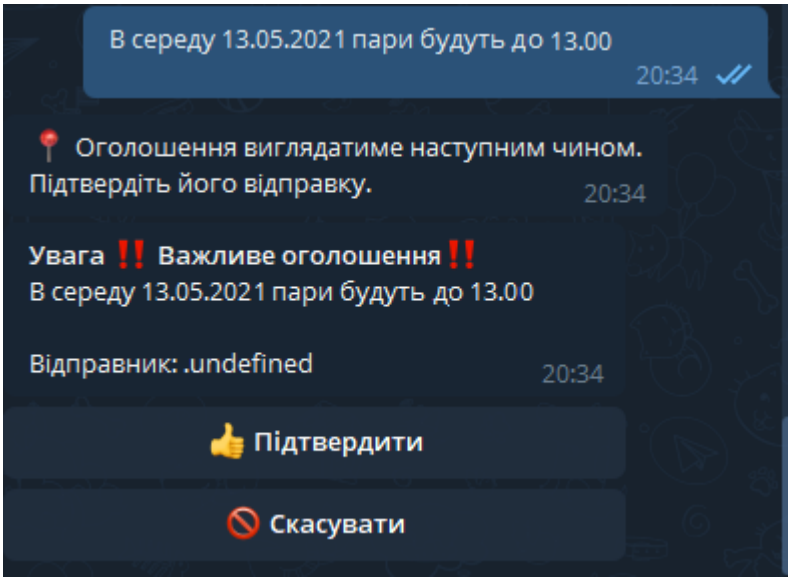


Рис. 2.30. Підтвердження відправки оголошення

Якщо натиснути «Скасувати», то відправку повідомлення буде скасовано і користувач повернеться до попереднього меню. Якщо підтвердити, то оголошення буде надіслане кожному користувачу бота і адміністратор отримає повідомлення, зображене на рис. 2.31.

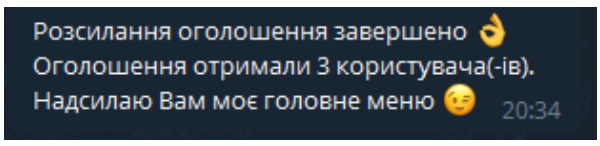


Рис. 2.31. Повідомлення про завершення розсилання оголошення

Якщо в головному меню обрати «Налаштування», то користувач отримає меню, зображене на рис. 2.32.

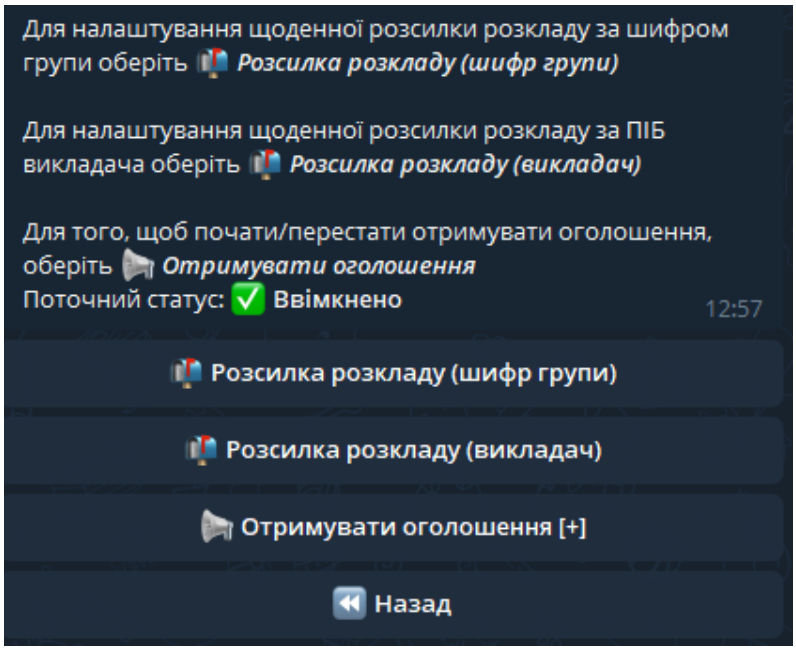


Рис. 2.32. Меню налаштувань

Для того щоб почати, або перестати отримувати оголошення від адміністратора, необхідно натиснути кнопку «Отримувати оголошення».

Для того, щоб налаштувати щоденну розсилку розкладу за викладачем, необхідно натиснути кнопку «Розсилка розкладу (викладач)». Користувач отримає меню, зображене на рис. 2.33.

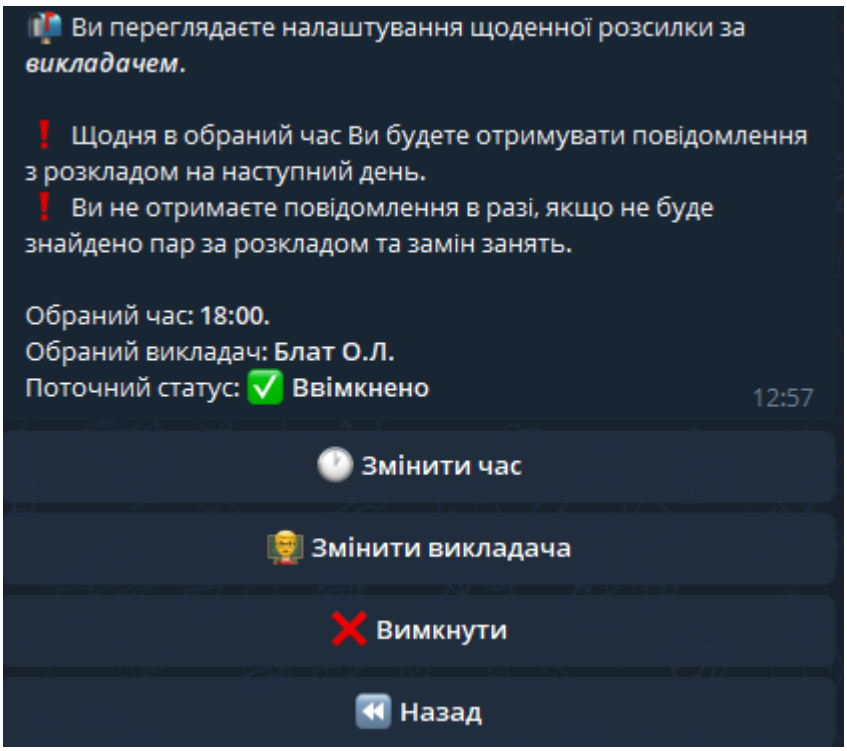


Рис. 2.33. Меню налаштування розсилки розкладу за викладачем

Для того, щоб обрати викладача, необхідно натиснути кнопку «Змінити викладача». Введення ПІБ викладача аналогічне введенню ПІБ при перегляді розкладу, часткове введення також підтримується.

Для того, щоб встановити час для щоденної розсилки, необхідно обрати пункт меню «Змінити час», виведеться меню, зображене на рис. 2.34 За допомогою стрілок необхідно обрати час та натиснути «Підтвердити». Час можна встановлювати з інтервалом в 30 хвилин.

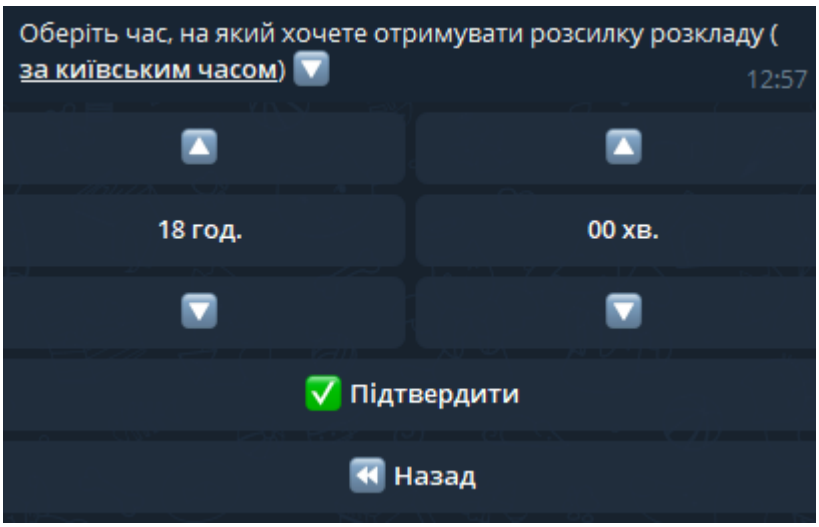


Рис. 2.34. Встановлення часу для щоденної розсилки розкладу

Для того, щоб ввімкнути або вимкнути розсилку, необхідно обрати відповідний пункт меню.

Налаштування щоденної розсилки за шифром групи здійснюється таким самим чином.

2.6.5. Тестування системи на наявність помилок

При розробці програмного забезпечення велику увагу було приділено перевірці коректності вхідних даних.

В разі, якщо при введенні ПІБ викладача, його не було знайдено, то користувач отримає повідомлення, зображене на рис. 2.35.

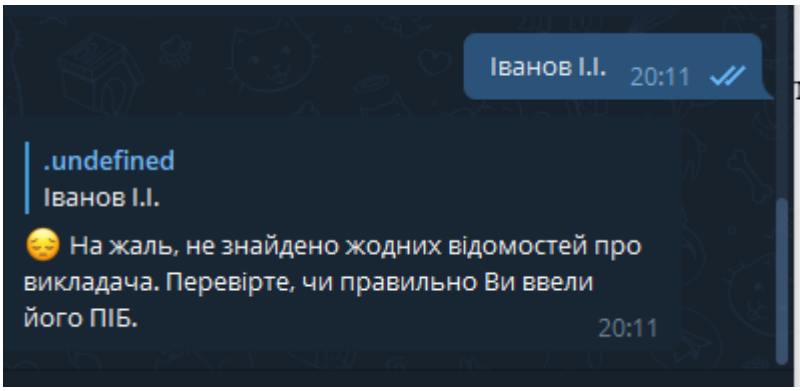


Рис. 2.35. Повідомлення про неправильний ПІБ викладача

В разі, якщо при введенні шифру групи, її не було знайдено, то користувач отримає повідомлення, зображене на рис. 2.36.

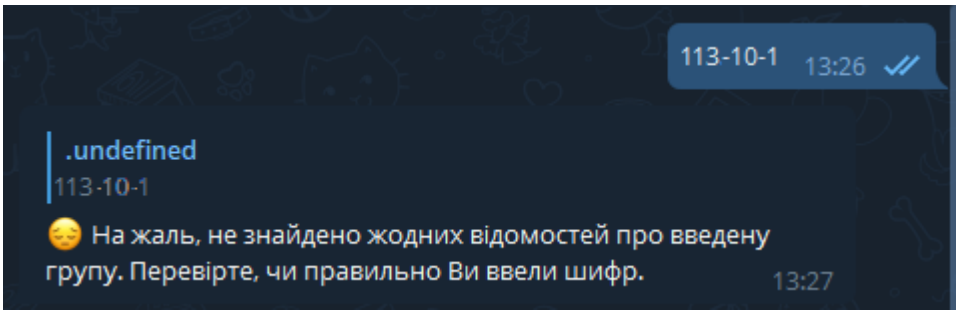


Рис. 2.36. Повідомлення про неправильний шифр групи

При пошуку за шифром групи відбувається видалення зайвих символів-роздільників та приведення шифру до спільного регістру.

Якщо при оновленні розкладу, графіка навчального процесу або замін занять завантажити не Excel файл, користувач отримає повідомлення, зображене на рис. 2.37.

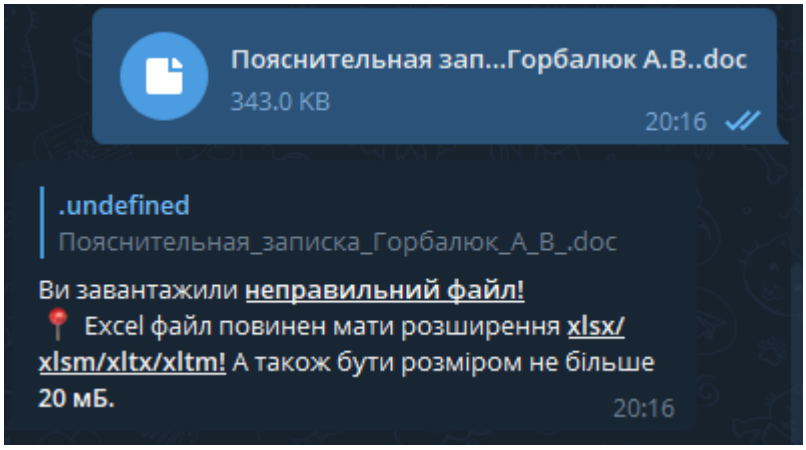


Рис. 2.37. Повідомлення про неправильний формат файлу

Якщо при оновленні розкладу або графіка навчального процесу було введено назву листа, який відсутній в завантаженому файлі, буде виведено повідомлення, зображене на рис. 2.38.

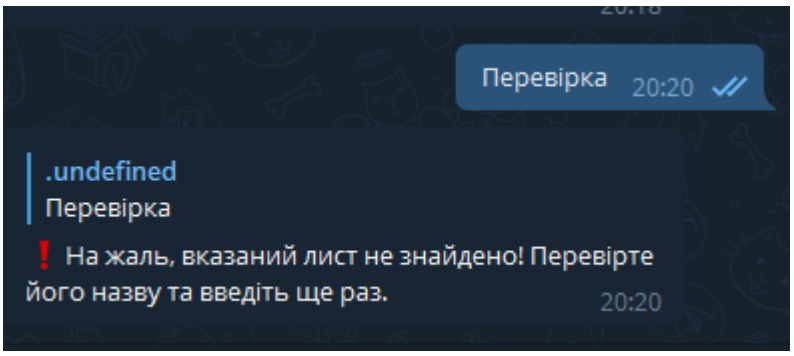


Рис. 2.38. Повідомлення про неправильну назву листа

Якщо при налаштуванні розсилки за викладачем користувач спробує увімкнути розсилку, не обравши викладача, він отримає повідомлення, зображене на рис. 2.39.

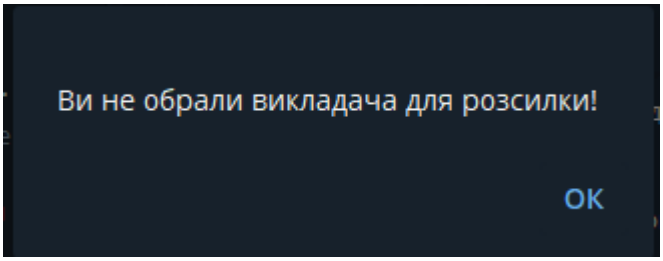


Рис. 2.39. Повідомлення про необхідність обрати викладача

Якщо при налаштуванні розсилки користувач спробує увімкнути розсилку, не обравши час для розсилки, він отримає повідомлення, зображене на рис. 2.40.

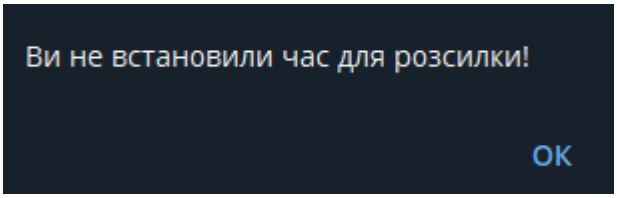


Рис. 2.40. Повідомлення про необхідність обрати час для розсилки

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Вхідні дані:

- передбачуване число операторів - 1500;
- коефіцієнт складності програми - 2;
- коефіцієнт корекції програми в ході її розробки - 0,08;
- годинна заробітна плата програміста, грн./год - 30.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_u + t_a + t_n + t_{отл} + t_д, \text{ людино-годин,} \quad (3.1)$$

де to – витрати праці на підготовку й опис поставленої задачі (50),

ти – витрати праці на дослідження алгоритму рішення задачі,

та – витрати праці на розробку блок-схеми алгоритму,

тп – витрати праці на програмування по готовій блок-схемі,

тотл – витрати праці на налагодження програми на ЕОМ,

тд – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \times C \times (1+ p) \text{ , людино-годин,} \quad (3.2)$$

де q - передбачуване число операторів,

C - коефіцієнт складності програми;

p - коефіцієнт кореляції програми в ході її розробки.

$$Q = 1500 \cdot 2 \cdot (1 + 0{,}08) = 3240 \text{ людино-годин.} \quad (3.3)$$

Витрати праці на вивчення опису задачі ti визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{QB}{(75...85)K}, \text{ людино-годин,} \quad (3.4)$$

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі; B=1.2 … 1.5,

k – коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності.

Витрати праці на розробку алгоритму рішення задачі:

$$t_u = \frac{3240 \cdot 1{,}3}{80 \cdot 1{,}2} = 44, \text{ людино-годин.} \quad (3.5)$$

Витрати на складання програми по готовій блок-схемі:

$$t_a = \frac{Q}{(20...25)K} \quad \text{людино-годин.} \quad (3.6)$$

$$t_a = \frac{3240}{22 \cdot 1{,}2} = 123 \text{ людино-годин.} \quad (3.7)$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25)K} \quad \text{людино-годин.} \tag{3.8}$$

$$t_n = \frac{3240}{23 \cdot 1{,}2} = 117 \text{ людино-годин.} \tag{3.9}$$

Витрати праці на налагодження програми на ЕОМ:

$$t_{отл} = \frac{Q}{(4...5)K} \quad \text{людино-годин.} \tag{3.10}$$

$$t_{отл} = \frac{3240}{5 \cdot 1{,}2} = 540 \text{ людино-годин.} \tag{3.11}$$

Витрати праці на підготовку документації:

$$t_д = t_{др} + t_{до}, \text{ людино-годин,} \tag{3.12}$$

де tдр - трудомісткість підготовки матеріалів і рукопису.

$$t_{др} = \frac{Q}{(15...20)K}, \text{ людино-годин.} \tag{3.13}$$

$$t_{др} = \frac{3240}{18 \cdot 1{,}2} = 150 \text{ людино-годин,} \tag{3.14}$$

де tдо - трудомісткість редагування, печатки й оформлення документації.

$$t_{до} = 0{,}75 \cdot t_{др}, \text{ людино-годин.} \tag{3.15}$$

$$t_{до} = 150 + 73 = 223 \text{ людино-годин.} \tag{3.16}$$

Отримуємо трудомісткість розробки програмного забезпечення:

$$t = 50 + 44 + 123 + 117 + 540 + 223 = 1097 \text{ людино-годин.} \tag{3.17}$$

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ Кпо включають витрати на заробітну плату виконавця програми Зз/п і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$K_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн.} \quad (3.18)$$

Заробітна плата виконавців визначається за формулою:

$$З_{ЗП} = t \cdot C_{ПР}, \text{ грн,} \quad (3.19)$$

де t - загальна трудомісткість, людино-годин,

Спр - середня годинна заробітна плата програміста, грн/година.

$$З_{ЗП} = 1097 \cdot 30 = 32910 \text{ грн.} \quad (3.20)$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{МВ} = t_{отл} \times C_{М}, \text{ грн,} \quad (3.21)$$

де tотл - трудомісткість налагодження програми на ЕОМ, год.,

Смч - вартість машино-години ЕОМ, грн/год.

Визначені в такий спосіб витрати на створення програмного забезпечення є частиною одноразових капітальних витрат на створення АСУП.

$$З_{МВ} = 540 \times 5 = 2700 \text{ грн.}, \quad (3.22)$$

$$Кпо = 2700 + 32910 = 35610 \text{ грн.} \quad (3.23)$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p} \text{ міс.,} \tag{3.24}$$

де Bk - число виконавців,

Fp - місячний фонд робочого часу (при 40 годинному робочому тижні Fp=176 годин).

$$T = \frac{1097}{1 \cdot 176} = 6.23 \text{ міс.} \tag{3.25}$$

Визначено трудомісткість розробленої інформаційної системи (1097 люд-год), проведений підрахунок вартості роботи по створенню програми (39610 грн.) та розраховано час на його створення (6,23 міс).

ВИСНОВКИ

Під час роботи над кваліфікаційною роботою на тему «Розробка чат-бота на базі месенджеру Telegram для моніторингу розкладу занять в закладі освіти засобами середовища Microsoft Visual Studio Code» було розроблено програмний додаток, який надає допомогу студентам та викладачам в освітньому процесі.

Додаток реалізований з використанням таких ролей користувачів: «Адміністратор» та «Користувач з правом перегляду».

Для ролі «Адміністратор» реалізовано функції оновлення графіка навчального процесу, розкладу занять та їх замін. Для цього користувач може перейти у відповідне меню та перенести готовий файл в діалог з чат-ботом. Також реалізовано функцію для надсилання оголошень, які отримуватимуть всі користувачі бота.

Для всіх користувачів реалізовано функції перегляду розкладу занять за вказаним шифром групи або ПІБ викладача. Інформація надана з урахуванням графіка освітнього процесу та існуючих замін на обрану дату.

Також для всіх користувачів є можливість підписки на щоденну розсилку розкладу занять. Для цього користувачеві необхідно обрати викладача або групу та зручний час для отримання повідомлень. У випадку, якщо для обраного викладача або групи не було знайдено занять за розкладом та замін на наступний день, чат-бот не надсилає жодних повідомлень.

Чат-бот буде корисним для студентів та викладачів любого закладу, оскільки надає можливість легко та швидко отримувати актуальну інформацію щодо розкладу занять, їх замін та графіка навчального процесу.

Програмний додаток має інтуїтивно-зрозумілий графічний інтерфейс та містить підказки для користувача. Його розроблено таким чином, щоб користувачеві потрібно було вводити якомога менше інформації вручну, а отже, можливість здійснення помилки буде мінімальною.

Розроблена пояснювальна записка містить корисну інформацію як для

розробників, які можуть супроводжувати проєкт в майбутньому, так і для користувачів, щоб полегшити їх досвід користування чат-ботом.

Для реалізації додатку кваліфікаційної роботи обрано мову програмування Python. Для реалізації бази даних додатку було обрано СУБД MongoDB. В якості середовища розробки обрано Microsoft Visual Studio Code.

В економічному розділі визначено трудомісткість розробленої інформаційної системи (1097 люд-год), проведений підрахунок вартості роботи по створенню програми (39610 грн.) та розраховано час на його створення (6,23 міс).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання : (ГОСТ 7.1-2003, IDT) : ДСТУ ГОСТ 7.1:2006. – Чинний з 2007–07–01. – К. : Держспоживстандарт України, 2007. – 47 с. – (Система стандартів з інформації, бібліотечної та видавничої справи) (Національний стандарт України).

2. Бусигін Б.С., Коротенко Г.М., Коротенко Л.М. Прикладна інформатика. Підручник для студентів комп'ютерних спеціальностей. – Дніпропетровськ: Видавництво НГУ, 2004. – 559 с. URL: http://www.programmer.dp.ua/book-ua-k01.php. дата звернення: 15.03.2019.

3. Бусыгин Б.С., Дивизинюк М.М., Коротенко Г.М., Коротенко Л.М. Введение в современную информатику. Учебник. – Севастополь: Издательство СНУЯЭиП, 2005. – 644 с. / URL: http://www.programmer.dp.ua/book-ru-k02.php. дата звернення: 15.01.2018.

4. Бэнкер Кайл. MongoDB в действии : пер. с англ. / Кайл Бэнкер - ДМК Пресс, 2014. - 394 с. - ISBN 978-5-97060-057-3

5. Вахула Б. Я. Аксіоматичні ядра головних соціологічних підходів у дослідженні соціальних інтернет-мереж/ Б. Я. Вахула // Львівський національний університет імені Івана Франка. – 2013. – URL: http://www.ukr-socium.org.ua/Arhiv/Stati/1_2013/33-42.pdf Дата звернення: 15.01.2020

6. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення : ДСТУ 3008-95. – Чинний від 1996–01–01. – К. : Держстандарт України, 1996. – 39 с.

7. Доусон М. Программируем на Python : пер. с англ. / М. Доусон - СПб.: Питер, 2012. - 432 с. - ISBN 978-5-459-00314-7

8. ДСТУ 2394-94 Інформація та документація. Комплектування фонду, бібліографічний опис, аналіз документів. Терміни та визначення. – Чинний від 01.01.1995. - Київ : Держстандарт України, 1994. – 88 с.

9. Методичні вказівки з виконання економічного розділу в дипломних проектах студентів спеціальності “Комп’ютерні системи ” / Укладачі О.Г. Вагонова, Нікітіна О.Б. Н.Н. Романюк – Дніпропетровськ: Національний гірничий університет. – 2013. – 23с.

10. Методичні рекомендації до виконання кваліфікаційних робіт бакалаврів напряму підготовки 122 «Комп’ютерні науки» галузі знань 12 Інформаційні технології/, Л.М. Коротенко , О.С. Шевцова; Нац. гірн. ун-т. – Д : ДВНЗ НГУ, 2019. – 65 с.

11. Методичні рекомендації щодо написання, оформлення та представлення учнівських науково-дослідницьких робіт учнів – членів Малої академії наук України / Г.Г. Півняк, Л.М. Коротенко, І.М. Удовик, Є.М. Головня – Д.: ДВНЗ «Національний гірничий університет», 2017. – 24 с.

12. Офіційний сайт середовища розробки Visual Studio Code URL: документація - Режим доступу : https://code.visualstudio.com/docs. дата звернення 11.04.21

13. Офіційний сайт СУБД MongoDB. URL: сторінка про графічну оболонку MongoDB Compass - Режим доступу : https://www.mongodb.com/ products/compass. дата звернення 11.04.21

14. Саммерфилд Марк. Python на практике : пер. с англ. / Марк Саммерфилд - М.: ДМК Пресс, 2014. - 338 с. - ISBN 978-5-9706-0095-5

15. Скорочення слів в українській мові у бібліографічному описі. Загальні правила та вимоги : ДСТУ 3582-97. – Чинний від 1998–07–01. – К. : Держстандарт України, 1998. – 24 с. – (Державний стандарт України).

16. Фаронов В., Изд.: Питер 2006 г.Дж. Боуман, С.Эмерсон, М.Дарновски. Практическое руководство SQL; 2000. – 321 с.

17. Якимчук О. Соціальні мережі та їх аналіз / О. Якимчук // Релігія та соціум. – 2012. – №1. – С. 146–153.

18. Benjamin Pierce. Types and Programming Languages. — MIT Press, 2002. – 221с.

19. SQL. URL: https://ru.wikipedia.org/wiki/SQL. Дата звернення: 15.01.2021

20. Telegram. URL: https://uk.wikipedia.org/wiki/Telegram. дата звернення 11.04.21

ДОДАТОК А

КОД ПРОГРАМИ

Реалізація алгоритму завантаження графіка навчального процесу

```
graph_collection_name = 'graphic'
async def parse_graph_file(path, sheet_name):
    with open(path, "rb") as file:
        in_mem_file = io.BytesIO(file.read())
    workbook = load_workbook(filename=in_mem_file,
                             read_only=True,
                             data_only=True,
                             keep_links=False)
    if not sheet_name in workbook.sheetnames:
        raise ParsingError('Помилка при оновленні графіку.' \
                'Не знайдено вказаний лист.')
    sheet = workbook[sheet_name]
    header_row = get_header_row(sheet)
    options_row, options = get_options(sheet, header_row)
    groups_list = get_groups_list(sheet, header_row, options_row)
    start_parse_col, dates_list = get_dates_list(sheet, header_row)
    await start_parse(sheet, options, groups_list,
                dates_list, header_row + 3, start_parse_col)

def get_header_row(sheet):
    for row in sheet.iter_rows( min_col=1,
                                max_col=1):
        if '№' in str(row[0].value):
            return row[0].row
    raise ParsingError('Помилка при оновленні графіку.' \
                    ' Перевірте шапку таблиці (не знайдено колонки "№").')

def get_options(sheet, header_row):
    for row in sheet.iter_rows( min_row=header_row + 2,
                                min_col=1,
                                max_col=1):
        if row[0].value \
                and not sheet['A{}'.format(row[0].row + 1)].value \
                and not sheet['A{}'.format(row[0].row + 2)].value:
            options_row = row[0].row + 3
            break
    options = {}
    i = 0
    for row in sheet.iter_rows( min_row=options_row,
                                max_row=options_row + 6):
        col = 0
        for cell in row:
            if cell.value and len(cell.value) > 3:
                key_col = col - 1
                try:
```

```
                while row[key_col].border.left.style is None:
                    key_col -= 1
            except:
                pass
            if row[key_col].value:
                key = ' '.join(row[key_col].value.split())
            else:
                key = row[key_col].value
            val = cell.value
            ###
            if sheet.cell(row = cell.row + 1, column=col + 1).value:
                val += ' ' + sheet.cell(row = cell.row + 1, column=col + 1).value
            ###
            params = {  'Value': ' '.join(val.split()),
                        'Key': key,
                        'Fill': row[key_col].fill,
                        'Border': {
                            'DiagonalUp': row[key_col].border.diagonalUp,
                            'DiagonalDown': row[key_col].border.diagonalDown,
                        }
                     }
            options[i] = params
            i += 1
        col += 1
    """for option in options:
        print(options.get(option).get('Value'), \
            options.get(option).get('Key'))"""
    return options_row, options

def get_groups_list(sheet, header_row, options_row):
    for row in sheet.iter_rows( min_row=header_row,
                                max_row=header_row):
        groups_col = 1
        finded = False
        for cell in row:
            if cell.value and cell.value.lower().find('шифр групи') != -1:
                finded = True
                break
            groups_col += 1
        if not finded:
            raise ParsingError('Помилка при оновленні графіку.' \
                ' Перевірте шапку таблиці (не знайдено колонки "Шифр групи").')
    groups = []
    for row in sheet.iter_rows( min_row=header_row + 1,
                                max_row=options_row - 1,
                                min_col=groups_col,
                                max_col=groups_col):
        if row[0].value:
            group = ''.join(row[0].value.lower().split())
            group = group.replace('–', '-')
            group = group.replace('—', '-')
            groups.append(group)
```

```
    return groups

def get_all_months_list():
    months = {
        'Січень': 1,
        'Лютий': 2,
        'Березень': 3,
        'Квітень': 4,
        'Травень': 5,
        'Червень': 6,
        'Липень': 7,
        'Серпень': 8,
        'Вересень': 9,
        'Жовтень': 10,
        'Листопад': 11,
        'Грудень': 12
    }
    return months

def get_dates_list(sheet, header_row):
    for row in sheet.iter_rows( min_row=header_row,
                                max_row=header_row):
        start_parse_col = 1
        for cell in row:
            if cell.value and cell.value.capitalize() in get_all_months_list():
                break
            start_parse_col += 1

    dates = {}
    for i in range(0, 3):
        for row in sheet.iter_rows( min_row=header_row + i,
                                    max_row=header_row + i,
                                    min_col=start_parse_col):
            j = 0
            for cell in row:
                if i == 0:
                    if cell.border.left.style == 'medium' \
                        and cell.value is None:
                        raise ParsingError('Помилка при оновленні графіку.'
                                ' Перевірте шапку таблиці (місяці).'
                                ' Не повинно бути пустих клітинок ({})' \
                                .format(cell.coordinate))
                    dates[j] = {
                        'Month': cell.value
                    }
                if i == 1:
                    month = dates.get(j).get('Month')
                    dates[j] = {
                        'Month': month,
                        'Start_date': cell.value
                    }
                if i == 2:
```

```
                month = dates.get(j).get('Month')
                date = dates.get(j).get('Start_date')
                dates[j] = {
                    'Month': month,
                    'Start_date': date,
                    'Last_date': cell.value
                }
            j += 1
    dates_ready = {}
    for date in dates:
        if dates.get(date).get('Month'):
            current_month = dates.get(date).get('Month')
        start_day = dates.get(date).get('Start_date')
        last_day = dates.get(date).get('Last_date')
        dates_ready[date] = {
            'Month': ' '.join(current_month.capitalize().split()),
            'Start_date': start_day,
            'Last_date': last_day
        }
    dates_ready = dates_to_usual_format(dates_ready)
    dates_ready = dates_to_unix_format(dates_ready)
    return start_parse_col, dates_ready

def dates_to_usual_format(dates):
    months_list = get_all_months_list()
    today_month = datetime.datetime.now().month
    today_year = datetime.datetime.now().year
    temp_dates = {}
    for i in dates:
        month = dates.get(i).get('Month')
        if month not in months_list:
            continue
        month = months_list[month]
        if today_month >= 8 and month >= 8:
            year = today_year
        elif today_month >= 8 and month < 8:
            year = today_year + 1
        elif today_month < 8 and month >= 8:
            year = today_year - 1
        elif today_month < 8 and month < 8:
            year = today_year
        start_day = dates.get(i).get('Start_date')
        last_day = dates.get(i).get('Last_date')
        try:
            start_date = datetime.date(year, month, start_day)
            last_date = datetime.date(year, month, last_day)
        except TypeError:
            raise ParsingError('Помилка при оновленні графіку.'
                               ' Перевірте шапку таблиці (числа).'
                               ' В клітинках повинні бути тільки цілі числа.')
        #swap months
        if start_date > last_date:
```

```
            try:
                start_date = start_date \
                            .replace(month = months_list[dates.get(i - 1) \
                                        .get('Month')])
            except:
                start_date = start_date \
                            .replace(month = start_date.month - 1)
            try:
                last_date = last_date \
                            .replace(month = months_list[dates.get(i + 1) \
                                        .get('Month')])
            except:
                last_date = last_date \
                            .replace(month = last_date.month + 1)
        temp_dates[i] = {
            'Start_date': start_date,
            'Last_date': last_date
        }
    temp_dates = dates_fix_year(temp_dates)
    return temp_dates

def dates_fix_year(dates):
    #swap years
    for i in dates:
        start_date = dates.get(i).get('Start_date')
        last_date = dates.get(i).get('Last_date')
        if start_date > last_date:
            try:
                start_date = start_date \
                            .replace(year = dates
                                        .get(i - 1)
                                        .get('Last_date')
                                        .year)
            except:
                start_date = start_date \
                            .replace(year = start_date.year - 1)
            try:
                last_date = last_date \
                            .replace(year = dates
                                        .get(i + 1)
                                        .get('Last_date')
                                        .year)
            except:
                last_date = last_date \
                            .replace(year = last_date.year + 1)
            dates[i] = {
            'Start_date': start_date,
            'Last_date': last_date
            }
    return dates

def dates_to_unix_format(dates):
```

```
    ready_dates = {}
    for i in dates:
        start_date = dates.get(i).get('Start_date')
        start_date = datetime.datetime( start_date.year,
                                        start_date.month,
                                        start_date.day)
        start_date_unix = start_date \
                          .replace(tzinfo=datetime.timezone.utc) \
                          .timestamp()
        last_date = dates.get(i).get('Last_date')
        last_date = datetime.datetime( last_date.year,
                                       last_date.month,
                                       last_date.day)
        last_date_unix = last_date \
                         .replace(tzinfo=datetime.timezone.utc) \
                         .timestamp()
        ready_dates[i] = {
            'Start_date': start_date_unix,
            'Last_date': last_date_unix
        }
    return ready_dates

async def start_parse(sheet, options, groups, dates, start_row, start_col):
    await data_worker.clear_collection(graph_collection_name)
    for i in range(0, len(groups)):
    #for i in range(45, 46):
        num = 0
        records = []
        for row in sheet.iter_rows( min_row=start_row + i*2,
                                    max_row=start_row + i*2 + 1,
                                    min_col=start_col,
                                    max_col=len(dates) + start_col - 1):
            col = 0
            for cell in row:
                option = ''
                if cell.border.left.style is None \
                    and row[col - 1].border.right.style is None:
                    option = records[num - 1].get('Option')
                if cell.border.top.style is None \
                    and sheet.cell( row=cell.row - 1, \
                                    column=start_col + col) \
                                        .border \
                                        .bottom \
                                        .style is None:
                    option = records[num - len(dates)].get('Option')
                if not option:
                    try:
                        value = ' '.join(cell.value.lower().split())
                    except:
                        value = cell.value
                    for j in options:
                        opt = options.get(j).get('Key')
```

```
                    if opt is not None:
                        opt = opt.lower()
                    if not options.get(j) \
                            .get('Border') \
                            .get('DiagonalUp') == cell.border.diagonalUp \
                       or not options.get(j) \
                            .get('Border') \
                            .get('DiagonalDown') == cell.border.diagonalDown:
                        continue
                    if opt == value \
                        and options.get(j).get('Fill') == cell.fill \
                        or opt == value \
                        and compare_fills(options.get(j).get('Fill'), cell.fill):
                        option = options.get(j).get('Value')
                        break
                if not option:
                    option = 'теоретичне навчання'
                num += 1
                if num <= len(dates):
                    start_date, last_date = set_dates(dates.get(col).get('Start_date'),
                                                      dates.get(col).get('Last_date'))
                else:
                    start_date = records[col].get('Last_date') + 1
                    last_date = dates.get(col).get('Last_date') + 86399
                records.append({
                    'Group': groups[i],
                    'Start_date': start_date,
                    'Last_date': last_date,
                    'Numerator': col % 2,
                    'Option': option
                })
                col += 1
        await data_worker.insert_many_records(graph_collection_name, records)

def set_dates(start_date, last_date):
    start_date_weekday = datetime.datetime.fromtimestamp(start_date,
                                                         tz=datetime.timezone.utc) \
                                                         .isoweekday()
    if start_date_weekday in [1, 2, 3]:
        k = 3
    elif start_date_weekday in [4, 5, 6]:
        k = 5
    elif start_date_weekday == 7:
        k = 4
    last_date = start_date + k*86400 - 1
    return start_date, last_date

def compare_fills(f_fill, s_fill):
    try:
        if f_fill.start_color.index is not None:
            if COLOR_INDEX[f_fill.start_color.index] == WHITE \
                and s_fill.fgColor.rgb == '00000000':
```

```
            return True
    except:
        pass
    try:
        if s_fill.start_color.index is not None:
            if COLOR_INDEX[s_fill.start_color.index] == WHITE \
                and f_fill.fgColor.rgb == '00000000':
                return True
    except:
        pass
    GRAY_1 = -0.499984740745262
    GRAY_2 = -0.3499862666707358
    try:
        if f_fill.start_color.index is not None:
            if f_fill.start_color.index == 23 \
                and (s_fill.fgColor.tint == GRAY_1 \
                or s_fill.fgColor.tint == GRAY_2) \
                and s_fill.fgColor.type == 'theme':
                return True
    except:
        pass
    try:
        if s_fill.start_color.index is not None:
            if s_fill.start_color.index == 23 \
                and (f_fill.fgColor.tint == GRAY_1 \
                or f_fill.fgColor.tint == GRAY_2) \
                and f_fill.fgColor.type == 'theme':
                return True
    except:
        pass
    return False
```

Реалізація алгоритму завантаження розкладу занять

```
schedule_collection_name = 'schedule'

async def parse_schedule_file(path, sheet_name):
    with open(path, "rb") as file:
        in_mem_file = io.BytesIO(file.read())
    workbook = load_workbook(filename=in_mem_file,
                             read_only=True,
                             data_only=True,
                             keep_links=False)
    if not sheet_name in workbook.sheetnames:
        raise ParsingError('Помилка при оновленні розкладу.' \
                'Не знайдено вказаний лист.')
    sheet = workbook[sheet_name]
    header_row = get_header_row(sheet)
    await start_parse(sheet, header_row)

def get_header_row(sheet):
```

```
    for row in sheet.iter_rows(min_row=2):
        if row[0].value and 'дні' in str(row[0].value).lower():
            return row[0].row
    raise ParsingError('Помилка при оновленні розкладу.'
                       ' Перевірте шапку таблиці (не знайдено колонки "Дні").')

async def start_parse(sheet, header_row):
    groups_list = get_groups_list(sheet, header_row)
    #all spaces deleted from header
    exceptions = ['дні', 'пара', 'часзанять']
    step = 4
    days = get_days_list(sheet, header_row, step)
    lessons = get_lessons_list(sheet, header_row, step)
    await data_worker.clear_collection(schedule_collection_name)
    empty_rec = {
        'Subject': None,
        'Teacher': None,
        'Classroom': 'не визначено'
    }
    for col in range(0, sheet.max_column):
    #for col in range(92, 98):
        if groups_list[col] in exceptions \
                or isinstance(groups_list[col], int) \
                or groups_list[col] is None:
                continue
        for row in range(header_row + 1, sheet.max_row, step):
            cell = sheet.cell(row=row, column=col + 1)
            if isinstance(cell, EmptyCell) \
                or row not in days \
                or row not in lessons:
                continue
            if sheet.cell(row=row + 1, column=col + 1).border.bottom.style is None \
                and sheet.cell(row=row + 2, column=col + 1).border.top.style is None:
                subject = find_subject(sheet, row, col + 1)
                if subject is None:
                    numerator = empty_rec
                else:
                    numerator = {
                        'Subject': subject,
                        'Teacher': find_teacher(sheet, row + 3, col + 1),
                        'Classroom': find_classroom(sheet, row, col + 2)
                    }
                denominator = numerator
            else:
                subject = find_subject(sheet, row, col + 1)
                if subject is None:
                    numerator = empty_rec
                else:
                    numerator = {
                        'Subject': subject,
                        'Teacher': find_teacher(sheet, row + 1, col + 1),
                        'Classroom': find_classroom(sheet, row, col + 2)
```

```
                }
            subject = find_subject(sheet, row + 2, col + 1)
            if subject is None:
                denominator = empty_rec
            else:
                denominator = {
                    'Subject': subject,
                    'Teacher': find_teacher(sheet, row + 3, col + 1),
                    'Classroom': find_classroom(sheet, row + 2, col + 2)
                }
        record = {
            'Group': groups_list[col],
            'Day': get_num_by_day(days.get(row)),
            'Lesson': lessons.get(row),
            'Numerator': numerator,
            'Denominator': denominator
        }
        await data_worker.insert_one_record(schedule_collection_name, record)
        """print('**********')
        print(record.get('Group'), record.get('Day'), record.get('Lesson'))
        print(record.get('Numerator').get('Subject'), \
            record.get('Numerator').get('Teacher'), \
            record.get('Numerator').get('Classroom'))
        print(record.get('Denominator').get('Subject'), \
            record.get('Denominator').get('Teacher'), \
            record.get('Denominator').get('Classroom'))"""

def get_day_col(sheet, header_row):
    days_col = None
    for row in sheet.iter_rows(min_row=header_row,
                               max_row=header_row):
        col = 1
        for cell in row:
            if not isinstance(cell.value, str):
                continue
            if 'дні' in ' '.join(cell.value.lower().split()):
                days_col = col
            if days_col is not None:
                break
            col += 1
    if days_col is None:
        raise ParsingError('Помилка при оновленні розкладу.'
                    ' Перевірте шапку таблиці (не знайдено колонки "Дні").')
    return days_col

def get_lesson_col(sheet, header_row):
    num_lesson_col = None
    for row in sheet.iter_rows(min_row=header_row,
                               max_row=header_row):
        col = 0
        for cell in row:
            col += 1
```

```
            if not isinstance(cell.value, str):
                continue
            if 'пара' in ' '.join(cell.value.lower().split()):
                num_lesson_col = col
            if num_lesson_col is not None:
                break
    if num_lesson_col is None:
        raise ParsingError('Помилка при оновленні розкладу.'
                   ' Перевірте шапку таблиці (не знайдено колонки "Пара").')
    return num_lesson_col

def get_groups_list(sheet, header_row):
    groups_list = []
    for row in sheet.iter_rows( min_row=header_row,
                            max_row=header_row):
        for cell in row:
            if isinstance(cell.value, str):
                group = ''.join(cell.value.lower().split())
                group = group.replace('–', '-')
                group = group.replace('—', '-')
                groups_list.append(group)
            else:
                groups_list.append(cell.value)
    return groups_list

def get_days_list(sheet, header_row, step):
    days = {}
    col = get_day_col(sheet, header_row)
    for row in range(header_row + 1, sheet.max_row, step):
        cell = sheet.cell(row=row, column=col)
        if isinstance(cell, EmptyCell):
            continue
        if cell.border.top.style == 'medium' \
            and cell.value is None:
            raise ParsingError('Помилка при оновленні розкладу.' \
                ' Перевірте колонку з днями тижня.' \
                ' Там не повинно бути пустих клітинок ({}).' \
                .format(cell.coordinate))
        if cell.value is None:
            days[row] = days.get(row - 4)
        else:
            days[row] = ' '.join(cell.value.lower().split())
    return days

def get_lessons_list(sheet, header_row, step):
    lessons = {}
    col = get_lesson_col(sheet, header_row)
    for row in range(header_row + 1, sheet.max_row, step):
        cell = sheet.cell(row=row, column=col)
        if isinstance(cell, EmptyCell):
            continue
        if not cell.value:
```

```
            raise ParsingError('Помилка при оновленні розкладу.'
                  ' Перевірте колонку "Пара", не повинно бути пустих клітинок ({}).' \
                  .format(cell.coordinate))
        lessons[row] = ' '.join(cell.value.lower().split())
    return lessons

def get_num_by_day(day):
    days_of_week = {
        'понеділок': 1,
        'вівторок': 2,
        'середа': 3,
        'четвер': 4,
        "п'ятниця": 5,
        'субота': 6,
        'неділя': 7
    }
    try:
        return days_of_week[day]
    except KeyError:
        return -1

def find_classroom(sheet, row, col):
    cell = sheet.cell(row=row, column=col)
    if cell.value:
        return ' '.join(str(cell.value).lower().split())
    cell = sheet.cell(row=row, column=col + 1)
    while cell.border.right.style is None:
        col += 1
        cell = sheet.cell(row=row, column=col + 1)
    if cell.value is None:
        cell = sheet.cell(row=row, column=col + 2)
        if cell.value:
            return ' '.join(str(cell.value).lower().split())
        cell = sheet.cell(row=row + 1, column=col + 2)
    cell = sheet.cell(row=row + 1, column=col)
    if cell.value:
        return ' '.join(str(cell.value).lower().split())
    return 'не визначено'

def find_subject(sheet, row, col):
    cell = sheet.cell(row=row, column=col)
    if cell.value:
        return ' '.join(cell.value.split())
    while cell.border.left.style is None:
        col -= 1
        cell = sheet.cell(row=row, column=col)
    if cell.value:
        return ' '.join(cell.value.split())
    else:
        return cell.value

def fix_name(name):
```

```
    teacher = ' '.join(name.lower().split())
    if not '-' in teacher:
        if teacher[len(teacher) - 1] != '.':
            teacher += '.'
        if teacher[len(teacher) - 3] != '.':
            teacher = teacher[0:len(teacher) - 2] + '.' + teacher[len(teacher) - 2:]
        if teacher[len(teacher) - 5] != ' ':
            teacher = teacher[0:len(teacher) - 4] + ' ' + teacher[len(teacher) - 4:]
    return teacher

def find_teacher(sheet, row, col):
    cell = sheet.cell(row=row, column=col)
    if cell.value:
        teacher = fix_name(cell.value)
        return teacher
    while cell.border.left.style is None:
        col -= 1
        cell = sheet.cell(row=row, column=col)
    if cell.value:
        teacher = fix_name(cell.value)
        return teacher
    else:
        return cell.value
```

Реалізація алгоритму завантаження замін занять

```
changes_collection_name = 'changes'

async def parse_changes_file(path):
    with open(path, "rb") as file:
        in_mem_file = io.BytesIO(file.read())
    workbook = load_workbook(filename=in_mem_file,
                             read_only=True,
                             data_only=True,
                             keep_links=False)
    dates = get_dates_list(workbook)
    for date in dates:
        await start_parse(workbook[date], dates[date])

def get_header_values():
    return ['шифр групи', '№ пари', 'хто заміняє', 'кого заміняє', 'ауд']

def get_dates_list(workbook):
    sheets = {}
    for sheet in workbook.sheetnames:
        form = ''
        splitted = sheet.split('.')
        if sheet.count('.') == 2:
            if len(splitted) == 3 and len(splitted[2]) == 2:
                form = '%d.%m.%y'
            else:
                form = '%d.%m.%Y'
```

```
        try:
            date = datetime.strptime(sheet, form)
            date = date.replace(tzinfo=timezone.utc).timestamp()
            now = datetime.now().replace(hour=0, minute=0, second=0, microsecond=0,
                                         tzinfo=timezone.utc).timestamp()
            if now <= date:
                sheets[sheet] = date
        except ValueError:
            continue
    return sheets

async def start_parse(sheet, date):
    header_row, header_cols = get_header_row(sheet)
    header_values = get_header_values()
    group_col = header_values[0]
    group_col = [col for col, value in header_cols.items() if value == group_col]
    group_col = group_col[0]
    if not header_row:
        return
    for row in sheet.iter_rows(min_row=header_row + 1):
        if not isinstance(row[group_col].value, str):
            continue
        groups, lesson, teacher, reason, classroom = [''] * 5
        for i in header_cols:
            if not isinstance(row[i].value, str) \
                and not isinstance(row[i].value, int):
                continue
            if header_values[0] in header_cols[i]:
                groups = (''.join(row[i].value.lower().split())).split(',')
                for j in range(1, len(groups)):
                    if len(groups[j]) < 2:
                        groups[j] = groups[j - 1][:len(groups[j - 1]) - 1] \
                                + groups[j][:len(groups[j])]
            elif header_values[1] in header_cols[i]:
                lesson = row[i].value
                if isinstance(lesson, str):
                    lesson = ' '.join(lesson.upper().split())
            elif header_values[2] in header_cols[i]:
                teacher = ' '.join(row[i].value.lower().split())
                if not '-' in teacher:
                    if teacher[len(teacher) - 1] != '.':
                        teacher += '.'
                    if teacher[len(teacher) - 3] != '.':
                        teacher = teacher[0:len(teacher) - 2] + '.' + teacher[len(teacher) - 2:]
                    if teacher[len(teacher) - 5] != ' ':
                        teacher = teacher[0:len(teacher) - 4] + ' ' + teacher[len(teacher) - 4:]
            elif header_values[3] in header_cols[i]:
                reason = ' '.join(row[i].value.lower().split())
            elif header_values[4] in header_cols[i]:
                classroom = row[i].value
                if isinstance(classroom, str):
                    classroom = ' '.join(classroom.split())
```

```
        if not groups or not date or not lesson or not teacher:
            continue
        if not classroom:
            classroom = 'не визначено'
        if not reason:
            reason = 'не визначено'
        for group in groups:
            group = group.replace('–', '-')
            group = group.replace('—', '-')
            record = {
                'Group': group,
                'Date': date,
                'Lesson': lesson
            }
            await data_worker.delete_records(changes_collection_name, record)
            record['Teacher'] = teacher
            record['Reason'] = reason
            record['Classroom'] = classroom
            await data_worker.insert_one_record(changes_collection_name, record)

def get_header_row(sheet):
    header_values = get_header_values()
    header_row = 0
    header_cols = {}
    for row in sheet.iter_rows():
        col = 1
        for cell in row:
            if isinstance(cell.value, str):
                cell_value = ' '.join(cell.value.lower().split())
                for item in header_values:
                    if item in cell_value:
                        if not header_row:
                            header_row = cell.row
                        header_cols[col - 1] = item
            col += 1
    return header_row, header_cols

async def clear_old_changes():
    now = datetime.now()
    now = now.replace(hour=0, minute=0, second=0, microsecond=0)
    params = {
        'Date': { '$lt': now \
                         .replace(tzinfo=timezone.utc) \
                         .timestamp()}
    }
    cursor = data_worker.find_many_records(changes_collection_name, params)
    for document in await cursor.to_list(length=None):
        await data_worker.delete_records(changes_collection_name, document)
```

Текст програми додатку

#main.py

```
async def startup(dp: Dispatcher):
    await changes_file_handler.clear_old_changes()
    start_scheduler()

async def shutdown_storage(dp: Dispatcher):
    await dp.storage.close()
    await dp.storage.wait_closed()
    shutdown_scheduler()

if __name__ == '__main__':
    dp = Dispatcher(bot, storage=storage)
    log.setup_logging()
    log.write_logs(log.logs_levels.INFO.value, '-------------------------------')
    log.write_logs(log.logs_levels.INFO.value, 'Bot successfull started')
    log.write_logs(log.logs_levels.INFO.value, '-------------------------------')
    load_admins_ID()
    load_support_ID()
    load_website_link()
    handlers_registration.setup(dp)
    try:
        executor.start_polling(dp, on_startup=startup, on_shutdown=shutdown_storage)
    except Exception as Error:
        print('--ERROR-- Could not start polling.')
        log.write_logs( log.logs_levels.ERROR.value,
                    'Could not start polling. Error: {}'.format(Error))

#-----------------
#fsm.py
def load_fsm_settings():
    try:
        tree = ['redis']
        tags = ['host', 'port', 'password', 'db']
        host, port, password, db = xml_parser.get_list_tags_text(tree, tags)
        tree.append('FSM')
        tags = ['poolSize', 'prefix']
        pool_size, prefix = xml_parser.get_list_tags_text(tree, tags)
        return host, int(port), password, int(db), int(pool_size), prefix
    except AttributeError as error:
        log.write_logs(log.logs_levels.ERROR.value, error)
try:
    host, port, password, db, pool_size, prefix = load_fsm_settings()
    storage = RedisStorage2(host,
                            port,
                            password=password,
                            db=db,
                            pool_size=pool_size,
                            prefix=prefix)
except Exception as Error:
```

```
    log.write_logs(log.logs_levels.ERROR.value,
                   'Could not connect to FSM storage. ' \
                   'Check settings file. Error: {}'.format(Error))
class States(StatesGroup):
    MAIN_MENU = State()
    CHOOSE_WAY_SHOW_SCHEDULE = State()
    SEARCH_ENTER_GROUP_CIPHER = State()
    SEARCH_BY_GROUP_CHOOSE_DAY = State()
    SEARCH_BY_GROUP_PICK_DATE = State()
    SEARCH_ENTER_TEACHER_NAME = State()
    SEARCH_BY_TEACHER_CHOOSE_DAY = State()
    SEARCH_BY_TEACHER_PICK_DATE = State()
    UPLOAD_FILES_MENU = State()
    UPLOAD_GRAPH_FILE = State()
    UPLOAD_SCHEDULE_FILE = State()
    UPLOAD_CHANGES_FILE = State()
    ENTER_GRAPH_SHEET = State()
    ENTER_SCHEDULE_SHEET = State()
    CONFIRM_UPDATE_GRAPH = State()
    CONFIRM_UPDATE_SCHEDULE = State()
    CONFIRM_UPDATE_CHANGES = State()
    ENTER_ADVERTISEMENT = State()
    CONFIRM_ADVERTISEMENT = State()
    SETTINGS_MENU = State()
    MAILING_BY_GROUP_MENU = State()
    MAILING_CHOOSE_GROUP = State()
    MAILING_BY_GROUP_CHOOSE_TIME = State()
    MAILING_BY_TEACHER_MENU = State()
    MAILING_CHOOSE_TEACHER = State()
    MAILING_BY_TEACHER_CHOOSE_TIME = State()

#-----------------
#database.connection.py
def load_db_settings():
    try:
        tree = ['mongoDB']
        tags = ['DBname', 'username', 'password']
        return xml_parser.get_list_tags_text(tree, tags)
    except AttributeError as error:
        log.write_logs(log.logs_levels.ERROR.value, error)
connection_string = 'mongodb+srv://{}:{}@krkmbot-s59lc.mongodb.net' \
                    '/{}?retryWrites=true&w=majority'
try:
    name, user, password = load_db_settings()
    client = AsyncIOMotorClient(connection_string.format(user, password, name))
except Exception as Error:
    log.write_logs(log.logs_levels.ERROR.value,
                   'Could not connect to MongoDB server. ' \
                   'Check settings file. Error: {}'.format(Error))

def get_conn():
    return client
```

```
#-----------------
#database.data_worker.py
async def insert_one_record(collection_name, record):
    client = get_conn()
    db = client[name]
    collection = db[collection_name]
    await collection.insert_one(record)

async def insert_many_records(collection_name, records):
    client = get_conn()
    db = client[name]
    collection = db[collection_name]
    await collection.insert_many(records)

async def clear_collection(collection_name):
    client = get_conn()
    db = client[name]
    await db[collection_name].delete_many({})

async def delete_records(collection_name, params):
    client = get_conn()
    db = client[name]
    await db[collection_name].delete_many(params)

async def find_one_record(collection_name, params):
    client = get_conn()
    db = client[name]
    return await db[collection_name].find_one(params)

def find_many_records(collection_name, params):
    client = get_conn()
    db = client[name]
    return db[collection_name].find(params)

async def update_one_record(collection_name, filter_, params):
    client = get_conn()
    db = client[name]
    await db[collection_name].update_one(filter_, params)

#-----------------
#handlers.callback_get_schedule_menu.py

def transform_name(name):
    if '.' in name:
        str_list = list(name)
        for i in [0, len(str_list) - 4, len(str_list) - 2]:
            str_list[i] = str_list[i].upper()
        return ''.join(str_list)
    elif '-' in name:
        str_list = name.split('-')
        return '-'.join(item.capitalize() for item in str_list)
```

```
async def handle_group_cipher(call: types.CallbackQuery):
    await bot.answer_callback_query(call.id)
    params = {
        '_id': call.message.chat.id
    }
    document = await data_worker.find_one_record(users_collection_name, params)
    if document:
        last_group = document.get('Last_group')
        if last_group:
            await bot.send_message(call.message.chat.id,
                                   get_choose_proposed_option_msg(),
                                   reply_markup=reply_button.get_keyboard(last_group))
    msg = get_enter_group_cipher_msg()
    if is_graph_updating() or is_schedule_updating():
        msg += '\n' + get_may_be_errors_msg()
    await bot.edit_message_text(chat_id=call.message.chat.id,
                                message_id=call.message.message_id,
                                text=msg,
                                reply_markup=back_button.get_keyboard())
    await States.SEARCH_ENTER_GROUP_CIPHER.set()

async def handle_teacher_name(call: types.CallbackQuery):
    await bot.answer_callback_query(call.id)
    params = {
        '_id': call.message.chat.id
    }
    document = await data_worker.find_one_record(users_collection_name, params)
    if document:
        last_teacher = document.get('Last_teacher')
        if last_teacher:
            last_teacher = transform_name(last_teacher)
            await bot.send_message(call.message.chat.id,
                                   get_choose_proposed_option_msg(),
                                   reply_markup=reply_button.get_keyboard(last_teacher))
    msg = get_enter_teacher_name_msg()
    if is_graph_updating() or is_schedule_updating():
        msg += '\n' + get_may_be_errors_msg()
    await bot.edit_message_text(chat_id=call.message.chat.id,
                                message_id=call.message.message_id,
                                text=msg,
                                reply_markup=back_button.get_keyboard())
    await States.SEARCH_ENTER_TEACHER_NAME.set()

async def handle_backto_get_schedule_menu(call: types.CallbackQuery, state: FSMContext):
    await bot.answer_callback_query(call.id)
    await bot.edit_message_text(chat_id=call.message.chat.id,
                                message_id=call.message.message_id,
                                text=emojize('Повертаю до попереднього меню :ok_hand:'))
    state = await state.get_state()
    state = state.split(':')
    if state[1] in [States.SEARCH_ENTER_GROUP_CIPHER._state,
                    States.SEARCH_ENTER_TEACHER_NAME._state]:
```

```
        message = await bot.send_message(call.message.chat.id,
                        'В процесі..',
                        reply_markup=types.ReplyKeyboardRemove())
        await message.delete()
    await bot.send_message(call.message.chat.id,
                        get_choose_way_get_schedule_msg(),
                        reply_markup=get_schedule_menu.get_keyboard())
    await States.CHOOSE_WAY_SHOW_SCHEDULE.set()

async def enter_group_cipher(message: types.Message):
    group = ''.join(message.text.lower().split())
    group = group.replace('–', '-')
    group = group.replace('—', '-')
    params = {
        'Group': group
    }
    if not await data_worker.find_one_record(graph_collection_name, params) \
        and not await data_worker.find_one_record(schedule_collection_name, params):
        await find_similar_groups(message, group)
    else:
        await message.reply(emojize('Чудово, група знайдена :smile:'),
                        reply_markup=types.ReplyKeyboardRemove())
        params = {
            '_id': message.chat.id
        }
        document = await data_worker.find_one_record(users_collection_name, params)
        if document:
            await data_worker.update_one_record(users_collection_name,
                                        params,
                                        {'$set': {'Last_group': group}})
        else:
            document = {
                '_id': message.chat.id,
                'Last_group': group
            }
            await data_worker.insert_one_record(users_collection_name, document)
        await bot.send_message(message.chat.id,
                           get_search_schedule_choose_day_msg(),
                           reply_markup=schedule_days_menu.get_keyboard())
        await States.SEARCH_BY_GROUP_CHOOSE_DAY.set()

async def find_similar_groups(message: types.Message, group):
    groups = []
    params = {
        'Group': {
            '$regex': '.*{}.*'.format(group)
        }
    }
    cursor = data_worker.find_many_records(schedule_collection_name, params)
    for document in await cursor.to_list(length=None):
        group = document.get('Group')
        if not group in groups and group is not None:
```

```
            groups.append(group)
    cursor = data_worker.find_many_records(graph_collection_name, params)
    for document in await cursor.to_list(length=None):
        group = document.get('Group')
        if not group in groups and group is not None:
            groups.append(group)
    if not groups:
        await message.reply(emojize(':pensive: На жаль, не знайдено жодних'
                        ' відомостей про введену групу.'
                        ' Перевірте, чи правильно Ви ввели шифр.'))
    else:
        def_locale = locale.getlocale(locale.LC_COLLATE)
        locale.setlocale(locale.LC_COLLATE, 'uk_UA')
        groups.sort(key=cmp_to_key(locale.strcoll))
        locale.setlocale(locale.LC_COLLATE, def_locale)
        await message.reply(emojize(':thumbsup: Знайдено декілька збігів.' \
                        ' Оберіть, яку саме групу Ви мали на увазі.'),
                        reply_markup=reply_button.get_keyboard_by_list(groups))

async def enter_teacher_name(message: types.Message):
    teacher = ' '.join(message.text.lower().split())
    params_num = {
        'Numerator.Teacher': teacher
    }
    params_den = {
        'Denominator.Teacher': teacher
    }

    if not await data_worker.find_one_record(schedule_collection_name, params_num) \
        or not await data_worker.find_one_record(schedule_collection_name, params_den):
        await find_similar_teachers(message, teacher)
    else:
        await message.reply(emojize('Чудово, відомості про викладача знайдені :smile:'),
                        reply_markup=types.ReplyKeyboardRemove())
        params = {
            '_id': message.chat.id
        }
        document = await data_worker.find_one_record(users_collection_name, params)
        if document:
            await data_worker.update_one_record(users_collection_name,
                                        params,
                                        {'$set': {'Last_teacher': teacher}})
        else:
            document = {
                '_id': message.chat.id,
                'Last_teacher': teacher
            }
            await data_worker.insert_one_record(users_collection_name, document)
        await bot.send_message(message.chat.id,
                           get_search_schedule_choose_day_msg(),
                           reply_markup=schedule_days_menu.get_keyboard())
        await States.SEARCH_BY_TEACHER_CHOOSE_DAY.set()
```

```
async def find_similar_teachers(message: types.Message, teacher):
    teachers = []
    params_num = {
        'Numerator.Teacher': {
            '$regex': '.*{}.*'.format(teacher)
        }
    }
    cursor = data_worker.find_many_records(schedule_collection_name, params_num)
    for document in await cursor.to_list(length=None):
        teacher = document.get('Numerator').get('Teacher')
        if '-' in teacher:
            continue
        if not teacher in teachers and teacher is not None:
            teachers.append(teacher)
    params_den = {
        'Denominator.Teacher': {
            '$regex': '.*{}.*'.format(teacher)
        }
    }
    cursor = data_worker.find_many_records(schedule_collection_name, params_den)
    for document in await cursor.to_list(length=None):
        teacher = document.get('Denominator').get('Teacher')
        if '-' in teacher:
            continue
        if not teacher in teachers and teacher is not None:
            teachers.append(teacher)
    if not teachers:
        await message.reply(emojize(':pensive: На жаль, не знайдено жодних'
                            ' відомостей про викладача.'
                            ' Перевірте, чи правильно Ви ввели його ПІБ.'))
    else:
        def_locale = locale.getlocale(locale.LC_COLLATE)
        locale.setlocale(locale.LC_COLLATE, 'uk_UA')
        teachers.sort(key=cmp_to_key(locale.strcoll))
        locale.setlocale(locale.LC_COLLATE, def_locale)
        for i in range(0, len(teachers)):
            teachers[i] = transform_name(teachers[i])
        await message.reply(emojize(':thumbsup: Знайдено декілька збігів.' \
                            ' Оберіть, кого саме Ви мали на увазі.'),
                            reply_markup=reply_button.get_keyboard_by_list(teachers))

#-----------------
#handlers.callback_group_schedule_menu.py
async def handle_search_by_group(call: types.CallbackQuery):
    await bot.answer_callback_query(call.id)
    await bot.edit_message_text(chat_id=call.message.chat.id,
                                message_id=call.message.message_id,
                                text = emojize('Починаю збирати інформацію :ok_hand:'))
    if call.data == 'search_today':
        date = datetime.now()
    elif call.data == 'search_tomorrow':
```

```
        date = datetime.now()
        date += timedelta(days=1)
    await bot.send_message(call.message.chat.id,
                           await gather_information_by_group(call.message.chat.id, date),
                           parse_mode='HTML')
    await bot.send_message(call.message.chat.id,
                           get_search_schedule_choose_day_msg(),
                           reply_markup=schedule_days_menu.get_keyboard())

async def handle_search_by_group_pick_date(call: types.CallbackQuery):
    await bot.answer_callback_query(call.id)
    date = datetime.date(datetime.now())
    #date = date.replace(2019, 12, 30)
    #date = date.replace(2020, 1, 2)
    await bot.edit_message_reply_markup(chat_id=call.message.chat.id,
                                        message_id=call.message.message_id,
                                        reply_markup=pick_date.get_keyboard(date))
    await States.SEARCH_BY_GROUP_PICK_DATE.set()

def get_day_by_number(num):
    days = {
        1: 'понеділок',
        2: 'вівторок',
        3: 'середа',
        4: 'четвер',
        5: "п'ятниця",
        6: 'субота',
        7: 'неділя'
    }
    return days.get(num)

async def gather_information_by_group(user_id, date, mailing=False, group=None):
    if not group:
        group = await get_user_param(user_id, 'Last_group')
    result_msg, numerator = await get_graph_msg(group, date)
    schedule_msg = await get_schedule_msg(group, date, numerator)
    changes_msg = await get_changes_msg(group, date)
    if schedule_msg is None and changes_msg is None and mailing:
        return None
    if numerator == 0:
        numer_msg = 'чисельник'
    elif numerator == 1:
        numer_msg = 'знаменник'
    else:
        numer_msg = 'чисельник/знаменник не визначено'
    result_msg += emojize('<strong>\n\n:clipboard: ' \
                  'Розклад занять (<em>{}</em>):</strong>').format(numer_msg)
    if numerator is None:
        result_msg += '<strong><em> не знайдено інформації про розклад групи' \
              ' (чисельник/знаменник не визначено).</em></strong>'
    elif schedule_msg is None:
        result_msg += '<strong><em> не знайдено інформації про розклад групи.</em></strong>'
```

```
    else:
        result_msg += schedule_msg
    result_msg += emojize('<strong>\n\n:memo: Заміни занять: </strong>')
    if changes_msg:
        result_msg += changes_msg
    else:
        result_msg += '<strong><em>не знайдено.</em></strong>'
    return result_msg

async def get_user_param(user_id, param):
    params = {
        '_id': user_id
    }
    group = await data_worker.find_one_record(users_collection_name, params)
    return group.get(param)

def get_header_msg(group, date):
    msg = '<strong>Група: {}.\n</strong>'.format(group)
    msg += '<strong>Дата: {}.\n</strong>'.format(datetime.date(date).strftime('%d.%m.%Y'))
    msg += '<strong>День тижня: {}.</strong>'.format(get_day_by_number(date \
                                                    .isoweekday()))
    if is_graph_updating() or is_schedule_updating():
        msg += '\n' + get_may_be_errors_msg()
    msg += '\n\n'
    return msg

async def get_graph_msg(group, date):
    msg = get_header_msg(group, date)
    msg += emojize(':date: За графіком навчального процесу: ')
    date = date.replace(tzinfo=timezone.utc).timestamp()
    result_graph = await get_from_graph_info(date, group)
    if not result_graph:
        msg += '<strong><em>не визначено</em></strong>.'
        result_graph = await get_from_graph_info(date)
    else:
        msg += '<strong><em>{}</em></strong>.'.format(result_graph.get('Option'))
    if result_graph:
        numerator = result_graph.get('Numerator')
    else:
        numerator = None
    return msg, numerator

async def get_from_graph_info(date, group=None):
    if group:
        params = {
            'Group': group,
            'Start_date': {'$lte': date},
            'Last_date': {'$gte':date}
        }
    else:
        params = {
            'Start_date': {'$lte': date},
```

```
            'Last_date': {'$gte':date}
        }
    return await data_worker.find_one_record(graph_collection_name, params)

async def get_schedule_msg(group, date, numerator):
    weekday = date.isoweekday()
    result_schedule = await get_from_schedule_info(group, weekday)
    if not result_schedule:
        return None
    elif numerator is None:
        return None
    else:
        msg = ''
        for item in result_schedule:
            if numerator == 0:
                numerator = 'Numerator'
            elif numerator == 1:
                numerator = 'Denominator'
            subject = item.get(numerator).get('Subject')
            teacher = item.get(numerator).get('Teacher')
            classroom = item.get(numerator).get('Classroom')
            lesson = item.get('Lesson')
            timing = item.get('Timing')
            if not timing:
                timing = 'час не визначено'
            if not subject:
                msg += emojize('\n\n:clock3:<strong> {} ({} пара)</strong>\nВікно.') \
                       .format(timing, lesson)
            else:
                teacher = transform_name(teacher)
                msg += emojize('\n\n:clock3:<strong> {} ({} пара)</strong>' \
                       '\n{}. Викладач: {}, аудиторія: {}.') \
                       .format(timing,
                               lesson,
                               subject,
                               teacher,
                               classroom)
        return msg

async def get_from_schedule_info(group, weekday):
    params = {
        'Group': group,
        'Day': weekday
    }
    cursor = data_worker.find_many_records(schedule_collection_name, params)
    lessons = []
    for document in await cursor.to_list(length=10):
        lessons.append(document)
    return lessons

async def get_changes_msg(group, date):
    date = date.replace(hour=0, minute=0, second=0, microsecond=0,
```

```
                    tzinfo=timezone.utc).timestamp()
    params = {
        'Group': group,
        'Date': date
    }
    cursor = data_worker.find_many_records(changes_collection_name, params)
    i = 1
    msg = ''
    for document in await cursor.to_list(length=10):
        teacher = transform_name(document.get('Teacher'))
        reason = document.get('Reason')
        if reason.count('.') == 2:
            reason = transform_name(reason)
        msg += '\n{}) № пари: <strong>{}</strong>. Хто заміняє: <strong>{}</strong>,' \
               ' кого заміняє/причина: <strong>{}</strong>, аудиторія: {}.'.format(i,
                                            document.get('Lesson'),
                                            teacher,
                                            reason,
                                            document.get('Classroom'))
        i += 1
    if i == 1:
        return None
    return msg

#-----------------
#handlers.callback_mailing_group.py
async def handle_change_time_group(call: types.CallbackQuery):
    await bot.answer_callback_query(call.id)
    params = {
        '_id': call.message.chat.id
    }
    document = await data_worker.find_one_record(users_collection_name, params)
    time = hours = minutes = None
    if document:
        document = document.get('MailingByGroup')
        if document:
            time = document.get('Time')
    if time:
        hours = time.get('Hours')
        minutes = time.get('Minutes')
    if hours is None or minutes is None:
        time = datetime.time(16, 00)
    else:
        time = datetime.time(hours, minutes)
    await bot.edit_message_text(chat_id=call.message.chat.id,
                                message_id=call.message.message_id,
                                text=get_choose_time_mailing_msg(),
                                parse_mode='HTML',
                                reply_markup=pick_time.get_keyboard(time))
    await States.MAILING_BY_GROUP_CHOOSE_TIME.set()

async def handle_change_group(call: types.CallbackQuery):
```

```
    await bot.answer_callback_query(call.id)
    msg = get_enter_group_cipher_msg()
    if is_graph_updating() or is_schedule_updating():
        msg += '\n' + get_may_be_errors_msg()
    await bot.edit_message_text(chat_id=call.message.chat.id,
                                message_id=call.message.message_id,
                                text=msg,
                                reply_markup=back_button.get_keyboard())
    await States.MAILING_CHOOSE_GROUP.set()

async def handle_turn_mailing_group(call: types.CallbackQuery):
    params = {
        '_id': call.message.chat.id
    }
    document = await data_worker.find_one_record(users_collection_name, params)
    group = hours = minutes = enabled = None
    if document:
        document = document.get('MailingByGroup')
        if document:
            enabled = document.get('Enabled')
            group = document.get('Group')
            time = document.get('Time')
            if time:
                hours = time.get('Hours')
                minutes = time.get('Minutes')
        if enabled:
            await bot.answer_callback_query(call.id,
                                text='Щоденна розсилка за групою вимкнена!',
                                show_alert=True)
            await data_worker.update_one_record(users_collection_name,
                                        params,
                                        {'$set': {'MailingByGroup.Enabled': False}})
            remove_mailing_by_group(call.message.chat.id)
            return await handle_mailing_by_group(call)
    if hours is None or minutes is None:
        return await bot.answer_callback_query(call.id,
                                text='Ви не встановили час для розсилки!',
                                show_alert=True)
    elif not group:
        return await bot.answer_callback_query(call.id,
                                text='Ви не встановили групу для розсилки!',
                                show_alert=True)
    await bot.answer_callback_query(call.id,
                                text='Щоденна розсилка за групою {} увімкнена!' \
                                    .format(group),
                                show_alert=True)
    await data_worker.update_one_record(users_collection_name,
                                params,
                                {'$set': {'MailingByGroup.Enabled': True}})
    add_mailing_by_group(call.message.chat.id, group, hours, minutes)
    return await handle_mailing_by_group(call)
```

```
async def handle_enter_group(message: types.Message):
    group = ''.join(message.text.lower().split())
    group = group.replace('–', '-')
    group = group.replace('—', '-')
    params = {
        'Group': group
    }
    if not await data_worker.find_one_record(graph_collection_name, params) \
        and not await data_worker.find_one_record(schedule_collection_name, params):
        await find_similar_groups(message, group)
    else:
        await message.reply(emojize('Чудово, група знайдена :smile:'),
                            reply_markup=types.ReplyKeyboardRemove())
        params = {
            '_id': message.chat.id
        }
        document = await data_worker.find_one_record(users_collection_name, params)
        if document:
            await data_worker.update_one_record(users_collection_name,
                                                params,
                                                {'$set': {'MailingByGroup.Group': group}})
            if document.get('MailingByGroup') \
                and document.get('MailingByGroup').get('Enabled') is True:
                change_mailing_group(message.chat.id, group)
        else:
            document = {
                '_id': message.chat.id,
                'MailingByGroup': {
                    'Group': group
                }
            }
            await data_worker.insert_one_record(users_collection_name, document)
        enabled, msg = await find_mailing_group_info(message.chat.id)
        await bot.send_message(message.chat.id,
                               msg,
                               reply_markup=mailing_menu.get_keyboard('group', enabled),
                               parse_mode='HTML')
        await States.MAILING_BY_GROUP_MENU.set()

#-----------------
#handlers.callback_make_advertisement.py
async def handle_advertisement_text(message: types.Message):
    if len(message.text) > 3000:
        return await message.reply(emojize(':exclamation: Завеликий текст оголошення!' \
                ' Його довжина повинна бути не більше, ніж 3000 символів.' \
                '\nВведіть оголошення ще раз.'))
    await bot.send_message(message.chat.id,
            emojize(':round_pushpin: Оголошення виглядатиме наступним чином.' \
                    '\nПідтвердіть його відправку.'))
    msg = emojize('<strong>Увага :bangbang: Важливе оголошення :bangbang:\n</strong>{}' \
        '\n\nВідправник: ').format(message.text)
    if message.chat.first_name:
```

```
        msg += '{} '.format(message.chat.first_name)
    if message.chat.last_name:
        msg += message.chat.last_name
    await bot.send_message( message.chat.id,
                            msg,
                            parse_mode='HTML',
                            reply_markup=confirm_action.get_keyboard())
    await States.CONFIRM_ADVERTISEMENT.set()

async def handle_confirm_advertisement(call: types.CallbackQuery):
    await bot.answer_callback_query(call.id)
    await bot.send_chat_action(call.message.chat.id, 'typing')
    msg = emojize('Чудово :ok_hand:\nПочинаю розсилати оголошення користувачам.')
    await bot.edit_message_text(chat_id=call.message.chat.id,
                                message_id=call.message.message_id,
                                text=msg)
    total_users = await send_advertisement(call.message.chat.id, call.message.text)
    msg = emojize('Розсилання оголошення завершено :ok_hand:\nОголошення отримали {}' \
        ' користувача(-ів).' \
        '\nНадсилаю Вам моє головне меню :wink:').format(str(total_users))
    log_user_action(logs_levels.INFO.value,
                    call.message.chat.id,
                    call.message.chat.username,
                    call.message.chat.first_name,
                    call.message.chat.last_name,
                    'made an advertisement.')
    await bot.edit_message_text(chat_id=call.message.chat.id,
                                message_id=call.message.message_id,
                                text=msg)
    await cmd_menu(call.message)

async def handle_reject_advertisement(call: types.CallbackQuery):
    await bot.answer_callback_query(call.id)
    await bot.edit_message_text(chat_id=call.message.chat.id,
                        message_id=call.message.message_id - 1,
                        text=emojize(':no_entry_sign: Ви скасували відправку оголошення.'))
    await bot.edit_message_text(chat_id=call.message.chat.id,
                                message_id=call.message.message_id,
                                text=get_enter_advertisement_msg(),
                                parse_mode='HTML',
                                reply_markup=back_button.get_keyboard())
    await States.ENTER_ADVERTISEMENT.set()

async def send_advertisement(sender, message):
    users_ID = await storage.get_states_list()
    mailing_ID = []
    for ID in users_ID:
        ID = int(ID[1])
        if ID == sender or ID in mailing_ID:
            continue
        document = {
            '_id': ID
```

```
        }
        result = await data_worker.find_one_record(users_collection_name, document)
        if not result or result.get('MailingEnabled') is not False:
            mailing_ID.append(ID)
    sent_mails = 0
    for ID in mailing_ID:
        try:
            await bot.send_message(ID, message)
            sent_mails += 1
            await sleep(0.08)
        except BotBlocked:
            pass
    return sent_mails

#-----------------
#handlers.upload_graph.py
async def handle_messages_when_uploading(message: types.Message):
    await message.reply(get_graph_already_uploading_msg(), parse_mode='HTML')

async def handle_graph_file(message: types.Message):
    if not is_file_excel(message.document.mime_type):
        return await message.reply(get_wrong_file_msg(), parse_mode='HTML')
    try:
        await download_data_file(message.document.file_id,
                                 message.document.file_name)
        save_file_info('graphFileName__{}'.format(message.chat.id),
                       message.document.file_name)
    except AttributeError as error:
        msg = str(error)
    else:
        msg = emojize(':memo: Введіть назву листа, на якому знаходиться' \
            ' <strong><em>графік навчального процесу</em></strong>.')
    await bot.send_message( message.chat.id,
                            msg,
                            parse_mode='HTML',
                            reply_markup=back_button.get_keyboard())
    await States.ENTER_GRAPH_SHEET.set()
    await bot.send_message( message.chat.id,
                            '\nАбо Ви можете обрати запропонований варіант.',
                            parse_mode='HTML',
                            reply_markup=reply_button.get_keyboard('Графік'))

async def handle_sheet_name(message: types.Message):
    founded_sheet = is_sheet_exists('graphFileName__{}' \
                                    .format(message.chat.id),
                                    message.text)
    if not founded_sheet:
        return await message.reply(emojize(':exclamation:На жаль, вказаний лист' \
                                    ' не знайдено!' \
                                    ' Перевірте його назву та введіть ще раз.'))
    await message.reply(emojize(':ok_hand: Вказаний лист знайдено!'),
                        reply_markup=types.ReplyKeyboardRemove())
```

```
    msg = emojize(':round_pushpin: Ви впевнені, що хочете оновити' \
        ' <strong><em>графік навчального процесу?</em></strong>' \
        ' Всі поточні дані будуть видалені.' \
        '\nЛист з графіком: <strong>{}</strong>').format(founded_sheet)
    await States.CONFIRM_UPDATE_GRAPH.set()
    await bot.send_message( message.chat.id,
                            msg,
                            parse_mode='HTML',
                            reply_markup=confirm_action.get_keyboard())

async def handle_callbacks_when_uploading(call: types.CallbackQuery):
    await bot.answer_callback_query(call.id)
    await bot.send_message( call.message.chat.id,
                            get_graph_already_uploading_msg(),
                            parse_mode='HTML')

async def handle_back_button(call: types.CallbackQuery):
    await bot.answer_callback_query(call.id)
    new_message = await bot.send_message(call.message.chat.id,
                                         'Процес...',
                                         reply_markup=types.ReplyKeyboardRemove())
    await new_message.delete()
    await bot.edit_message_text(chat_id=call.message.chat.id,
                                message_id=call.message.message_id,
                                text=get_upload_graph_msg(),
                                parse_mode='HTML',
                                reply_markup=back_button.get_keyboard())
    await States.UPLOAD_GRAPH_FILE.set()
    remove_data_file('graphFileName__{}' \
                     .format(call.message.chat.id))

async def handle_confirm_update_graph(call: types.CallbackQuery):
    await bot.answer_callback_query(call.id)
    global is_graph_already_parsing
    is_graph_already_parsing = True
    msg = emojize('Чудово :ok_hand:\nПочинаю оновлювати графік навчального процесу.' \
        ' Це довгий процес, тому можливо доведеться зачекати.')
    await bot.edit_message_text(chat_id=call.message.chat.id,
                                message_id=call.message.message_id,
                                text=msg)
    await bot.send_chat_action(call.message.chat.id, 'upload_document')
    target_text = 'Лист з графіком: '
    sheet = call.message.text[call.message.text.rfind(target_text) + len(target_text):]
    msg = ''
    try:
        await parse_graph_file(get_file_path('graphFileName__{}' \
                                             .format(call.message.chat.id)),
                               sheet)
    except ParsingError as error:
        msg = emojize(':bangbang: ' + str(error) + \
            '\nСпробуйте ще раз.' \
            '\nПовертаю Вас до попереднього меню :wink:')
```

```
    except Exception as error:
        log.write_logs(log.logs_levels.ERROR.value,
                       'Could not update graph. Error: {}'.format(error))
        msg = get_undefined_error_msg()
    else:
        log.log_user_action(log.logs_levels.INFO.value,
                            call.message.chat.id,
                            call.message.chat.username,
                            call.message.chat.first_name,
                            call.message.chat.last_name,
                            'uploaded new graph of studying proccess.')
        msg = emojize('Оновлення графіка навчального процесу' \
            ' успішно завершено :ok_hand:\nПовертаю Вас до попереднього меню :wink:')
    is_graph_already_parsing = False
    await bot.edit_message_text(chat_id=call.message.chat.id,
                                message_id=call.message.message_id,
                                text=msg)
    await bot.send_message( call.message.chat.id,
                            get_upload_files_menu_msg(),
                            reply_markup=upload_files_menu.get_keyboard())
    await States.UPLOAD_FILES_MENU.set()
    remove_data_file('graphFileName__{}' \
                     .format(call.message.chat.id))

async def handle_reject_update_graph(call: types.CallbackQuery):
    await bot.answer_callback_query(call.id)
    await bot.edit_message_text(chat_id=call.message.chat.id,
                                message_id=call.message.message_id,
                                text=get_upload_graph_msg(),
                                parse_mode='HTML',
                                reply_markup=back_button.get_keyboard())
    await States.UPLOAD_GRAPH_FILE.set()
    remove_data_file('graphFileName__{}' \
                     .format(call.message.chat.id))

#-----------------
#handlers.upload_files_service.py
def is_file_excel(file_type):
    extensions_list = ['application/vnd.openxmlformats-officedocument.spreadsheetml.sheet',
                       'application/vnd.ms-excel.sheet.macroEnabled.12',
                       'application/vnd.openxmlformats-officedocument.spreadsheetml.template',
                       'application/vnd.ms-excel.template.macroEnabled.12']
    if file_type in extensions_list:
        return True
    else:
        return False

def get_file_path(tag):
    try:
        tree = [files_info_tag, tag]
        file_name = xml_parser.get_tag_text(tree)
        path = os.path.join(os.getcwd(), dir_with_data_files, file_name)
```

```
        return path
    except AttributeError as error:
        log.write_logs(log.logs_levels.ERROR.value, error)

def is_sheet_exists(target_tag_name, target_sheet):
    with open(get_file_path(target_tag_name), "rb") as file:
        in_mem_file = io.BytesIO(file.read())
    workbook = load_workbook(filename=in_mem_file,
                             read_only=True,
                             data_only=True,
                             keep_links=False)
    for sheet in workbook.sheetnames:
        if ' '.join(sheet.lower().split()) == ' '.join(target_sheet.lower().split()):
            return sheet
    return False

async def download_data_file(file_id, file_name):
    try:
        file_info = await bot.get_file(file_id)
        source = os.path.join(os.getcwd(), dir_with_data_files)
        if not os.path.exists(source):
            os.mkdir(source)
        source = os.path.join(source, file_name)
        await bot.download_file_by_id(file_info.file_id, source)
    except IOError as Error:
        log.write_logs(log.logs_levels.ERROR.value,
              'Could not get from user graph file. Error: {}'.format(Error))
        raise AttributeError(get_undefined_error_msg())

def save_file_info(tag_file_name, file_name):
    try:
        tree = [files_info_tag]
        if not xml_parser.is_tag_exists(tree):
            xml_parser.create_new_tag([], files_info_tag)
        xml_parser.create_new_tag(tree, tag_file_name, text=file_name)
    except AttributeError as error:
        log.write_logs(log.logs_levels.ERROR.value, error)
        raise AttributeError(get_undefined_error_msg())

def remove_data_file(tag):
    try:
        tree = [files_info_tag, tag]
        file_path = os.path.join(os.getcwd(),
                 dir_with_data_files,
                 xml_parser.get_tag_text(tree))
        if os.path.exists(file_path):
            os.remove(file_path)
        xml_parser.delete_tag(tree)
    except AttributeError as error:
        log.write_logs(log.logs_levels.ERROR.value, error)
```

ДОДАТОК Б

ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
Диплом_ doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Диплом_.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Program.rar	Архів. Містить коди програми і откомпільовану програму
Презентація	
Презентація_.ppt	Презентація кваліфікаційної роботи