Державний торговельно-економічний університет
Кафедра інженерії програмногозабезпечення та кібербезпеки

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Автоматизація інформаційної підтримки освітнього процесу засобами асинхронного програмування»

Студента 2м курсу, 2 групи,
спеціальності 121 «Інженерія
програмного забезпечення»
освітня програма «Інженерія
програмного забезпечення» Полігушка Андрія
Юрійовича

_____
підпис студента

Науковий керівник
кандидат економічних наук,
доцент кафедри інженерії
програмного забезпечення та
кібербезпеки Тищенко Дмитро
Олександрович

_____
підпис керівника

Гарант освітньої програми
кандидат педагогічних наук,
доцент кафедри інженерії
програмного забезпечення та
кібербезпеки Котенко Наталія
Олексіївна

_____
підпис гаранта

КИЇВ – 2023

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь магістр
Спеціальність 121 «Інженерія програмного забезпечення»

Затверджую
Зав. кафедри інженерії програмного забезпечення та кібербезпеки
Криворучко О. В.
«13» грудня 2022 р.

Завдання
на випускну кваліфікаційну роботу студентові

Полігушку Андрію Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема випускної кваліфікаційної роботи «Автоматизація інформаційної підтримки освітнього процесу засобами асинхронного програмування»

Затверджена наказом ректора від «06» грудня 2022 р. № 3285

2. Строк здачі студентом закінченої роботи 1 грудня 2023 р.

3. Цільова установка та вихідні дані до роботи

Мета роботи: розробка і впровадження інноваційної інформаційної системи для автоматизації та поліпшення управління та взаємодії між викладачами та студентами у вищих навчальних закладах з використанням асинхронного програмування

Об'єкт дослідження – система інформаційної підтримки освітнього процесу в вищих навчальних закладах

Предмет дослідження – Розробка та впровадження функціональності бота для підтримки комунікації між вчителями та студентами, спрощення відстеження та оцінювання домашніх завдань, а також поліпшення доступу до інформації про спеціальності та розклад занять за допомогою асинхронного програмування

4. Консультанти роботи із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускної кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. СУЧАСНІ ТЕНДЕНЦІЇ АВТОМАТИЗАЦІЇ УПРАВЛІНСЬКОГО І НАВЧАЛЬНОГО ПРОЦЕСІВ

1.1. Аналіз сучасних тенденцій автоматизації управлінських процесів у закладах вищої освіти

1.1.1. Застосування інформаційних технологій у процесі управління

1.1.2. Використання електронних систем управління навчальним процесом

1.1.3. Автоматизація процесу планування та контролю за навчальним процесом

1.2. Тенденції використання інформаційних технологій у навчальному процесі

1.2.1. Електронні підручники та навчальні посібники

1.2.2. Відеоуроки та вебінари

1.2.3. Електронні системи контролю знань

1.3. Переваги та недоліки використання автоматизації управлінського та навчального процесів

1.3.1. Переваги застосування інформаційних технологій

1.3.2. Недоліки автоматизації управління та навчання

1.4. Висновки до першого розділу

РОЗДІЛ 2. МЕТОДИ ТА ЗАСОБИ АВТОМАТИЗАЦІЇ ІНФОРМАЦІЙНОЇ ПІДТРИМКИ ОСВІТНЬОГО ПРОЦЕСУ З ВИКОРИСТАННЯМ АСИНХРОННОГО ПРОГРАМУВАННЯ

2.1. Обґрунтування вибору способів рішення задачі

2.1.1. Визначення вимог і потреб освітнього процесу

2.1.2. Огляд і порівняння різних підходів до автоматизації інформаційної підтримки освітнього процесу

2.1.3. Обґрунтування вибору асинхронного програмування як підходу для автоматизації інформаційної підтримки освітнього процесу

2.2. Засоби розробки

2.2.1. Вибір мови програмування для реалізації асинхронного програмування

6. Календарний план виконання роботи

№ пор.	Назва етапів випускної кваліфікаційної роботи	Строк виконання етапів роботи	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускної кваліфікаційної роботи*	*07.11.2022*	*07.11.2022*
2.	*Розробка та затвердження завдання на роботу магістра (стац/заоч)*	*13.12.2022*	*13.12.2022*
3.	*Вступ та перелік літературних джерел*	*24.02.2023*	*24.02.2023*
4.	*Розробка технічного завдання*	*15.03.2023*	*15.03.2023*
5.	*Розділ 1. Огляд методів захисту даних*	*10.04.2023*	*10.04.2023*
6.	*Розділ 2. Аналіз та оцінка основних методів захисту особистого ключа*	*24.05.2023*	*24.05.2023*
7.	*Розділ 3. Підтвердження ефективності використання ЕЦП*	*06.09.2023*	*06.09.2023*
8.	*Розробка програми та методики тестування*	*18.10.2023*	*18.10.2023*
9.	*Написання наукової статті*	*17.05.2023*	*17.05.2023*
10.	*Керівництво користувача*	*25.10.2023*	*25.10.2023*
11.	*Висновки та пропозиції*	*01.11.2023*	*01.11.2023*
12.	*Здача випускної кваліфікаційної роботи на кафедру (перша перевірка)*	*06.11.2023*	*06.11.2023*
13.	*Підготовка автореферату та презентації доповіді*	*06.11.2023*	*06.11.2023*
14.	*Попередній захист випускної кваліфікаційної роботи*	*20.11.2023 – 24.11.2023*	*20.11.2023 – 24.11.2023*
15.	*Здача зброшурованої випускної кваліфікаційної роботи*	*01.12.2023*	*01.12.2023*
16.	*Зовнішнє рецензування випускної кваліфікаційної роботи*	*02.12.2023*	*02.12.2023*
17.	*Підготовка до публічного захисту випускної кваліфікаційної роботи*	*05.12.2023-06.12.2023*	*05.12.2023-06.12.2023*

7. Дата видачі завдання «13» грудня 2022 р.

8. Науковий керівник випускної кваліфікаційної роботи

Тищенко Д.О.

(прізвище, ініціали, підпис)

9. Гарант освітньої програми Котенко Н.О.

(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент Полігушко А.Ю.

(прізвище, ініціали, підпис)

11. Відгук керівника випускної кваліфікаційної роботи

Науковий керівник випускної кваліфікаційної роботи
_____Тищенко Д.О._____
(підпис, дата)

Відмітка про попередній захист _____
(ПІБ, підпис, дата)

12. Висновок про випускну кваліфікаційну роботу

Випускна кваліфікаційна робота студента _____Полігушко А.Ю._____
(прізвище, ініціали)
може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми _____Котенко Н.О._____
(прізвище, ініціали, підпис)

Завідувач кафедри _____Криворучко О. В._____
(підпис, прізвище, ініціали)

«_____»_____2023 р.

АНОТАЦІЯ

Відповідно до мети дослідження, робота присвячена розробці та впровадженню інформаційної системи для автоматизації та поліпшення управління та взаємодії між викладачами та студентами у вищих навчальних закладах.

В ході дослідження було проведено аналіз існуючих проблем у взаємодії між викладачами та студентами, виявлено необхідність впровадження асинхронного програмування для покращення доступу до інформації та оптимізації адміністративних процесів в навчальних закладах.

В результаті роботи розроблено та впроваджено бота у месенджер Telegram, який сприяє покращенню комунікації, ефективному веденню обліку домашніх завдань та полегшенню доступу до інформації про спеціальності та розклад занять.

Ключові слова: автоматизація, інформаційна система, асинхронне програмування.

ABSTRACT

According to the purpose of the study, this work is dedicated to the development and implementation of an information system aimed at automating and enhancing the management and interaction between teachers and students in higher education institutions.

Throughout the study, an analysis of existing challenges in the interaction between teachers and students was conducted, highlighting the necessity of implementing asynchronous programming to improve access to information and optimize administrative processes in educational institutions.

As a result of the research, a Telegram messenger bot was developed and implemented. This bot contributes to the improvement of communication, efficient homework tracking, and facilitates access to information about specialties and class schedules..

Keywords: automation, information system, asynchronous programming.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

LMS – Learning Management System

SQL – мова структурованих запитів

TOKEN – токен (унікальний ідентифікатор)

ЗМІСТ

					ДТЕУ 121 02-15.МР			
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*				
Зав. каф.		Криворучко О.В.		19.09.23	*Автоматизація інформаційної підтримки освітнього процесу засобами асинхронного програмування*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Тищенко Д.О.		19.09.23		*Зміст*	*3*	*41*
Гарант		Котенко Н.О.		19.09.23		*Факультет інформаційних технологій 2м курс, 2 група*		
Розробив		Полігушко А.Ю.		19.09.23	*Зміст*			

ВСТУП

Актуальність. Сучасний розвиток технологій невід'ємно впливає на сферу освіти, змінюючи та вдосконалюючи підходи до навчання та взаємодії між викладачами та студентами. У контексті цього, актуальність роботи обумовлена необхідністю вдосконалення інформаційної підтримки освітнього процесу з використанням сучасних засобів асинхронного програмування.

Мета дослідження. Метою даної роботи є розробка та впровадження інформаційної системи, спрямованої на автоматизацію та поліпшення взаємодії між викладачами та студентами в умовах вищих навчальних закладів.

Об'єкт дослідження. Об'єктом дослідження є процес взаємодії викладачів та студентів у вищих навчальних закладах, який включає у себе адміністративні процеси, облік домашніх завдань та загальний доступ до необхідної інформації.

Предмет дослідження. Предметом дослідження є розробка та впровадження інформаційної системи з використанням засобів асинхронного програмування для оптимізації управління освітнім процесом.

Предметна область. Робота розглядає сферу вищої освіти та взаємодії між членами освітнього процесу, фокусуючись на використанні сучасних технологій для покращення комунікації, оптимізації адміністративних завдань та полегшенні доступу до необхідної інформації

					ДТЕУ 121 02-15.МР			
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*				
Зав. каф.		Криворучко О.В.		19.09.23	*Автоматизація інформаційної підтримки освітнього процесу засобами асинхронного програмування*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Тищенко Д.О.		19.09.23		*В*	*5*	*41*
Гарант		Котенко Н.О.		19.09.23		*Факультет інформаційних технологій 2м курс, 2 група*		
Розробив		Полігушко А.Ю.		19.09.23	*Вступ*			

РОЗДІЛ 1

СУЧАСНІ ТЕНДЕНЦІЇ АВТОМАТИЗАЦІЇ УПРАВЛІНСЬКОГО І НАВЧАЛЬНОГО ПРОЦЕСІВ

1.1. Аналіз сучасних тенденцій автоматизації управлінських процесів у закладах вищої освіти

1.1.1. Застосування інформаційних технологій у процесі управління

Застосування інформаційних технологій є необхідним елементом автоматизації управлінських процесів в закладах вищої освіти. Завдяки використанню різноманітних програмних засобів та інформаційних систем, можливо автоматизувати процеси збору, обробки та збереження інформації, що дозволяє значно полегшити роботу управлінських структур та підвищити їх ефективність.

Один із прикладів застосування інформаційних технологій у процесі управління в закладах вищої освіти - це використання систем електронного документообігу. Це дозволяє значно спростити процес обміну документами між управлінськими структурами, забезпечити швидкий доступ до необхідної інформації та підвищити рівень безпеки збереження даних.

Також інформаційні технології дозволяють автоматизувати процеси планування та контролю навчального процесу, що дозволяє ефективно використовувати ресурси закладу та забезпечити якісну підготовку студентів. Наприклад, системи електронного навчання дають можливість вчителям та

студентам здійснювати навчання та контроль знань в режимі онлайн, що значно полегшує процес навчання та контролює успішність студентів. Отже, використання інформаційних технологій є важливим елементом автоматизації управління в закладах вищої освіти, що сприяє поліпшенню якості управління та підвищенню ефективності процесів[1].

1.1.2. Використання електронних систем управління навчальним процесом

В сучасних умовах університети активно використовують електронні системи управління навчальним процесом. Ці системи забезпечують можливість зберігання, обробки та аналізу великих обсягів інформації, що сприяє покращенню якості навчання та оптимізації роботи викладачів.

Однією з найбільш поширених електронних систем управління навчальним процесом є Learning Management System (LMS), такою, наприклад, є «МІА: Освіта». Ця система дозволяє студентам отримувати доступ до електронних матеріалів, викладачам - контролювати навчальний процес та взаємодіяти зі студентами в онлайн-режимі, а адміністрації - контролювати виконання навчальних планів та аналізувати результати навчання.

Електронні системи управління навчальним процесом дозволяють автоматизувати процеси планування навчальних курсів, відстежування виконання навчальних завдань та оцінювання студентів. Це дозволяє покращити якість навчання, сприяє більш ефективному використанню часу викладачів та студентів, а також зменшує кількість адміністративної роботи, пов'язаної з управлінням навчальним процесом. Використання електронних систем управління навчальним процесом дозволяє створити зручні інструменти для спілкування між викладачами та студентами, а також

дозволяє відстежувати прогрес студентів та реагувати на їх потреби у реальному часі. Крім того, використання цих систем дозволяє забезпечити доступ до навчальних матеріалів з будь-якого пристрою[6].

1.1.3. Автоматизація процесу планування та контролю за навчальним процесом

Автоматизація процесу планування та контролю за навчальним процесом є ще однією важливою тенденцією в управлінні вищою освітою. Застосування інформаційних технологій в цих процесах дозволяє збільшити ефективність та точність планування, зменшити кількість помилок та зайвих витрат часу. Це дозволяє викладачам та адміністраторам швидко та зручно створювати розклади занять, враховуючи наявність аудиторій, наявність викладачів, потреби студентів тощо. Також автоматизація дозволяє забезпечити оптимальне використання ресурсів та максимальну віддачу від проведених занять.

Автоматизація процесу контролю за виконанням навчальних планів дозволяє вчителям швидко та ефективно відстежувати успішність студентів, виявляти проблеми та надавати необхідну допомогу. Крім того, це дозволяє студентам отримувати негайний зворотний зв'язок щодо їхньої успішності та відразу коригувати свої дії.

Контроль за навчальним процесом також може бути автоматизованим. Системи автоматичної оцінки та відстеження успішності студентів можуть використовувати алгоритми для оцінки та аналізу навчальних результатів, що дозволяє підвищити якість контролю та сприяти розвитку індивідуальних методів навчання. Наприклад, такі програмні засоби як "Moodle" або "Microsoft Teams" дозволяють викладачам створювати та організовувати віртуальні класи, додавати в них матеріали, викладати завдання та оцінювати

роботи студентів. Також ці системи можуть бути інтегровані з електронними бібліотеками та іншими базами даних для забезпечення швидкого та зручного доступу до інформації. Такі інструменти як автоматизовані інформаційні системи, програмне забезпечення та електронні платформи дозволяють забезпечити високий рівень якості навчання та ефективність управління в закладах вищої освіти. Вони дозволяють прискорити процес навчання, забезпечити доступ до великої кількості інформації та зменшити затрати на управління навчальним процесом.

1.2. Тенденції використання інформаційних технологій у навчальному процесі

1.2.1. Електронні підручники та навчальні посібники

Електронні підручники та навчальні посібники є однією з основних тенденцій використання інформаційних технологій у навчальному процесі в закладах вищої освіти. Ці матеріали можуть містити різноманітні форми навчання, такі як відеоуроки, аудіозаписи, інтерактивні тести та завдання, що дозволяє студентам засвоювати матеріал на свій власний темп та з будь-якого місця з доступом до Інтернету. Крім того, електронні підручники та навчальні посібники можуть бути оновлені та доповнені в режимі онлайн, що дозволяє студентам мати доступ до актуальної та свіжої інформації. Такі матеріали можуть бути дуже корисні для дистанційного навчання, яке стало набагато популярнішим в останні роки.

Також електронні підручники та навчальні посібники можуть бути інтегровані з іншими електронними системами управління навчальним процесом, такими як система електронного навчання чи система автоматизованого тестування, що забезпечує більш зручну та ефективну організацію навчального процесу для студентів та викладачів.

1.2.2. Відеоуроки та вебінари

Ще однією з тенденцій використання інформаційних технологій у навчальному процесі є використання відеоуроків та вебінарів. Це особливо актуально у зв'язку з пандемією COVID і російсько-українською війною, через що заклади освіти були вимушені перейти на дистанційну форму навчання.

Відеоуроки є корисним інструментом для вивчення нового матеріалу або поглиблення знань в певній темі. Вони можуть бути розміщені на різних платформах, таких як YouTube, або спеціалізовані платформи навчання, які надають доступ до відеоуроків від провідних фахівців у різних галузях, таких як Coursera або Prometheus.

Вебінари - це живі онлайн-презентації, під час яких провідний експерт пояснює певну тему та відповідає на запитання учасників. Вони можуть бути проведені за допомогою різних платформ, таких як Zoom, Google Meeting, Microsoft Teams, або спеціалізовані платформи для вебінарів.

Відеоуроки та вебінари забезпечують доступ до навчання з будь-якого місця, де є доступ до Інтернету. Вони дозволяють учням вивчати матеріал у власному темпі та зручний для них час. Крім того, вони можуть бути записані та використані в якості додаткового матеріалу для підготовки до екзаменів або інших важливих подій[2].

1.2.3. Електронні системи контролю знань

Електронні системи контролю знань є ще однією тенденцією використання інформаційних технологій у навчальному процесі. Ці системи можуть бути використані для проведення онлайн тестування, оцінювання

завдань, роботи з електронними тестами та іншими методами оцінювання знань студентів.

Електронні системи контролю знань дозволяють автоматизувати процес оцінювання та зберігання результатів, що полегшує роботу викладачів та адміністраторів навчального закладу. Крім того, такі системи можуть надавати додаткові можливості для аналізу даних про навчальні досягнення студентів та виявлення проблемних місць у навчанні. Також електронні системи контролю знань можуть бути інтегровані з іншими електронними системами управління навчальним процесом, що дозволяє створювати єдину інформаційну систему для ефективного управління навчальним процесом в цілому.

1.3. Переваги та недоліки використання автоматизації управлінського та навчального процесів

1.3.1. Переваги застосування інформаційних технологій

Застосування інформаційних технологій управління та навчання має ряд переваг, серед яких:

- Збільшення ефективності та швидкості обробки інформації. Інформаційні технології дозволяють обробляти інформацію значно швидше, ніж людина, що забезпечує швидке та точне прийняття рішень.
- Зменшення витрат на паперову документацію та збереження даних. Інформаційні системи дозволяють зберігати та обробляти інформацію в електронному вигляді, що зменшує витрати на паперову документацію та зберігання даних.

– Забезпечення доступності інформації для користувачів. Інформаційні технології дозволяють швидко та легко забезпечити доступ до інформації користувачам, що робить процес управління та навчання більш доступним та ефективним.
– Підвищення якості навчання та забезпечення індивідуального підходу. Застосування інформаційних технологій дозволяє забезпечити індивідуальний підхід до навчання, забезпечити різноманітність форм та методів навчання, що підвищує якість навчання.

1.3.2. Недоліки автоматизації управління та навчання

Хоча автоматизація управлінських та навчальних процесів має багато переваг, вона також має певні недоліки, які слід враховувати:

– Високі витрати на розробку та впровадження інформаційних систем. Розробка, підтримка та оновлення програмного забезпечення можуть бути дуже витратними, особливо для закладів з обмеженим бюджетом.
– Залежність від технології. Якщо система автоматизації навчання відмовляє, це може призвести до затримок у проведенні занять та оцінюванні.
– Віддаленість від людського контакту. Деякі студенти можуть відчувати віддаленість від викладачів та інших студентів, особливо якщо навчання відбувається виключно в онлайн-форматі.
– Використання застарілих технологій. Багато закладів вищої освіти мають застарілу техніку та програмне забезпечення, що може призвести до проблем зі сумісністю та ефективністю автоматизованих систем.

1.4. Висновки до розділу 1

Автоматизація управлінських та навчальних процесів є актуальною та перспективною тенденцією в сучасній вищій освіті. Застосування інформаційних технологій дозволяє знизити витрати часу та зусиль на адміністративні та рутинні процеси, підвищити якість та ефективність управління, забезпечити доступність та якість навчання.

Проте, також є недоліки та обмеження в застосуванні інформаційних технологій, такі як можливість технічних збоїв та проблем з безпекою даних, необхідність кваліфікованого персоналу та відповідних знань та навичок користування цими технологіями. Тобто, можна стверджувати, що використання інформаційних технологій у вищій освіті є невід'ємною складовою сучасного навчально-виховного процесу, який підвищує якість та ефективність навчання, а також сприяє покращенню управління закладами вищої освіти.

РОЗДІЛ 2

МЕТОДИ ТА ЗАСОБИ АВТОМАТИЗАЦІЇ ІНФОРМАЦІЙНОЇ ПІДТРИМКИ ОСВІТНЬОГО ПРОЦЕСУ З ВИКОРИСТАННЯМ АСИНХРОННОГО ПРОГРАМУВАННЯ

2.1. Обґрунтування вибору способів рішення задачі

2.1.1. Визначення вимог і потреб освітнього процесу

Під час проведення аналізу і визначення вимог і потреб, які стосуються автоматизації інформаційної підтримки освітнього процесу були визначені наступні вимоги і потреби:

1. Забезпечення ефективного спілкування між викладачами та студентами:
   a. Можливість надсилання та отримання повідомлень, що дозволяє викладачам та студентам обмінюватись інформацією без обмежень у часі.
   b. Можливість створення та управління форумами, де студенти можуть обговорювати навчальні матеріали та задавати питання викладачам.
2. Забезпечення доступу до навчальних матеріалів та ресурсів:
   a. Можливість завантаження та збереження навчальних матеріалів у різних форматах (текстові файли, відео, аудіо тощо) для подальшого використання студентами.
   b. Розподілення прав доступу до різних навчальних ресурсів відповідно до потреб студентів та викладачів.

3. Забезпечення зручності та ефективності управління навчальним процесом:
    a. Можливість створення та планування навчальних курсів, включаючи розклад занять, список завдань та дедлайни для студентів.
    b. Автоматизоване відстеження прогресу студентів, їхніх досягнень та оцінювання з можливістю генерації звітів для викладачів та адміністраторів.

2.1.2. Огляд і порівняння різних підходів до автоматизації інформаційної підтримки освітнього процесу

Приклади різних підходів та їхніх характеристик, недоліків і переваг:

1. Веб-платформа для дистанційного навчання:
    a. Характеристики:
        i. Централізована система, що надає доступ до навчальних матеріалів та інтерактивних курсів.
        ii. Забезпечує можливість спілкування між викладачами та студентами через форуми або чати.
        iii. Має можливість контролю та оцінювання студентів.
    b. Переваги:
        i. Забезпечує доступ до навчання в будь-який час та з будь-якого місця.
        ii. Дозволяє студентам самостійно управляти своїм навчанням.
        iii. Зручний інтерфейс для викладачів та студентів.
    c. Недоліки:

i. Потребує стабільного Інтернет-з'єднання для доступу до платформи.
ii. Обмежена можливість взаємодії в реальному часі між викладачами та студентами.

2. Мобільний додаток для навчання:
   a. Характеристики:
      i. Доступний на мобільних пристроях.
      ii. Містить навчальні матеріали, завдання та тести.
      iii. Забезпечує можливість взаємодії через обмін повідомленнями або форумами.
   b. Переваги:
      i. Забезпечує мобільний доступ до навчального контенту.
      ii. Можливість отримання сповіщень та нагадувань про завдання та дедлайни.
      iii. Інтерактивність та гейміфікація можуть сприяти більшій мотивації студентів.
   c. Недоліки:
      i. Обмежений простір екрану мобільного пристрою може ускладнити сприйняття навчального матеріалу.
      ii. Обмежена можливість використання складних інтерактивних елементів.
3. Використання електронних журналів та системи онлайн-оцінювання:
   a. Характеристики:
      i. Автоматизована система для ведення журналів та оцінювання студентів.
      ii. Доступ до оцінок та коментарів через онлайн-інтерфейс.
      iii. Можливість зберігання студентських робіт та генерація звітів.

b. Переваги:
    i. Спрощує процес ведення журналу та оцінювання студентів для викладачів.
    ii. Доступність оцінок та коментарів для студентів і батьків.
    iii. Забезпечує архівування даних та зручну аналітику.
c. Недоліки:
    i. Можлива складність інтеграції з іншими системами університету.
    ii. Потребує навчання викладачів та студентів використовувати систему.

Огляд і порівняння цих підходів дозволить зробити висновки щодо вибору найбільш підходящого рішення для розробки системи автоматизованої інформаційної підтримки освітнього процесу з використанням асинхронного програмування.

2.1.3. Обґрунтування вибору асинхронного програмування як підходу для автоматизації інформаційної підтримки освітнього процесу

Асинхронне програмування є потужним інструментом, який може бути використаний для автоматизації інформаційної підтримки освітнього процесу. Оглянемо приклади, що обґрунтовують вибір асинхронного програмування:

1. Обробка багатопоточності: Асинхронне програмування дозволяє ефективно керувати багатопоточним середовищем, де різні операції можуть виконуватись паралельно. У випадку освітнього процесу, можуть бути одночасно виконувані різні завдання, такі як завантаження навчальних матеріалів, взаємодія студентів та викладачів

тощо. Асинхронність дозволяє оптимізувати виконання цих завдань і забезпечити більш ефективну роботу системи.

2. Розподілена архітектура: Асинхронне програмування також підходить для розподіленої архітектури, де різні компоненти системи можуть знаходитись на різних вузлах мережі. У випадку освітнього процесу, можуть існувати різні сервери для зберігання навчального контенту, системи оцінювання та взаємодії. Асинхронність дозволяє ефективно взаємодіяти з цими розподіленими компонентами і забезпечити швидку передачу даних.
3. Обробка асинхронних подій: В освітньому процесі часто виникають асинхронні події, такі як запити студентів, завантаження матеріалів, відправлення повідомлень тощо. Асинхронне програмування дозволяє ефективно обробляти ці події і відповідати на них без блокування основного потоку виконання. Це підвищує реагування системи та забезпечує зручний користувацький досвід.
4. Масштабованість: Асинхронне програмування також підходить для масштабованих систем, де можлива залучення багатьох ресурсів для виконання завдань. В освітньому процесі можуть бути великі навчальні групи, багато взаємодіючих студентів та викладачів. Асинхронність дозволяє розподіляти завдання між різними ресурсами та забезпечувати оптимальне використання системних ресурсів.

Обґрунтування вибору асинхронного програмування для автоматизації інформаційної підтримки освітнього процесу базується на його здатності ефективно керувати багатопоточним середовищем, розподіленою архітектурою, обробкою асинхронних подій та масштабованістю. Використання асинхронного програмування допоможе побудувати потужну

та ефективну систему, яка забезпечить ефективну інформаційну підтримку освітнього процесу.

2.2 Засоби розробки

2.2.1 Вибір мови програмування для реалізації асинхронного програмування

При реалізації асинхронного програмування для автоматизації інформаційної підтримки освітнього процесу, важливо обрати відповідну мову програмування. Для реалізації автоматизації інформаційної підтримки освітнього процесу засобами асинхронного програмування буде використано асинхронного бота Telegram.

Один із популярних варіантів для реалізації асинхронного бота Telegram - мова програмування Python. Python є широко використовуваною мовою програмування з багатим екосистемою бібліотек і фреймворків для розробки асинхронних додатків. Для створення асинхронного бота Telegram ми будемо використовувати бібліотеку "aiogram"[4].

Python є популярною мовою для реалізації асинхронних програм засобами своєї асинхронної бібліотеки asyncio та різноманітних фреймворків, що спрощують розробку асинхронних додатків.

2.2.2. Обґрунтування вибору конкретних засобів розробки для реалізації системи автоматизованої інформаційної підтримки освітнього процесу

При реалізації системи автоматизованої інформаційної підтримки освітнього процесу, вибір конкретних засобів розробки є важливим кроком. Обґрунтування вибору засобів розробки для реалізації Telegram бота з використанням бібліотеки aiogram:

1. aiogram: aiogram є популярною бібліотекою для розробки Telegram ботів на мові програмування Python. Вибір цієї бібліотеки обґрунтований кількома факторами:
    - Простота використання: aiogram надає зручний та інтуїтивно зрозумілий інтерфейс для розробки Telegram ботів. Вона має документацію та приклади, що спрощують початок роботи та швидкий розвиток проекту.
    - Багатофункціональність: aiogram надає широкі можливості для розробки різноманітних функцій Telegram ботів. Вона підтримує обробку повідомлень, команд, клавіатур, фотографій, відео та багато іншого, що дозволяє розширити функціональність системи автоматизованої інформаційної підтримки.
    - Асинхронна підтримка: aiogram побудована з використанням асинхронного програмування на основі бібліотеки asyncio. Це дозволяє ефективно керувати багатопотоковими операціями та забезпечує швидку відповідь бота на запити.
2. Python: Вибір мови програмування Python для реалізації Telegram бота також має свої обґрунтування:
    - Простота та зручність: Python є однією з найпопулярніших мов програмування, що характеризується простотою та зрозумілістю синтаксису. Це сприяє швидкому розвитку проекту та полегшує спільну роботу над ним.
    - Багатість бібліотек: Python має широкий вибір сторонніх бібліотек, що полегшує розробку різноманітних функцій Telegram бота. Багато бібліотек, в тому числі й aiogram, забезпечують зручний інтерфейс та функціональність для роботи з Telegram API.

- Спільнота розробників: Python має активну спільноту розробників, що сприяє доступності допомоги, обміну досвідом та пошуком рішень під час розробки. Це може значно полегшити процес розробки та вирішення проблем.

3. Telegram бот для освітнього процесу: Обрання Telegram бота як засобу автоматизованої інформаційної підтримки освітнього процесу має свої переваги:
   - Широке поширення: Telegram є популярною месенджерською платформою з великою кількістю активних користувачів. Це забезпечує широку доступність системи автоматизованої інформаційної підтримки для студентів та педагогічного персоналу.
   - Зручність та простота використання: Telegram має зрозумілий та інтуїтивно зрозумілий інтерфейс, який не вимагає складних налаштувань або додаткового навчання для користувачів. Це дозволяє зручно використовувати систему автоматизованої інформаційної підтримки.
   - Функціональність: Telegram надає можливості для обміну повідомленнями, використання клавіатур, відправки фотографій, відео та інших медіа-файлів. Це дозволяє розширити функціональність системи автоматизованої інформаційної підтримки та надати більше можливостей користувачам.

Загалом, обрання бібліотеки aiogram для реалізації Telegram бота та використання мови програмування Python є обґрунтованим вибором, оскільки ці засоби надають зручність в розробці, багатофункціональність та асинхронну підтримку. Крім того, Telegram бот є зручним для автоматизованої інформаційної підтримки освітнього процесу.

2.3. Висновки до розділу 2

Було обґрунтовано вибір мови програмування Python для реалізації системи автоматизованої інформаційної підтримки освітнього процесу. Python був обрано через його простоту та зручність, багатий вибір сторонніх бібліотек і активну спільноту розробників.

А також вибір бібліотеки aiogram для реалізації Telegram бота. Aiogram надає зручний інтерфейс та функціональність для роботи з Telegram API, що дозволяє ефективно розробляти функції, необхідні для автоматизованої інформаційної підтримки.

Було обґрунтовано вибір Telegram бота як засобу автоматизованої інформаційної підтримки освітнього процесу - Telegram є популярною месенджерською платформою з великою кількістю користувачів, що забезпечує широку доступність системи підтримки. Крім того, зручність використання та функціональність Telegram, такі як обмін повідомленнями, медіа-файли і клавіатури, сприяють покращенню досвіду користувачів.

Враховуючи ці обґрунтування, вибір мови програмування Python, бібліотеки aiogram і Telegram бота виявляється вдалим для реалізації системи автоматизованої інформаційної підтримки освітнього процесу. Ці засоби забезпечують зручність в розробці, багатофункціональність та асинхронну підтримку, що сприяє вдосконаленню та ефективності інформаційного процесу.

РОЗДІЛ 3

ОПИС КЛАСІВ ЯКІ ВИКОРИСТОВУЮТЬСЯ В ДОДАТКУ ДЛЯ ЗАБЕЗПЕЧЕННЯ ФУНКЦІОНУВАННЯ ПРОГРАМИ

3.1. Модель Даних та База Даних (DbContext)

Структура методу з DbContext:

async def method_name():

Виконання якогось коду

await cursor.execute("SQL запит")

Виконання якогось коду

return result

3.1.1 Клас AsyncContext

Цей клас відповідає за управління асинхронним з'єднанням з базою даних. Забезпечує можливість використання конструкції async with [3], яка автоматично відкриває та закриває з'єднання. Використовує бібліотеку asyncpg для асинхронного з'єднання з PostgreSQL[5] базою даних.

3.1.2. Функції для отримання даних з бази даних

get_users(): Отримує всіх користувачів з таблиці BotUser.

get_user_states(): Отримує стани користувачів з таблиці UserState.

get_user_state(telegram_id): Отримує стан користувача за його telegram_id.

set_user_state(telegram_id, state): Встановлює стан для користувача.

get_specialities(): Отримує всі спеціальності з таблиці Speciality.

3.1.3. Функції для реєстрації користувачів

register_student(...): Реєструє студента у таблицях BotUser, Student та StudentSpeciality.

register_teacher(...): Реєструє вчителя у таблицях BotUser та Teacher.

register_admin(...): Реєструє адміністратора у таблиці BotUser.

3.1.4. Функції для отримання інформації про студентів

get_student_subjects(telegram_id): Отримує предмети, які вивчає студент.

get_student_grades(telegram_id, subject_name): Отримує оцінки студента за конкретний предмет.

get_student_homeworks(telegram_id, subject_name): Отримує домашні завдання студента за конкретний предмет.**get_group_students(telegram_id):** Отримує список студентів у групі студента.

get_group_rating(telegram_id): Отримує середні оцінки всіх студентів у групі студента.

3.1.5. Функції для отримання інформації про вчителів

get_teachers(): Отримує список всіх вчителів разом із списком предметів, які вони викладають.

get_teacher_by_subject(subject_name): Отримує вчителя, який викладає певний предмет.

3.1.6. Функції для роботи зі спеціальностями та предметами

get_all_specialities(): Отримує всі спеціальності.

add_speciality(...): Додає нову спеціальність у таблиці Speciality та SpecialitySubject.

delete_speciality(speciality_code): Видаляє спеціальність. **edit_speciality(...):** Редагує інформацію про спеціальність. **get_all_subjects():** Отримує всі предмети.

add_subject(...): Додає новий предмет у таблицю Subject.

delete_subject(subject_id): Видаляє предмет.

edit_subject(...): Редагує інформацію про предмет.

3.1.7. Функції для отримання інформації про користувачів

get_all_students(): Отримує інформацію про всіх студентів.

delete_student(telegram_id): Видаляє студента.

edit_student(...): Редагує інформацію про студента.

get_all_teachers(): Отримує інформацію про всіх вчителів.

delete_teacher(telegram_id): Видаляє вчителя.

edit_teacher(...): Редагує інформацію про вчителя.

3.1.8. Інші функції

get_schedule_image(speciality_id): Отримує ID фотографії розкладу для конкретної спеціальності.

upload_homework(telegram_id, subject_id, file_id, date): Завантажує домашнє завдання для студента.

rate_student_homework(...): Оцінює домашнє завдання студента.

upload_homework_as_teacher(...): Завантажує домашнє завдання для всіх студентів конкретного предмету та курсу.

add_schedule(...): Додає розклад для конкретної спеціальності.

3.2. Сервіс Користувачів (UsersService)

Структура методу з UserService:

async def method_name():

Виконання якогось коду

await DbContext.method_name()

Клас UsersService виконує роль сервісного шару (бізнес-логіки) для взаємодії з користувачами в телеграм-боті. Цей клас служить як проміжний рівень між інтерфейсом бота та базою даних, реалізуючи логіку обробки запитів користувачів та взаємодії з базою даних за допомогою методів класу DbContext. Це дозволяє виконувати операції реєстрації, отримання даних і т.д. через асинхронний інтерфейс.Основні методи цього класу:

is_registered(telegram_id): Перевіряє, чи зареєстрований користувач з вказаним telegram_id.

get_user_state(telegram_id): Отримує стан користувача за його telegram_id.

set_user_state(telegram_id, state): Встановлює стан для користувача.

is_speciality_exists(speciality_code): Перевіряє, чи існує спеціальність з вказаним кодом.

register_student(…): Реєструє студента.

get_speciality_terms_count(speciality_code): Отримує кількість термінів для спеціальності.

register_teacher(telegram_id, first_name, second_name, last_name): Реєструє вчителя.

register_admin(telegram_id, first_name, second_name, last_name): Реєструє адміністратора.

get_student_subjects(telegram_id): Отримує предмети, які вивчає студент.

get_student_grades(telegram_id, subject_name): Отримує оцінки студента за конкретний предмет.

get_student_homeworks(telegram_id, subject_name): Отримує домашні завдання студента за конкретний предмет.

get_group_students(telegram_id): Отримує список студентів у групі студента.

get_group_rating(telegram_id): Отримує середні оцінки всіх студентів у групі студента.

get_specialities(): Отримує список всіх спеціальностей.

get_teachers_with_subjects(): Отримує список вчителів разом із списком предметів, які вони викладають.

get_teachers_by_speciality(speciality_code): Отримує вчителів для конкретної спеціальності.

3.3. Обробники Запитів та Команд (Handlers)

Структура методу message в класі Handlers:

```
@router.message(F.text == "/command")
async def command(message: types.Message):
    log("Ви обрали команду /command (command_handler.py)")
    *Виконання якогось коду*
    await message.answer("Відповідь на команду /command")
```

Структура методу callback в класі Handlers:

```
@router.callback_query(F.data == "callback")
async def callback(callback: types.CallbackQuery):
    log("Ви обрали callback callback (command_handler.py)")
    *Виконання якогось коду*
    await callback.message.edit_text("Відповідь на callback callback")
```

3.3.1. Клас command_handler

Клас command_handler використовується для обробки команд та повідомлень користувачів. Цей клас допомагає маршрутизувати команди та повідомлення користувачів до відповідних обробників, забезпечуючи взаємодію з різними частинами функціоналу бота відповідно до їхніх дій та ролей:

get_markup(user_id): Генерує і повертає розмітку клавіатури для користувача в залежності від його стану (роль студента, вчителя, адміністратора або адміна з певним станом).

@router.message(F.text == "/start"): Обробляє команду /start і вітає користувача, запрошуючи його зареєструватися.

@router.message(F.text == "/register"): Обробляє команду /register і перевіряє, чи користувач вже зареєстрований. Якщо так, повідомляє користувача. Якщо ні, надсилає вибір ролі.

@router.message(F.text == "/menu"): Обробляє команду /menu і перевіряє, чи користувач зареєстрований. Якщо ні, повідомляє користувача про необхідність реєстрації. Якщо так, надсилає меню відповідно до ролі користувача.

@router.callback_query(F.data == "back_to_menu"): Обробляє клавішу "Назад до меню" в інтерфейсі користувача.

@router.message(F.document): Обробляє документи, розпізнаючи їх за іменем файлу. Якщо це домашнє завдання, викликає метод **upload_homework з student_handler**. Якщо це інший документ, викликає метод **admin_add_schedule з admin_handler**.

@router.message(): Обробляє інші текстові повідомлення. Використовує словник state_actions для визначення методів обробки повідомлень в залежності від поточного стану користувача.

3.3.2. Клас register_handler

Клас register_handler містить обробники для реєстрації користувачів різних ролей (студент, викладач, адміністратор). Цей клас відповідає за обробку запитів та команд, пов'язаних із реєстрацією різних користувачів в системі:

async def is_registered(state, callback): Перевіряє, чи користувач вже зареєстрований з певною роллю. Якщо так, відправляє повідомлення з відповідним текстом та клавіатурою.

@router.callback_query(F.data == "register_as_student"): Обробник для реєстрації як студент. Перевіряє стан користувача і викликає метод реєстрації, який очікує введення даних.

@router.callback_query(F.data == "register_as_teacher"): Обробник для реєстрації як викладач. Перевіряє стан користувача і викликає метод реєстрації, який очікує введення даних.

@router.callback_query(F.data == "register_as_admin"): Обробник для реєстрації як адміністратор. Перевіряє стан користувача і викликає метод реєстрації, який очікує введення даних та паролю адміністратора.

async def register_as_student(message: types.Message): Метод реєстрації студента. Перевіряє правильність введення даних та спеціальності, а потім викликає метод реєстрації студента в базі даних.

async def register_as_teacher(message: types.Message): Метод реєстрації викладача. Перевіряє правильність введення даних і викликає метод реєстрації викладача в базі даних.

async def register_as_admin(message: types.Message): Метод реєстрації адміністратора. Перевіряє правильність введення даних та паролю адміністратора, а потім викликає метод реєстрації адміністратора в базі даних.

3.3.3. Клас admin_handler

Клас admin_handler містить обробники для адміністративних операцій у системі університетського бота, таких як видалення/редагування спеціальностей, предметів та студентів. Ідентичні обробники і методи використовуються також в класах student_handler і teacher_handler, тому щоб не дублювати ці методи, тут будуть вказані укікальні методи для admin_handler. Ці методи дозволяють адміністраторам системи виконувати різноманітні завдання, такі як редагування викладачів, додавання розкладів та інші операції для забезпечення функціональності бота:

@router.callback_query(F.data == "admin_delete_speciality"): Обробник для видалення спеціальності. Відображає список спеціальностей та очікує введення коду спеціальності для видалення.

async def admin_delete_speciality(message: types.Message): Метод для видалення спеціальності. Отримує введений код спеціальності, перевіряє його наявність та викликає метод видалення спеціальності в базі даних.

@router.callback_query(F.data == "admin_edit_speciality"): Обробник для редагування спеціальності. Відображає список спеціальностей та очікує введення даних для редагування.

async def admin_edit_speciality_input_data(message: types.Message): Метод для введення нових даних спеціальності та виклику методу редагування спеціальності в базі даних.

@router.callback_query(F.data == "admin_view_all_subjects"): Обробник для перегляду всіх предметів. Відображає список предметів.

@router.callback_query(F.data == "admin_add_subject"): Обробник для додавання нового предмета. Очікує введення назви предмета та його семестрового діапазону.

async def admin_add_subject(message: types.Message): Метод для додавання нового предмета в базу даних.

@router.callback_query(F.data == "admin_delete_subject"): Обробник для видалення предмета. Відображає список предметів та очікує введення ID предмета для видалення.

async def admin_delete_subject(message: types.Message): Метод для видалення предмета з бази даних.

@router.callback_query(F.data == "admin_edit_subject"): Обробник для редагування предмета. Відображає список предметів та очікує введення даних для редагування.

async def admin_edit_subject_input_data(message: types.Message): Метод для введення нових даних предмета та виклику методу редагування предмета в базі даних.

@router.callback_query(F.data == "admin_view_all_students"): Обробник для перегляду всіх студентів. Відображає список студентів.

@router.callback_query(F.data == "admin_delete_student"): Обробник для видалення студента. Відображає список студентів та очікує введення ID

@router.callback_query(F.data == "admin_edit_teacher"): Обробник для редагування викладача. Відображає список викладачів та очікує введення даних для редагування.

async def admin_edit_teacher_input_data(message: types.Message): Метод для введення нових даних викладача та виклику методу редагування викладача в базі даних.

@router.callback_query(F.data == "admin_add_schedule"): Обробник для додавання розкладу. Відображає повідомлення та очікує надсилання файлу з розкладом у форматі .xlsx з номером спеціальності.

async def admin_add_schedule(message: types.Message): Метод для додавання розкладу до бази даних. Перевіряє, чи надіслано файл, отримує

ідентифікатор файлу та номер спеціальності з імені файлу, і викликає метод для додавання розкладу в базу даних.

3.4. Клавіатури та Інтерфейси (Keyboards)

3.4.1. Клас admin_keyboards

Клас admin_keyboards визначає різні клавіатури для взаємодії з адміністраторським інтерфейсом. Кожна клавіатура має свою структуру з рядків та кнопок, які надають адміністратору зручний інтерфейс для взаємодії з різними функціональними можливостями бота:

admin_menu_keyboard: Основне меню адміністратора з кнопками для перегляду інформації про користувачів, спеціальності, предмети, студентів та викладачів.

admin_users_info_keyboard: Меню для перегляду інформації про користувачів з опціями для показу всіх користувачів, студентів, викладачів, адміністраторів і кнопкою "Назад".

admin_specialities_info_keyboard: Меню для перегляду інформації про спеціальності з опціями для показу всіх спеціальностей, додавання, видалення та редагування спеціальностей, а також можливість додати розклад для спеціальності та кнопкою "Назад".

admin_subjects_info_keyboard: Меню для перегляду інформації про предмети з опціями для показу всіх предметів, додавання, видалення та редагування предметів, а також кнопкою "Назад".

admin_students_info_keyboard: Меню для перегляду інформації про студентів з опціями для показу всіх студентів, видалення та редагування студентів, а також кнопкою "Назад".

admin_teachers_info_keyboard: Меню для перегляду інформації про викладачів з опціями для показу всіх викладачів, видалення та редагування викладачів, а також кнопкою "Назад".

3.4.2. Клас student_keyboads

Клас student_keyboards визначає різні клавіатури для взаємодії з інтерфейсом студента. Ці клавіатури надають студентам зручний інтерфейс для отримання інформації та виконання операцій в межах бота:

student_menu_keyboard: Основне меню для студента з кнопками для перегляду інформації про себе, групу, спеціальність та університет.

student_self_info_keyboard: Меню для перегляду особистої інформації студента, включаючи оцінки та домашні завдання, а також кнопку "Назад".

view_self_student_marks: Меню для перегляду оцінок студента з можливістю повернутися назад.

view_self_student_homework: Меню для перегляду домашніх завдань студента з можливістю повернутися назад.

build_student_homework_keyboard(subject_name): Метод для створення клавіатури для завантаження домашнього завдання з конкретного предмету.

group_info_keyboard: Меню для перегляду інформації про групу з кнопками для списку студентів, рейтингу та кнопкою "Назад".

view_group_info: Меню для перегляду інформації про групу з можливістю повернутися назад.

speciality_info_keyboard: Меню для перегляду інформації про спеціальність з кнопками для розкладу, викладачів, предметів та кнопкою "Назад".

view_speciality_info: Меню для перегляду інформації про спеціальність з можливістю повернутися назад.

college_info_keyboard: Меню для перегляду інформації про університет з кнопками для спеціальностей, викладачів та кнопкою "Назад".

view_college_info: Меню для перегляду інформації про університет з можливістю повернутися назад.

build_subjects_keyboard(subjects, target, callback_data): Метод для створення клавіатури для перегляду предметів, домашніх завдань чи оцінок з можливістю повернутися назад.

3.4.3. Клас teacher_keyboards

Структура клавіатури:

```
keyboard = InlineKeyboardMarkup(inline_keyboard=[
  [InlineKeyboardButton(text="Текст кнопки 1", callback_data="callback_1")],
  [InlineKeyboardButton(text="Текст кнопки 2", callback_data="callback_2")],
  [InlineKeyboardButton(text="Текст кнопки 3", callback_data="callback_3")], ])
```

Клас teacher_keyboards визначає різні клавіатури для взаємодії з інтерфейсом вчителя. Ці клавіатури допомагають вчителям отримати доступ до різних функцій та інформації у боті:

teacher_menu_keyboard: Основне меню для вчителя з кнопками для перегляду інформації про себе, спеціальності, домашні завдання та університет.

teacher_self_info_keyboard: Меню для перегляду особистої інформації вчителя, зокрема списку предметів, які він викладає, та кнопкою "Назад".

teacher_back_self_info_keyboard: Меню для повернення назад зі сторінки перегляду особистої інформації вчителя.

speciality_info_keyboard: Меню для перегляду інформації про спеціальність з кнопками для перегляду списку студентів на спеціальності та кнопкою "Назад".

speciality_back_info_keyboard: Меню для повернення назад зі сторінки перегляду інформації про спеціальності.

homework_info_keyboard: Меню для перегляду інформації про домашні завдання з кнопками для перегляду домашніх завдань студентів, оцінювання та завантаження домашніх завдань для предметів та кнопкою "Назад".

homework_back_info_keyboard: Меню для повернення назад зі сторінки перегляду інформації про домашні завдання.

college_info_keyboard: Меню для перегляду інформації про університет з кнопками для перегляду списку спеціальностей та викладачів університету та кнопкою "Назад".

3.4.4. Клас service_keyboards

Клас service_keyboards визначає клавіатуру для обрання ролі при реєстрації користувача. Основна клавіатура цього класу:

roles_keyboard: Містить три кнопки для вибору ролі: "Студент", "Викладач" і "Адміністратор". Кожна кнопка має відповідний callback_data для подальшої обробки в коді бота.

Ця клавіатура дозволяє користувачам обрати свою роль під час реєстрації в системі.

3.5. Логування та Відлагодження (Logger)

Структура методу з logger:

```
def method_name():
    print("Текст повідомлення")
```

Клас logger визначає простий логгер, який виводить повідомлення із вказаним текстом та датою в консоль.

log(text): Цей метод отримує текстове повідомлення text, а потім додає поточну дату та час. Результат виводиться у вигляді рядка в консоль. Використовується формат дати та часу "%d/%m/%Y %H:%M:%S". – день/місяць/рік/година/хвилина/секунда.

3.6. Основний Клас та Налаштування (Main та Settings)

3.6.1. Клас main

Структура методу з main:

async def main():

Виконання якогось коду

await dp.start_polling(bot)

Код класу main використовує бібліотеку aiogram для створення та налаштування телеграм-бота. Основні елементи коду:

async def main():: Це асинхронна функція, яка є точкою входу для запуску бота. Вона створює об'єкт бота та об'єкт диспетчера, додає обробники повідомлень з різних роутерів (admin, command, register, student, teacher), та починає процес отримання оновлень через polling.

if __name__ == '__main__':: Ця умовна конструкція перевіряє, чи файл запускається напряму (а не імпортується в інший скрипт). Якщо це так, то викликається функція asyncio.run(main()).

try:: Використовується для перехоплення винятків.

except KeyboardInterrupt:: Якщо користувач(програміст) натискає Ctrl+C для зупинки бота, виводиться повідомлення про завершення роботи бота.

Загальний функціонал коду включає в себе налаштування бота, встановлення обробників повідомлень та початок асинхронного процесу отримання оновлень бота через метод start_polling.

3.6.2. Клас settings

У файлі settings визначені ряд ключових налаштувань, таких як TOKEN, AdministratorPassword, host, port, database, user, і password:

TOKEN: Токен, який надається боту під час реєстрації в Telegram API. Використовується для аутентифікації бота при кожному запиті до Telegram.

AdministratorPassword: Пароль адміністратора, який, ймовірно, використовується для аутентифікації адміністратора бота. Потрібно забезпечити безпеку таких паролів та уникати їхнього використання у виробничих системах.

host, port, database, user, password: Параметри для підключення до бази даних PostgreSQL. Бот, можливо, використовує цю базу даних для зберігання та отримання інформації. Переконайтеся, що ці налаштування вірні та відповідають налаштуванням вашого PostgreSQL сервера.

3.7. Висновки до розділу 3

Ефективність комунікації: Розроблений бот значно полегшив процес комунікації між вчителями та студентами. Взаємодія через команди та клавіші дозволила зробити обмін інформацією швидким та зручним.

Зручний перегляд інформації: Модулі бота, присвячені перегляду інформації про викладачів, студентів, предмети та домашні завдання, роблять процес моніторингу більш структурованим та зрозумілим.

*Систематизація домашніх завдань:*Можливість завантаження та оцінювання домашніх завдань в одній інформаційній системі сприяє ефективній взаємодії між викладачами та студентами у процесі навчання.

Управління групами та спеціальностями: Додаткові функції, такі як перегляд списків студентів за спеціальностями та виставлення оцінок, спрощують адміністративні аспекти освітнього процесу.

Динамічний розвиток: Архітектурна структура бота забезпечує легкість розширення та модифікації функціоналу відповідно до змінних потреб освітнього середовища.

Використання клавіш та меню: Використання клавіш та меню дозволяє користувачам, навіть з мінімальним досвідом користування ботами, легко та ефективно взаємодіяти з розробленим ботом.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

ВИСНОВКИ:

Функціоналість:

- Бот реалізує обробку команд від користувачів різних ролей: студентів, викладачів і адміністраторів.
- Для кожної ролі визначено відповідний набір команд і можливостей через відповідні обробники (handlers).

Структура проекту:

- Проект використовує модульну структуру, розділяючи обробників для кожної ролі, а також окремий обробник для реєстрації користувачів.
- Використано бібліотеку aiogram для зручного роботи з API Telegram.
- Важливі компоненти, такі як клавіатури та логер, винесені у власні файли для збереження структури та зрозумілості коду.

Безпека:

- У файлі settings є конфіденційні дані, такі як токен та дані для доступу до бази даних. Необхідно забезпечити безпеку цих даних, особливо при публікації коду чи його використанні в реальних умовах.
- Пароль адміністратора (AdministratorPassword) може потребувати більш безпечного управління для забезпечення захисту доступу.

Робота з базою даних:

- Проект використовує базу даних PostgreSQL для зберігання інформації про користувачів, викладачів, студентів, предмети та інше.

- Важливо переконатися, що дані у файлі settings вірно відображають конфігурацію бази даних.

ПРОПОЗИЦІЇ:

Безпека:

Рекомендується використовувати змінні середовища для зберігання конфіденційних даних, таких як токен та дані для доступу до бази даних, замість зберігання їх прямо в коді.

Документація:

Додати або розширити коментарі до коду, щоб зробити його більш зрозумілим для інших розробників або для самого себе у майбутньому.

Інтернаціоналізація:

У разі планування використання бота в різних мовах рекомендується розглянути питання інтернаціоналізації для легшого впровадження нових мов.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Дослідження за темою "Ефективність використання електронних платформ для онлайн-навчання в закладах вищої освіти", автори: Шевченко Н.М., Козловський О.І.

2. Дослідження за темою "Ефективність використання електронних систем управління навчальним процесом в університеті", автори: Іванова О.М., Петренко І.В., Сидоренко В.О.)

3. Бібліотека aiogram - github.com/aiogram/aiogram

4. Бібліотека asyncio - docs.python.org/3/library/asyncio.html

5. PostgresSQL - www.postgresql.org/

6. Використання інформаційних систем управління в діяльності вищого навчального закладу - um.co.ua/6/6-11/6-115701.html

ДОДАТКИ

Додаток А

DbContext.py

```
import asyncpg

from settings import host, port, database, user, password


class AsyncContext:
    def __init__(self):
        self.conn = None

    async def __aenter__(self):
        self.conn = await asyncpg.connect(
            user=user,
            password=password,
            database=database,
            host=host,
            port=port
        )
        return self.conn

    async def __aexit__(self, exc_type, exc_val, exc_tb):
        await self.conn.close()


async_context = AsyncContext()


async def get_users():
    async with async_context as conn:
        query = "SELECT * FROM BotUser"
        result = await conn.fetch(query)
        return result


async def get_user_states():
```

```
    async with async_context as conn:
        query = "SELECT * FROM UserState"
        result = await conn.fetch(query)
        return result


async def get_user_state(telegram_id):
    user_states = await get_user_states()
    for user_state in user_states:
        if user_state[0] == telegram_id:
            return user_state[1]
    return None


async def set_user_state(telegram_id, state):
    async with async_context as conn:
        # if state is not created, create it
        if await get_user_state(telegram_id) is None:
            await conn.execute("INSERT INTO UserState (telegramUserId, state) VALUES ($1, $2)", telegram_id,
state)
        await conn.execute("UPDATE UserState SET state = $1 WHERE telegramUserId = $2", state, telegram_id)


async def get_specialities():
    async with async_context as conn:
        query = "SELECT * FROM Speciality"
        result = await conn.fetch(query)
        return result


async def register_student(telegram_id, first_name, second_name, last_name, speciality, course):
    async with async_context as conn:
        await conn.execute("INSERT INTO BotUser (telegramUserId, firstName, secondName, lastName, position) "
                           "VALUES ($1, $2, $3, $4, $5)",
                           telegram_id, first_name, second_name, last_name, 'Student')
        await conn.execute("INSERT INTO Student (telegramUserId, course) VALUES ($1, $2)",
                           telegram_id, course)
```

```
    speciality_id = await conn.fetch("SELECT id FROM Speciality WHERE code = $1", speciality)
    speciality_id = speciality_id[0][0]
    await conn.execute("INSERT INTO StudentSpeciality (studentId, specialityId) VALUES ($1, $2)",
                       telegram_id, speciality_id)


async def register_teacher(telegram_id, first_name, second_name, last_name):
  async with async_context as conn:
    await conn.execute("INSERT INTO BotUser (telegramUserId, firstName, secondName, lastName, position) "
                       "VALUES ($1, $2, $3, $4, $5)",
                       telegram_id, first_name, second_name, last_name, 'Teacher')
    await conn.execute("INSERT INTO Teacher (telegramUserId) VALUES ($1)",
                       telegram_id)


async def register_admin(telegram_id, first_name, second_name, last_name):
  async with async_context as conn:
    await conn.execute("INSERT INTO BotUser (telegramUserId, firstName, secondName, lastName, position) "
                       "VALUES ($1, $2, $3, $4, $5)",
                       telegram_id, first_name, second_name, last_name, 'Admin')


async def get_student_subjects(telegram_id):
  async with async_context as conn:
    subjects_query = await conn.fetch("SELECT s.name FROM Subject s "
                                      "JOIN SpecialitySubject ss ON s.id = ss.subjectId "
                                      "JOIN Speciality sp ON sp.id = ss.specialityId "
                                      "JOIN StudentSpeciality ss2 ON sp.id = ss2.specialityId "
                                      "WHERE ss2.studentId = $1", telegram_id)

    subjects = [row[0] for row in subjects_query]
    return subjects


async def get_student_grades(telegram_id, subject_name):
  async with async_context as conn:
    subject_id = await conn.fetchval("SELECT id FROM Subject WHERE name = $1", subject_name)
```

```
        grades_with_dates = await conn.fetch(
            "SELECT grade, date, isFinalGrade FROM StudentGrades WHERE studentId = $1 AND subjectId = $2",
            telegram_id, subject_id)
        return grades_with_dates


async def get_student_homeworks(telegram_id, subject_name):
    async with async_context as conn:
        subject_id = await conn.fetchval("SELECT id FROM Subject WHERE name = $1", subject_name)
        homeworks_with_dates = await conn.fetch(
            "SELECT text, date, isDone FROM StudentSubjectHomeWork WHERE studentId = $1 AND subjectId =
$2",
            telegram_id, subject_id)
        return homeworks_with_dates


async def get_group_students(telegram_id):
    async with async_context as conn:
        # select from tables Student and StudentSpeciality where telegramUserId = telegram_id
        speciality_id = await conn.fetchval("SELECT specialityId FROM StudentSpeciality WHERE studentId = $1",
                                            telegram_id)
        students = await conn.fetch(
            "SELECT b.firstName, b.secondName, b.lastName, b.telegramUserId FROM BotUser b "
            "JOIN Student s ON b.telegramUserId = s.telegramUserId "
            "JOIN StudentSpeciality ss ON s.telegramUserId = ss.studentId "
            "WHERE ss.specialityId = $1", speciality_id)
        return students


async def get_group_rating(telegram_id):
    async with async_context as conn:
        students = await get_group_students(telegram_id)
        avg_grades = []
        for student in students:
            student_id = await conn.fetchval("SELECT telegramUserId FROM BotUser WHERE telegramUserId = $1",
                                             student[3])
            avg_grade = await conn.fetchval("SELECT AVG(grade) FROM StudentGrades WHERE studentId = $1",
student_id)
```

```
        avg_grades.append((student, avg_grade))
    avg_grades = [(avg_grade[0][0], avg_grade[0][1], avg_grade[0][2], avg_grade[1]) for avg_grade in
avg_grades]
    avg_grades = [list(avg_grade) for avg_grade in avg_grades]
    return avg_grades


async def get_teachers():
  async with async_context as conn:
    teachers = await conn.fetch(
      "SELECT b.firstName, b.secondName, b.lastName, b.telegramUserId FROM BotUser b WHERE b.position
= $1",
      'Teacher')
    teachers_with_subjects = []
    for teacher in teachers:
      teacher_id = teacher[3]
      subjects = await conn.fetch("SELECT s.name FROM Subject s "
                    "JOIN TeacherSubject ts ON s.id = ts.subjectId "
                    "JOIN Teacher t ON t.telegramUserId = ts.teacherId "
                    "WHERE t.telegramUserId = $1", teacher_id)
      subjects = [subject[0] for subject in subjects]
      teachers_with_subjects.append((teacher[0], teacher[1], teacher[2], subjects))
    return teachers_with_subjects


async def get_student_speciality_code(telegram_id):
  async with async_context as conn:
    speciality_code = await conn.fetchval("SELECT sp.code FROM Speciality sp "
                    "JOIN StudentSpeciality ss ON sp.id = ss.specialityId "
                    "JOIN Student s ON s.telegramUserId = ss.studentId "
                    "WHERE s.telegramUserId = $1", telegram_id)
    return speciality_code


async def get_teachers_by_speciality(speciality_code):
  async with async_context as conn:
    teachers = await conn.fetch(
```

```
        "SELECT b.firstName, b.secondName, b.lastName, b.telegramUserId FROM BotUser b WHERE b.position = $1",
        'Teacher')
    # from teachers select those who have subjects from speciality
    teachers_with_subjects = []
    for teacher in teachers:
        teacher_id = teacher[3]
        subjects = await conn.fetch("SELECT s.name FROM Subject s "
                                    "JOIN TeacherSubject ts ON s.id = ts.subjectId "
                                    "JOIN Teacher t ON t.telegramUserId = ts.teacherId "
                                    "JOIN SpecialitySubject ss ON s.id = ss.subjectId "
                                    "JOIN Speciality sp ON sp.id = ss.specialityId "
                                    "WHERE t.telegramUserId = $1 AND sp.code = $2", teacher_id, speciality_code)
        subjects = [subject[0] for subject in subjects]
        teachers_with_subjects.append((teacher[0], teacher[1], teacher[2], subjects))
    return teachers_with_subjects


async def get_subjects_names_by_speciality(speciality_code):
    async with async_context as conn:
        subjects = await conn.fetch("SELECT s.name FROM Subject s "
                                    "JOIN SpecialitySubject ss ON s.id = ss.subjectId "
                                    "JOIN Speciality sp ON sp.id = ss.specialityId "
                                    "WHERE sp.code = $1", speciality_code)
        subjects = [subject[0] for subject in subjects]
        return subjects


async def get_subjects_ids_by_speciality(speciality_code):
    async with async_context as conn:
        subjects = await conn.fetch("SELECT s.id FROM Subject s "
                                    "JOIN SpecialitySubject ss ON s.id = ss.subjectId "
                                    "JOIN Speciality sp ON sp.id = ss.specialityId "
                                    "WHERE sp.code = $1", speciality_code)
        subjects = [subject[0] for subject in subjects]
        return subjects
```

```
async def get_teacher_by_subject(subject_name):
    async with async_context as conn:
        teacher = await conn.fetchrow("SELECT b.firstName, b.secondName, b.lastName, b.telegramUserId FROM
BotUser b "
                                      "JOIN TeacherSubject ts ON b.telegramUserId = ts.teacherId "
                                      "JOIN Subject s ON s.id = ts.subjectId "
                                      "WHERE s.name = $1", subject_name)
        teacher = (teacher[0], teacher[1], teacher[2], [subject_name])
        return teacher


async def get_student_course(telegram_id):
    async with async_context as conn:
        course = await conn.fetchval("SELECT course FROM Student WHERE telegramUserId = $1", telegram_id)
        return course


async def get_schedule_image(speciality_id):
    async with async_context as conn:
        schedule_image_id = await conn.fetchval("SELECT telegramPhotoId FROM Schedule "
                                                "WHERE specialityId = $1",
                                                speciality_id)
        return schedule_image_id


async def get_student_speciality_id(telegram_id):
    async with async_context as conn:
        speciality_id = await conn.fetchval("SELECT specialityId FROM StudentSpeciality WHERE studentId = $1",
                                            telegram_id)
        return speciality_id


async def get_user_waiting_homework_state(telegram_id):
    async with async_context as conn:
        subject_name = await conn.fetchval("SELECT subjectName FROM StudentHomeworkWaitingStatus "
                                           "WHERE studentId = $1", telegram_id)
```

```
        return subject_name


async def set_user_waiting_homework_state(telegram_id, subject_name):
    async with async_context as conn:
        if await get_user_waiting_homework_state(telegram_id) is None:
            await conn.execute("INSERT INTO StudentHomeworkWaitingStatus (studentId, subjectName) "
                               "VALUES ($1, $2)", telegram_id, subject_name)
        else:
            await conn.execute("UPDATE StudentHomeworkWaitingStatus SET subjectName = $1 WHERE studentId
= $2",
                               subject_name, telegram_id)


async def get_subject_id_by_name(subject_name):
    async with async_context as conn:
        subject_id = await conn.fetchval("SELECT id FROM Subject WHERE name = $1", subject_name)
        return subject_id


async def upload_homework(telegram_id, subject_id, file_id, date):
    async with async_context as conn:
        await conn.execute("UPDATE StudentSubjectHomeWork SET isDone = $1, telegramPhotoId = $2 "
                           "WHERE studentId = $3 AND subjectId = $4 AND date = $5", True, file_id,
                           telegram_id, subject_id, date)


async def get_all_users():
    async with async_context as conn:
        query = "SELECT * FROM BotUser"
        result = await conn.fetch(query)
        return result


async def get_all_specialities():
    async with async_context as conn:
        specialities = await conn.fetch("SELECT * FROM Speciality")
```

```
    return specialities


async def add_speciality(name, code, terms, subjects):
    async with async_context as conn:
        await conn.execute("INSERT INTO Speciality (name, code, terms) VALUES ($1, $2, $3)", name, code, terms)
        speciality_id = await conn.fetchval("SELECT id FROM Speciality WHERE code = $1", code)
        for subject in subjects:
            await conn.execute("INSERT INTO SpecialitySubject (specialityId, subjectId) VALUES ($1, $2)",
                               speciality_id, subject)


async def delete_speciality(speciality_code):
    async with async_context as conn:
        await conn.execute("DELETE FROM Speciality WHERE code = $1", speciality_code)


async def edit_speciality(name, code, terms, subjects):
    async with async_context as conn:
        await conn.execute("UPDATE Speciality SET name = $1, terms = $2 WHERE code = $3", name, terms, code)
        speciality_id = await conn.fetchval("SELECT id FROM Speciality WHERE code = $1", code)
        await conn.execute("DELETE FROM SpecialitySubject WHERE specialityId = $1", speciality_id)
        for subject in subjects:
            await conn.execute("INSERT INTO SpecialitySubject (specialityId, subjectId) VALUES ($1, $2)",
                               speciality_id, subject)


async def get_all_subjects():
    async with async_context as conn:
        subjects = await conn.fetch("SELECT * FROM Subject")
        return subjects


async def add_subject(name, start_term, end_term):
    async with async_context as conn:
        await conn.execute("INSERT INTO Subject (name, startTerm, endTerm) VALUES ($1, $2, $3)", name,
start_term,
```

```
                end_term)


async def delete_subject(subject_id):
    async with async_context as conn:
        await conn.execute("DELETE FROM Subject WHERE id = $1", subject_id)


async def edit_subject(subject_id, name, start_term, end_term):
    async with async_context as conn:
        await conn.execute("UPDATE Subject SET name = $1, startTerm = $2, endTerm = $3 WHERE id = $4",
                           name, start_term, end_term, subject_id)


async def get_all_students():
    # out as id | first_name | second_name | last_name | speciality_name | speciality_code | course
    async with async_context as conn:
        students = await conn.fetch("SELECT b.telegramUserId, b.firstName, b.secondName, b.lastName, sp.name, "
                                    "sp.code, s.course FROM BotUser b "
                                    "JOIN Student s ON b.telegramUserId = s.telegramUserId "
                                    "JOIN StudentSpeciality ss ON s.telegramUserId = ss.studentId "
                                    "JOIN Speciality sp ON sp.id = ss.specialityId")
        return students


async def delete_student(telegram_id):
    async with async_context as conn:
        await conn.execute("DELETE FROM BotUser WHERE telegramUserId = $1", telegram_id)


async def edit_student(telegram_id, name, second_name, last_name, speciality_code, course):
    async with async_context as conn:
        await conn.execute(
            "UPDATE BotUser SET firstName = $1, secondName = $2, lastName = $3 WHERE telegramUserId = $4",
            name, second_name, last_name, telegram_id)
        speciality_id = await conn.fetchval("SELECT id FROM Speciality WHERE code = $1", speciality_code)
        await conn.execute("UPDATE Student SET course = $1 WHERE telegramUserId = $2", course, telegram_id)
```

```
    await conn.execute("UPDATE StudentSpeciality SET specialityId = $1 WHERE studentId = $2",
                       speciality_id, telegram_id)


async def get_all_teachers():
    async with async_context as conn:
        teachers = await conn.fetch("SELECT b.firstName, b.secondName, b.lastName, b.telegramUserId FROM
BotUser b "
                                    "WHERE b.position = $1", 'Teacher')
        teachers_with_subjects = []
        for teacher in teachers:
            teacher_id = teacher[3]
            subjects = await conn.fetch("SELECT s.name FROM Subject s "
                                        "JOIN TeacherSubject ts ON s.id = ts.subjectId "
                                        "JOIN Teacher t ON t.telegramUserId = ts.teacherId "
                                        "WHERE t.telegramUserId = $1", teacher_id)
            subjects = [subject[0] for subject in subjects]
            teachers_with_subjects.append((teacher[0], teacher[1], teacher[2], subjects, teacher[3]))
        return teachers_with_subjects


async def delete_teacher(telegram_id):
    async with async_context as conn:
        await conn.execute("DELETE FROM BotUser WHERE telegramUserId = $1", telegram_id)


async def edit_teacher(telegram_id, name, surname, patronymic, subjects):
    async with async_context as conn:
        await conn.execute(
            "UPDATE BotUser SET firstName = $1, secondName = $2, lastName = $3 WHERE telegramUserId = $4",
            name, surname, patronymic, telegram_id)
        await conn.execute("DELETE FROM TeacherSubject WHERE teacherId = $1", telegram_id)
        for subject in subjects:
            subject_id = await conn.fetchval("SELECT id FROM Subject WHERE name = $1", subject)
            await conn.execute("INSERT INTO TeacherSubject (teacherId, subjectId) VALUES ($1, $2)",
                               telegram_id, subject_id)
```

```
async def get_teacher_id(telegram_id):
    async with async_context as conn:
        teacher_id = await conn.fetchval("SELECT telegramUserId FROM Teacher WHERE telegramUserId = $1",
telegram_id)
        return teacher_id


async def get_teacher_subjects(teacher_id):
    async with async_context as conn:
        subjects = await conn.fetch("SELECT s.name FROM Subject s "
                                    "JOIN TeacherSubject ts ON s.id = ts.subjectId "
                                    "JOIN Teacher t ON t.telegramUserId = ts.teacherId "
                                    "WHERE t.telegramUserId = $1", teacher_id)
        subjects = [subject[0] for subject in subjects]
        return subjects


async def get_students_by_speciality(speciality_code):
    async with async_context as conn:
        students = await conn.fetch("SELECT b.firstName, b.secondName, b.lastName, b.telegramUserId FROM
BotUser b "
                                    "JOIN Student s ON b.telegramUserId = s.telegramUserId "
                                    "JOIN StudentSpeciality ss ON s.telegramUserId = ss.studentId "
                                    "JOIN Speciality sp ON sp.id = ss.specialityId "
                                    "WHERE sp.code = $1", speciality_code)
        return students


async def get_homeworks_by_subject_and_course(subject_id, course):
    async with async_context as conn:
        homeworks = await conn.fetch("SELECT text, date, isDone FROM StudentSubjectHomeWork "
                                     "WHERE subjectId = $1 AND course = $2", subject_id, course)
        return homeworks


async def get_student_homework_file_id(student_id, subject_id, date):
    async with async_context as conn:
```

```
    file_id = await conn.fetchval("SELECT telegramPhotoId FROM StudentSubjectHomeWork "
                                  "WHERE studentId = $1 AND subjectId = $2 AND date = $3",
                                  student_id, subject_id, date)
    return file_id


# create table StudentGrades
# (
#     id          serial    not null primary key,
#     studentId   int       not null,
#     subjectId   int       not null,
#     grade       int       not null check (grade >= 0 and grade <= 100),
#     date        timestamp not null default current_timestamp,
#     isFinalGrade boolean  not null default false,
#     foreign key (studentId) references Student (telegramUserId) on update cascade on delete cascade,
#     foreign key (subjectId) references Subject (id) on update cascade on delete cascade
# );
async def rate_student_homework(student_id, subject_id, date, grade, is_final):
  async with async_context as conn:
    # in StudentGrades table add new grade
    await conn.execute("INSERT INTO StudentGrades (studentId, subjectId, grade, date, isFinalGrade) "
                       "VALUES ($1, $2, $3, $4, $5)", student_id, subject_id, int(grade), date, bool(is_final))


# create type UserPosition as enum ('Student', 'Teacher', 'Admin');
#
# create table UserState
# (
#     telegramUserId serial      not null primary key,
#     state          varchar(255) not null
# );
#
# create table BotUser
# (
#     telegramUserId serial      not null primary key,
#     firstName      varchar(255) not null,
#     secondName     varchar(255) not null,
```

```
#     lastName       varchar(255) not null,
#     position       UserPosition not null
# );
#
# create table Student
# (
#     telegramUserId serial not null primary key,
#     course         int    not null default 1,
#     foreign key (telegramUserId) references BotUser (telegramUserId) on update cascade on delete cascade
# );
#
# create table Teacher
# (
#     telegramUserId serial not null primary key,
#     foreign key (telegramUserId) references BotUser (telegramUserId) on update cascade on delete cascade
# );
#
# create table Subject
# (
#     id        serial       not null primary key,
#     name      varchar(255) not null,
#     startTerm int          not null check (startTerm > 0),
#     endTerm   int          not null check (endTerm > 0)
# );
#
# create table Speciality
# (
#     id    serial       not null primary key,
#     name  varchar(255) not null,
#     code  varchar(255) not null,
#     terms int          not null check (terms > 0)
# );
#
# create table SpecialitySubject
# (
#     id           serial not null primary key,
#     specialityId int    not null,
```

```
#     subjectId    int    not null,
#     foreign key (specialityId) references Speciality (id) on update cascade on delete cascade,
#     foreign key (subjectId) references Subject (id) on update cascade on delete cascade
# );
#
# create table TeacherSubject
# (
#     id       serial not null primary key,
#     teacherId int    not null,
#     subjectId int    not null,
#     foreign key (teacherId) references Teacher (telegramUserId) on update cascade on delete cascade,
#     foreign key (subjectId) references Subject (id) on update cascade on delete cascade
# );
#
# create table StudentSpeciality
# (
#     id          serial not null primary key,
#     studentId    int    not null,
#     specialityId int    not null,
#     foreign key (studentId) references Student (telegramUserId) on update cascade on delete cascade,
#     foreign key (specialityId) references Speciality (id) on update cascade on delete cascade
# );
#
# create table StudentGrades
# (
#     id          serial    not null primary key,
#     studentId    int       not null,
#     subjectId    int       not null,
#     grade        int       not null check (grade >= 0 and grade <= 100),
#     date         timestamp not null default current_timestamp,
#     isFinalGrade boolean   not null default false,
#     foreign key (studentId) references Student (telegramUserId) on update cascade on delete cascade,
#     foreign key (subjectId) references Subject (id) on update cascade on delete cascade
# );
#
# create table StudentSubjectHomeWork
# (
```

```
#     id            serial      not null primary key,
#     studentId     int         not null,
#     subjectId     int         not null,
#     date          varchar(255) not null,
#     text          text        not null,
#     isDone        boolean     not null default false,
#     telegramPhotoId varchar(255) not null,
#     foreign key (studentId) references Student (telegramUserId) on update cascade on delete cascade,
#     foreign key (subjectId) references Subject (id) on update cascade on delete cascade
# );
#
# create table StudentHomeworkWaitingStatus
# (
#     id          serial      not null primary key,
#     studentId   int         not null,
#     subjectName varchar(255) not null,
#     foreign key (studentId) references Student (telegramUserId) on update cascade on delete cascade
# );
#
# create table Schedule
# (
#     id            serial      not null primary key,
#     specialityId  int         not null,
#     telegramPhotoId varchar(255) not null,
#     foreign key (specialityId) references Speciality (id) on update cascade on delete cascade
# );
async def upload_homework_as_teacher(subject_id, speciality_code, course, date, homework_text):
    async with async_context as conn:
        # create new StudentSubjectHomeWork
        students = await conn.fetch("SELECT b.telegramUserId FROM BotUser b "
                                    "JOIN Student s ON b.telegramUserId = s.telegramUserId "
                                    "JOIN StudentSpeciality ss ON s.telegramUserId = ss.studentId "
                                    "JOIN Speciality sp ON sp.id = ss.specialityId "
                                    "WHERE sp.code = $1 AND s.course = $2", speciality_code, course)
        for student in students:
            await conn.execute("INSERT INTO StudentSubjectHomeWork (studentId, subjectId, date, text) "
                               "VALUES ($1, $2, $3, $4)", student[0], subject_id, date, homework_text)
```

```
async def add_schedule(file_id, speciality_code):
    async with async_context as conn:
        speciality_id = await conn.fetchval("SELECT id FROM Speciality WHERE code = $1", speciality_code)
        await conn.execute("INSERT INTO Schedule (specialityId, telegramPhotoId) VALUES ($1, $2)",
                           speciality_id, file_id)
```

UsersService.py

```
import Db.DbContext as DbContext


async def is_registered(telegram_id):
    users = await DbContext.get_users()
    for user in users:
        if user[0] == telegram_id:
            return True
    return False


async def get_user_state(telegram_id):
    userStates = await DbContext.get_user_states()
    for userState in userStates:
        if userState[0] == telegram_id:
            return userState[1]
    return None


async def set_user_state(telegram_id, state):
    # set user state in database
    await DbContext.set_user_state(telegram_id, state)


async def is_speciality_exists(speciality_code):
    specialities = await DbContext.get_specialities()
    for speciality in specialities:
        if speciality[2] == speciality_code:
            return True
```

```
    return False


async def register_student(telegram_id, first_name, second_name, last_name, speciality, course):
    await DbContext.register_student(telegram_id, first_name, second_name, last_name, speciality, course)


async def get_speciality_terms_count(speciality_code):
    specialities = await DbContext.get_specialities()
    for speciality in specialities:
        if speciality[2] == speciality_code:
            return speciality[3]
    return 0


async def register_teacher(telegram_id, first_name, second_name, last_name):
    await DbContext.register_teacher(telegram_id, first_name, second_name, last_name)


async def register_admin(telegram_id, first_name, second_name, last_name):
    await DbContext.register_admin(telegram_id, first_name, second_name, last_name)


async def get_student_subjects(telegram_id):
    subjects = await DbContext.get_student_subjects(telegram_id)
    return subjects


async def get_student_grades(telegram_id, subject_name):
    grades_with_dates = await DbContext.get_student_grades(telegram_id, subject_name)
    return grades_with_dates


async def get_student_homeworks(telegram_id, subject_name):
    homeworks_with_dates = await DbContext.get_student_homeworks(telegram_id, subject_name)
    return homeworks_with_dates
```

```
async def get_group_students(telegram_id):
    students = await DbContext.get_group_students(telegram_id)
    return students


async def get_group_rating(telegram_id):
    students_with_avg_grades = await DbContext.get_group_rating(telegram_id)
    for student in students_with_avg_grades:
        student[3] = float(student[3])
    students_with_avg_grades.sort(key=lambda x: x[3], reverse=True)
    return students_with_avg_grades


async def get_specialities():
    specialities = await DbContext.get_specialities()
    return specialities


async def get_teachers_with_subjects():
    teachers_with_subjects = await DbContext.get_teachers()
    return teachers_with_subjects


async def get_teachers_by_speciality(speciality_code):
    teachers_with_subjects = await DbContext.get_teachers_by_speciality(speciality_code)
    return teachers_with_subjects


async def get_student_speciality_code(telegram_id):
    speciality_code = await DbContext.get_student_speciality_code(telegram_id)
    return speciality_code


async def get_subjects_by_speciality(speciality_code):
    subjects = await DbContext.get_subjects_names_by_speciality(speciality_code)
    return subjects
```

```
async def get_subjects_ids_by_speciality(speciality_code):
    subjects = await DbContext.get_subjects_ids_by_speciality(speciality_code)
    return subjects


async def get_teacher_by_subject(subject_name):
    teacher = await DbContext.get_teacher_by_subject(subject_name)
    return teacher


async def get_student_course(telegram_id):
    course = await DbContext.get_student_course(telegram_id)
    return course


async def get_schedule_image(speciality_id):
    image = await DbContext.get_schedule_image(speciality_id)
    return image


async def get_student_speciality_id(telegram_id):
    speciality_id = await DbContext.get_student_speciality_id(telegram_id)
    return speciality_id


async def set_user_waiting_homework_state(telegram_id, subject_name):
    await DbContext.set_user_waiting_homework_state(telegram_id, subject_name)


async def get_user_waiting_homework_state(telegram_id):
    subject_name = await DbContext.get_user_waiting_homework_state(telegram_id)
    return subject_name


async def get_subject_id_by_name(subject_name):
```

```
    subject_id = await DbContext.get_subject_id_by_name(subject_name)
    return subject_id


async def upload_homework(telegram_id, subject_id, file_id, date):
    await DbContext.upload_homework(telegram_id, subject_id, file_id, date)


async def get_all_users():
    users = await DbContext.get_all_users()
    return users


async def get_students():
    users = await DbContext.get_all_users()
    students = []
    for user in users:
        if user[4] == "Student":
            students.append(user)
    return students


async def get_teachers():
    users = await DbContext.get_all_users()
    teachers = []
    for user in users:
        if user[4] == "Teacher":
            teachers.append(user)
    return teachers


async def get_admins():
    users = await DbContext.get_all_users()
    admins = []
    for user in users:
        if user[4] == "Admin":
            admins.append(user)
```

```
    return admins


async def get_all_specialities():
    specialities = await DbContext.get_all_specialities()
    return specialities


async def add_speciality(name, code, terms, subjects):
    await DbContext.add_speciality(name, code, terms, subjects)


async def delete_speciality(speciality_code):
    await DbContext.delete_speciality(speciality_code)


async def edit_speciality(name, code, terms, subjects):
    await DbContext.edit_speciality(name, code, terms, subjects)


async def get_all_subjects():
    subjects = await DbContext.get_all_subjects()
    return subjects


async def add_subject(name, start_term, end_term):
    await DbContext.add_subject(name, start_term, end_term)


async def is_subject_exists(subject_id):
    subjects = await DbContext.get_all_subjects()
    for subject in subjects:
        subject = list(subject)
        if int(subject[0]) == int(subject_id):
            return True
    return False
```

```
async def delete_subject(subject_id):
    await DbContext.delete_subject(subject_id)


async def edit_subject(subject_id, name, start_term, end_term):
    await DbContext.edit_subject(subject_id, name, start_term, end_term)


async def get_all_students():
    students = await DbContext.get_all_students()
    return students


async def is_student_exists(telegram_id):
    students = await DbContext.get_all_students()
    for student in students:
        if student[0] == telegram_id:
            return True
    return False


async def delete_student(telegram_id):
    await DbContext.delete_student(telegram_id)


async def edit_student(telegram_id, name, second_name, last_name, speciality_code, course):
    await DbContext.edit_student(telegram_id, name, second_name, last_name, speciality_code, course)


async def get_all_teachers():
    teachers = await DbContext.get_all_teachers()
    return teachers


async def is_teacher_exists(telegram_id):
    teachers = await DbContext.get_all_teachers()
```

```
    for teacher in teachers:
        if teacher[4] == telegram_id:
            return True
    return False


async def delete_teacher(telegram_id):
    await DbContext.delete_teacher(telegram_id)


async def edit_teacher(telegram_id, name, surname, patronymic, subjects):
    await DbContext.edit_teacher(telegram_id, name, surname, patronymic, subjects)


async def get_teacher_id(telegram_id):
    teacher_id = await DbContext.get_teacher_id(telegram_id)
    return teacher_id


async def get_teacher_subjects(telegram_id):
    teacher_id = await DbContext.get_teacher_id(telegram_id)
    subjects = await DbContext.get_teacher_subjects(teacher_id)
    return subjects


async def get_students_by_speciality(speciality_code):
    students = await DbContext.get_students_by_speciality(speciality_code)
    return students


async def get_homeworks_by_subject_and_course(subject_id, course):
    homeworks = await DbContext.get_homeworks_by_subject_and_course(subject_id, course)
    return homeworks


async def get_student_homework_file_id(student_id, subject_id, date):
    file_id = await DbContext.get_student_homework_file_id(student_id, subject_id, date)
```

```
    return file_id


async def rate_student_homework(student_id, subject_id, date, grade, is_final):
    await DbContext.rate_student_homework(student_id, subject_id, date, grade, is_final)


async def upload_homework_as_teacher(subject_id, speciality_code, course, date, homework_text):
    await DbContext.upload_homework_as_teacher(subject_id, speciality_code, course, date, homework_text)


async def add_schedule(file_id, speciality_code):
    await DbContext.add_schedule(file_id, speciality_code)
```

admin_handler.py

```
from aiogram import Router, types, F
from Keyboards import admin_keyboards
import Db.UsersService as UsersService
from logger import log

router = Router()


@router.callback_query(F.data == "admin_view_users_info")
async def admin_view_users_info(callback: types.CallbackQuery):
    log("Ви обрали інформацію про користувачів (admin_handler.py)")
    await UsersService.set_user_state(callback.from_user.id, "registered_as_admin")
    await callback.message.edit_text("Інформація про користувачів",
                                     reply_markup=admin_keyboards.admin_users_info_keyboard)


@router.callback_query(F.data == "admin_view_specialities_info")
async def admin_view_specialities_info(callback: types.CallbackQuery):
    log("Ви обрали подивитися інформацію про спеціальності (admin_handler.py)")
    await UsersService.set_user_state(callback.from_user.id, "registered_as_admin")
    await callback.message.edit_text("Інформація про спеціальності",
                                     reply_markup=admin_keyboards.admin_specialities_info_keyboard)
```

```
@router.callback_query(F.data == "admin_view_subjects_info")
async def admin_view_subjects_info(callback: types.CallbackQuery):
    log("Ви обрали інформацію про предмети (admin_handler.py)")
    await UsersService.set_user_state(callback.from_user.id, "registered_as_admin")
    await callback.message.edit_text("Інформація про предмети",
                                     reply_markup=admin_keyboards.admin_subjects_info_keyboard)


@router.callback_query(F.data == "admin_view_students_info")
async def admin_view_students_info(callback: types.CallbackQuery):
    log("Ви обрали інформацію про студентів (admin_handler.py)")
    await UsersService.set_user_state(callback.from_user.id, "registered_as_admin")
    await callback.message.edit_text("Інформація про студентів",
                                     reply_markup=admin_keyboards.admin_students_info_keyboard)


@router.callback_query(F.data == "admin_view_teachers_info")
async def admin_view_teachers_info(callback: types.CallbackQuery):
    log("Ви обрали подивитися інформацію про викладачів (admin_handler.py)")
    await UsersService.set_user_state(callback.from_user.id, "registered_as_admin")
    await callback.message.edit_text("Інформація про викладачів",
                                     reply_markup=admin_keyboards.admin_teachers_info_keyboard)


@router.callback_query(F.data == "admin_view_all_users")
async def admin_view_all_users(callback: types.CallbackQuery):
    log("Ви вибрали подивитися всіх користувачів (admin_handler.py)")
    users = await UsersService.get_all_users()
    if len(users) == 0:
        await callback.message.edit_text("Користувачів немає",
                                         reply_markup=admin_keyboards.back_to_admin_users_info_keyboard)
        return
    text = ""
    for user in users:
        text += f"ID: {user[0]}\n" \
                f"Ім'я: {user[1]}\n" \
```

```
            f"Прізвище: {user[2]}\n" \
            f"По батькові: {user[3]}\n" \
            f"Роль: {user[4]}\n" \
            f"-------------------------\n"

    await callback.message.edit_text("Користувачі:\n" + text,
                                     reply_markup=admin_keyboards.back_to_admin_users_info_keyboard)


@router.callback_query(F.data == "admin_view_students")
async def admin_view_students(callback: types.CallbackQuery):
    log("Ви обрали подивитися студентів (admin_handler.py)")
    users = await UsersService.get_students()
    if len(users) == 0:
        await callback.message.edit_text("Студентів немає",
                                         reply_markup=admin_keyboards.back_to_admin_users_info_keyboard)
        return
    text = ""
    for user in users:
        text += f"ID: {user[0]}\n" \
            f"Ім'я: {user[1]}\n" \
            f"Прізвище: {user[2]}\n" \
            f"По батькові: {user[3]}\n" \
            f"Роль: {user[4]}\n" \
            f"-------------------------\n"

    await callback.message.edit_text("Студенти:\n" + text,
                                     reply_markup=admin_keyboards.back_to_admin_users_info_keyboard)


@router.callback_query(F.data == "admin_view_teachers")
async def admin_view_teachers(callback: types.CallbackQuery):
    log("Ви обрали подивитися викладачів (admin_handler.py)")
    users = await UsersService.get_teachers()
    if len(users) == 0:
        await callback.message.edit_text("Викладачів немає",
                                         reply_markup=admin_keyboards.back_to_admin_users_info_keyboard)
```

```
        return
    text = ""
    for user in users:
        text += f"ID: {user[0]}\n" \
                f"Ім'я: {user[1]}\n" \
                f"Прізвище: {user[2]}\n" \
                f"По батькові: {user[3]}\n" \
                f"Роль: {user[4]}\n" \
                f"-------------------------\n"

    await callback.message.edit_text("Викладачі:\n" + text,
                                     reply_markup=admin_keyboards.back_to_admin_users_info_keyboard)


@router.callback_query(F.data == "admin_view_admins")
async def admin_view_admins(callback: types.CallbackQuery):
    log("Ви обрали подивитися адміністраторів (admin_handler.py)")
    users = await UsersService.get_admins()
    if len(users) == 0:
        await callback.message.edit_text("Адміністраторів немає",
                                         reply_markup=admin_keyboards.back_to_admin_users_info_keyboard)
        return
    text = ""
    for user in users:
        text += f"ID: {user[0]}\n" \
                f"Ім'я: {user[1]}\n" \
                f"Прізвище: {user[2]}\n" \
                f"По батькові: {user[3]}\n" \
                f"Роль: {user[4]}\n" \
                f"-------------------------\n"

    await callback.message.edit_text("Адміністратори:\n" + text,
                                     reply_markup=admin_keyboards.back_to_admin_users_info_keyboard)


@router.callback_query(F.data == "admin_view_all_specialities")
async def admin_view_all_specialities(callback: types.CallbackQuery):
```

```
    log("Ви обрали подивитися всі спеціальності (admin_handler.py)")
    specialities = await UsersService.get_all_specialities()
    if len(specialities) == 0:
        await callback.message.edit_text("Спеціальностей немає",
                                         reply_markup=admin_keyboards.back_to_admin_specialities_info_keyboard)
        return
    text = ""
    for speciality in specialities:
        subjects_names = await UsersService.get_subjects_by_speciality(speciality[2])
        subjects_ids = await UsersService.get_subjects_ids_by_speciality(speciality[2])
        subjects = []
        for i in range(len(subjects_names)):
            subjects.append((subjects_ids[i], subjects_names[i]))
        text += f"ID: {speciality[0]}, Назва: {speciality[1]}, Код: {speciality[2]}, Семестрів: {speciality[3]}\n" \
                f"Предмети:\n"
        for subject in subjects:
            text += f"ID: {subject[0]}, Назва: {subject[1]};"
        text += f"\n-------------------------\n"

    await callback.message.edit_text("Спеціальності:\n" + text,
                                     reply_markup=admin_keyboards.back_to_admin_specialities_info_keyboard)


@router.callback_query(F.data == "admin_add_speciality")
async def admin_add_speciality_input(callback: types.CallbackQuery):
    log("Ви вибрали додати спеціальність (admin_handler.py)")
    await callback.message.edit_text(
        "Введіть назву спеціальності, код та кількість семестрів через пробіл, а потім ID предметів, які будуть
вивчатися (наприклад, Спеціальність 100 8 32 23 43)")
    await UsersService.set_user_state(callback.from_user.id, "admin_add_speciality")


async def admin_add_speciality(message: types.Message):
    log("Ви вибрали додати спеціальність (admin_handler.py)")
    data = message.text.split()
    if len(data) < 3:
        await message.answer("Неправильний формат, спробуйте ще раз",
```

```
                reply_markup=admin_keyboards.back_to_admin_specialities_info_keyboard)
    await UsersService.set_user_state(message.from_user.id, "registered_as_admin")
    return
  if await UsersService.is_speciality_exists(data[1]):
    await message.answer("Спеціальність з таким кодом вже існує",
                reply_markup=admin_keyboards.back_to_admin_specialities_info_keyboard)
    await UsersService.set_user_state(message.from_user.id, "registered_as_admin")
    return
  name = data[0]
  code = data[1]
  terms_count = int(data[2])
  subjects = []
  not_exists_subjects = []
  for i in range(3, len(data)):
    subject_id = int(data[i])
    if not await UsersService.is_subject_exists(subject_id):
      not_exists_subjects.append(subject_id)
    else:
      subjects.append(subject_id)
  await UsersService.add_speciality(name, code, terms_count, subjects)
  await UsersService.set_user_state(message.from_user.id, "registered_as_admin")
  not_existed_text = "" if len(
    not_exists_subjects) == 0 else f"\nДеякі предмети не були додані, оскільки не існують."
  await message.answer("Спеціальність успішно додана" + not_existed_text,
              reply_markup=admin_keyboards.back_to_admin_specialities_info_keyboard)


@router.callback_query(F.data == "admin_delete_speciality")
async def admin_delete_speciality_input(callback: types.CallbackQuery):
  log("Ви вибрали видалити спеціальність (admin_handler.py)")
  specialities = await UsersService.get_all_specialities()
  if len(specialities) == 0:
    await callback.message.edit_text("Спеціальностей немає",
                    reply_markup=admin_keyboards.back_to_admin_specialities_info_keyboard)
    return
  specialities = [[speciality[0], speciality[1], speciality[2], speciality[3]] for speciality in specialities]
  text = "Спеціальності:\n"
```

```
    for speciality in specialities:
        text += f"{speciality[1]} - {speciality[2]}\n"
    await callback.message.edit_text(text + "Введіть код спеціальності, яку хочете видалити")
    await UsersService.set_user_state(callback.from_user.id, "admin_delete_speciality")


async def admin_delete_speciality(message: types.Message):
    log("Ви вибрали видалити спеціальність (admin_handler.py)")
    speciality_code = message.text
    if not await UsersService.is_speciality_exists(speciality_code):
        await message.answer("Спеціальність з таким кодом не знайдена",
                             reply_markup=admin_keyboards.back_to_admin_specialities_info_keyboard)
        await UsersService.set_user_state(message.from_user.id, "registered_as_admin")
        return
    await UsersService.delete_speciality(speciality_code)
    await message.answer("Спеціальність успішно видалена",
                         reply_markup=admin_keyboards.back_to_admin_specialities_info_keyboard)
    await UsersService.set_user_state(message.from_user.id, "registered_as_admin")


@router.callback_query(F.data == "admin_edit_speciality")
async def admin_edit_speciality_input(callback: types.CallbackQuery):
    log("Ви вибрали редагувати спеціальність (admin_handler.py)")
    specialities = await UsersService.get_all_specialities()
    if len(specialities) == 0:
        await callback.message.edit_text("Спеціальностей немає",
                                         reply_markup=admin_keyboards.back_to_admin_specialities_info_keyboard)
        return
    specialities = [[speciality[0], speciality[1], speciality[2], speciality[3]] for speciality in specialities]
    text = "Спеціальності:\n"
    for speciality in specialities:
        text += f"ID: {speciality[0]}, Назва: {speciality[1]}, Код: {speciality[2]}, Семестрів: {speciality[3]}\n"
    text += "--------------------------------\n"
    subjects = await UsersService.get_all_subjects()
    subjects = [[subject[0], subject[1], subject[2], subject[3]] for subject in subjects]
    sub_text = "Предмети:\n"
    for subject in subjects:
```

```
    sub_text += f"ID: {subject[0]} - {subject[1]}\n"
  sub_text += "-----------------------------------\n"
  await callback.message.edit_text(text + sub_text +
                                   "Введіть код спеціальності, її нову назву, кількість семестрів та ID предметів через
пробіл (наприклад, 100 Спеціальність 8 32 23 43)")
  await UsersService.set_user_state(callback.from_user.id, "admin_edit_speciality_input_data")


async def admin_edit_speciality_input_data(message: types.Message):
  log("Ви вибрали редагувати спеціальність (admin_handler.py)")
  data = message.text.split()
  if len(data) < 4:
    await message.answer("Неправильний формат, спробуйте ще раз",
                         reply_markup=admin_keyboards.back_to_admin_specialities_info_keyboard)
    await UsersService.set_user_state(message.from_user.id, "registered_as_admin")
    return
  if not await UsersService.is_speciality_exists(data[0]):
    await message.answer("Спеціальність з таким кодом не знайдена",
                         reply_markup=admin_keyboards.back_to_admin_specialities_info_keyboard)
    await UsersService.set_user_state(message.from_user.id, "registered_as_admin")
    return
  name = data[0]
  code = data[1]
  terms_count = int(data[2])
  subjects = []
  for i in range(3, len(data)):
    subject_id = int(data[i])
    if await UsersService.is_subject_exists(subject_id):
      subjects.append(subject_id)
  await UsersService.edit_speciality(code, name, terms_count, subjects)
  await message.answer("Спеціальність успішно відредагована",
                       reply_markup=admin_keyboards.back_to_admin_specialities_info_keyboard)
  await UsersService.set_user_state(message.from_user.id, "registered_as_admin")


@router.callback_query(F.data == "admin_view_all_subjects")
async def admin_view_all_subjects(callback: types.CallbackQuery):
```

```
    log("Ви обрали подивитися всі предмети (admin_handler.py)")
    subjects = await UsersService.get_all_subjects()
    if len(subjects) == 0:
        await callback.message.edit_text("Предметів немає",
                                         reply_markup=admin_keyboards.back_to_admin_subjects_info_keyboard)
        return
    text = ""
    for subject in subjects:
        text += f"ID: {subject[0]}, Назва: {subject[1]}, Стартовий семестр: {subject[2]}, Кінцевий семестр:
{subject[3]}\n" \
                f"-------------------------\n"

    await callback.message.edit_text("Предмети:\n" + text,
                                     reply_markup=admin_keyboards.back_to_admin_subjects_info_keyboard)


@router.callback_query(F.data == "admin_add_subject")
async def admin_add_subject_input(callback: types.CallbackQuery):
    log("Ви вибрали додати предмет (admin_handler.py)")
    await callback.message.edit_text(
        "Введіть назву предмета, стартовий та кінцевий семестр через пробіл (наприклад, Математика 1 2)")
    await UsersService.set_user_state(callback.from_user.id, "admin_add_subject")


async def admin_add_subject(message: types.Message):
    log("Ви вибрали додати предмет (admin_handler.py)")
    data = message.text.split()
    if len(data) != 3:
        await message.answer("Неправильний формат, спробуйте ще раз",
                             reply_markup=admin_keyboards.back_to_admin_subjects_info_keyboard)
        await UsersService.set_user_state(message.from_user.id, "registered_as_admin")
        return
    await UsersService.add_subject(data[0], int(data[1]), int(data[2]))
    await UsersService.set_user_state(message.from_user.id, "registered_as_admin")
    await message.answer("Предмет успішно доданий",
                         reply_markup=admin_keyboards.back_to_admin_subjects_info_keyboard)
```

```
@router.callback_query(F.data == "admin_delete_subject")
async def admin_delete_subject_input(callback: types.CallbackQuery):
    log("Ви вибрали видалити предмет (admin_handler.py)")
    subjects = await UsersService.get_all_subjects()
    if len(subjects) == 0:
        await callback.message.edit_text("Предметів немає",
                                         reply_markup=admin_keyboards.back_to_admin_subjects_info_keyboard)
        return
    subjects = [[subject[0], subject[1], subject[2], subject[3]] for subject in subjects]
    text = "Предмети:\n"
    for subject in subjects:
        text += f"{subject[1]} - {subject[0]}\n"
    await callback.message.edit_text(text + "Введіть ID предмета, який хочете видалити")
    await UsersService.set_user_state(callback.from_user.id, "admin_delete_subject")


async def admin_delete_subject(message: types.Message):
    log("Ви вибрали видалити предмет (admin_handler.py)")
    subject_id = int(message.text)
    if not await UsersService.is_subject_exists(subject_id):
        await message.answer("Предмет з таким ID не знайдено",
                             reply_markup=admin_keyboards.back_to_admin_subjects_info_keyboard)
        await UsersService.set_user_state(message.from_user.id, "registered_as_admin")
        return
    await UsersService.delete_subject(subject_id)
    await message.answer("Предмет успішно видалено",
                         reply_markup=admin_keyboards.back_to_admin_subjects_info_keyboard)
    await UsersService.set_user_state(message.from_user.id, "registered_as_admin")


@router.callback_query(F.data == "admin_edit_subject")
async def admin_edit_subject_input(callback: types.CallbackQuery):
    log("Ви вибрали редагувати предмет (admin_handler.py)")
    subjects = await UsersService.get_all_subjects()
    if len(subjects) == 0:
        await callback.message.edit_text("Предметів немає",
```

```
                                reply_markup=admin_keyboards.back_to_admin_subjects_info_keyboard)
        return
    subjects = [[subject[0], subject[1], subject[2], subject[3]] for subject in subjects]
    text = "Предмети:\n"
    for subject in subjects:
        text += f"ID: {subject[0]}, Назва: {subject[1]}, Стартовий семестр: {subject[2]}, Кінцевий семестр:
{subject[3]}\n"
    await callback.message.edit_text(text +
                                     "Введіть ID предмета, а потім його нову назву, стартовий та кінцевий семестр через
пробіл (наприклад, 321 Математика 1 2)")
    await UsersService.set_user_state(callback.from_user.id, "admin_edit_subject_input_data")


async def admin_edit_subject_input_data(message: types.Message):
    log("Ви вибрали редагувати предмет (admin_handler.py)")
    data = message.text.split()
    if len(data) != 4:
        await message.answer("Неправильний формат, спробуйте ще раз",
                             reply_markup=admin_keyboards.back_to_admin_subjects_info_keyboard)
        await UsersService.set_user_state(message.from_user.id, "registered_as_admin")
        return
    if not await UsersService.is_subject_exists(int(data[0])):
        await message.answer("Предмет з таким ID не знайдено",
                             reply_markup=admin_keyboards.back_to_admin_subjects_info_keyboard)
        await UsersService.set_user_state(message.from_user.id, "registered_as_admin")
        return

    await UsersService.edit_subject(int(data[0]), data[1], int(data[2]), int(data[3]))
    await message.answer("Предмет успішно відредаговано",
                         reply_markup=admin_keyboards.back_to_admin_subjects_info_keyboard)
    await UsersService.set_user_state(message.from_user.id, "registered_as_admin")


@router.callback_query(F.data == "admin_view_all_students")
async def admin_view_all_students(callback: types.CallbackQuery):
    log("Ви обрали подивитися всіх студентів (admin_handler.py)")
    students = await UsersService.get_all_students()
    if len(students) == 0:
```

```
    await callback.message.edit_text("Студентів немає",
                                     reply_markup=admin_keyboards.back_to_admin_students_info_keyboard)
    return
  text = ""
  for student in students:
    text += f"ID: {student[0]}, Ім'я: {student[1]}, Прізвище: {student[2]}, По батькові: {student[3]}, " \
        f"Спеціальність: {student[4]} {student[5]}, Курс: {student[6]}\n" \
        f"-------------------------\n"

  await callback.message.edit_text("Студенти:\n" + text,
                                   reply_markup=admin_keyboards.back_to_admin_students_info_keyboard)


@router.callback_query(F.data == "admin_delete_student")
async def admin_delete_student_input(callback: types.CallbackQuery):
  log("Ви обрали видалити студента (admin_handler.py)")
  students = await UsersService.get_all_students()
  if len(students) == 0:
    await callback.message.edit_text("Студентів немає",
                                     reply_markup=admin_keyboards.back_to_admin_students_info_keyboard)
    return
  students = [[student[0], student[1], student[2], student[3], student[4], student[5], student[6]] for student in
        students]
  text = "Студенти:\n"
  for student in students:
    text += f"{student[1]} {student[2]} {student[3]} ({student[4]} {student[5]}) - {student[0]}\n"
  await callback.message.edit_text(text + "Введіть ID студента, якого хочете видалити")
  await UsersService.set_user_state(callback.from_user.id, "admin_delete_student")


async def admin_delete_student(message: types.Message):
  log("Ви обрали видалити студента (admin_handler.py)")
  student_id = int(message.text)
  if not await UsersService.is_student_exists(student_id):
    await message.answer("Студент з таким ID не знайдено",
                         reply_markup=admin_keyboards.back_to_admin_students_info_keyboard)
    await UsersService.set_user_state(message.from_user.id, "registered_as_admin")
```

```
        return
    await UsersService.delete_student(student_id)
    await message.answer("Студент успішно видалено",
                         reply_markup=admin_keyboards.back_to_admin_students_info_keyboard)
    await UsersService.set_user_state(message.from_user.id, "registered_as_admin")


@router.callback_query(F.data == "admin_edit_student")
async def admin_edit_student_input(callback: types.CallbackQuery):
    log("Ви обрали редагувати студента (admin_handler.py)")
    students = await UsersService.get_all_students()
    if len(students) == 0:
        await callback.message.edit_text("Студентів немає",
                                         reply_markup=admin_keyboards.back_to_admin_students_info_keyboard)
        return
    students = [[student[0], student[1], student[2], student[3], student[4], student[5], student[6]] for student in
                students]
    text = "Студенти:\n"
    for student in students:
        text += f"{student[1]} {student[2]} {student[3]} ({student[4]} {student[5]} - {student[6]} курс) -
{student[0]}\n"
    await callback.message.edit_text(text +
                                     "Введіть ID студента, а потім його нове ім'я, прізвище, по батькові, код спеціальності
та поточний курс через пропуск (наприклад, 321 Іванов Іван Іванович 125 2)")
    await UsersService.set_user_state(callback.from_user.id, "admin_edit_student_input_data")


async def admin_edit_student_input_data(message: types.Message):
    log("Ви обрали редагувати студента (admin_handler.py)")
    data = message.text.split()
    if len(data) != 6:
        await message.answer("Неправильний формат, спробуйте ще раз",
                             reply_markup=admin_keyboards.back_to_admin_students_info_keyboard)
        await UsersService.set_user_state(message.from_user.id, "registered_as_admin")
        return
    if not await UsersService.is_student_exists(int(data[0])):
        await message.answer("Студент з таким ID не знайдено",
                             reply_markup=admin_keyboards.back_to_admin_students_info_keyboard)
```

```
        await UsersService.set_user_state(message.from_user.id, "registered_as_admin")
        return
    speciality_terms = await UsersService.get_speciality_terms_count(data[4])
    if int(data[5]) < 1 or int(data[5]) > int(speciality_terms):
        await message.answer("Будь ласка, вкажіть номер курсу в діапазоні від 1 до " + str(speciality_terms),
                             reply_markup=admin_keyboards.back_to_admin_students_info_keyboard)
        await UsersService.set_user_state(message.from_user.id, "registered_as_admin")
        return
    await UsersService.edit_student(int(data[0]), data[1], data[2], data[3], data[4], int(data[5]))
    await message.answer("Студент успішно відредаговано",
                         reply_markup=admin_keyboards.back_to_admin_students_info_keyboard)
    await UsersService.set_user_state(message.from_user.id, "registered_as_admin")


@router.callback_query(F.data == "admin_view_all_teachers")
async def admin_view_all_teachers(callback: types.CallbackQuery):
    log("Ви обрали подивитися всіх викладачів (admin_handler.py)")
    teachers = await UsersService.get_all_teachers()
    if len(teachers) == 0:
        await callback.message.edit_text("Викладачів немає",
                                         reply_markup=admin_keyboards.back_to_admin_teachers_info_keyboard)
        return
    text = ""
    for teacher in teachers:
        text += f"ID: {teacher[4]}, Ім'я: {teacher[0]}, Прізвище: {teacher[1]}, По батькові: {teacher[2]}, Предмети:
{teacher[3]}\ n" \
                f"-------------------------\n"

    await callback.message.edit_text("Викладачі:\n" + text,
                                     reply_markup=admin_keyboards.back_to_admin_teachers_info_keyboard)


@router.callback_query(F.data == "admin_delete_teacher")
async def admin_delete_teacher_input(callback: types.CallbackQuery):
    log("Ви вибрали видалити викладача (admin_handler.py)")
    teachers = await UsersService.get_all_teachers()
    if len(teachers) == 0:
```

```
    await callback.message.edit_text("Викладачів немає",
                                     reply_markup=admin_keyboards.back_to_admin_teachers_info_keyboard)
    return
  teachers = [[teacher[0], teacher[1], teacher[2], teacher[3], teacher[4]] for teacher in teachers]
  text = "Викладачі:\n"
  for teacher in teachers:
    text += f"{teacher[0]} {teacher[1]} {teacher[2]} - {teacher[4]}\n"
  await callback.message.edit_text(text + "Введіть ID викладача, якого хочете видалити")
  await UsersService.set_user_state(callback.from_user.id, "admin_delete_teacher")


async def admin_delete_teacher(message: types.Message):
  log("Ви вибрали видалити викладача (admin_handler.py)")
  teacher_id = int(message.text)
  if not await UsersService.is_teacher_exists(teacher_id):
    await message.answer("Викладач з таким ID не знайдено",
                         reply_markup=admin_keyboards.back_to_admin_teachers_info_keyboard)
    await UsersService.set_user_state(message.from_user.id, "registered_as_admin")
    return
  await UsersService.delete_teacher(teacher_id)
  await message.answer("Викладач успішно видалено",
                       reply_markup=admin_keyboards.back_to_admin_teachers_info_keyboard)
  await UsersService.set_user_state(message.from_user.id, "registered_as_admin")


@router.callback_query(F.data == "admin_edit_teacher")
async def admin_edit_teacher_input(callback: types.CallbackQuery):
  log("Ви обрали редагувати викладача (admin_handler.py)")
  teachers = await UsersService.get_all_teachers()
  if len(teachers) == 0:
    await callback.message.edit_text("Викладачів немає",
                                     reply_markup=admin_keyboards.back_to_admin_teachers_info_keyboard)
    return
  teachers = [[teacher[0], teacher[1], teacher[2], teacher[3], teacher[4]] for teacher in teachers]
  text = "Викладачі:\n"
  for teacher in teachers:
    text += f"{teacher[4]} - {teacher[0]} {teacher[1]} {teacher[2]} ({teacher[3]})\n"
```

```
    await callback.message.edit_text(text +
                                     "Введіть ID викладача, а потім його нове ім'я, прізвище, по батькові через прогалину, а далі предмети, які він веде через прогалину (наприклад, 321 Іванов Іван Іванович Математика Фізика)")
    await UsersService.set_user_state(callback.from_user.id, "admin_edit_teacher_input_data")


async def admin_edit_teacher_input_data(message: types.Message):
    log("Ви обрали редагувати викладача (admin_handler.py)")
    data = message.text.split()
    if len(data) < 5:
        await message.answer("Неправильний формат, спробуйте ще раз",
                             reply_markup=admin_keyboards.back_to_admin_teachers_info_keyboard)
        await UsersService.set_user_state(message.from_user.id, "registered_as_admin")
        return
    if not await UsersService.is_teacher_exists(int(data[0])):
        await message.answer("Викладач з таким ID не знайдено",
                             reply_markup=admin_keyboards.back_to_admin_teachers_info_keyboard)
        await UsersService.set_user_state(message.from_user.id, "registered_as_admin")
        return
    telegram_id = int(data[0])
    name = data[1]
    surname = data[2]
    patronymic = data[3]
    subjects = data[4:]
    await UsersService.edit_teacher(telegram_id, name, surname, patronymic, subjects)
    await message.answer("Викладач успішно відредаговано",
                         reply_markup=admin_keyboards.back_to_admin_teachers_info_keyboard)
    await UsersService.set_user_state(message.from_user.id, "registered_as_admin")


@router.callback_query(F.data == "admin_add_schedule")
async def admin_add_schedule_input(callback: types.CallbackQuery):
    log("Ви вибрали додати розклад (admin_handler.py)")
    await callback.message.edit_text(
        "Надіслати файл з розкладом у форматі .xlsx з номером спеціальності, наприклад 100.xlsx",
        reply_markup = admin_keyboards.back_to_admin_specialities_info_keyboard)
    await UsersService.set_user_state(callback.from_user.id, "admin_add_schedule")
```

```
async def admin_add_schedule(message: types.Message):
    log("Ви вибрали додати розклад (admin_handler.py)")
    if not message.document:
        await message.answer(
            "Будь ласка, надішліть файл із розкладом у форматі .xlsx з номером спеціальності, наприклад 100.xlsx",
            reply_markup=admin_keyboards.back_to_admin_specialities_info_keyboard)
        await UsersService.set_user_state(message.from_user.id, "registered_as_admin")
        return
    file_id = message.document.file_id
    file_name = message.document.file_name
    speciality_code = file_name.split(".")[0]
    await UsersService.add_schedule(file_id, speciality_code)
    await message.answer("Розклад успішно доданий",
                         reply_markup=admin_keyboards.back_to_admin_specialities_info_keyboard)
    await UsersService.set_user_state(message.from_user.id, "registered_as_admin")
```

command_handler.py

```
from aiogram import Router, types, F

from Handlers import student_handler, teacher_handler, admin_handler, register_handler
from Keyboards import service_keyboards
from Keyboards import student_keyboards
from Keyboards import teacher_keyboards
from Keyboards import admin_keyboards
import Db.UsersService as UsersService
from logger import log
import Db.UsersService as UserService

router = Router()


async def get_markup(user_id):
    markup = None
    state = await UsersService.get_user_state(user_id)
    if state == "registered_as_student":
        markup = student_keyboards.student_menu_keyboard
```

```
    elif state == "registered_as_teacher":
        markup = teacher_keyboards.teacher_menu_keyboard
    elif state == "registered_as_admin" or state.startswith("admin_"):
        markup = admin_keyboards.admin_menu_keyboard
    return markup


# Команда /start для початку роботи з ботом
@router.message(F.text == "/start")
async def start(message: types.Message):
    log("Ви обрали почати роботу з ботом (command_handler.py)")
    await message.answer("Привіт, я бот для університету. Для початку роботи вам потрібно ввести /register")


# Команда /register для авторизації
@router.message(F.text == "/register")
async def start(message: types.Message):
    log("Ви обрали авторизуватися (command_handler.py)")
    # check if user is already registered
    isRegistered = await UsersService.is_registered(message.from_user.id)
    if isRegistered:
        await message.answer("Ви вже зареєстровані")
        return

    await message.answer("Виберіть роль", reply_markup=service_keyboards.roles_keyboard)


# Команда /menu для переходу до меню
@router.message(F.text == "/menu")
async def start(message: types.Message):
    log("Ви вибрали перейти в меню (command_handler.py)")
    isRegistered = await UsersService.is_registered(message.from_user.id)
    if not isRegistered:
        await message.answer("Ви не зареєстровані, будь ласка, введіть /register")
        return

    await message.answer("Меню", reply_markup=await get_markup(message.from_user.id))
```

```
@router.callback_query(F.data == "back_to_menu")
async def student_info(callback: types.CallbackQuery):
    log("Ви обрали перехід у меню (command_handler.py)")
    await callback.message.edit_text("Меню", reply_markup=await get_markup(callback.from_user.id))


@router.message(F.document)
async def route_documents(message: types.Message):
    if message.document.file_name.startswith("homework_"):
        await student_handler.upload_homework(message)
    else:
        await admin_handler.admin_add_schedule(message)


@router.message()
async def route_messages(message: types.Message):
    log("Ви написали повідомлення (command_handler.py)")
    state = await UserService.get_user_state(message.from_user.id)
    state_actions = {
        "waiting_for_register_as_student": register_handler.register_as_student,
        "waiting_for_register_as_teacher": register_handler.register_as_teacher,
        "waiting_for_register_as_admin": register_handler.register_as_admin,
        "admin_add_speciality": admin_handler.admin_add_speciality,
        "admin_delete_speciality": admin_handler.admin_delete_speciality,
        "admin_edit_speciality_input_data": admin_handler.admin_edit_speciality_input_data,
        "admin_add_subject": admin_handler.admin_add_subject,
        "admin_delete_subject": admin_handler.admin_delete_subject,
        "admin_edit_subject_input_data": admin_handler.admin_edit_subject_input_data,
        "admin_delete_student": admin_handler.admin_delete_student,
        "admin_edit_student_input_data": admin_handler.admin_edit_student_input_data,
        "admin_delete_teacher": admin_handler.admin_delete_teacher,
        "admin_add_schedule": admin_handler.admin_add_schedule,
        "admin_edit_teacher_input_data": admin_handler.admin_edit_teacher_input_data,
        "view_speciality_students": teacher_handler.view_speciality_students_by_code,
        "view_group_homework": teacher_handler.view_student_homework,
```

```
        "rate_homework": teacher_handler.rate_homework_by_data,
        "upload_homework": teacher_handler.upload_homework_by_data,
    }
    if state in state_actions:
        await state_actions[state](message)
    else:
        await message.answer("Невідома команда, спробуйте ще раз, або введіть /menu")
```

register_handler.py

```
from aiogram import Router, types, F
from Keyboards import service_keyboards
from Keyboards import student_keyboards
from Keyboards import teacher_keyboards
from Keyboards import admin_keyboards
import Db.UsersService as UserService
import settings

router = Router()


async def is_registered(state, callback):
    if state == "registered_as_student":
        await callback.answer("Ви вже зареєстровані як студент",
                              reply_markup=student_keyboards.student_menu_keyboard)
        return True
    elif state == "registered_as_teacher":
        await callback.answer("Ви вже зареєстровані як викладач",
                              reply_markup=teacher_keyboards.teacher_menu_keyboard)
        return True
    elif state == "registered_as_admin":
        await callback.answer("Ви вже зареєстровані як адміністратор",
                              reply_markup=admin_keyboards.admin_menu_keyboard)
        return True
    return False


@router.callback_query(F.data == "register_as_student")
async def register_student(callback: types.CallbackQuery):
```

```
    state = await UserService.get_user_state(callback.from_user.id)
    if await is_registered(state, callback):
        return

    await UserService.set_user_state(callback.from_user.id, "waiting_for_register_as_student")
    await callback.message.edit_text(
        "Введіть ваше ім'я, прізвище, по батькові, код спеціальності та поточний курс у форматі: Іванов Іван Іванович 125 2")


@router.callback_query(F.data == "register_as_teacher")
async def register_teacher(callback: types.CallbackQuery):
    state = await UserService.get_user_state(callback.from_user.id)
    if await is_registered(state, callback):
        return

    await UserService.set_user_state(callback.from_user.id, "waiting_for_register_as_teacher")
    await callback.message.edit_text(
        "Введіть ваше ім'я, прізвище, по-батькові у форматі: Іванов Іван Іванович")


@router.callback_query(F.data == "register_as_admin")
async def register_admin(callback: types.CallbackQuery):
    state = await UserService.get_user_state(callback.from_user.id)
    if await is_registered(state, callback):
        return

    await UserService.set_user_state(callback.from_user.id, "waiting_for_register_as_admin")
    await callback.message.edit_text(
        "Введіть ваше ім'я, прізвище, по батькові та пароль адміністратора у форматі: Іванов Іван Іванович Пароль")


async def register_as_student(message: types.Message):
    try:
        user_input = message.text.split()
        if len(user_input) != 5:
            await message.answer("Будь ласка, введіть дані у правильному форматі.")
```

```
            return

        first_name, second_name, last_name, speciality, course = user_input
        isSpecialityExists = await UserService.is_speciality_exists(speciality)
        if not isSpecialityExists:
            await message.answer("Будь ласка, вкажіть існуючу спеціальність.")
            return
        specialityTermsCount = await UserService.get_speciality_terms_count(speciality)
        mod = specialityTermsCount % 2
        specialityCourseCount = specialityTermsCount // 2
        if mod != 0:
            specialityCourseCount += 1
        if not course.isdigit():
            await message.answer("Будь ласка, вкажіть номер курсу у числовому форматі.")
            return
        if int(course) < 1 or int(course) > specialityCourseCount:
            await message.answer(
                "Будь ласка, вкажіть номер курсу в діапазоні від 1 до " + str(specialityCourseCount))
            return
        await UserService.register_student(message.from_user.id, first_name, second_name, last_name, speciality,
                                           int(course))
        await UserService.set_user_state(message.from_user.id, "registered_as_student")

        await message.answer("Дякую! Ви успішно зареєстровані як студент.",
                             reply_markup=student_keyboards.student_menu_keyboard)

    except Exception as e:
        print(e)
        await message.answer("Сталася помилка при збереженні даних. Будь ласка, спробуйте знову.")


async def register_as_teacher(message: types.Message):
    try:
        user_input = message.text.split()
        if len(user_input) != 3:
            await message.answer("Будь ласка, введіть дані у правильному форматі.")
            return
```

```
        first_name, second_name, last_name = user_input
        await UserService.register_teacher(message.from_user.id, first_name, second_name, last_name)
        await UserService.set_user_state(message.from_user.id, "registered_as_teacher")
        await message.answer("Дякую! Ви успішно зареєстровані як викладач.",
                             reply_markup=teacher_keyboards.teacher_menu_keyboard)

    except Exception as e:
        print(e)
        await message.answer("Сталася помилка при збереженні даних. Будь ласка, спробуйте знову.")


async def register_as_admin(message: types.Message):
    try:
        user_input = message.text.split()
        if len(user_input) != 4:
            await message.answer("Будь ласка, введіть дані у правильному форматі.")
            return

        first_name, second_name, last_name, password = user_input
        if password != settings.AdministratorPassword:
            await UserService.set_user_state(message.from_user.id, "waiting_for_register")
            await message.answer("Ви ввели неправильний пароль. Будь ласка, виберіть роль",
                                 reply_markup=service_keyboards.roles_keyboard)
            return
        await UserService.register_admin(message.from_user.id, first_name, second_name, last_name)
        await UserService.set_user_state(message.from_user.id, "registered_as_admin")
        await message.answer("Дякую! Ви успішно зареєстровані як адміністратор.",
                             reply_markup=admin_keyboards.admin_menu_keyboard)

    except Exception as e:
        print(e)
        await message.answer("Сталася помилка при збереженні даних. Будь ласка, спробуйте знову.")
```

student_handler.py

```
from aiogram import Router, types, F

import Db.UsersService as UsersService
```

```
from Keyboards import student_keyboards
from logger import log

router = Router()


@router.callback_query(F.data == "view_self_student_info")
async def student_info(callback: types.CallbackQuery):
    log("Ви обрали перегляд інформації про себе (student_handler.py)")
    await callback.message.edit_text("Яку інформацію ви хочете подивитися?",
                                     reply_markup=student_keyboards.student_self_info_keyboard)


@router.callback_query(F.data == "view_group_info")
async def group_info(callback: types.CallbackQuery):
    log("Ви обрали перегляд інформації про групу (student_handler.py)")
    await callback.message.edit_text("Яку інформацію ви хочете подивитися?",
                                     reply_markup=student_keyboards.group_info_keyboard)


@router.callback_query(F.data == "view_speciality_info")
async def speciality_info(callback: types.CallbackQuery):
    log("Ви обрали перегляд інформації про спеціальність (student_handler.py)")
    await callback.message.edit_text("Яку інформацію ви хочете подивитися?",
                                     reply_markup=student_keyboards.speciality_info_keyboard)


@router.callback_query(F.data == "view_college_info")
async def college_info(callback: types.CallbackQuery):
    log("Ви обрали перегляд інформації про університет (student_handler.py)")
    await callback.message.edit_text("Яку інформацію ви хочете подивитися?",
                                     reply_markup=student_keyboards.college_info_keyboard)


@router.callback_query(F.data == "view_self_student_marks")
async def view_self_student_marks(callback: types.CallbackQuery):
    log("Ви обрали перегляд оцінок (student_handler.py)")
```

```
    subjects = await UsersService.get_student_subjects(callback.from_user.id)
    await callback.message.edit_text("Виберіть предмет",
                                     reply_markup=student_keyboards.build_subjects_keyboard(subjects, "student_marks",
                                                                                            "back_to_student_grades_menu"))


@router.callback_query(F.data == "back_to_student_grades_menu")
async def view_self_student_marks(callback: types.CallbackQuery):
    log("Ви обрали повернення в меню оцінок (student_handler.py)")
    await callback.message.edit_text("Яку інформацію ви хочете подивитися?",
                                     reply_markup=student_keyboards.student_self_info_keyboard)


@router.callback_query(F.data.startswith("view_self_student_marks_"))
async def view_self_student_grade(callback: types.CallbackQuery):
    log("Ви обрали перегляд оцінки (student_handler.py)")
    subject_name = callback.data.split("_")[4]
    grades_with_dates = await UsersService.get_student_grades(callback.from_user.id, subject_name)
    if len(grades_with_dates) == 0:
        await callback.message.edit_text("У вас немає оцінок з цього предмету",
                                         reply_markup=student_keyboards.view_self_student_marks)
    else:
        text = "Оцінки з предмета " + subject_name + ":\n"
        for grade_with_date in grades_with_dates:
            text += grade_with_date[1] + " - " + str(grade_with_date[0])
            if grade_with_date[2]:
                text += "(підсумкова)"
            text += "\n"
        await callback.message.edit_text(text, reply_markup=student_keyboards.view_self_student_marks)


@router.callback_query(F.data == "view_self_student_homeworks")
async def view_self_student_homeworks(callback: types.CallbackQuery):
    log("Ви обрали перегляд домашніх завдань (student_handler.py)")
    await UsersService.set_user_waiting_homework_state(callback.from_user.id, "None")
    subjects = await UsersService.get_student_subjects(callback.from_user.id)
    await callback.message.edit_text("Виберіть предмет",
```

```
                reply_markup=student_keyboards.build_subjects_keyboard(subjects,
                                                                       "student_homework",
                                                                       "back_to_student_homework_menu"))


@router.callback_query(F.data == "back_to_student_homework_menu")
async def view_self_student_homeworks(callback: types.CallbackQuery):
    log("Ви вибрали повернення в меню домашніх завдань (student_handler.py)")
    await callback.message.edit_text("Яку інформацію ви хочете подивитися?",
                                     reply_markup=student_keyboards.student_self_info_keyboard)


@router.callback_query(F.data.startswith("view_self_student_homework_"))
async def view_self_student_homework(callback: types.CallbackQuery):
    log("Ви обрали перегляд домашнього завдання (student_handler.py)")
    subject_name = callback.data.split("_")[4]
    homeworks = await UsersService.get_student_homeworks(callback.from_user.id, subject_name)
    if len(homeworks) == 0:
        await callback.message.edit_text("У вас немає домашніх завдань з цього предмету",
                                         reply_markup=student_keyboards.view_self_student_homework)
    else:
        not_uploaded = 0
        text = "Домашні завдання з предмета " + subject_name + ":\n"
        for homework in homeworks:
            if homework[2] is False:
                not_uploaded += 1
            text += homework[1] + " - " + homework[0] + (" (здано)" if homework[2] is True else "") + "\n"
        if not_uploaded > 0:
            keyboard = student_keyboards.build_student_homework_keyboard(subject_name)
            await callback.message.edit_text(text, reply_markup=keyboard)
        else:
            await callback.message.edit_text(text, reply_markup=student_keyboards.view_self_student_homework)


@router.callback_query(F.data.startswith("upload_self_student_homework_"))
async def upload_self_student_homework(callback: types.CallbackQuery):
    log("Ви обрали завантаження домашнього завдання (student_handler.py)")
```

```
    subject_name = callback.data.split("_")[4]
    await UsersService.set_user_waiting_homework_state(callback.from_user.id, subject_name)
    await callback.message.edit_text("Надішліть файл з домашнім завданням з ім'ям homework_dd.mm",
                                     reply_markup=student_keyboards.view_self_student_homework)


async def upload_homework(message: types.Message):
    log("Ви завантажили домашнє завдання (student_handler.py)")
    student_id = message.from_user.id
    subject_name = await UsersService.get_user_waiting_homework_state(student_id)
    if subject_name == "None":
        await message.answer("Ви не вибрали предмет або у вас немає домашнього завдання на цей предмет",
                             reply_markup=student_keyboards.view_self_student_homework)
        return
    subject_id = await UsersService.get_subject_id_by_name(subject_name)
    file_id = message.document.file_id
    date = message.document.file_name.split("_")[1]
    await UsersService.upload_homework(message.from_user.id, subject_id, file_id, date)
    await UsersService.set_user_waiting_homework_state(message.from_user.id, "None")
    await message.answer("Домашнє завдання успішно завантажено",
                         reply_markup=student_keyboards.view_self_student_homework)


@router.callback_query(F.data == "view_group_students")
async def view_group_students(callback: types.CallbackQuery):
    log("Ви обрали перегляд списку студентів (student_handler.py)")
    student_list = await UsersService.get_group_students(callback.from_user.id)
    if len(student_list) == 0:
        await callback.message.edit_text("У вашій групі немає студентів",
                                         reply_markup=student_keyboards.view_group_info)
    else:
        text = "Список студентів:\n"
        for student in student_list:
            text += student[0] + " " + student[1] + " " + student[2] + "\n"
        await callback.message.edit_text(text, reply_markup=student_keyboards.view_group_info)
```

```
@router.callback_query(F.data == "view_group_rating")
async def view_group_rating(callback: types.CallbackQuery):
    log("Ви обрали перегляд рейтингу (student_handler.py)")
    rating = await UsersService.get_group_rating(callback.from_user.id)
    if len(rating) == 0:
        await callback.message.edit_text("У вашій групі немає студентів",
                                         reply_markup=student_keyboards.view_group_info)
    else:
        text = "Рейтинг (за середнім балом):\n"
        for student in rating:
            text += student[0] + " " + student[1] + " " + student[2] + " - " + str(student[3]) + "\n"
        await callback.message.edit_text(text, reply_markup=student_keyboards.view_group_info)


@router.callback_query(F.data == "view_speciality_teachers")
async def view_speciality_teachers(callback: types.CallbackQuery):
    log("Ви обрали перегляд викладачів (student_handler.py)")
    speciality_code = await UsersService.get_student_speciality_code(callback.from_user.id)
    teachers_with_subjects = await UsersService.get_teachers_by_speciality(
        speciality_code)  # name, secondName, lastName, list(subjects)
    if len(teachers_with_subjects) == 0:
        await callback.message.edit_text("У вашій спеціальності немає викладачів",
                                         reply_markup=student_keyboards.view_speciality_info)
    else:
        text = "Список викладачів:\n"
        for teacher in teachers_with_subjects:
            subject_list = ""
            for subject in teacher[3]:
                subject_list += subject + ","
            text += teacher[0] + " " + teacher[1] + " " + teacher[2] + " - " + subject_list[:-2] + "\n"
        await callback.message.edit_text(text, reply_markup=student_keyboards.view_speciality_info)


@router.callback_query(F.data == "view_speciality_subjects")
async def view_speciality_subjects(callback: types.CallbackQuery):
    log("Ви обрали перегляд предметів (student_handler.py)")
    speciality_code = await UsersService.get_student_speciality_code(callback.from_user.id)
```

```
    subjects = await UsersService.get_subjects_by_speciality(speciality_code)
    if len(subjects) == 0:
        await callback.message.edit_text("У вашій спеціальності немає предметів",
                                         reply_markup=student_keyboards.view_speciality_info)
    else:
        text = "Список предметів:\n"
        for subject in subjects:
            teacher_subject = await UsersService.get_teacher_by_subject(subject)
            text += subject + " - " + teacher_subject[0] + " " + teacher_subject[1] + " " + teacher_subject[2] + "\n"
        await callback.message.edit_text(text, reply_markup=student_keyboards.view_speciality_info)


@router.callback_query(F.data == "view_speciality_schedule")
async def view_speciality_schedule(callback: types.CallbackQuery):
    log("Ви обрали перегляд розкладу (student_handler.py)")
    speciality_id = await UsersService.get_student_speciality_id(callback.from_user.id)
    student_course = await UsersService.get_student_course(callback.from_user.id)
    image_telegram_id = await UsersService.get_schedule_image(speciality_id)

    if image_telegram_id is None:
        await callback.message.edit_text("У вашій спеціальності немає розкладу",
                                         reply_markup=student_keyboards.view_speciality_info)
    else:
        await callback.message.answer_document(document=image_telegram_id)
        await callback.message.delete()
        text = "Розклад для + str(student_course) + курсу вашої спеціальності"
        await callback.message.answer(text, reply_markup=student_keyboards.view_speciality_info)


@router.callback_query(F.data == "view_college_specialities")
async def view_college_specialities(callback: types.CallbackQuery):
    log("Ви обрали перегляд спеціальностей (student_handler.py)")
    specialities = await UsersService.get_specialities()
    if len(specialities) == 0:
        await callback.message.edit_text("У вашому університеті немає спеціальностей",
                                         reply_markup=student_keyboards.view_college_info)
    else:
```

```
    text = "Список спеціальностей:\n"
    for speciality in specialities:
        text += speciality[1] + " " + speciality[2] + " - " + str(speciality[3]) + " семестрів\n"
    await callback.message.edit_text(text, reply_markup=student_keyboards.view_college_info)


@router.callback_query(F.data == "view_college_teachers")
async def view_college_teachers(callback: types.CallbackQuery):
    log("Ви обрали перегляд викладачів (student_handler.py)")
    teachers_with_subjects = await UsersService.get_teachers_with_subjects()  # name, secondName, lastName,
list(subjects)
    if len(teachers_with_subjects) == 0:
        await callback.message.edit_text("У вашому університеті немає викладачів",
                                         reply_markup=student_keyboards.view_college_info)
    else:
        text = "Список викладачів:\n"
        for teacher in teachers_with_subjects:
            subject_list = ""
            for subject in teacher[3]:
                subject_list += subject + ","
            text += teacher[0] + " " + teacher[1] + " " + teacher[2] + " - " + subject_list[:-2] + "\n"
        await callback.message.edit_text(text, reply_markup=student_keyboards.view_college_info)
```

teacher_handler.py

```
from aiogram import Router, types, F

import Db.UsersService as UsersService
from Keyboards import teacher_keyboards
from logger import log

router = Router()


@router.callback_query(F.data == "view_self_teacher_info")
async def teacher_info(callback: types.CallbackQuery):
    log("Ви обрали перегляд інформації про себе (teacher_handler.py)")
    await callback.message.edit_text("Яку інформацію ви хочете подивитися?",
                                     reply_markup=teacher_keyboards.teacher_self_info_keyboard)
```

```
@router.callback_query(F.data == "view_specialities_info")
async def group_info(callback: types.CallbackQuery):
    log("Ви обрали перегляд інформації про групи (teacher_handler.py)")
    await callback.message.edit_text("Яку інформацію ви хочете подивитися?",
                                     reply_markup=teacher_keyboards.speciality_info_keyboard)


@router.callback_query(F.data == "view_homework_info")
async def group_info(callback: types.CallbackQuery):
    log("Ви обрали перегляд інформації про домашні завдання (teacher_handler.py)")
    await callback.message.edit_text("Яку інформацію ви хочете подивитися?",
                                     reply_markup=teacher_keyboards.homework_info_keyboard)


@router.callback_query(F.data == "view_college_info")
async def group_info(callback: types.CallbackQuery):
    log("Ви обрали перегляд інформації про університет (teacher_handler.py)")
    await callback.message.edit_text("Яку інформацію ви хочете подивитися?",
                                     reply_markup=teacher_keyboards.college_info_keyboard)


@router.callback_query(F.data == "view_self_teacher_subjects")
async def view_self_teacher_subjects(callback: types.CallbackQuery):
    log("Ви обрали перегляд своїх предметів (teacher_handler.py)")
    subjects = await UsersService.get_teacher_subjects(callback.from_user.id)
    if len(subjects) == 0:
        await callback.message.edit_text("У вас немає предметів",
                                         reply_markup=teacher_keyboards.teacher_back_self_info_keyboard)
        return
    text = "Ваші предмети:\n"
    for subject in subjects:
        text += f"{subject}\n"
    await callback.message.edit_text(text, reply_markup=teacher_keyboards.teacher_back_self_info_keyboard)


@router.callback_query(F.data == "back_to_self_teacher_info")
```

```
async def back_to_self_teacher_info(callback: types.CallbackQuery):
    log("Ви обрали повернутися в меню перегляду інформації про себе (teacher_handler.py)")
    await callback.message.edit_text("Яку інформацію ви хочете подивитися?",
                                     reply_markup=teacher_keyboards.teacher_self_info_keyboard)


@router.callback_query(F.data == "view_speciality_students")
async def view_speciality_students(callback: types.CallbackQuery):
    log("Ви обрали перегляд студентів спеціальності (teacher_handler.py)")
    specialities = await UsersService.get_all_specialities()  # [<Record id=1 name='Спеціальність' code='100'
terms=8>]
    specialities = [[speciality[0], speciality[1], speciality[2], speciality[3]] for speciality in specialities]
    if len(specialities) == 0:
        await callback.message.answer("В університеті немає спеціальностей",
                                      reply_markup=teacher_keyboards.speciality_back_info_keyboard)
        return
    text = "Спеціальності університету:\n"
    for speciality in specialities:
        text += f"{speciality[1]} - {speciality[2]}\n"
    specialities
    await callback.message.edit_text(text + "\nВведіть код спеціальності")
    await UsersService.set_user_state(callback.from_user.id, "view_speciality_students")


@router.callback_query(F.data == "back_to_specialities_info")
async def back_to_specialities_info(callback: types.CallbackQuery):
    log("Ви обрали повернутися в меню перегляду інформації про спеціальності (teacher_handler.py)")
    await callback.message.edit_text("Яку інформацію ви хочете подивитися?",
                                     reply_markup=teacher_keyboards.speciality_info_keyboard)


async def view_speciality_students_by_code(message: types.Message):
    log("Ви ввели код спеціальності (teacher_handler.py)")
    speciality_code = message.text
    students = await UsersService.get_students_by_speciality(speciality_code)
    if len(students) == 0:
        await message.answer("У цій спеціальності немає студентів",
```

```
                reply_markup=teacher_keyboards.speciality_back_info_keyboard)
        await UsersService.set_user_state(message.from_user.id, "registered_as_teacher")
        return
    text = "Студенти спеціальності:\n"
    for student in students:
        text += f"ID: {student[3]} - {student[0]} {student[1]} {student[2]}\n"
    await message.answer(text, reply_markup=teacher_keyboards.speciality_back_info_keyboard)
    await UsersService.set_user_state(message.from_user.id, "registered_as_teacher")


@router.callback_query(F.data == "view_student_homework_by_subject")
async def view_student_homework_by_subject(callback: types.CallbackQuery):
    log("Ви обрали перегляд домашніх завдань (teacher_handler.py)")
    students = await UsersService.get_all_students()
    students = [[student[0], student[1], student[2], student[3]] for student in students]
    if len(students) == 0:
        await callback.message.answer("В університеті немає студентів",
                                      reply_markup=teacher_keyboards.homework_info_keyboard)
        return
    text = "Студенти університету:\n"
    for student in students:
        text += f"{student[1]} {student[2]} {student[3]} - {student[0]}\n"
    await callback.message.edit_text(
        text + "Введіть ID студента, назву предмета та дату, наприклад 16543 Математика 01.12")
    await UsersService.set_user_state(callback.from_user.id, "view_group_homework")


async def view_student_homework(message: types.Message):
    log("Ви ввели ID студента та назву предмета (teacher_handler.py)")
    data = message.text.split()
    if len(data) != 3:
        await message.answer("Неправильний формат введення",
reply_markup=teacher_keyboards.homework_info_keyboard)
        await UsersService.set_user_state(message.from_user.id, "registered_as_teacher")
        return
    student_id = int(data[0])
    subject_name = data[1]
```

```
    date = data[2]
    subject_id = await UsersService.get_subject_id_by_name(subject_name)
    homework_file_id = await UsersService.get_student_homework_file_id(student_id, subject_id, date)
    if homework_file_id is None:
        await message.answer("У студента немає домашнього завдання з цього предмета на цю дату",
                             reply_markup=teacher_keyboards.homework_info_keyboard)
        await UsersService.set_user_state(message.from_user.id, "registered_as_teacher")
        return
    await message.answer_document(document=homework_file_id,
reply_markup=teacher_keyboards.homework_back_info_keyboard)
    await UsersService.set_user_state(message.from_user.id, "registered_as_teacher")


@router.callback_query(F.data == "back_to_homework_info")
async def back_to_homework_info(callback: types.CallbackQuery):
    log("Ви вибрали повернутися в меню перегляду інформації про домашні завдання (teacher_handler.py)")
    await callback.message.edit_text("Яку інформацію ви хочете подивитися?",
                                     reply_markup=teacher_keyboards.homework_info_keyboard)


@router.callback_query(F.data == "rate_homework")
async def rate_homework(callback: types.CallbackQuery):
    log("Ви обрали оцінити домашнє завдання (teacher_handler.py)")
    students = await UsersService.get_all_students()
    students = [[student[0], student[1], student[2], student[3]] for student in students]
    if len(students) == 0:
        await callback.message.answer("В університеті немає студентів",
                                      reply_markup=teacher_keyboards.homework_back_info_keyboard)
        return
    text = "Студенти університету:\n"
    for student in students:
        text += f"{student[1]} {student[2]} {student[3]} - {student[0]}\n"
    await callback.message.answer(text +
                                  "Введіть ID студента, назву предмета, дату та оцінку (якщо іспит, то напишіть exam в
самому кінці), наприклад 134575 Математика 01.12 100 exam")
    await UsersService.set_user_state(callback.from_user.id, "rate_homework")
```

```
async def rate_homework_by_data(message: types.Message):
    log("Ви ввели ID студента та назву предмета та дату та оцінку (teacher_handler.py)")
    data = message.text.split()
    if len(data) < 4:
        await message.answer("Неправильний формат введення",
reply_markup=teacher_keyboards.homework_back_info_keyboard)
        await UsersService.set_user_state(message.from_user.id, "registered_as_teacher")
        return
    student_id = int(data[0])
    subject_name = data[1]
    subject_id = await UsersService.get_subject_id_by_name(subject_name)
    date = data[2]
    grade = int(data[3])
    if grade < 0 or grade > 100:
        await message.answer("Оцінка має бути від 0 до 100",
                             reply_markup=teacher_keyboards.homework_back_info_keyboard)
        await UsersService.set_user_state(message.from_user.id, "registered_as_teacher")
        return
    if len(data) == 5 and data[4] == "exam":
        await UsersService.rate_student_homework(student_id, subject_id, date, grade, True)
    else:
        await UsersService.rate_student_homework(student_id, subject_id, date, grade, False)
    await message.answer("Оцінка виставлена", reply_markup=teacher_keyboards.homework_back_info_keyboard)
    await UsersService.set_user_state(message.from_user.id, "registered_as_teacher")


@router.callback_query(F.data == "upload_homework")
async def upload_homework(callback: types.CallbackQuery):
    log("Ви обрали завантажити домашнє завдання (teacher_handler.py)")
    await callback.message.edit_text(
        "Надіслати текст домашнього завдання, на самому початку вкажіть:\n"
        "Предмет: назва предмета\n"
        "Спеціальність: код спеціальності\n"
        "Курс: номер курсу\n"
        "Дата: дата у форматі дд.мм, наприклад 01.12")
    await UsersService.set_user_state(callback.from_user.id, "upload_homework")
```

```
async def upload_homework_by_data(message: types.Message):
    log("Ви надіслали текст із домашнім завданням (teacher_handler.py)")
    text = message.text
    data = text.split("\n")
    if len(data) < 4:
        await message.answer("Неправильний формат введення",
reply_markup=teacher_keyboards.homework_back_info_keyboard)
        await UsersService.set_user_state(message.from_user.id, "registered_as_teacher")
        return
    subject_name = data[0].split(":")[1].strip()
    subject_id = await UsersService.get_subject_id_by_name(subject_name)
    speciality_code = data[1].split(":")[1].strip()
    course = int(data[2].split(":")[1].strip())
    date = data[3].split(":")[1].strip()
    homework_text = "\n".join(data[4:])
    if len(homework_text) == 0:
        await message.answer("Немає тексту домашнього завдання",
                             reply_markup=teacher_keyboards.homework_back_info_keyboard)
        await UsersService.set_user_state(message.from_user.id, "registered_as_teacher")
        return
    if not await UsersService.is_speciality_exists(speciality_code):
        await message.answer("Такої спеціальності немає",
reply_markup=teacher_keyboards.homework_back_info_keyboard)
        await UsersService.set_user_state(message.from_user.id, "registered_as_teacher")
        return
    if not await UsersService.is_subject_exists(subject_id):
        await message.answer("Такого предмета немає",
reply_markup=teacher_keyboards.homework_back_info_keyboard)
        await UsersService.set_user_state(message.from_user.id, "registered_as_teacher")
        return
    await UsersService.upload_homework_as_teacher(subject_id, speciality_code, course, date, homework_text)
    await message.answer("Домашнє завдання завантажено",
reply_markup=teacher_keyboards.homework_back_info_keyboard)
    await UsersService.set_user_state(message.from_user.id, "registered_as_teacher")


@router.callback_query(F.data == "teacher_view_college_specialities")
async def view_college_specialities(callback: types.CallbackQuery):
```

```
    log("Ви обрали перегляд спеціальностей (teacher_handler.py)")
    specialities = await UsersService.get_all_specialities()
    if len(specialities) == 0:
        await callback.message.answer("В університеті немає спеціальностей",
                                      reply_markup=teacher_keyboards.college_info_keyboard)
        return
    text = "Спеціальності університету:\n"
    for speciality in specialities:
        text += f"{speciality[0]} {speciality[1]}\n"
    await callback.message.answer(text, reply_markup=teacher_keyboards.college_info_keyboard)


@router.callback_query(F.data == "teacher_view_college_teachers")
async def view_college_teachers(callback: types.CallbackQuery):
    log("Ви обрали перегляд викладачів (teacher_handler.py)")
    teachers = await UsersService.get_teachers()
    if len(teachers) == 0:
        await callback.message.answer("В університеті немає викладачів",
                                      reply_markup=teacher_keyboards.college_info_keyboard)
        return
    text = "Викладачі університету:\n"
    for teacher in teachers:
        text += f"{teacher[0]} {teacher[1]} {teacher[2]}\n"
    await callback.message.answer(text, reply_markup=teacher_keyboards.college_info_keyboard)
```

admin_keyboards.py

```
from aiogram.types import InlineKeyboardMarkup, InlineKeyboardButton

admin_menu_keyboard = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="Інформація про користувачів", callback_data="admin_view_users_info")],
    [InlineKeyboardButton(text="Інформація про спеціальності", callback_data="admin_view_specialities_info")],
    [InlineKeyboardButton(text="Інформація про предмети", callback_data="admin_view_subjects_info")],
    [InlineKeyboardButton(text="Інформація про студентів", callback_data="admin_view_students_info")],
    [InlineKeyboardButton(text="Інформація про викладачів", callback_data="admin_view_teachers_info")],
])

admin_users_info_keyboard = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="Показати всіх користувачів", callback_data="admin_view_all_users")],
```

```
    [InlineKeyboardButton(text="Показати студентів", callback_data="admin_view_students")],
    [InlineKeyboardButton(text="Показати викладачів", callback_data="admin_view_teachers")],
    [InlineKeyboardButton(text="Показати адміністраторів", callback_data="admin_view_admins")],
    [InlineKeyboardButton(text="Назад", callback_data="back_to_menu")]
])

back_to_admin_users_info_keyboard = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="Назад", callback_data="admin_view_users_info")]
])

admin_specialities_info_keyboard = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="Показати всі спеціальності", callback_data="admin_view_all_specialities")],
    [InlineKeyboardButton(text="Додати спеціальність", callback_data="admin_add_speciality")],
    [InlineKeyboardButton(text="Видалити спеціальність", callback_data="admin_delete_speciality")],
    [InlineKeyboardButton(text="Редагувати спеціальність", callback_data="admin_edit_speciality")],
    [InlineKeyboardButton(text="Додати розклад для спеціальності", callback_data="admin_add_schedule")],
    [InlineKeyboardButton(text="Назад", callback_data="back_to_menu")]
])

back_to_admin_specialities_info_keyboard = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="Назад", callback_data="admin_view_specialities_info")]
])

admin_subjects_info_keyboard = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="Показати всі предмети", callback_data="admin_view_all_subjects")],
    [InlineKeyboardButton(text="Додати предмет", callback_data="admin_add_subject")],
    [InlineKeyboardButton(text="Видалити предмет", callback_data="admin_delete_subject")],
    [InlineKeyboardButton(text="Редагувати предмет", callback_data="admin_edit_subject")],
    [InlineKeyboardButton(text="Назад", callback_data="back_to_menu")]
])

back_to_admin_subjects_info_keyboard = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="Назад", callback_data="admin_view_subjects_info")]
])

admin_students_info_keyboard = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="Показати всіх студентів", callback_data="admin_view_all_students")],
```

```
    [InlineKeyboardButton(text="Видалити студента", callback_data="admin_delete_student")],
    [InlineKeyboardButton(text="Редагувати студента", callback_data="admin_edit_student")],
    [InlineKeyboardButton(text="Назад", callback_data="back_to_menu")]
])

back_to_admin_students_info_keyboard = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="Назад", callback_data="admin_view_students_info")]
])

admin_teachers_info_keyboard = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="Показати всіх викладачів", callback_data="admin_view_all_teachers")],
    [InlineKeyboardButton(text="Видалити викладача", callback_data="admin_delete_teacher")],
    [InlineKeyboardButton(text="Редагувати викладача", callback_data="admin_edit_teacher")],
    [InlineKeyboardButton(text="Назад", callback_data="back_to_menu")]
])

back_to_admin_teachers_info_keyboard = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="Назад", callback_data="admin_view_teachers_info")]
])
```

Service_keyboards.py

```
from aiogram.types import InlineKeyboardMarkup, InlineKeyboardButton

roles_keyboard = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="Студент", callback_data="register_as_student")],
    [InlineKeyboardButton(text="Викладач", callback_data="register_as_teacher")],
    [InlineKeyboardButton(text="Адміністратор", callback_data="register_as_admin")]
])
```

Student_keyboards.py

```
from aiogram.types import InlineKeyboardMarkup, InlineKeyboardButton

student_menu_keyboard = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="Подивитися інформацію про себе", callback_data="view_self_student_info")],
    [InlineKeyboardButton(text="Подивитися інформацію про групу", callback_data="view_group_info")],
    [InlineKeyboardButton(text="Подивитися інформацію про спеціальність",
callback_data="view_speciality_info")],
    [InlineKeyboardButton(text="Подивитися інформацію про університет", callback_data="view_college_info")]
```

```
])

student_self_info_keyboard = InlineKeyboardMarkup(inline_keyboard=[
  [InlineKeyboardButton(text="Оцінки", callback_data="view_self_student_marks")],
  [InlineKeyboardButton(text="Домашні завдання", callback_data="view_self_student_homeworks")],
  [InlineKeyboardButton(text="Назад", callback_data="back_to_menu")]
])

view_self_student_marks = InlineKeyboardMarkup(inline_keyboard=[
  [InlineKeyboardButton(text="Назад", callback_data="view_self_student_marks")]
])

view_self_student_homework = InlineKeyboardMarkup(inline_keyboard=[
  [InlineKeyboardButton(text="Назад", callback_data="view_self_student_homeworks")]
])


def build_student_homework_keyboard(subject_name):
  return InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="Завантажити", callback_data="upload_self_student_homework_" +
subject_name)],
    [InlineKeyboardButton(text="Назад", callback_data="view_self_student_homeworks")]
  ])


group_info_keyboard = InlineKeyboardMarkup(inline_keyboard=[
  [InlineKeyboardButton(text="Список студентів", callback_data="view_group_students")],
  [InlineKeyboardButton(text="Рейтинг", callback_data="view_group_rating")],
  [InlineKeyboardButton(text="Назад", callback_data="back_to_menu")]
])

view_group_info = InlineKeyboardMarkup(inline_keyboard=[
  [InlineKeyboardButton(text="Назад", callback_data="view_group_info")]
])

speciality_info_keyboard = InlineKeyboardMarkup(inline_keyboard=[
  [InlineKeyboardButton(text="Розклад", callback_data="view_speciality_schedule")],
```

```
    [InlineKeyboardButton(text="Викладачі", callback_data="view_speciality_teachers")],
    [InlineKeyboardButton(text="Предмети", callback_data="view_speciality_subjects")],
    [InlineKeyboardButton(text="Назад", callback_data="back_to_menu")]
])

view_speciality_info = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="Назад", callback_data="view_speciality_info")]
])

college_info_keyboard = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="Спеціальності", callback_data="view_college_specialities")],
    [InlineKeyboardButton(text="Викладачі", callback_data="view_college_teachers")],
    [InlineKeyboardButton(text="Назад", callback_data="back_to_menu")]
])

view_college_info = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="Назад", callback_data="view_college_info")]
])


def build_subjects_keyboard(subjects, target, callback_data):
    keyboard = []
    for subject in subjects:
        keyboard.append([InlineKeyboardButton(text=subject, callback_data="view_self_" + target + "_" + subject)])
    keyboard.append([InlineKeyboardButton(text="Назад", callback_data=callback_data)])
    return InlineKeyboardMarkup(inline_keyboard=keyboard)
```

teacher_keyboards.py

```
from aiogram.types import InlineKeyboardMarkup, InlineKeyboardButton

teacher_menu_keyboard = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="Подивитися інформацію про себе", callback_data="view_self_teacher_info")],
    [InlineKeyboardButton(text="Подивитися інформацію про спеціальності",
callback_data="view_specialities_info")],
    [InlineKeyboardButton(text="Подивитися інформацію про домашні завдання",
callback_data="view_homework_info")],
    [InlineKeyboardButton(text="Подивитися інформацію про університет", callback_data="view_college_info")]
```

```
])

teacher_self_info_keyboard = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="Мої предмети", callback_data="view_self_teacher_subjects")],
    [InlineKeyboardButton(text="Назад", callback_data="back_to_menu")]
])

teacher_back_self_info_keyboard = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="Назад", callback_data="back_to_self_teacher_info")]
])

speciality_info_keyboard = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="Переглянути студентів спеціальності",
callback_data="view_speciality_students")],
    [InlineKeyboardButton(text="Назад", callback_data="back_to_menu")]
])

speciality_back_info_keyboard = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="Назад", callback_data="back_to_specialities_info")]
])

homework_info_keyboard = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="Переглянути домашні завдання студента з предмета",
                callback_data="view_student_homework_by_subject")],
    [InlineKeyboardButton(text="Оцінити домашнє завдання", callback_data="rate_homework")],
    [InlineKeyboardButton(text="Завантажити домашнє завдання для предмета",
callback_data="upload_homework")],
    [InlineKeyboardButton(text="Назад", callback_data="back_to_menu")]
])

homework_back_info_keyboard = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="Назад", callback_data="back_to_homework_info")]
])

college_info_keyboard = InlineKeyboardMarkup(inline_keyboard=[
    [InlineKeyboardButton(text="Спеціальності", callback_data="teacher_view_college_specialities")],
    [InlineKeyboardButton(text="Викладачі", callback_data="teacher_view_college_teachers")],
    [InlineKeyboardButton(text="Назад", callback_data="back_to_menu")]
```

```
])
```

Logger.py

```
from datetime import datetime


def log(text):
    current_time = datetime.now().strftime("%d/%m/%Y %H:%M:%S")
    print("||" + current_time + "|| - " + text)
```

Main.py

```
import asyncio

from aiogram import Bot, Dispatcher

from Handlers.admin_handler import router as ar
from Handlers.command_handler import router as cr
from Handlers.register_handler import router as rr
from Handlers.student_handler import router as sr
from Handlers.teacher_handler import router as tr
from logger import log
from settings import TOKEN


async def main():
    log("Starting bot...")
    bot = Bot(token=TOKEN)
    dp = Dispatcher()
    dp.include_routers(cr, rr, sr, tr, ar)
    await dp.start_polling(bot)


if __name__ == '__main__':
    try:
        asyncio.run(main())
    except KeyboardInterrupt:
        log("Shutting down...")
```

Settings.py

```
TOKEN = "6606675146:AAEeSQXGUEWOliDPPbG8YmGMq_xopWh-36U"
AdministratorPassword = "0000"

host = "localhost"  # server address
port = "5432"  # server port
database = "StudenBot"  # database name
user = "postgres"  # name of the user
password = "password"
```