Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Телеграм-бот для автоматизації інформаційного обслуговування студентів

(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ-21-2
Владислав ШЕВЕЛЄВ
(власне ім’я, прізвище)

Спеціальність
123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп’ютерна інженерія
(повна назва освітньої програми)

Керівник: ас. Антон ГАВРАШЕНКО
(посада, власне ім’я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім’я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп’ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Шевелєву Владиславу Романовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Телеграм-бот для автоматизації інформаційного обслуговування студентів

затверджена наказом по університету від “ 26 ” травня 2025 р. № 424 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 17 червня 2025 р.

3. Вхідні дані до роботи 1) мова програмування Java; 2) Фреймворк Spring; Boot 3) Telegram API; 4) джерело даних:БД PostgreSQL, HTML, JSON.

4. Перелік питань, що потрібно опрацювати у роботі
1) аналіз проблеми та огляд існуючих рішень;
2) вибір технології розробки та інструментальних засобів;
3) розробка алгоритмічного забезпечення;
4) розробка програмних модулів;
5) відлагодження програмних модулів;
6) висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій

Слайд-презентація – 14 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Аналіз проблеми та огляд існуючих рішень	27.05.25-30.05.25	
2	Вибір технології розробки та інструменталь-	31.05.25-02.06.25	
	них засобів		
3	Розробка алгоритмічного забезпечення	03.06.25-05.06.25	
4	Розробка та відлагодження програмного	06.06.25-09.06.25	
	забезпечення		
5	Оформлення матеріалів кваліфікаційної роботи	10.06.25-11.06.25	
6	Подання атестаційної роботи керівникові та	12.06.25-13.06.25	
	її попередній захист		
7	Подання кваліфікаційної роботи на рецензу-	14.06.25-16.06.25	
	вання		

Дата видачі завдання " 26 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ас. Антон ГАВРАШЕНКО
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 80 с., 8 рис., 8 табл., 1 дод., 35 джерел.

JAVA, SPRING BOOT, FRAMEWORK, BOT API, POSTGRESQL, HTTP.

Метою кваліфікаційної роботи є створення та розгортання інформаційного телеграм бота.

У ході виконання кваліфікаційної роботи проаналізовано проблеми інформаційного обслуговування студентів в українських ЗВО та існуючі засоби інформування. Проаналізовано вибір платформи для розробки бота.

Для реалізації обрано технологічний стек Java, Spring Boot та PostgreSQL. Спроєктовано архітектуру та розроблено функціонал Telegram-бота, що включає надання розкладу, оповіщення (о змінах в розкладі, новинах тощо) та надання довідкової інформації, FAQ тощо. Проведено тестування. Результатом є функціонуючий бот для автоматизації інформаційного обслуговування студентів.

ABSTRACT

Bachelor’s thesis: 80 pages, 8 figures, 8 tables, 1 appendices, 35 sources.

JAVA, SPRING BOOT, FRAMEWORK, BOT API, POSTGRESQL, HTTP.

The major goal of this thesis is to create and deploy an informational Telegram bot.

In order to complete the qualification thesis, the problems of information services for students in Ukrainian higher education institutions (HEIs) and existing means of information dissemination were analyzed. The choice of a platform for bot development was also analyzed.

The technological stack of Java, Spring Boot, and PostgreSQL was chosen for implementation. The architecture was designed and the functionality of the Telegram bot was developed, which includes providing schedules, notifications (about schedule changes, news, etc.), and providing reference information, FAQs, etc. Testing was conducted. The result is a functioning bot for automating information services for students.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД – база даних

ЗВО – заклад вищої освіти

ООП – об'єктно-орієнтований підхід (або об'єктно-орієнтоване програмування)

СДН – системи дистанційного навчання

СУБД – система управління базами даних

ACID – Atomicity, Consistency, Isolation, Durability

API – Application Programming Interface

CTE – Common Table Expressions

DL – Distance Learning

FAQ – Frequently Asked Questions

HTTP – HyperText Transfer Protocol

IDE – Integrated Development Environment

JSON – JavaScript Object Notation

JPA – Jakarta Persistence API

JVM – Java Virtual Machine

MVC – Model-View-Controller

MVCC – Multi-Version Concurrency Control

VPS – Virtual Private Server

ВСТУП

Сучасний освітній процес у закладах вищої освіти (ЗВО) нерозривно пов'язаний з ефективним інформаційним забезпеченням студентів. Своєчасний доступ до актуальної навчальної, організаційної та адміністративної інформації є критично важливим для успішного навчання, планування часу та повноцінної участі студентів у академічному житті. Однак, в українських ЗВО ця сфера часто стикається з певними труднощами. Студенти нерідко вказують на проблеми з оперативним інформуванням про зміни в розкладі чи важливі університетські події, наявність бюрократичних перепон та недостатню ефективність традиційних каналів комунікації, таких як дошки оголошень чи навіть офіційні веб-сайти, які не завжди забезпечують зручний та швидкий доступ до потрібної інформації. Водночас спостерігається стрімкий розвиток цифрових технологій та їх проникнення у повсякденне життя. Месенджери, зокрема Telegram, набули надзвичайної популярності в Україні, ставши одним з основних каналів комунікації та отримання інформації для молоді. Це створює сприятливі умови для впровадження нових інструментів інформаційного обслуговування на базі цих платформ. Чат-боти, як програмні агенти, здатні автоматизувати відповіді на типові запитання, надавати персоналізовану інформацію та забезпечувати цілодобовий доступ до необхідних даних, розглядаються як перспективний напрямок для покращення комунікації між університетом та студентами. Розробка спеціалізованого Telegram-бота, адаптованого до потреб конкретного факультету чи університету, може суттєво підвищити ефективність інформаційного обслуговування, зменшити навантаження на адміністративний персонал та покращити загальний досвід студентів.

Метою даної дипломної роботи є розробка Telegram-бота для автоматизації та покращення процесу надання актуальної адміністративної та навчальної інформації студентам університету.

Розробка та програмна реалізація Telegram-бота для автоматизації надання інформаційних послуг студентам з використанням мови програмування Java, фреймворку Spring Boot та системою управління базами даних (СУБД) PostgreSQL.

Для вирішення поставлених завдань у роботі використано комплекс методів: аналіз науково-технічної літератури та інтернет-джерел для вивчення проблеми та існуючих рішень; системний аналіз для визначення вимог до системи та проєктування її архітектури; об'єктно-орієнтоване програмування для реалізації програмного коду; методи тестування програмного забезпечення для перевірки коректності функціонування розробленого бота; методи порівняльного аналізу для обґрунтування вибору технологій.

Наукова частина полягає у розробці архітектури та програмної реалізації інформаційного Telegram-бота для студентів факультету КІУ з використанням стеку технологій Java/Spring Boot/PostgreSQL, що дозволяє інтегрувати різні джерела інформації (розклад, новини, FAQ) в єдиний зручний інтерфейс та забезпечує можливості для подальшого розширення функціоналу та інтеграції з іншими університетськими системами.

Розроблений Telegram-бот може бути впроваджений у діяльність університету для покращення якості та оперативності інформування студентів. Це дозволить зменшити час, який студенти витрачають на пошук інформації, знизити навантаження на деканат та кураторів щодо відповідей на типові запитання, а також підвищити загальну задоволеність студентів інформаційними сервісами університету. Програмний продукт може слугувати основою для створення аналогічних ботів для інших факультетів або ЗВО.

1 АНАЛІЗ ПРОБЛЕМИ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Актуальність проблеми інформаційного обслуговування студентів в українських ЗВО

Ефективне функціонування сучасного закладу вищої освіти неможливе без налагодженої системи комунікації та своєчасного інформування всіх учасників освітнього процесу. Для студентів доступ до актуальної та повної інформації є важливим фактором не лише для успішного навчання, але й для соціальної адаптації в університетському середовищі, планування часу та повноцінної участі в академічному житті [1].

Аналіз свідчить про наявність низки поширених викликів, з якими стикаються студенти у процесі навчання. Іноді студенти зіштовхуються з затримками у видачі необхідних документів, змін у розкладі або іншими організаційними бар’єрами, що ускладнюють участь у програмах мобільності чи вирішення побутових питань [2]. У таких умовах навчання нерідко супроводжується потребою в значній самостійності, адже молоді люди змушені самостійно долати частину адміністративних труднощів, що, утім, сприяє розвитку навичок адаптації та відповідальності.

Хоча ця проблема напряму не пов'язана з оперативним інформуванням, вона ілюструє загальний контекст інформаційних викликів, з якими стикається студентство [3].

Розробка автоматизованих інформаційних інструментів, таких як чат-бот, може суттєво покращити комунікацію та полегшити доступ до інформації, але не здатна самостійно усунути першопричини цих системних недоліків. Тим не менш, в умовах, коли студенти значною мірою покладаються на самоосвіту через організаційні та методичні недоліки університету, цінність оперативного джерела інформації значно зростає. Автоматизований помічник, що надає швидкий доступ до розкладу,

різноманітної довідкової інформації та давати змогу університету вмить масово сповіщати про новини та зміни, може частково компенсувати ці недоліки, допомагаючи студентам ефективніше планувати свій час, концентруватися на навчанні та зменшувати витрати зусиль на пошук базової інформації.

Отже, роблема ефективного інформаційного супроводу студентів у закладах вищої освіти України є актуальною, багатовимірною та такою, що впливає на якість навчального процесу [4]. Це обґрунтовує необхідність пошуку та впровадження сучасних технологічних рішень для її подолання.

1.2 Аналіз існуючих каналів та засобів інформування студентів

Для комунікації зі студентами українські ЗВО використовують різноманітні канали, які можна умовно поділити на традиційні та цифрові.

До традиційних каналів належать:

- дошки оголошень: розміщуються в деканатах, на кафедрах, у коридорах навчальних корпусів. Основний недолік – обмежена доступність (лише при фізичній присутності в університеті) та потенційна застарілість інформації;

- особисте звернення: спілкування з викладачами, співробітниками деканату, кураторами. Цей канал є важливим, але не завжди ефективним для отримання оперативної інформації та може вимагати значних витрат часу;

- старости груп: часто виступають посередниками у передачі інформації від адміністрації до студентів. Ефективність залежить від відповідальності старости та оперативності отримання ним інформації;

Цифрові канали набули значного поширення, але також мають свої обмеження:

- офіційні веб-сайти університетів: є основним джерелом офіційної інформації. Наприклад, сайт Харківського національного університету радіоелектроніки (ХНУРЕ) nure.ua містить загальну інформацію про

університет, новини, структуру, контактні дані. Однак, ефективність таких сайтів може бути обмеженою. Рейтинги інформативності сайтів ЗВО показують значні відмінності між університетами [5]. Навіть сайти з високим рейтингом не завжди забезпечують своєчасне оновлення даних, мають складну навігацію або незручну структуру, що ускладнює швидке отримання специфічної інформації, необхідної студентам у повсякденному житті (наприклад, актуальний розклад конкретної групи або термінове оголошення деканату) [6];

- спеціалізовані веб-ресурси університету: деякі університети мають окремі сайти для специфічних потреб, наприклад, сайт з розкладом занять ХНУРЕ cist.nure.ua, на який є посилання з основного сайту. Хоча такі ресурси надають цільову інформацію, вони все ще вимагають від студента активного пошуку та перевірки оновлень і не забезпечують інтерактивної взаємодії чи сповіщень;

- сторінки в соціальних мережах: багато факультетів та кафедр мають свої сторінки (наприклад, у Facebook або Instagram). Вони можуть бути корисними для поширення новин та оголошень, але водночас формат соціальних мереж не завжди сприяє впорядкованому поданню інформації: контент часто є фрагментарним, неструктурованим і може залишатися непоміченим серед загального інформаційного потоку;

- електронна пошта: використовується для офіційних розсилок (info@nure.ua як приклад), але не завжди є зручним каналом для швидкого отримання оперативної інформації чи відповідей на запитання;

- системи дистанційного навчання (СДН): платформи типу Moodle, Classroom, DL використовуються переважно для навчальних матеріалів, завдань та комунікації в рамках конкретних курсів, а не для загальноуніверситетської інформації;

- вебінари: є ефективним інструментом для проведення лекцій, семінарів та тренінгів, особливо в умовах дистанційного навчання, забезпечуючи доступ до навчальних матеріалів [7]. Однак, вони не

призначені для оперативного інформування про адміністративні зміни чи надання довідкової інформації;

- загальні мобільні додатки для студентів: існує багато додатків, що допомагають студентам у навчанні та організації часу, наприклад, планувальники розкладу та завдань (iStudiez, Todo, Trello, Any.do), додатки для вивчення мов (Duolingo), розв'язання математичних задач (PhotoMath, Mathway) та управління фінансами (Mint.com) [8]. Ці інструменти є корисними, але вони не інтегровані з інформаційними системами конкретного університету і не надають доступу до офіційної академічної та адміністративної інформації [9].

Неофіційні мобільні додатки, розроблені студентами або сторонніми ініціативами, досить поширені в університетському середовищі. Їх основною метою зазвичай є забезпечення зручного доступу до розкладу занять. Наприклад, для Харківського національного університету радіоелектроніки (ХНУРЕ) існують додатки NURE Timetable для платформ Android та iOS. Варто зауважити, що подібні рішення, як правило, не мають офіційного статусу, що може викликати запитання щодо надійності їхньої підтримки, стабільності роботи та актуальності поданої інформації. Попри практичну користь у вирішенні конкретного завдання, тобто доступу до розкладу з мобільного пристрою, то функціональність таких застосунків зазвичай є обмеженою та не охоплює інші важливі аспекти інформаційного обслуговування (довідкова інформація, зв’язок з адміністрацією, оголошення деканату тощо).

На рисунку 1.1 зображено сайт ХНУРЕ.

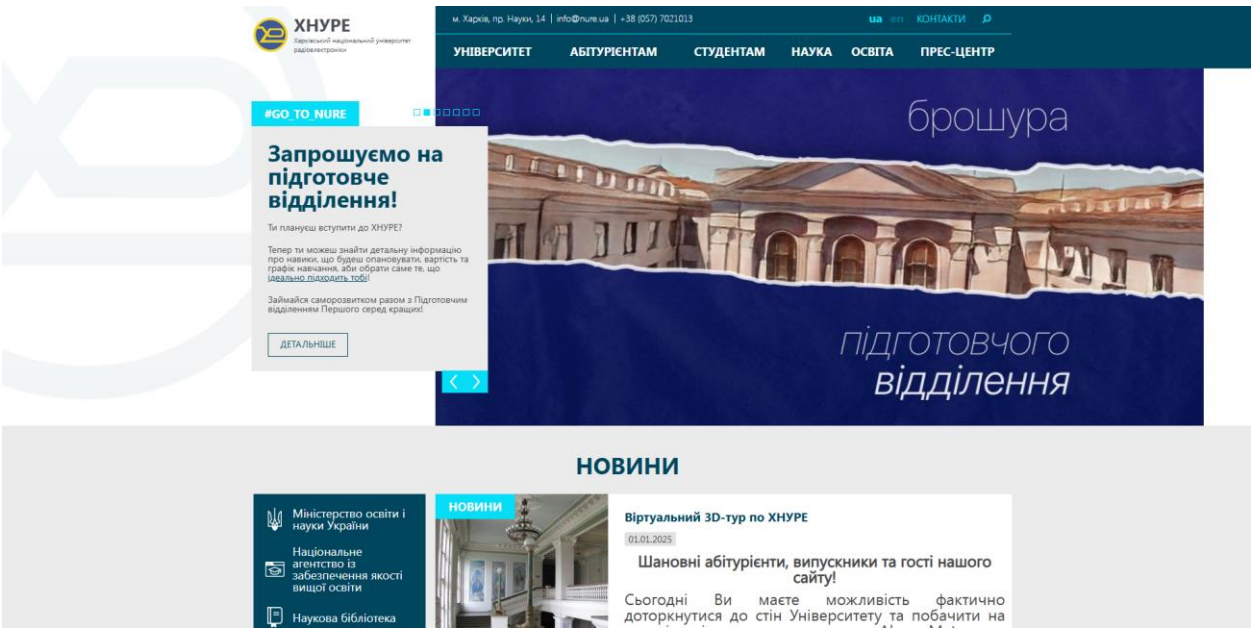


Рисунок 1.1 – Сайт університету

І також для прикладу наведемо зображення сайту розкладу (рисунок 1.2).

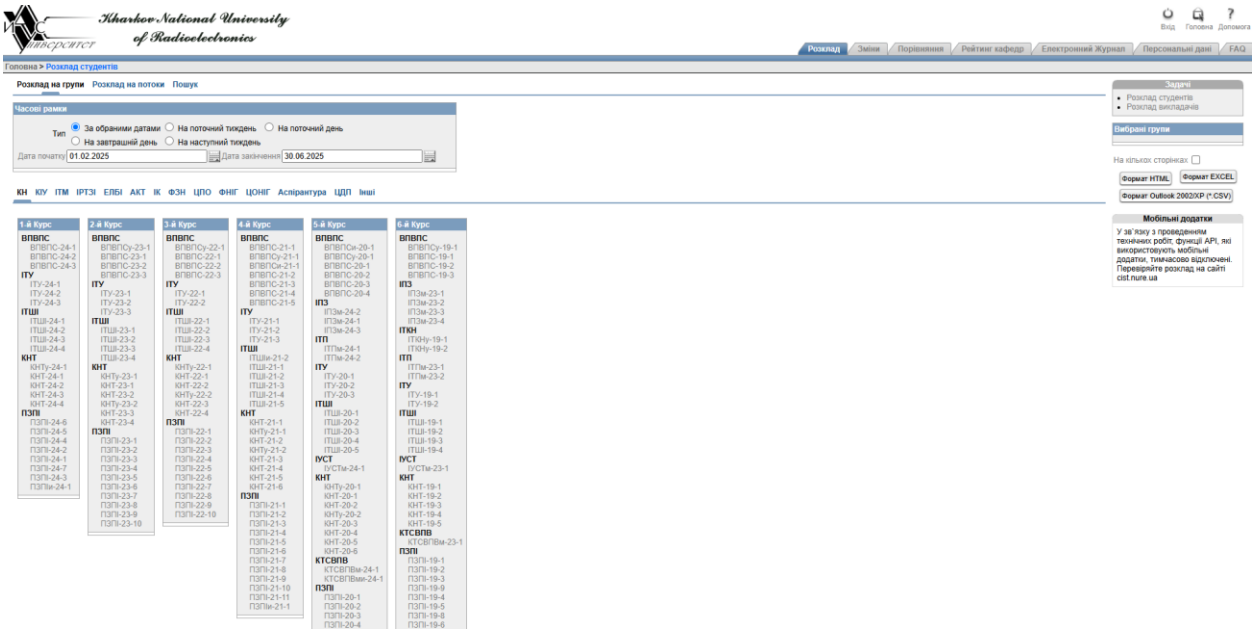


Рисунок 1.2 – Сайт розкладу університету

Для наочного порівняння основних каналів інформування, розуміння їх різниці, переваг, недоліків, актуальності та доступності, далі наведено таблицю 1.1.

Таблиця 1.1 – Порівняльний аналіз існуючих каналів інформування студентів

Канал	Переваги	Недоліки	Актуальність	Доступність
Сайт	Централізоване	Складність, без персоналізації	Середня	Висока
Сайт розкладу	Надання цільової інформації	Потребує активного пошуку, відсутність сповіщень	Середня	Висока
Дошка оголошень	Простота	Обмежена	Низька	Низька
Email	Офіційний канал	Незручність, потрапляння пошти в спам	Середня	Висока
Соцмережі	Швидкі новини, неформальне спілкування	Фрагментація, неофіційний	Висока	Висока
Мобільні додатки	Зручність	Без інтеграції, не надають адмін. інформацію	Середня	Висока
Неофіційні додатки розкладу	Мобільний доступ до розкладу	Неофіційний, нестабільність, обмежений функціонал	Висока	Висока

Аналіз існуючих каналів показує, що цифрові канали, такі як веб-сайти та соціальні мережі, мають такі проблеми фрагментації даних, нерегулярних оновлень та відсутності зручного інтерфейсу для швидкого пошуку потрібної інформації [3]. Навіть спеціалізовані ресурси, як-от сайт розкладу cist.nure.ua чи неофіційні мобільні додатки, хоч і надають доступ до конкретних даних, не вирішують проблему комплексно та не гарантують офіційної підтримки та повноти інформації. Це створює об'єктивну потребу в легкодоступному та інтерактивному джерелі актуальної університетської інформації.

1.3 Огляд аналогічних програмних рішень та чат-ботів у сфері освіти

Використання чат-ботів у сфері освіти є світовим трендом, що набуває поширення і в Україні. Чат-боти розглядаються як перспективний інструмент для автоматизації рутинних завдань, покращення комунікації та надання підтримки студентам [10]. Загальні можливості чат-ботів в освітньому процесі включають [11]:

- надання інформації: відповіді на поширені запитання студентів (FAQ), надання інформації про розклад занять, навчальні плани, університетські події;
- підтримка навчання: допомога у вивченні нових тем через надання додаткових матеріалів (статей, відео), проведення тестів, нагадування про завдання;
- збір зворотного зв'язку: проведення опитувань щодо якості курсів, викладачів, збір відгуків та пропозицій;
- організація та планування: допомога студентам у плануванні завдань, нагадування про дедлайни;
- підтримка дистанційного навчання: забезпечення швидкого доступу до інформації та відповідей на запитання в умовах відсутності прямого контакту з викладачами та адміністрацією.

В Україні вже існують приклади впровадження чат-ботів в

університетському середовищі. Яскравим прикладом є чат-бот Національного університету «Полтавська політехніка імені Юрія Кондратюка» (@NuppBot) в Telegram [12]. Цей бот, розроблений студентами університету, дозволяє користувачам: отримувати розклад, вмикати/вимикати нагадування про початок занять тощо.

Існування та успішне функціонування подібних спеціалізованих університетських ботів, як @NuppBot, є свідченням доцільності таких рішень. Це демонструє, що розробка та інтеграція бота з університетськими системами є цілком досяжною метою, а саме рішення відповідає сучасним тенденціям покращення сервісів для студентів.

Крім специфічних університетських ботів, існує низка інших чат-ботів, розроблених в Україні, які можуть бути корисними для студентів у різних аспектах життя, хоча й не пов'язані безпосередньо з навчальним процесом. Наприклад, боти для перевірки якості повітря (SaveEcoBot), допомоги у сортуванні сміття (NoWaste_Chat_bot), відстеження поштових відправлень (QTrackerBot), пошуку фільмів (Kinoreminderbot) чи навіть фітнес-тренувань (Gymbossbot) [13]. Це свідчить про загальну популярність та універсальність технології чат-ботів для вирішення різноманітних завдань.

Аналіз функціоналу існуючих освітніх ботів [14] (переважно відповіді на запитання, надання розкладу, нагадування) дозволяє визначити базовий набір можливостей, який очікується від подібного інструменту. Водночас, це відкриває можливості для диференціації та створення більш комплексного рішення. Розроблюваний бот може розширити свій функціонал, інтегруючи, наприклад, новини, інформацію про викладачів (години консультацій, контакти), важливі терміни (здача курсових, реєстрація на вибіркові дисципліни), а також надавати відповіді запитання щодо адміністративних процедур (отримання довідок, поселення в гуртожиток тощо), спираючись на проблеми, окреслені в підрозділі (підрозділ 1.1)

Таким чином, чат-боти є доведеним та перспективним інструментом для покращення інформаційного обслуговування в ЗВО [15]. Існуючі

приклади підтверджують їхню ефективність, але залишається простір для розробки більш комплексних та адаптованих рішень.

1.4 Постановка завдання

На основі проведеного аналізу проблеми, огляду каналів комунікації та аналогічних рішень, формулюється мета та завдання даної дипломної роботи.

Мета роботи: розробити Telegram-бота для автоматизації та покращення процесу надання актуальної адміністративної та навчальної інформації студентам університету.

Для досягнення поставленої мети мають бути вирішені наступні задачі:

- проаналізувати специфічні інформаційні потреби студентів;
- спроєктувати архітектуру програмного комплексу, що включає Telegram-бота, серверну частину та базу даних;
- обґрунтувати вибір технологічного стеку для розробки: платформа Telegram, мова програмування Java, фреймворк Spring Boot, СУБД PostgreSQL;

Реалізувати основний функціонал бота, що включає щонайменше:

- надання розкладу занять для обраної групи;
- можливість робити оголошення (новини, зміни часу пар тощо);
- можливість створювати довідкову інформацію;
- розробити програмний код з використанням обраних технологій;
- налаштувати взаємодію серверної частини з Telegram Bot API;
- створити структуру бази даних (БД) та забезпечити її наповнення необхідною інформацією (розклади, новини, FAQ тощо);
- провести тестування розробленого програмного продукту;
- підготувати документацію до дипломної роботи до чинних стандартів.

Очікуваний результат це функціонуючий Telegram-бот, доступний для використання, який демонструє вирішення поставлених завдань.

2 ВИБІР ТЕХНОЛОГІЇ РОЗРОБКИ ТА ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ

2.1 Аналіз та вибір платформи

Вибір платформи для розгортання чат-бота є ключовим рішенням, що впливає на доступність, функціональність та потенційне охоплення аудиторії. Для даного проєкту було обрано платформу Telegram, виходячи з сукупності технічних та соціально-демографічних факторів.

По-перше, Telegram є популярним месенджером в Україні. Згідно з соціологічними дослідженнями 2024 року, Telegram очолює рейтинг соціальних мереж за популярністю, ним користується 78.1% українських респондентів. Цей показник демонструє стабільне зростання, особливо в контексті споживання новин (зростання на 6.8% порівняно з попереднім роком) [16]. Така висока проникність серед населення, включаючи цільову аудиторію – студентів, є вагомою перевагою. Розробка бота саме для Telegram мінімізує бар'єр входу для користувачів, оскільки їм не потрібно встановлювати жодних додаткових застосунків – бот буде доступний у звичному для них середовищі. Це суттєво відрізняється від підходу розробки окремого мобільного додатку, який потребує встановлення та може сприйматися як менш зручний [17].

По-друге, Telegram надає потужний, гнучкий та безкоштовний інтерфейс програмування додатків (API) для розробки ботів – Telegram Bot API [18]. Ключові технічні характеристики та переваги цього API включають:

- HTTP-орієнтованість: взаємодія з API відбувається через стандартні HTTP-запити по захищеному протоколу HTTPS. Це забезпечує сумісність з будь-якою мовою програмування та фреймворком, що підтримує HTTP-клієнти;

- формат даних JSON: усі дані, що передаються до API та

отримуються від нього, мають формат JSON, який є стандартом для веб-сервісів;

- механізми отримання оновлень: API підтримує два основні способи отримання повідомлень та подій від користувачів: Long Polling (періодичні запити getUpdates) та Webhooks (отримання POST-запитів від Telegram на вказаний URL у реальному часі при надходженні оновлення) [19];

- гнучке налаштування подій: можливість підписуватися лише на певні типи оновлень (наприклад, нові повідомлення, редаговані повідомлення, натискання кнопок), що дозволяє оптимізувати обробку;

- багатий функціонал: API дозволяє не лише надсилати та отримувати текстові повідомлення, але й працювати з різними типами контенту (зображення, відео, документи, аудіо, геолокація), створювати інтерактивні клавіатури (звичайні та вбудовані), редагувати повідомлення, керувати групами та каналами, отримувати інформацію про користувачів та багато іншого [20];

- наявність бібліотек: існує велика кількість готових бібліотек для різних мов програмування (включаючи Java [21]), які спрощують взаємодію з API;

- документація та спільнота: telegram надає детальну офіційну документацію та має активну спільноту розробників;

- можливість локального сервера API: для специфічних потреб (робота з великими файлами, зменшення затримок) можна розгорнути власний локальний сервер Bot API.

Гнучкість Telegram Bot API дозволяє створювати не просто інформаційні канали, а повноцінні інтерактивні додатки всередині месенджера. Можливість використання кнопок, меню, команд, отримання різних типів даних від користувача дозволяє спроєктувати зручний та інтуїтивно зрозумілий користувацький інтерфейс, що значно покращує досвід взаємодії порівняно з традиційними каналами, такими як веб-сайти чи дошки оголошень.

Для порівняння Telegram з іншими популярними в Україні платформами (Viber та Facebook Messenger), що підтримують ботів, наведено таблицю 2.1.

Таблиця 2.1 – Порівняння месенджер-платформ для розробки ботів

Платформа	Популярність	Можливості API	Обмеження
Telegram	78.1% [16]	HTTP API, JSON, Webhooks/Long Polling, різний контент	Ліміти на частоту надсилання повідомлень [18]
Viber	~42% [22]	HTTP API, JSON, Webhooks, обмежений контент	Можлива плата за бізнес-повідомлення, суворіші політики щодо розсилок
Facebook Messenger	44.6% [22]	Graph API, Webhooks, різний контент. інтеграція з Facebook Platform	Складніша політика, прив'язка до екосистеми

Як видно з таблиці, Telegram поєднує найвищу популярність в Україні з потужним, гнучким та безкоштовним API, а також відмінною підтримкою для розробників. Це робить його оптимальним вибором для розробки.

2.2 Аналіз та вибір мови програмування та фреймворку

Вибір мови програмування та фреймворку для розробки бота є важливим архітектурним рішенням, що впливає на швидкість розробки,

надійність, продуктивність та можливості подальшого розвитку системи. Для даного проєкту було обрано мову програмування Java у поєднанні з фреймворком Spring Boot.

Java є однією з найпопулярніших та найпоширеніших мов програмування у світі, особливо у сфері розробки великих корпоративних систем [23]. Її переваги включають [24]:

- об'єктно-орієнтований підхід (ООП): сприяє створенню структурованого, модульного та перевикористовуваного коду;
- строга типізація: дозволяє виявляти багато помилок на етапі компіляції, підвищуючи надійність програми;
- платформонезалежність: принцип "Write Once, Run Anywhere" (WORA) завдяки використанню віртуальної машини Java (JVM) дозволяє запускати скомпільований код на різних операційних системах без змін;
- велика екосистема: наявність величезної кількості зрілих бібліотек, фреймворків та інструментів для вирішення різноманітних завдань;
- продуктивність: сучасні реалізації JVM забезпечують високу продуктивність, що важливо для серверних додатків, які обслуговують багато запитів;
- велика спільнота: наявність великої спільноти розробників полегшує пошук інформації, вирішення проблем та найм кваліфікованих фахівців.

Хоча для розробки Telegram-ботів часто використовується Python через його лаконічність та швидкість прототипування [25], Java була обрана з огляду на потенційну складність та вимоги до надійності університетського інформаційного сервісу. Java зазвичай має кращу продуктивність у порівнянні з Python завдяки компіляції в байт-код і використанню JVM. Це може бути важливо для масштабованості, якщо в майбутньому бот буде обробляти великий обсяг запитів та мати потребу у високій швидкості реакції. Її строгість та орієнтація на ООП сприяють створенню більш надійних та підтримуваних систем у довгостроковій перспективі. Важливим фактором є також наявність якісних та добре підтримуваних Java-бібліотек

для роботи з Telegram Bot API [21], що спрощує інтеграцію. Варто зазначити, що Telegram API є HTTP-інтерфейсом, тому технічно бота можна розробити практично будь-якою мовою програмування, що підтримує веб-запити

Spring Boot є розширенням популярного фреймворку Spring, розробленим для спрощення створення автономних, готових до запуску додатків на основі Spring [26]. Він значно прискорює процес розробки, дозволяючи розробникам зосередитися на бізнес-логіці, а не на конфігурації інфраструктури. Ключові переваги Spring Boot для даного проєкту включають [27]:

- автоконфігурація: Spring Boot автоматично налаштовує багато компонентів Spring та сторонніх бібліотек на основі залежностей, присутніх у проєкті, мінімізуючи необхідність ручної XML або Java-конфігурації;

- вбудовані сервери: можливість вбудовувати веб-сервери (Tomcat, Jetty, Undertow) безпосередньо в JAR-архів додатку, що робить розгортання надзвичайно простим (просто запуск JAR-файлу);

- стартери (Starters): набори залежностей, що спрощують підключення та налаштування певних функціональних можливостей (наприклад, spring-boot-starter-web для веб-додатків, spring-boot-starter-data-jpa для роботи з БД через JPA). Існують також стартери, що полегшують інтеграцію з Telegram Bot API (наприклад, TelegramBots-Spring-Boot-Starter [28]), або ж можна інтегрувати сторонні бібліотеки у Spring Boot додаток;

- підтримка мікросервісної архітектури: хоча даний бот може бути реалізований як монолітний додаток, Spring Boot надає всі необхідні інструменти для побудови мікросервісів, що може бути корисним при подальшому розвитку системи;

- інтеграція з екосистемою Spring: легкий доступ до всіх можливостей Spring Framework [29], включаючи Spring Data (для спрощеної роботи з БД), Spring Security (для забезпечення безпеки), Spring MVC (для створення REST API) тощо;

- велика спільнота та документація: як і сам Spring, Spring Boot має

величезну спільноту та вичерпну документацію.

Використання Java у поєднанні зі Spring Boot дозволяє створити не просто скрипт для обробки повідомлень Telegram, а повноцінний, надійний та масштабований сервіс. Це особливо важливо в контексті університетського середовища, де може виникнути потреба в інтеграції з іншими інформаційними системами, забезпеченні високої доступності сервісу для великої кількості користувачів та можливості подальшого розширення функціоналу. Наявність спеціалізованих стартерів або легкість інтеграції бібліотек для Telegram [30] є ключовим фактором ефективності, оскільки дозволяє абстрагуватися від низькорівневих деталей взаємодії з Telegram API та зосередитися на реалізації основної логіки бота: обробці команд, взаємодії з БД та формуванні відповідей користувачам.

У таблиці 2.2 підсумовано основні переваги Spring Boot стосовно розробки даного Telegram-бота.

Таблиця 2.2 – Переваги Spring Boot для розробки Telegram-бота

Перевага	Опис переваги	Релевантність для проєкту
Автоконфігурація	Автоматичне налаштування компонентів на основі залежностей, зменшення шаблонного коду	Прискорює початкове налаштування проєкту, конфігурацію підключення до БД, налаштування веб-сервера та інтеграцію з Telegram API.
Вбудовані сервери	Можливість пакувати веб-сервер (Tomcat) разом з додатком у єдиний JAR-файл [26]	Спрощує розгортання та запуск бота на сервері, не потребує встановлення та налаштування.

Продовження таблиці 2.2

Перевага	Опис переваги	Релевантність для проєкту
Стартери	Спрощене керування залежностями та початкова конфігурація для різних модулів (web, data, security, Telegram) [28, 30]	Дозволяє швидко додати необхідний функціонал (веб-сервіс, робота з БД, інтеграція з Telegram API), зменшуючи час на налаштування.
Підтримка мікросервісів	Спрощує створення незалежних, невеликих сервісів	Забезпечує гнучкість для майбутнього розвитку, якщо виникне потреба розділити функціонал на окремі сервіси
Екосистема Spring	Легка інтеграція з іншими модулями Spring (Data JPA, Security, MVC) [29]	Спрощує роботу з БД (Spring Data JPA), реалізацію REST API для адміністрування (Spring MVC), потенційне додавання автентифікації/авторизації (Spring Security).
Велика спільнота	Широка підтримка спільноти, велика кількість навчальних матеріалів та прикладів	Полегшує процес навчання, пошук рішень для типових проблем та розробку в цілому.

Отже, вибір Java та Spring Boot забезпечує надійну, продуктивну та масштабовану основу для розробки Telegram-бота, суттєво прискорюючи та спрощуючи процес завдяки потужному інструментарію та інтеграції з широкою екосистемою Spring.

2.3 Аналіз та вибір СУБД

Для зберігання даних, необхідних для роботи Telegram-бота (інформація про користувачів, їхні налаштування, дані про групові чати тощо), потрібна надійна та ефективна СУБД. Для даного проєкту було обрано PostgreSQL.

PostgreSQL – це об'єктно-реляційна СУБД з відкритим вихідним кодом, яка має тривалу історію розвитку (понад 35 років) і заслужила репутацію надійної, відмовостійкої та високопродуктивної системи [31]. Вона широко використовується для різноманітних додатків, від невеликих проєктів до великих корпоративних систем.

Ключові переваги PostgreSQL, що роблять її привабливим вибором для даного проєкту, включають:

- відповідність стандартам та ACID (Atomicity, Consistency, Isolation, Durability): PostgreSQL суворо дотримується стандарту SQL та гарантує ACID-властивості для всіх транзакцій, що забезпечує цілісність та надійність даних;

- розширюваність: система дозволяє створювати власні типи даних, функції, оператори та індекси, а також використовувати процедурні мови (PL/pgSQL, PL/Python, PL/Perl та ін.), що надає велику гнучкість розробникам;

- робота зі складними запитами: PostgreSQL має розвинений оптимізатор запитів та підтримує широкий спектр можливостей SQL, включаючи Common Table Expressions (CTE), віконні функції, рекурсивні

запити та різноманітні типи індексів (B-tree, Hash, GiST, GIN, BRIN), що дозволяє ефективно виконувати складні аналітичні запити;

- підтримка різноманітних типів даних: окрім стандартних типів даних, PostgreSQL підтримує геометричні типи, мережеві адреси, перелічувані типи (ENUM), діапазони, XML, а також масиви будь-яких типів даних;

- ефективна робота з JSON: PostgreSQL надає два типи для зберігання JSON-даних: JSON (зберігає як текст з перевіркою валідності) та JSONB (зберігає у бінарному форматі). JSONB є особливо цінним, оскільки дозволяє індексувати дані всередині JSON-структури (за допомогою GIN або B-tree індексів) та ефективно виконувати пошук та маніпуляції з цими даними. Це надає значну гнучкість для зберігання напівструктурованої інформації, такої як налаштування користувачів, тексти оголошень чи інші дані зі змінною структурою, без необхідності проєктування надто складної реляційної моделі;

- конкурентний доступ (MVCC): PostgreSQL використовує механізм Multi-Version Concurrency Control (MVCC), який дозволяє операціям читання не блокувати операції запису, і навпаки. Це забезпечує високу продуктивність та паралелізм при великій кількості одночасних користувачів, що є важливим для бота, яким потенційно користуватимуться багато студентів одночасно;

- масштабованість та надійність: PostgreSQL добре масштабується для роботи з великими обсягами даних та високими навантаженнями. Вона має розвинені механізми реплікації (асинхронна, синхронна, логічна), журналювання Write-Ahead Logging (WAL) та відновлення на певний момент часу Point-in-Time Recovery (PITR), що забезпечує високу доступність та захист даних.

Порівнюючи PostgreSQL з іншою популярною СУБД з відкритим кодом MySQL [32], можна відзначити наступне: PostgreSQL загалом краще підходить для додатків зі складними запитами, частими операціями запису та

потребою у гнучких типах даних (як JSONB). MySQL може бути дещо простішим у налаштуванні та адмініструванні для початківців і вищу продуктивність [33] для простих операцій читання [34]. Однак, для інформаційного бота, який потенційно працюватиме з різноманітними типами даних та може потребувати складних запитів у майбутньому, переваги PostgreSQL роблять його більш обґрунтованим вибором.

У таблиці 2.3 наведено порівняльний аналіз PostgreSQL та MySQL з точки зору потреб проєкту.

Таблиця 2.3 – Порівняння PostgreSQL та MySQL для потреб додатку

Критерій	PostgreSQL	MySQL	Значення для бота
ACID відповідність	Завжди	Обмежена	Гарантує цілісність даних у всіх операціях,
MVCC	Вбудована	Обмежена	Забезпечує кращу продуктивність при одночасному читанні та записі даних багатьма користувачами бота
Складні запити	Повна підтримка	Обмежена	Важливо для майбутнього розширення функціоналу (аналітика, пошук, персоналізація)
JSONB	JSONB є	JSON, без JSONB	Напівструктуровані дані

Продовження таблиці 2.3

Критерій	PostgreSQL	MySQL	Значення для бота
Масиви	Нативна підтримка масивів будь-яких типів	Відсутня (потрібна нормалізація або зберігання як текст)	Може бути корисним для зберігання списків (наприклад, списки подій, теги)
Розширюваність	Дуже висока (типи даних, функції, оператори, мови)	Обмежена	Надає можливості для специфічних налаштувань та оптимізацій у майбутньому
Масштабованість	Добре масштабується (для великих обсягів даних та навантажень)	Добре масштабується, (для операцій читання)	Забезпечує запас міцності для зростання кількості користувачів та обсягу даних
Продуктивність	Висока для запису та складних запитів	Висока для читання та простих запитів	Баланс між читанням та записом робить MVCC PostgreSQL перевагою

Таким чином, PostgreSQL обрано як СУБД для проєкту завдяки її надійності (ACID, MVCC), гнучкості з різними типами даних (JSONB та масиви), високій продуктивності при змішаних навантаженнях та можливостям масштабування, що відповідає вимогам.

2.4 Аналіз та вибір інструментальних засобів розробки

Для ефективної розробки програмного забезпечення використовується набір інструментальних засобів, що автоматизують та спрощують різні етапи процесу. У рамках даного проєкту планується використання наступних стандартних інструментів:

а) інтегроване середовище розробки (IDE): IntelliJ IDEA (Community або Ultimate Edition). Це одна з найпопулярніших IDE для розробки на Java та Kotlin, що надає потужну підтримку для фреймворку Spring Boot, систем збирання Gradle та Maven, системи контролю версій Git, інструменти для роботи з базами даних, рефакторингу коду, налагодження та тестування. Її інтелектуальні функції значно прискорюють написання коду та зменшують кількість помилок;

б) система збирання (Build System): Maven [35]. Цей інструмент автоматизує процес компіляції коду, керування зовнішніми залежностями, запуску тестів та пакування додатку у виконуваний JAR-файл. Також має відмінну інтеграцію з Spring Boot;

в) система контролю версій (VCS): Git. Використання Git є стандартом де-факто для контролю версій у сучасній розробці. Він дозволяє відстежувати зміни у коді, повертатися до попередніх версій, створювати окремі гілки для розробки нових функцій без впливу на основну кодову базу, а також ефективно працювати в команді (навіть якщо розробник один, використання Git є хорошою практикою). Для зберігання Git-репозиторію можуть використовуватися платформи GitHub, GitLab або Bitbucket;

г) клієнт для роботи з БД (Database Client): DBeaver або pgAdmin. Ці інструменти надають графічний інтерфейс для взаємодії з БД PostgreSQL. Вони дозволяють переглядати структуру таблиць та дані, виконувати SQL-запити, керувати користувачами та правами доступу, створювати резервні копії під час розробки та тестування. DBeaver є універсальним клієнтом, що

підтримує багато СУБД, тоді як pgAdmin є офіційним інструментом адміністрування саме для PostgreSQL;

ґ) інструменти тестування:

1) JUnit: стандартна бібліотека для написання та запуску unit-тестів у Java;

2) Mockito: бібліотека для створення об'єктів-замінників (моків), що дозволяє ізолювати тестований компонент від його залежностей;

3) Spring Boot Test: надає утиліти та анотації для написання інтеграційних тестів для Spring Boot додатків, дозволяючи тестувати взаємодію між різними компонентами (наприклад, контролером, сервісом та репозиторієм);

д) інструмент для тестування API Postman. Якщо бот матиме в майбутньому додатковий REST API (наприклад, для адміністрування через веб-інтерфейс або для інтеграції з іншими системами), цей інструмент дозволить легко надсилати HTTP-запити та перевіряти відповіді API.

Використання цього стандартного набору інструментів не лише забезпечує ефективність та якість процесу розробки завдяки їх багатому функціоналу та взаємній інтеграції, але й відповідає загальноприйнятим практикам у Java/Spring Boot розробці. Це також знижує поріг входження для будь-яких майбутніх розробників, які можуть долучитися до підтримки або подальшого розвитку проєкту, оскільки ці інструменти є добре відомими в індустрії.

3 РОЗРОБКА АЛГОРИТМІЧНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Аналіз основних функцій системи

Програмний продукт InfoNureBot розроблявся з орієнтацією на потреби студентів, спрямованого на надання навчальної інформації. Основною метою бота є оперативне забезпечення користувачів даними, що стосуються навчального процесу.

Ключовий функіонал, який повинен мати InfoNureBot, включає:

- збереження, зміна, обробка даних про користувачів та групи (ID, username, код групи тощо);
- надання можливості користувачеві або групі вказати або змінити свою академічну групу для отримання релевантного розкладу;
- надання актуального розкладу занять для зазначеної користувачем академічної групи та дати;
- адміністративні функції (як на рівні бота, так і на рівні групових чатів), такі як можливість блокування певних користувачів, зміна коду групи та інше;
- можливість масової розсилки адміністрацією оголошень;
- зворотній зв’язок (як від користувача до адміністрації, так і навпаки).

3.2 Шаблон MVC, інструмент збірки Maven та бібліотека telegrambots

Проєкт дотримується архітектурного шаблону MVC, адаптованого для Telegram-ботів:

- model (Модель). Представляє дані. Цю роль виконують класи сутностей JPA з пакету model та класи даних розкладу з пакету parser;
- view (Представлення). Відповідає за представлення даних користувачеві. У контексті бота, це повідомлення та інтерактивні елементи.

Цю функцію реалізують класи-фабрики з пакету view;

- controller (Контролер). Приймає запити від користувача та взаємодіє з моделлю. Роль контролера виконують контролер та диспатчер;

- service (Сервіс). Додатковий шар, розташований між контролерами та репозиторіями.

Maven це інструмент, який автоматизує процес збірки та керування проєктами. Maven полегшує управління залежностями, налаштуваннями та структурою проєктів, що робить розробку більш впорядкованою і ефективною. Переваги використання Maven:

- декларативне управління залежностями. Залежності проєкту описуються у файлі pom.xml. Maven автоматично завантажує необхідні бібліотеки та їх транзитивні залежності з центрального або інших репозиторіїв;

- стандартизована структура проєкту. Maven визначає стандартну структуру каталогів для вихідного коду, ресурсів, тестів тощо, що полегшує розуміння проєкту новими розробниками;

- управління життєвим циклом збірки. Maven має визначений життєвий цикл збірки (compile, test, package, install, deploy), що дозволяє стандартизувати та автоматизувати ці процеси;

- велика кількість плагінів. Існує безліч плагінів для Maven, що розширюють його функціональність (наприклад, для генерації звітів, роботи з Docker, запуску застосунків).

Для спрощення взаємодії з API була використана бібліотека telegrambots, зокрема її інтеграція зі Spring Boot через telegrambots-spring-boot-starter. Ця бібліотека є обгорткою над Telegram Bot API, яка абстрагує розробника від низькорівневих HTTP-запитів та обробки JSON, надаючи зручні Java-класи та методи для роботи з об'єктами Telegram (Update, Message, User тощо) та надсилання команд (наприклад, SendMessage). Це значно прискорює процес розробки та зменшує кількість шаблонного коду.

3.3 Алгоритми роботи, обробки команд та взаємодії з користувачем

Процес роботи починається з ініціалізації додатку на базі Spring Boot. Spring створює та налаштовує всі необхідні компоненти (біни), включаючи сервіси, репозиторії, залежності, конфігурації та головний клас бота. Після успішної ініціалізації, бот реєструється на серверах Telegram за допомогою унікального токена та починає отримувати оновлення за допомогою механізму Long Polling.

Кожне оновлення, що надходить, містить інформацію про користувача. Система аналізує тип оновлення, виокремлює з нього дані та передає їх на обробку відповідному сервісу.

Взаємодія користувача з ботом відбувається через надсилання команд та текстових повідомлень у чаті. Загальна схема обробки вхідного повідомлення (об'єкта Update) побудована на основі отримання оновлення головним класом-контролером та подальшого делегування обробки спеціалізованим компонентам. Клас InfoNureBot виконує роль основного обробника вхідних оновлень від Telegram. Отримавши об'єкт Update, InfoNureBot передає його для подальшого аналізу та маршрутизації.

Загальний алгоритм роботи системи можна представити у вигляді блок-схеми (рисунок 3.1).

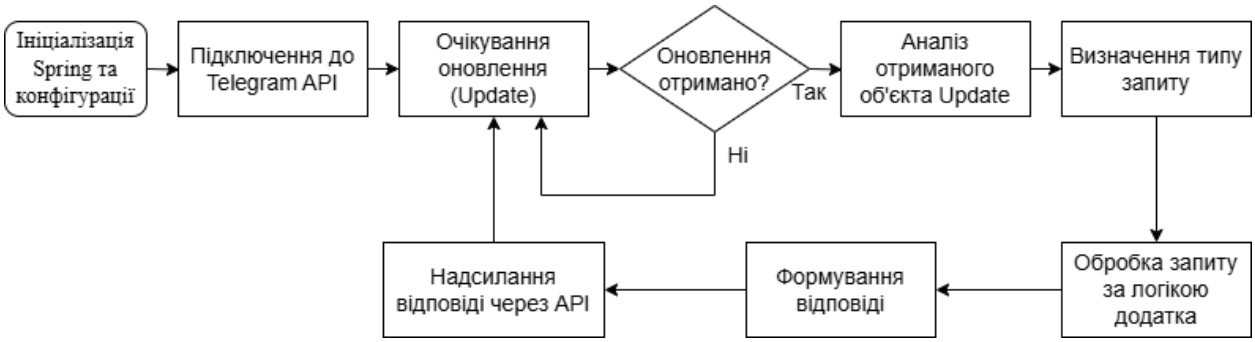


Рисунок 3.1 – Алгоритм роботи додатка

Схема описує роботу системи (Update, обробка і т.д.), головне з чим працює система – це об’єкт Update, без оновлень, бот просто чекає.

3.4 Алгоритм диспетчеризації оновлень (UpdateDispatcher) та обробка повідомлень

Ключову роль у маршрутизації вхідних оновлень відіграє клас UpdateDispatcher, що знаходиться в пакеті com.infonure.infonure_bot.handler. Цей клас реалізує логіку визначення типу отриманого оновлення та виклику відповідного обробника. Алгоритм роботи UpdateDispatcher можна представити наступним чином. Клас InfoNureBot передає об'єкт Update. Визначається тип оновлення через перевірку його полів (чи є текст, callback тощо). Залежно від типу, UpdateDispatcher викликає відповідний метод обробки для команд, звичайних повідомлень чи callback-запитів.

На концептуальному рівні UpdateDispatcher може реалізовувати елементи шаблонів проєктування "Ланцюжок відповідальності" (Chain of Responsibility)

3.4.1 Алгоритм обробки команд

Текстові команди є основним способом ініціювання дій з боку користувача. Алгоритм їх обробки включає наступні кроки:

- розпізнавання команди. Після того, як UpdateDispatcher визначив, що отримано текстове повідомлення, відбувається перевірка, чи є це повідомлення командою (зазвичай починається зі символа «/»);
- обробка команди та надсилання її до відповідного обробника (handler);
- подальша робота відповідно до команди, може бути просто надсилання повідомлення, наприклад, допоміжної інформації (/help), або робота з БД (наприклад, встановлення групи командою /group);
- надсилання відповіді.

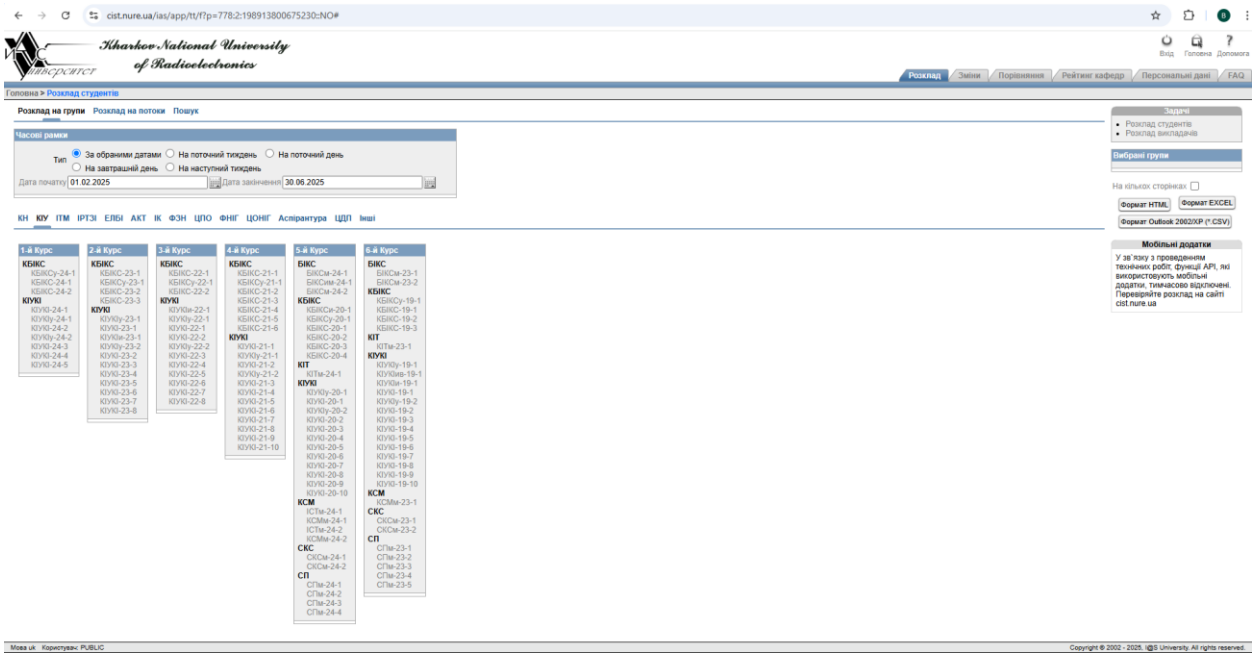
3.4.2 Алгоритм обробки запитів зворотного виклику (Callback Query)

Запити зворотного виклику генеруються при натисканні користувачем на кнопки. Алгоритм їх обробки виглядає так:

- отримання об'єкта CallbackQuery з Update;
- парсинг даних, переданих у полі callback_data кнопки, для ідентифікації дії;
- виконання відповідних дій. Надсилання нового повідомлення, взаємодія з сервісними класами для отримання даних;
- формування відповіді за допомогою MessageFactory та KeyboardFactory;
- надсилання відповіді на запит для підтвердження Telegram, що запит оброблено.

3.5 Алгоритми роботи з розкладом

У зв’язку з тим, що офіційний API університету виявився тимчасово недоступним для інтеграції (рисунок 3.2) та блокував часті запити було прийнято рішення використовувати метод парсингу HTML-файлу розкладу. Такий підхід дозволяє отримувати необхідну інформацію безпосередньо зі сторінки розкладу.



а)

Мобільні додатки

У зв`язку з проведенням технічних робіт, функції API, які використовують мобільні додатки, тимчасово відключені. Перевіряйте розклад на сайті cist.nure.ua

б)

Рисунок 3.2 – Сайт розкладу ХНУРЕ: а) сторінка розкладу; б) попередження про недоступність API

Центральним компонентом, відповідальним за логіку, пов'язану з розкладом, є ScheduleService з пакету com.infonure.infonure_bot.service.

Клас HtmlScheduleParser відповідає за вилучення інформації про розклад з HTML-документів. Роботу розкладу можна описати так:

а) завантаження HTML-контенту. Метод парсингу приймає HTML-контент, з файлу;

б) використання бібліотеки для парсингу. Для розбору HTML-структури використовується спеціалізована бібліотека Jsoup;

в) вибірка ключових DOM-елементів. На основі аналізу структури HTML-сторінки, парсер використовує селектори (наприклад, CSS-селектори)

для знаходження елементів, що містять потрібну інформацію:

1) таблиця розкладу;

2) рядки, що відповідають дням тижня та парам;

3) комірки, що містять час, назву дисципліни, тип заняття, ПІБ викладача, аудиторію;

г) структурування вилучених даних. Вилучені текстові дані перетворюються та зберігаються в об'єктах спеціально створених класів моделі:

1) Schedule: кореневий об'єкт, що містить розклад;

2) DaySchedule: містить список занять на конкретний день (List<LessonInfo>);

3) LessonInfo: містить детальну інформацію про одне заняття: timeSlot, lessonName, lessonType, teacherName, auditorium;

4) TimeSlot: містить час початку (startTime) та закінчення (endTime) заняття;

ґ) після отримання даних розкладу ScheduleService готує їх для представлення користувачеві:

1) фільтрація розкладу. Залежно від запиту користувача, фільтрація розкладу на конкретний день тижня або на поточний/наступний тиждень;

2) форматування даних. Структуровані дані розкладу перетворюються на зручний для користувача текст;

3) додавання елементів керування. За допомогою KeyboardFactory до повідомлення з розкладом можуть додаватися inline-клавіатури з кнопками для навігації, що покращує користувацький досвід.

3.6 Алгоритми взаємодії з базою даних PostgreSQL засобами Spring Data JPA

Для взаємодії з базою даних PostgreSQL використовується Spring Data JPA, що абстрагує розробника від написання SQL-запитів вручну. Цей

фреймворк спрощує створення шару доступу до даних, автоматично генеруючи реалізації для інтерфейсів-репозиторії. Розробнику потрібно лише визначити інтерфейс, успадкований від JpaRepository та, за потреби, додати методи для специфічних запитів. Цей підхід дозволяє чітко розділити бізнес-логіку (в сервісах) та логіку доступу до даних (в репозиторіях), що відповідає принципам чистої архітектури та спрощує підтримку та тестування коду.

Алгоритм взаємодії з БД описано в блок-схемі (рисунок 3.3).

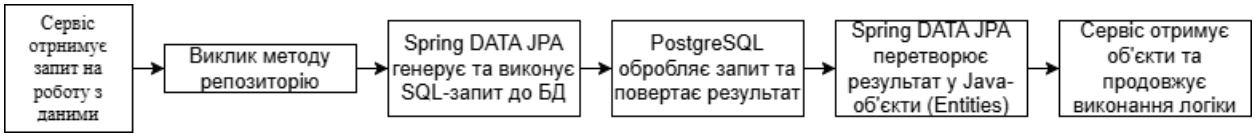


Рисунок 3.3 – Алгоритм взаємодії з БД

Описаний алгоритм пояснює всі етапи роботи при взаємодії з БД.

3.6.1 Операції з даними користувачів (user_data)

Реєстрація нового користувача. При першому контакті (/start) UserService перевіряє наявність користувача в БД за його ID Telegram. Якщо користувача немає, створюється новий екземпляр User, заповнюються поля id, username, created, group_code (спочатку строка null) і сутність зберігається в БД (userRepository.save(newUser)).

Оновлення даних користувача. Для зміни групи (/group) UserService знаходить користувача за id, встановлює нове значення groupCode та зберігає зміни (userRepository.save(user)), після чого код академічної групи для користувача змінює значення.

Отримання даних користувача. Для перевірки реєстрації або отримання group_code використовується userRepository.findById(userId), що повертає Optional<User> для безпечної обробки випадків, коли користувач не знайдений.

3.6.2 Операції з даними груп (group_data)

Збереження/оновлення довідкової інформації (ref_info). Адміністратор чату групи може використати команду /ref_info_edit. UserService (getReferenceInfoForChat) знаходить запис за його Telegram ID. Та оновлює його новою інформацією вказаним адміном. Збереження group_code для групового чату аналогічно до ref_info, оновлюється поле groupCode.

3.6.3 Операції зі списком заблокованих користувачів (banned_user)

Додавання користувача до списку заблокованих: команда /ban Telegram ID. UserService створює екземпляр BannedUser і зберігає його в БД. Перевірка блокування перед обробкою запиту. Видалення користувача зі списку відбувається за допогою команди /unban Telegram ID.

Структура бази даних описана в таблиці 3.1.

Таблиця 3.1 – Структура таблиць бази даних

Таблиця БД	Поля	Тип даних	Обмеження	Опис поля
banned_user	id	bigint	PK, NOT NULL	userId
	username	text	NOT NULL	username
group_data	id	bigint	PK, NOT NULL	chatId
	groupname	text	NOT NULL	Назва групи
	created	timestamp	NOT NULL	Час запису
	group_code	text	NULLABLE	Код групи
	ref_info	text	NULLABLE	Довідка

Продовження таблиці 3.1

Таблиця БД	Поля	Тип даних	Обмеження	Опис поля
user_data	id	bigint	PK, NOT NULL	userId
	username	text	NULLABLE	username
	created	timestamp	NOT NULL	Час запису
	group_code	text	NULLABLE	Код групи

Отже, бачимо, що система має структуровану та пропрацьовану БД, котра підходить для поточного проєкту.

3.7 Алгоритм управління станом користувача (UserState)

Клас UserState (стани перераховані в enum) у пакеті com.infonure.infonure_bot.model потрібен для використання механізму станів для реалізації більш складних, багатоетапних діалогів з користувачем. Стан зберігається в оперативній пам'яті у HashMap, де ключем є userId, а значенням об'єкт UserState. Цей підхід простий в реалізації, але стан втрачається при перезапуску бота. Але він потрібен лише в короткий проміжок часу, після обробки команди необхідність в ньому зникає, тому - збереження та постійне звернення до БД для отримання станів користувачів наразі важається не оптимальним варіантом. Приклад діалогу з використанням станів:

- користувач надсилає команду /timetable;
- бот переводить користувача в певний стан і зберігає цей стан;
- бот надсилає поступово користувачеві повідомлення з введенням дати;
- наступне текстове повідомлення від цього користувача

інтерпретується не як нова команда, а як введення дати;

- після отримання та обробки назви групи, стан користувача повертається до дефолтного або переходить до наступного етапу діалогу.

3.8 Структури даних

Програмний комплекс оперує різними структурами даних для представлення інформації та управління внутрішньою логікою. Основні класи моделей з пакету com.infonure.infonure_bot.model слугують для представлення сутностей даних у програмі та тісно пов'язані зі структурою таблиць у базі даних PostgreSQL:

- User. представляє користувача Telegram. Містить поля, такі як id (ідентифікатор користувача), username (@username користувача), created (час створення запису), groupCode (код обраної академічної групи);
- GroupData. представляє інформацію про групові чати. Включає поля id (ідентифікатор запису або чату групи), groupname (назва групи), created (час створення запису), groupCode (код групи), ref_info (довідкова текстова інформація);
- BannedUser. представляє заблокованого користувача. Містить id та username заблокованого користувача (якщо немає, зберігається символ «-»).

Для кожної таблиці в базі даних створено відповідну модель. Анотації JPA вказують, як саме поля відповідають стовпцям таблиці. UserService, який ми розглядали раніше, використовує Spring Data репозиторії для маніпуляції цими об'єктами, а Spring автоматично перетворює ці операції на SQL-запити. Реалізація за допомогою Spring Data JPA дозволяє абстрагуватися від написання ручних SQL-запитів і забезпечує чіткий зв'язок між структурою бази даних та об'єктною моделлю в Java. Для представлення даних розкладу використовуються класи з пакету com.infonure.infonure_bot.parser:

- Lesson: описує одне конкретне заняття;
- TimeSlot: описує один часовий слот (пару);

- DaySchedule: описує розклад на день;
- Schedule: корінь всієї моделі, агрегує повний розклад. Повертає один великий, гарно відформатований рядок String;
- UserState. описує перерахування (enum) можливих станів діалогу з ботом;
- стандартні колекції, такі як List, Map та Set.

Для систематизації інформації про ключові об'єкти даних, з якими оперує система, наведено таблицю 3.2.

Таблиця 3.2 – Основні сутності даних (моделі) та їх призначення

Клас моделі	Основні атрибути	Призначення/Опис
User	id, username, groupCode, created	Дані про користувача
GroupData	id, groupname, created, groupCode, ref_info	Дані про групові чати
BannedUser	id, username	Заблоковані користувачі.
UserState	enum	Опис доступних станів
Schedule	Колекція DaySchedule	Агрегує розклад
DaySchedule	Дата/день, колекція LessonInfo	Розклад на день
LessonInfo	Предмет, тип, час тощот	Пара в розкладі.
TimeSlot	Час початку та закінчення	Часовий інтервал заняття
Стандартні	List, Map, Set	Списки, словники тощо

Отже, бачимо що проєкт оперує різними класами моделей для роботи, але також використовує стандартні колекції.

4 РОЗРОБКА ТА ВІДЛАГОДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Опис основних компонентів системи та їх взаємозв'язків

Структура проєкту описана в таблиці 4.1, дозволяє виділити ключові компоненти (пакети) та визначити їх функціональне призначення.

Таблиця 4.1 – Ключові компоненти (пакети) системи та їх функціональне призначення

Компонент	Основні класи	Відповідальність
com.infonure.infonure_bot	InfoNureBotApplication	Головний клас
com.infonure.infonure_bot.config	BotInitializer	Ініціалізація бота
com.infonure.infonure_bot.controller	InfoNureBot	Отримує Update
com.infonure.infonure_bot.handler	UpdateDispatcher	Диспетчеризація вхідних оновлень
com.infonure.infonure_bot.model	User, GroupData, BannedUser, UserState	Класи сутностей даних
com.infonure.infonure_bot.parser	HtmlScheduleParser, Schedule і т.д.	Парсинг розкладу
com.infonure.infonure_bot.repository	UserRepository, GroupDataRepository	Інтерфейси для взаємодії з БД

Продовження таблиці 4.1

com.infonure.infonure_bot.service	ScheduleService, UserService	Бізнес-логіка
com.infonure.infonure_bot.view	KeyboardFactory, MessageFactory	Створення представлення
resources	application.properties, timetable.html	Ресурси застосунку

Дана таблиця дає опис структури проєкта, з чого саме складається проект та яка його частина за що відповідає.

4.2 Програмна реалізація

В даному розділі розглянуто безпосередньо програмну реалізацію додатку, його класи, методи їх принцип роботи тощо.

4.2.1 Конфігурація, ініціалізація та запуск бота

InfoNureBotApplication – точка входу. Це найголовніший клас у будь-якому Spring Boot додатку. Цей клас є точкою входу, з якої починається виконання всієї програми. Ключові елементи:

- @SpringBootApplication: це анотація для позначення головного класу додатку. Вона позначає клас, що відповідає за завантаження (bootstrapping) та налаштування програми;
- @Configuration: позначає клас як джерело конфігурації;
- @EnableAutoConfiguration: вмикає механізм автоконфігурації Spring Boot, який намагається автоматично налаштувати проєкт на основі підключених залежностей;

- @ComponentScan: вказує Spring сканувати поточний пакет (com.infonure.infonure_bot) та всі його підпакети на наявність компонентів (@Component, @Service, @Repository і т.д.), щоб створити їх та керувати;

- main(String[] args): стандартний метод запуску Java-програми. Виклик SpringApplication.run(...) фактично запускає весь фреймворк Spring, який, у свою чергу, створює та з'єднує всі частини бота.

BotInitializer – ініціалізація бота. Цей компонент відповідає за те, щоб запустити та зареєструвати сесію бота після того, як весь додаток Spring буде повністю налаштований. Ключові елементи:

- @EventListener({ContextRefreshedEvent.class}): це найважливіша частина. Ця анотація змушує метод init() виконатися тільки тоді, коли Spring завершить ініціалізацію всіх своїх компонентів і опублікує подію ContextRefreshedEvent. Це гарантує, що екземпляр бота (InfoNureBot) вже повністю створений і готовий до роботи;

- init(): у цьому методі створюється об'єкт TelegramBotsApi, який відповідає за керування сесіями ботів, і викликається telegramBotsApi.registerBot(bot). Саме цей виклик вмикає бота, і він починає отримувати оновлення від Telegram.

InfoNureBot – Головний клас бота. Клас, що безпосередньо взаємодіє з бібліотекою TelegramBots. Приймати оновлення від Telegram, делегувати їх обробку та відправляти відповіді назад. Ключові елементи:

- extends TelegramLongPollingBot: успадкування від цього класу надає всю базову логіку для отримання оновлень за допомогою механізму "довгих запитів" (Long Polling);

- конструктор: приймає UpdateDispatcher як залежність, а також токен, ім'я та ID адміністраторів з файлу конфігурації через анотацію @Value. Токен передається в конструктор батьківського класу super(botToken);

- onUpdateReceived(Update update): це головний метод-обробник. Коли Telegram надсилає оновлення (нове повідомлення, натискання кнопки), цей метод викликається. Принцип єдиного обов'язку: цей метод не містить

жодної бізнес-логіки. Він негайно передає об'єкт update в updateDispatcher.handleUpdate(update). Це чудовий приклад хорошої архітектури, де головний клас залишається чистим і лише делегує роботу. Після отримання списку відповідей від диспетчера він у циклі відправляє кожну з них за допомогою методу execute(response);

- registerBotMenu(): метод з анотацією @PostConstruct, який виконується один раз після створення класу. Він відповідає за реєстрацію списку команд (/start, /group і т.д.) у Telegram. Саме завдяки цьому користувачі бачать меню команд у чаті з ботом;

- isChatAdmin(Long chatId, Long userId): допоміжний метод, який використовується диспетчером для перевірки прав доступу.

Загальна послідовність запуску:

- запускається InfoNureBotApplication;

- Spring Boot сканує проєкт і створює всі компоненти (InfoNureBot, UpdateDispatcher, UserService, ScheduleService та інші);

- під час створення InfoNureBot спрацьовує @PostConstruct і реєструється меню команд в Telegram (registerBotMenu);

- коли всі компоненти готові, Spring публікує подію ContextRefreshedEvent;

- BotInitializer перехоплює цю подію і реєструє бота в Telegram API (registerBot);

- з цього моменту бот готовий приймати оновлення. Кожне оновлення потрапляє в onUpdateReceived, який передає його в UpdateDispatcher, запускаючи вже знайомий нам ланцюжок обробки.

4.2.2 Диспетчер

Цей клас є центральним компонентом (диспетчером), який відповідає за обробку всіх оновлень (updates), що надходять до Telegram-бота. Він вирішує, що робити з кожним повідомленням, натисканням кнопки чи

командою від користувача. У конструкторі він отримує кілька залежностей, які вказують на розподіл обов'язків:

- UserService: відповідає за роботу з користувачами та чатами (реєстрація, збереження групи, бани);
- ScheduleService: відповідає за отримання та обробку розкладу занять;
- KeyboardFactory: створює різні клавіатури для повідомлень;
- MessageFactory: створює готові до відправки об'єкти повідомлень;
- InfoNureBot: посилання на головний клас бота. @Lazy анотація використовується для уникнення циклічної залежності при ініціалізації.

Клас також має внутрішні Map для керування станом:

- userStates: ключова частина. Зберігає поточний стан для кожного користувача (наприклад, AWAITING_GROUP_NAME - бот очікує, що наступне повідомлення буде назвою групи). Це реалізація патерну "Машина станів";
- userSelectedStartDate, awaitingRefInfoForChatId та інші: тимчасові сховища даних для багатоетапних операцій (наприклад, збереження початкової дати при запиті розкладу на період).

Покроковий потік роботи. Точка входу: handleUpdate(Update update). Це головний публічний метод. Коли бот отримує будь-яке оновлення (нове повідомлення, натискання кнопки), цей метод викликається.

Створення списку відповідей: List<BotApiMethod<?>, готується список, куди будуть додаватися всі дії, які бот повинен виконати у відповідь (надіслати повідомлення, відредагувати і т.д.).

Обробка помилок: якщо виникає будь-яка помилка, вона логується, а користувачу надсилається повідомлення про помилку.

Розподіл за типом оновлення:

- якщо це повідомлення (update.hasMessage()): користувач (userService.regUser) та чат (userService.regChat) реєструються в базі даних. Це відбувається при кожному повідомленні, що дозволяє оновлювати дані (наприклад, username);

- перевірка бану: перевіряється, чи не забанений користувач (isEntityBanned(userId)) або чат (isEntityBanned(chatId)). Якщо так, метод просто завершує роботу, ігноруючи повідомлення.

Делегування: обробка передається далі до приватного методу handleIncomingMessage. Якщо це натискання кнопки (update.hasCallbackQuery()): обробка передається до методу handleCallbackQuery.

Обробка повідомлень: handleIncomingMessage(). Цей метод аналізує вміст повідомлення. Розпізнавання команди:

- якщо текст починається з / (text.startsWith("/")), він вважається командою;

- обробка команд у групах: код коректно обробляє команди формату /command@BotName. Він перевіряє, чи команда адресована саме цьому боту, і відсікає @BotName, щоб далі працювати з "чистою" командою;

- скидання стану: перед виконанням будь-якої нової команди (крім /cancel), стан користувача скидається. Це гарантує, що якщо користувач почав вводити групу, а потім викликав /start, його попередній незавершений діалог буде скасовано.

Виклик обробника команди: робота передається в handleCommand.Цей метод простий маршрутизатор, який на основі тексту команди (наприклад, /start) викликає відповідний метод-обробник. Для прикладу наведемо опис деяких команд:

- /start: надсилає вітальне повідомлення;

- /adt (advertisement): команда для адміна бота для масової розсилки оголошення всім користувачам та групам.

Обробка тексту відповідно до стану. Якщо текст не є командою, код перевіряє поточний стан користувача (currentState) за допомогою switch. Приклад: якщо стан UserState.AWAITING_GROUP_NAME, то текст, надісланий користувачем, передається в метод handleGroupNameInput, який розглядає його як назву групи. Аналогічно для інших станів:

AWAITING_START_DATE (очікування початкової дати розкладу), AWAITING_REPORT (очікування тексту скарги) і т.д.

Обробка вводу даних (методи handle...Input). Ці методи виконують логіку після того, як бот отримав дані, які очікував, також пов’язаний зі станами. Наведемо приклад: handleStartDateInput та handleEndDateInput. Реалізують двоетапний запит розкладу. handleStartDateInput перевіряє коректність формату дати, зберігає її в userSelectedStartDate і переводить користувача в стан AWAITING_END_DATE. handleEndDateInput отримує кінцеву дату, перевіряє її, бере початкову дату з userSelectedStartDate, запитує розклад у scheduleService і відправляє його користувачу. Після цього очищує стан і тимчасові дані.

Обробка натискання кнопок: handleCallbackQuery(CallbackQuery callbackQuery, ...). Цей метод спрацьовує, коли користувач натискає на кнопку під повідомленням (inline keyboard). Він отримує дані, зашиті в кнопку (callbackQuery.getData()). Це рядок, наприклад, TIMETABLE_TODAY або GROUP_INPUT_CANCEL.

Якщо дані починаються з TIMETABLE_, викликається handleTimetableInput, який обробляє запит на розклад (на сьогодні, завтра і т.д.). Якщо дані закінчуються на _CANCEL, дія скасовується, стан користувача очищується.

4.2.3 Сервіси

ScheduleService та UserService. Ці два класи є "сервісним шаром" додатку. Вони інкапсулюють основну бізнес-логіку:

- UserService відповідає за всі операції, пов'язані з даними користувачів, чатів та банів, взаємодіючи з базою даних;

- ScheduleService відповідає за завантаження, кешування та надання доступу до даних розкладу.

Диспетчер (UpdateDispatcher), який ми аналізували раніше,

використовує ці сервіси для виконання конкретних завдань.

Детальний аналіз ScheduleService. Цей сервіс повністю відповідає за роботу з файлом розкладу. Його головна мета забезпечити швидкий доступ до розкладу, мінімізуючи операції читання та розбору (парсингу) файлу. Також він робить кешування. Принцип роботи:

- при старті програми (@PostConstruct) метод init() викликає loadAndParseSchedule();

- цей метод один раз завантажує весь HTML-файл розкладу, розбирає його за допомогою HtmlScheduleParser і зберігає результат у пам'яті в полі scheduleCache;

- завдяки цьому, коли користувач запитує розклад, програмі не потрібно кожного разу читати та аналізувати великий HTML-файл. Вона просто звертається до вже готових даних у кеші, що є дуже швидким.

Покроковий аналіз методів:

а) init() та loadAndParseSchedule():

1) анотація @PostConstruct гарантує, що метод init виконається одразу після створення сервісу;

2) loadAndParseSchedule зчитує файл, шлях до якого вказано в конфігурації (timetable.filepath), з ресурсів проєкту (ClassPathResource);

3) він передає потік даних файлу в parser.parse(), який виконує всю "брудну" роботу з розбору HTML;

4) результат парсингу зберігається в scheduleCache;

5) метод має надійну обробку помилок: якщо файл не знайдено або його не вдалося розпарсити, у кеш записується об'єкт з повідомленням про помилку. Це запобігає "падінню" всього бота при запуску;

б) робота методу getScheduleForDateRange(...):

1) це головний метод, який використовує UpdateDispatcher для отримання розкладу;

2) перш за все, він перевіряє, чи кеш валідний. Якщо під час запуску сталася помилка, він робить ще одну спробу завантажити розклад,

викликавши loadAndParseSchedule(). Це механізм самовідновлення;

3) якщо все добре, він просто делегує запит об'єкту scheduleCache, викликаючи його власний метод getScheduleForDateRange. Це приклад хорошого дизайну: сервіс відповідає за завантаження та кешування, а сам об'єкт Schedule знає, як фільтрувати та форматувати свої дані;

в) робота методу getAllAvailableGroups():

1) цей метод надає набір всіх назв груп, які були знайдені у файлі розкладу;

2) UpdateDispatcher використовує цей метод для перевірки, чи є введена користувачем група правильною;

г) робота методу refreshSchedule(). Цей публічний метод дозволяє примусово оновити кеш, повторно завантаживши та розпарсивши файл.

Детальний аналіз UserService. Цей сервіс є посередником між логікою бота та базою даних. Він керує сутностями: User (користувач), GroupData (дані чату) та BannedUser (забанені). Анотація @Transactional над методами забезпечує цілісність даних (операція або виконується повністю, або не виконується взагалі). Основні функції та методи:

а) реєстрація та оновлення даних:

1) regUser(Long id, String username): при кожному повідомленні від користувача цей метод перевіряє, чи є він у базі. якщо так, він оновлює username (тільки якщо той змінився). Якщо ні, створює нового користувача;

2) regChat(Long chatId, String chatTitle): аналогічна логіка для групових чатів. Оновлює назву чату, якщо вона змінилася;

б) робота з групами:

1) setUserGroup(Long userId, String groupCode): встановлює персональну академічну групу для користувача;

2) setAcademicGroupForChat(...): встановлює академічну групу для всього чату;

3) getUserGroup(...) та getAcademicGroupForChat(...): отримують відповідні коди груп;

в) довідкова інформація (ref_info). Методи setReferenceInfoForChat(...) та getReferenceInfoForChat(...) дозволяють адміністраторам чату зберігати та отримувати довільну текстову інформацію (наприклад, корисні посилання), прив'язану до чату;

г) функціонал бану:

1) isEntityBanned(Long entityId): проста та швидка перевірка, чи існує запис про бан в базі даних;

2) banEntity(Long targetId): блокує користувача або чат. Цікава деталь: перед баном сервіс намагається знайти ім'я користувача (username) або назву чату (groupname), щоб зберегти їх разом з ID. Це дуже зручно для адміна, який бачитиме не просто голий ID, а й ім'я того, кого заблокували;

3) unbanEntity(Long targetId): видаляє запис із таблиці забанених;

ґ) отримання даних для розсилок. Методи getAllUserIds() та getAllGroupChatIdsWithAcademicGroup(): ці методи повертають списки ID всіх користувачів або чатів. Вони використовуються в UpdateDispatcher для команди /adt, щоб знати, кому саме надсилати оголошення.

4.2.4 Представлення

KeyboardFactory та MessageFactory. Ці два класи складають так званий "шар представлення" (View) у структурі бота. Вони відповідають за те, як виглядають повідомлення та елементи керування, які бачить кінцевий користувач.

KeyboardFactory. Цей клас, як і слідує з назви, є "фабрикою" для створення об'єктів клавіатур, які прикріплюються до повідомлень. Це дозволяє тримати логіку створення складних клавіатур в одному місці.

getTimetableOptionsKeyboard(). Цей метод створює та повертає готову клавіатуру (InlineKeyboardMarkup) з опціями для отримання розкладу. Клавіатура складається з трьох рядків кнопок. Перший ряд: "Сьогодні" та "Завтра". Другий ряд: "Поточний тиждень" та "Наступний тиждень". Третій

ряд: "Обрати діапазон дат". Ключова деталь (CallbackData): Для кожної кнопки встановлюється не тільки текст, який бачить користувач (setText), але і приховані дані для бота setCallbackData. Наприклад, для кнопки "Сьогодні" це дані «TIMETABLE_TODAY». Коли користувач натискає кнопку, бот отримує саме ці дані, і UpdateDispatcher розуміє, яку дію потрібно виконати.

getCancelKeyboard(String callbackDataPrefix). Це універсальний метод, який створює просту клавіатуру з однією кнопкою "Скасувати". Він приймає префікс для callbackData. Це дозволяє використовувати одну й ту ж фабрику для скасування різних дій. Якщо викликати getCancelKeyboard(«GROUP_INPUT»), дані кнопки будуть «GROUP_INPUT_CANCEL» і т.д. Такий підхід дозволяє UpdateDispatcher точно знати, який саме процес скасовує користувач.

MessageFactory.Ця фабрика є інструментом для створення об'єктів повідомлень. Вона вирішує кілька типових проблем: спрощує конструювання повідомлень, обробляє їх, перевіряє та оборобляє повідомлення перевищують ліміт Telegram, та допомагає уникнути помилок форматування.

Спрощення створення повідомлень (перевантажені методи createMessage). Клас надає кілька однойменних методів createMessage з різними параметрами для зручності. Це дозволяє UpdateDispatcher викликати потрібну версію, не створюючи об'єкт SendMessage вручну. Можна створити:

- просте текстове повідомлення;
- повідомлення з прикріпленою клавіатурою;
- повідомлення з форматуванням.

Всі ці методи використовують приватний метод buildMessage, що застосовує патерн "Будівельник" (SendMessage.builder()) для конструювання фінального об'єкта.

Обробка довгих повідомлень (createLongMessage та splitMessageText)

- Telegram має ліміт на довжину одного повідомлення (тут встановлено MAX_MESSAGE_LENGTH = 4000 символів). Якщо текст розкладу дуже великий, спроба відправити його одним повідомленням

призведе до помилки. Метод createLongMessage вирішує цю проблему. Він використовує утилітарний метод splitMessageText, щоб "нарізати" один великий текст на список менших частин, кожна з яких не перевищує ліміт. Потім він створює список об'єктів SendMessage по одному на кожну частину тексту. UpdateDispatcher отримує цей список і може надіслати повідомлення послідовно;

- метод splitMessageText розбиває текст спочатку по символу нового рядка (\n), а потім по пробілах, щоб не розривати слова посередині.

Безпечне форматування Markdown. Markdown-розмітка в Telegram вимагає, щоб символи форматування (наприклад, * для жирного тексту або _ для курсиву) були парними. Якщо в тексті зустрічається непарна кількість таких символів, Telegram видасть помилку. Метод sanitizeMarkdown аналізує текст. Якщо він знаходить непарну кількість спецсимволів, він автоматично "екранує" їх усі, додаючи перед ними зворотний слеш (\). Це гарантує, що повідомлення завжди буде відправлено коректно, навіть якщо вихідний текст розкладу містить "випадкові" символи форматування.

4.2.5 UserState та responses

Призначення та принцип роботи responses. В класі UpdateDispatcher, змінна responses (List<BotApiMethod<?>> responses = new ArrayList<>();) відіграє ключову роль у підготовці відповіді або серії відповідей бота користувачеві.

Кожна дія, яку бот повинен виконати у відповідь на запит користувача (надіслати повідомлення, надіслати клавіатуру тощо), представлена в бібліотеці TelegramBots об'єктом, що успадковує BotApiMethod. Наприклад, для надсилання текстового повідомлення використовується SendMessage. Метод handleUpdate у UpdateDispatcher (та методи, які він викликає, як-от handleIncomingMessage, handleCallbackQuery, handleCommand та інші обробники) аналізує запит користувача і формує один або декілька таких

об'єктів BotApiMethod, які додаються до списку responses. Принцип роботи:

- клас InfoNureBot у методі onUpdateReceived(Update update) отримує оновлення від Telegram;
- це оновлення передається до updateDispatcher.handleUpdate(update);
- UpdateDispatcher та його внутрішні методи визначають, які дії потрібно виконати. Замість негайного надсилання кожного повідомлення, що ускладнило б код і тестування, вони створюють відповідні об'єкти (SendMessage тощо) і додають їх до списку responses. Наприклад, якщо потрібно надіслати довгий розклад, який розбивається на частини, буде створено декілька об'єктів SendMessage;
- після того, як updateDispatcher.handleUpdate(update) завершив роботу і повернув список responses, InfoNureBot у циклі проходить по цьому списку і виконує кожен BotApiMethod за допомогою методу execute(response).

Переваги такого підходу:

- розподіл логіки: логіка формування відповіді відокремлена від логіки її фактичного надсилання. UpdateDispatcher вирішує, *що* сказати, а InfoNureBot – *як* це надіслати;
- пакетна обробка: дозволяє підготувати кілька дій (наприклад, кілька повідомлень) і повернути їх для послідовного виконання;
- тестованість: логіку в UpdateDispatcher легше тестувати, оскільки можна перевірити, які BotApiMethod він згенерував, не виконуючи реальних викликів до Telegram API.

Роль та механізм роботи UserState.UserState (перелічуваний тип enum) та Map<Long, UserState> userStates (де ключ – це ID користувача) є основою для керування станом діалогу з кожним користувачем. Це дозволяє боту розуміти контекст та обробляти багатоетапні взаємодії.

Щоб бот "пам'ятав" попередній крок діалогу з конкретним користувачем, необхідно зберігати цей стан. UserState визначає, якого типу інформацію бот очікує від користувача в його наступному повідомленні. Принцип роботи діапазону дат для /timetable:

- користувач обирає опцію "Обрати діапазон дат";

- бот надсилає "Введіть початкову дату..." і встановлює стан userStates.put(userId, UserState.AWAITING_START_DATE);

- користувач вводить дату. handleTextMessage через switch викликає handleStartDateInput;

- handleStartDateInput обробляє дату, надсилає "Тепер введіть кінцеву дату..." і змінює стан на userStates.put(userId, UserState.AWAITING_END_DATE);

- користувач вводить другу дату, handleIncomingMessage викликає handleEndDateInput, який обробляє запит, показує розклад і скидає стан.

4.2.6 Розклад

Головний клас парсера (HtmlScheduleParser) – це основний клас, відповідальний за перетворення HTML-файлу на структурований об'єкт Java. Його призначення це прочитати HTML-файл розкладу та агрегувати його вміст по об'єктах Schedule, DaySchedule і т.д. Бібліотека Jsoup дозволяє завантажувати HTML, знаходити потрібні елементи за допомогою CSS-селекторів та отримувати з них дані.

Метод parse(...) – це головний метод класу. Він приймає на вхід InputStream HTML-файлу і виконує наступні кроки:

а) завантаження HTML. Змінна doc: Jsoup.parse(htmlInputStream, "windows-1251", "") завантажує HTML-потік. Кодування "windows-1251" вказано явно, оскільки воно прописане у <meta>-тегу файлу. Jsoup перетворює HTML-текст на об'єкт Document;

б) ініціалізація результату:

1) змінна schedule: створюється порожній об'єкт Schedule. До нього вносяться всі виявлені дані;

2) пошук та ітерація по елементах;

3) назва групи: doc.selectFirst("table.header td") знаходить перший

<td> всередині таблиці з класом header і витягує з нього текст (наприклад, "КІУКІ-21-2");

4) основна таблиця: doc.selectFirst("table.MainTT") знаходить головну таблицю з розкладом;

5) рядки таблиці (<tr>): mainTable.select("> tbody > tr") отримує список всіх рядків (<tr>) всередині <tbody> головної таблиці;

в) цикл по рядках: далі парсер проходиться по кожному рядку в циклі for (Element row : rows) і визначає, який це тип рядка:

1) рядок з номерами тижнів. Перевірка: row.selectFirst("td.week") != null. Якщо в рядку є комірка (<td>) з класом week, це заголовок з тижнями. Парсер проходить по всіх таких комірках, витягує з них текст (1, 2, 3, ...), ігноруючи нечислові значення ("№", "Тижні"), та зберігає їх у списку weekNumbersHeader;

2) рядок з днем тижня і датами. Перевірка: row.selectFirst("td.date[colspan=2]") != null. Якщо в рядку є комірка з класом date і атрибутом colspan="2", це назва дня тижня. Змінна currentDaySchedule: створюється новий об'єкт DaySchedule, куди зберігається назва дня ("Понеділок"). Парсер знаходить у цьому ж рядку інші комірки з класом date (вже без colspan) і витягує з них дати. Кожна дата співставляється з номером тижня з weekNumbersHeader за індексом і зберігається у мапі datesByWeek об'єкта currentDaySchedule;

3) рядок з парою (заняттям). Перевірка: row.select("td.left").size() >= 2. Якщо в рядку є щонайменше дві комірки з класом left, це рядок із заняттям. Змінні lessonNumber, timeRange: з перших двох комірок td.left витягуються номер пари і час. Змінна currentTimeSlot: створюється об'єкт TimeSlot для цієї пари. Цикл по комірках занять: парсер проходить по всіх наступних комірках <td> у цьому рядку. Позиція кожної комірки (її індекс i) відповідає позиції тижня у заголовку weekNumbersHeader. Для кожної такої комірки він перевіряє, чи є всередині неї посилання <a> з класом linktt. Якщо посилання є: це означає, що на цьому тижні є заняття. Парсер витягує назву

предмету і тип (*КЗЗОТІ Лб), формат (DL), створює об'єкт Lesson і додає його в currentTimeSlot, прив'язуючи до відповідного номера тижня. Якщо посилання немає: заняття на цьому тижні немає. В currentTimeSlot для цього тижня додається null.

Результат: метод parse повертає повністю заповнений об'єкт schedule з усією структурою розкладу.

Класи моделі даних (пакет parser). Це прості Java-класи (POJO), що описують структуру розкладу:

- Lesson: описує одне конкретне заняття. Поля: subject (назва предмету), type (Лк, Пз, Лб), format (DL);

- TimeSlot: описує один часовий слот (пару). Поля: lessonNumber, timeRange і найважливіше Map<String, Lesson> lessonsByWeek. Це мапа (словник), де ключ це номер тижня (рядок, наприклад "5"), а значення це об'єкт Lesson, що проходить у цей час на цьому тижні;

- DaySchedule: описує розклад на цілий день. Поля: dayName (назва дня), Map<String, String> datesByWeek (мапа, що зв'язує номер тижня з конкретною датою для цього дня), і List<TimeSlot> (список всіх пар на цей день);

- Schedule: корінь всієї моделі, містить повний розклад. Поля: groupName, List<String> weekNumbers (список усіх номерів тижнів), List<DaySchedule> (список усіх днів з їх розкладами).

Метод getScheduleForDateRange(…). Формує фінальне текстове повідомлення для користувача. Алгоритм його роботи:

- перетворює вхідні рядки з датами (startDateStr, endDateStr) на об'єкти LocalDate для зручного порівняння. Одразу ж перевіряє їх на коректність;

- проходить у циклі по кожному DaySchedule (по кожному дню тижня);

- всередині кожного дня проходить по його мапі datesByWeek;

- для кожної дати з мапи перевіряє, чи входить вона в заданий користувачем діапазон;

- якщо дата входить у діапазон, то додає до фінального рядка назву дня і цю дату (напр., «--- Понеділок (03.03.2025) ---»);

- проходить по всіх TimeSlot (парах) цього дня;

- для кожної пари знаходить конкретне заняття (Lesson) для поточного номера тижня;

- якщо заняття є, додає інформацію про нього (номер, час, назву) до фінального рядка.

Результат: повертає один великий, гарно відформатований рядок String.

Сервісний Шар (ScheduleService). Цей клас є "мозком" для управління даними розкладу. Його призначення: виклик завантаження та парсингу файлу. Забезпечити кешування даних. Опис:

- змінна scheduleCache: це поле, де зберігається об'єкт Schedule після першого парсингу. Це кешування. Завдяки цьому ми не читаємо і не парсимо HTML-файл при кожному запиті користувача. Файл обробляється лише один раз при старті додатку;

- метод init() з анотацією @PostConstruct: Spring викликає цей метод автоматично одразу після створення біна ScheduleService. Він знаходить файл timetable.html у ресурсах проєкту, викликає HtmlScheduleParser.parse() і зберігає результат у scheduleCache;

- метод getScheduleForDateRange(...): це публічний метод, який викликає бот. Він не виконує жодного парсингу, а просто передає запит з датами до вже існуючого об'єкта scheduleCache і повертає готовий результат;

- метод refreshSchedule(): додаткова функція для примусового оновлення кешу (повторного парсингу файлу) через введення команди в чаті з ботом.

4.3 Тестування та відлагодження

В даному розділі розглянуто програмну тестування та відлагодження додатку. Це дозволяє детально проаналізувати код і з'ясувати проблеми та їх

вирішення, що забезпечує стабільну роботу додатку в реальних умовах.

4.3.1 Ручне тестування функціональних модулів

Ручне тестування було основним методом перевірки функціональності на всіх етапах розробки. Воно включало перевірку кожного реалізованого компонента та функції бота шляхом безпосередньої взаємодії з ним через клієнт Telegram. Робилися наступні дії:

а) перевірка коректності парсингу розкладу:

1) використовувався актуальний файл timetable.html, а також спеціально підготовлені тестові HTML-файли, що імітували різні ситуації: розклад з пропущеними парами, відсутність даних про викладача чи аудиторію, розклад для груп з різною кількістю занять, порожні дні;

2) ретельно перевірялася відповідність вилучених даних тим, що містилися в HTML-файлі;

3) особлива увага приділялася обробці крайових випадків, таких як розклад на перший та останній день тижня, наявність "вікон" між парами;

б) тестування обробки команд користувача та відповідей бота:

1) систематично перевірялися всі реалізовані команди: /start, /help, /timetable, /group, /set_chat_group, /ref_info_edit, /ref_info та інші;

2) команди тестувалися як з коректними аргументами, так і з навмисно не коректними або відсутніми аргументами для перевірки логіки обробки помилок;

3) тестувалася робота inline-клавіатур (наприклад, вибір дня тижня для розкладу) та коректність обробки відповідних callback-запитів;

4) оцінювалася зрозумілість, повнота та грамотність текстових відповідей бота;

5) перевірялася логіка роботи системи станів (UserState) для команд, що вимагають багатоетапного вводу даних від користувача;

в) тестування операцій CRUD з базою даних:

1) через команди бота перевірялися операції створення, читання, оновлення та видалення записів у таблицях user_data (реєстрація, зміна групи), group_data (встановлення довідкової інформації, коду групи для чату), banned_user (блокування/розблокування);

2) для верифікації результатів операцій використовувався клієнт PostgreSQL (pgAdmin), що дозволяло безпосередньо переглядати та аналізувати стан даних у таблицях після виконання відповідних команд ботом;

ґ) тестування функцій адміністрування:

1) перевірялася можливість блокування користувачів за ID командою /ban та їх подальшого розблокування командою /unban. Тестувалося, що заблокований користувач дійсно не може взаємодіяти з ботом;

2) тестувалася функція масової розсилки повідомлень командою /adt: перевірялася доставка повідомлень усім користувачам та коректність вмісту розісланого повідомлення;

3) особлива увага приділялася перевірці механізму обмеження доступу до адміністративних команд: спроби виконати ці команди звичайним користувачем блокувалися.

4.3.2 Тестування шляхом створення нетипових та стресових ситуацій

Метою цього тестування була перевірка стійкості бота до неочікуваних дій користувача, некоректних даних та інших умов: передача некоректних даних (відсутній аргумент, орфографічна помилка, не той тип даних для команди) або спроби неавторизованого доступу до адміністративних функцій.

Приклади тестів для нетипових ситуацій наведені в таблиці 4.2.

Таблиця 4.2 – Тести для нетипових ситуацій

Тип	Вхідні дані	Відповідь
Команда з помилкою	/timable -> КІУКІ-21-7	Невідома команда
Відсутній аргумент	/group -> к21	Групу не знайдено
Не авторизований доступ	Звичайний користувач надсилає /ban	Ця команда доступна адміністраторам бота.
Введення даних не того типу	Адміністратор бота вводить /ban Test	Бот повідомляє про некоректний формат

На рисунку 4.1 зображено реакцію бота на помилки.

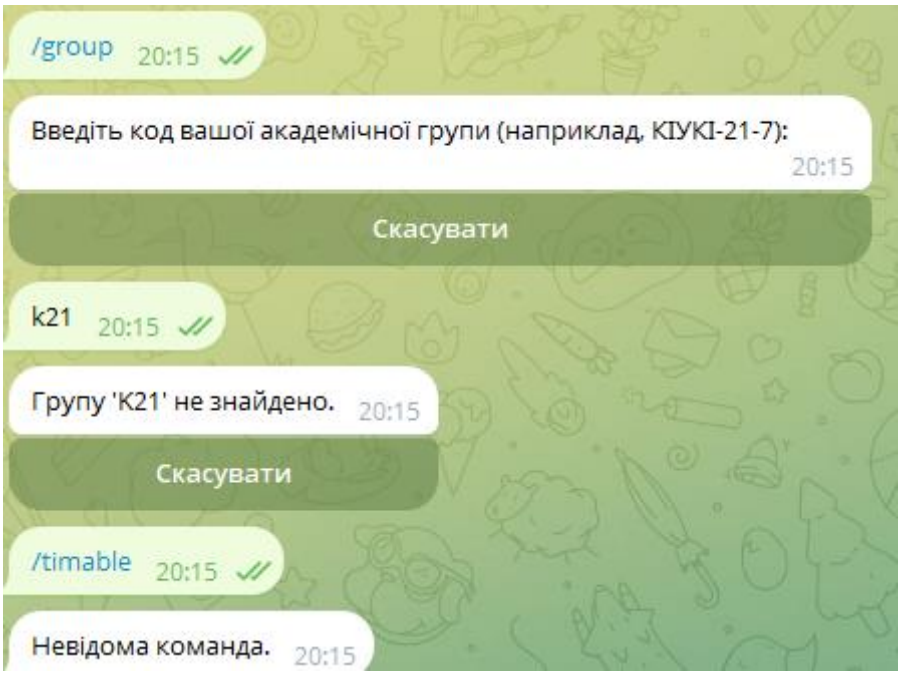


Рисунок 4.1 – Відповідь бота на неправильні команди

Отже, можна побачити роботу над помилками, бот оброблює їх та продовжує свою роботу не роблячи аварійної зупинки, що говорить про готовність програми до повноцінної роботи.

4.3.3 Тестування за участі кінцевих користувачів (збір зворотного зв'язку)

На завершальних етапах розробки було організовано тестування бота невеликою групою потенційних кінцевих користувачів. Метою такого тестування було:

- отримати відгуки щодо зручності використання (usability) інтерфейсу бота, зрозумілості команд та відповідей;
- виявити неочевидні помилки в логіці роботи або функціональні недоліки, які могли бути непомітні для розробника;
- зібрати пропозиції щодо покращення функціоналу або додавання нових можливостей.

Користувачам було запропоновано вільно взаємодіяти з ботом, виконуючи типові завдання: отримання розкладу, налаштування своєї групи, отримання довідкової інформації. Зворотний зв'язок збирався через особисте спілкування або через репорти, логи (рисунок 4.2).

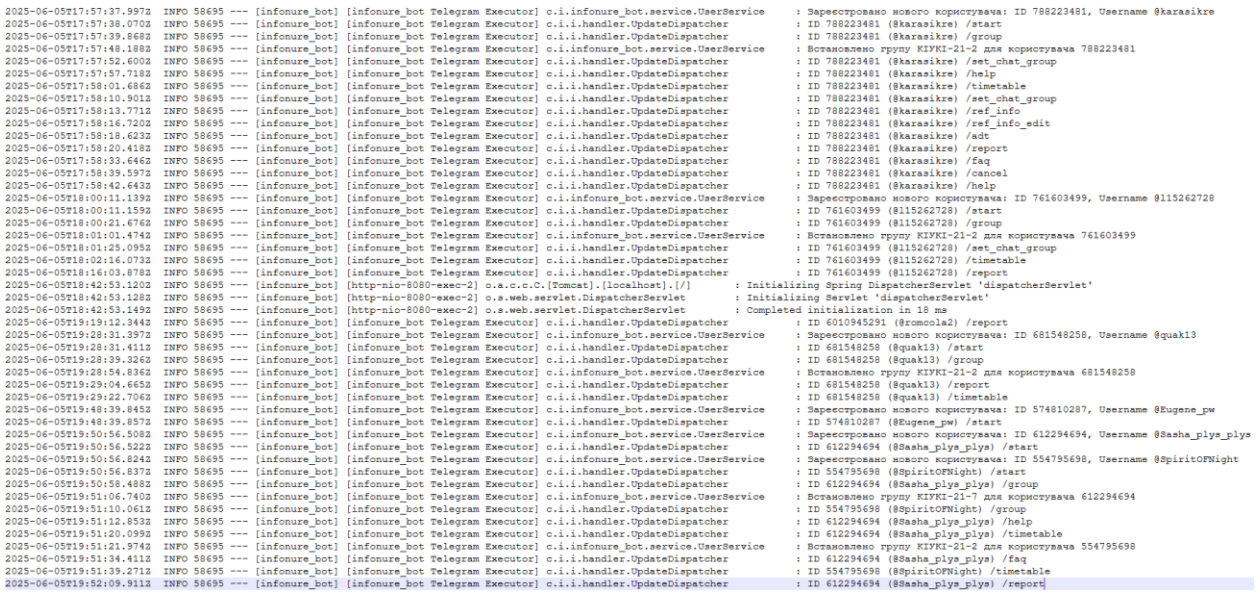


Рисунок 4.2 – Запис логів про активність користувачів

На рисунку зображено запис в лог-файлі додатка про активність користувачів, їх дані, час та тип активності.

4.3.4 Інструменти та підходи до відлагодження

У процесі розробки та тестування активно використовувалися інструменти та підходи для ефективного виявлення та усунення помилок (відлагодження):

а) логування. Було інтегровано стандартний для Spring Boot фреймворк логування SLF4J з реалізацією Logback. У ключових точках коду (обробка команд, взаємодія з сервісами, парсинг, робота з БД) були додані лог-повідомлення з різними рівнями деталізації:

1) INFO: для запису основних подій (наприклад, старт бота, виконання користувачем команди, успішне завершення операції);

2) WARN: для фіксації некритичних помилок або потенційно проблемних ситуацій (наприклад, не вдалося розпарсити частину HTML, але робота продовжується);

3) ERROR: для запису критичних помилок, що призвели до неможливості виконати операцію (наприклад, помилка підключення до БД, неперехоплений виняток у логіці);

б) вбудований дебагер IntelliJ IDEA:

1) активно використовувався режим відлагодження IDE для покрокового виконання коду;

2) встановлювалися точки зупину (breakpoints) у місцях передбачуваних помилок або для аналізу логіки роботи складних ділянок;

3) під час зупинки на breakpoint переглядалися значення локальних змінних, полів об'єктів, результати виконання виразів;

4) аналізувався стек викликів (call stack) для розуміння послідовності виклику методів та контексту виникнення помилки.

Такий підхід не тільки прискорює процес розробки, але й сприяє підвищенню загальної якості та надійності програмного коду, забезпечуючи стабільну та ефективну роботу програмних продуктів.

4.3.5 Аналіз результатів тестування та основні виявлені недоліки

Проведене комплексне тестування дозволило виявити та усунути низку недоліків, а також підтвердити працездатність основного функціоналу бота. Виявлені типові помилки, включали:

- помилки в логіці парсингу: некоректне вилучення даних з timetable.html при наявності непередбачених варіацій у структурі HTML (наприклад, об'єднані комірки, відсутність деяких атрибутів). Це вимагало уточнення CSS-селекторів та додавання більш гнучких перевірок у HtmlScheduleParser;

- неправильна обробка аргументів команд: проблеми з парсингом аргументів команд, особливо якщо вони містили пробіли або спеціальні символи. Було вдосконалено логіку розділення команди та її аргументів;

- проблеми з управлінням станами користувача: у деяких сценаріях стан користувача не скидався на DEFAULT після завершення багатоетапного діалогу, що призводило до невірної інтерпретації наступних команд, що було виправлено;

- NullPointerExceptions: виникали при спробі доступу до даних, які могли бути відсутні (наприклад, username користувача, якщо він прихований, або дані розкладу для неіснуючої групи). Що теж було виправлено;

На основі результатів тестування та зворотного зв'язку від користувачів було внесено такі зміни та виправлення:

- покращено стійкість парсера до незначних змін у HTML-структурі;
- виправлено логіку роботи з командами, що мають опціональні аргументи;
- посилено перевірку прав доступу до чутливих функцій;
- оптимізовано деякі запити до бази даних для підвищення швидкодії.

ВИСНОВКИ

У процесі виконання даної кваліфікаційній роботі було успішно розроблено та реалізовано Telegram-бот для автоматизації інформаційного обслуговування студентів. Проведений аналіз специфічних інформаційних потреб студентів у сучасному освітньому середовищі, а також доступних технологій розробки та інструментальних засобів дозволив чітко окреслити вимоги до функціоналу та обґрунтувати доцільність створення такого програмного рішення. Це підґрунтя стало основою для проектування та створення додатка, забезпечуючи системі гнучкість, надійність та потенціал для подальшого масштабування.

Протягом всіх етапів програмної реалізації, від детального проектування архітектури до написання коду та тестування, було забезпечено повну відповідність розробленого рішення технічному завданню. Вибір технологічного стеку, що охоплює платформу Telegram, мову програмування Java, фреймворк Spring Boot та СУБД PostgreSQL, виявився повністю обґрунтованим і сприявши ефективному досягненню поставлених цілей. Telegram, як обрана платформа, забезпечив зручну та широкодоступну взаємодію з користувачами. Водночас, використання Java в поєднанні з фреймворком Spring Boot та Telegram Bot API надало надійну та масштабовану основу для розробки, дозволивши ефективно реалізувати логіку та інтеграцію. СУБД PostgreSQL, своєю чергою, продемонструвала високу надійність, продуктивність та гнучкість у забезпеченні зберігання та ефективного управління даними, що було критично важливим для коректної роботи всіх функцій бота.

Практичне застосування розробленого програмного продукту підтвердило його стабільну та передбачувану роботу. Бот коректно обробляє команди користувачів, забезпечуючи оперативний доступ до актуального розкладу занять для обраних груп. Реалізовано функціонал масових

оголошень, що дозволяє адміністраторам ефективно та швидко доносити важливу інформацію, таку як новини, зміни в розкладі чи екстрені повідомлення тощо. Крім того, бот надає можливість створювати та відображати різноманітну довідкову інформацію, включаючи відповіді на поширені запитання та дані про налаштування бота, що значно спрощує самостійний пошук інформації студентами. Також бот має можливість працювати як особисто користувачами, так і в групових чатах. Усі виявлені під час розробки та тестування недоліки були своєчасно локалізовані та усунені.

Підтвердженням високого рівня готовності програмного забезпечення до повноцінної експлуатації стало його успішне розгортання на VPS за допомогою сервісу interserver.net. Стабільна робота сервера, відсутність затримок та збоїв позитивно вплинули на загальну надійність функціонування системи. Отриманий у результаті функціонуючий Telegram-бот повністю вирішує поставлені завдання та готовий до використання. У перспективі передбачається подальший розвиток програмного продукту, що може включати розширення функціоналу, наприклад, інтеграцію з іншими університетськими системами або можливість роботи через веб-інтерфейс. Розглядається також можливість локалізації інтерфейсу для іноземних студентів та масштабування рішення для використання в інших освітніх закладах, що сприятиме покращенню комунікації та цифровізації освітнього процесу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. К. В. Гораш, "Складові інформаційного забезпечення науково-дослідницької діяльності студентів вищих навчальних закладів," *Освіта: стратегія, політика, право*, вип. 2(41), 2020. URL:https://www.narodnaosvita.kiev.ua/?page_id=2179. (дата звернення: 28.05.2025).

2. "Проблеми сучасного студента," *Коло - новини Полтави*. URL:https://kolo.news/category/suspilstvo/979 (дата звернення: 28.05.2025).

3. Feng J, Yu B, Tan WH, Dai Z, Li Z. Key factors influencing educational technology adoption in higher education: A systematic review. PLOS Digit Health. 2025 Apr 29;4(4):e0000764. doi: 10.1371/journal.pdig.0000764. PMID: 40299977; PMCID: PMC12040101.

4. Z. Gilani, S. Al-Harfoushi, S. Al-Aufi, A. Al-Barwani, та H. Al-Sinawi, "Student communications: a review of current practices and scoping a new vision," *Tertiary Education and Management*, с. 1-18, 2024. doi: 10.1080/13603108.2024.2449395.

5. "Укладено рейтинг інформативності сайтів ВНЗ," *Освіта.UA*. URL:https://osvita.ua/vnz/rating/44894/ (дата звернення: 29.05.2025).

6. О. В. Дубів, "Рейтинги Інтернет-присутності ВНЗ та шляхи зміцнення позицій у них на прикладі ДВНЗ «УжНУ»," препринт, Ужгородський національний університет. URL: Дубів_О_В_Веб_рейтинги_ВНЗ_препринт.pdf (дата звернення: 29.05.2025).

7. "Вебінари у вищій освіті та їх використання," *MyOwnConference*. URL:https://myownconference.com/blog/uk/vebinary-u-vyshchii-osviti-ta-yikh-vykorystannia/ (дата звернення: 29.05.2025).

8. "14 додатків, які полегшать навчання в університеті," *Mudra.ua*. URL:https://mudra.ua/ua/articles/14-dodatkv-yak-polegshat-navchannya-v-unversitet/. (дата звернення: 29.05.2025).

9. Краснопольський, В. Е., Поліщук, О. А., & Демченко, О. М. (2024). Інтеграція мобільних додатків у освітній процес: аналіз ефективності та можливостей для здобувачів освіти. https://doi.org/10.5281/zenodo.11559587.

10. "Використання чат-ботів у освіті, створення чат-бота для освітнього закладу," *Gerabot*. URL:https://gerabot.com/article/vikoristannya_chatbotiv_u_osviti (дата звернення: 30.05.2025).

11. Peyton, K., Unnikrishnan, S. & Mulligan, B. A review of university chatbots for student support: FAQs and beyond. *Discov Educ* **4**, 21 (2025). https://doi.org/10.1007/s44217-025-00397-7

12. "Розклад занять завжди в телефоні: університет запустив на Telegram чат-бот для студентів," *Національний університет «Полтавська політехніка імені Юрія Кондратюка»*, 2020. URL:https://nupp.edu.ua/news/rozklad-zanyat-zavzhdi-v-telefoni-universitet-zapustiv-na-telegram-chat-bot-dlya-studentiv.html (дата звернення: 30.05.2025)..

13. "Українські боти для роботи та відпочинку," *Чорноморський національний університет імені Петра Могили*. URL:https://chmnu.edu.ua/ukrayinski-boti-dlya-roboti-ta-vidpochinku/ (дата звернення: 30.05.2025).

14. Baskara, F. R. (2023). Chatbots and Flipped Learning: Enhancing Student Engagement and Learning Outcomes through Personalised Support and Collaboration. *IJORER : International Journal of Recent Educational Research*, *4*(2), 223-238. https://doi.org/10.46245/ijorer.v4i2.331

15. ЖИХОРСЬКА, О. (2024). ВИКОРИСТАННЯ ЧАТ-БОТІВ У НАВЧАННІ СТУДЕНТІВ. *Вісник Київського національного університету імені Тараса Шевченка. Серія «Педагогіка»*, *2*(18). https://doi.org/10.17721/2415-3699.2023.18.06.

16. "«Опора»: найпопулярнішими соцмережами в Україні є телеграм, ютуб та фейсбук," *MediaSapiens*, 2024. URL:https://ms.detector.media/sotsmerezhi/post/35557/2024-07-16-opora-

naypopulyarnishymy-sotsmerezhamy-v-ukraini-ie-telegram-yutub-ta-feysbuk/ (дата звернення: 31.05.2025).

17. Nataliia Morze, Oksana Buinytska, Liliia Varchenko-Trotsenko, Use of Bot-Technologies for Educational Communication at the University, in: E-learning, Vol. 9, Effective Development of Teachers' Skills in the Area of ICT and E-learning, Scientific Editor Eugenia Smyrnova-Trybulska, Studio Noa for University of Silesia, Katowice-Cieszyn 2017, p. 239-248.

18. "Telegram Bot API Essential Guide," *Rollout, 2024*. URL:https://rollout.com/integration-guides/telegram-bot-api/api-essentials (дата звернення: 01.06.2025).

19. "Bot API," *grammY*, 2024. URL:https://grammy.dev/uk/guide/api (дата звернення: 01.06.2025).

20. "Telegram Bot API," *Telegram APIs*. URL:https://core.telegram.org/bots/api (дата звернення: 01.06.2025).

21. "pengrad/java-telegram-bot-api," *GitHub*. URL:https://github.com/pengrad/java-telegram-bot-api (дата звернення: 02.06.2025).

22. "Як змінилося медіаспоживання в Україні у 2024?," *Gradus Research*, 2024. URL:https://gradus.app/uk/open-reports/changes-media-consumption-ukraine-2024/ (дата звернення: 02.06.2025).

23. "Java Software," *Oracle*. URL:https://www.oracle.com/java/ (дата звернення: 02.06.2025).

24. "Overview of Java," *Oracle Help Center*. URL:https://docs.oracle.com/en/database/oracle/oracle-database/18/jjdev/Java-overview.html (дата звернення: 03.06.2025).

25. "Python vs Java: A Deep Dive into the Best Programming Language for You," *Dev.to*, 2023. URL:https://dev.to/respect17/python-vs-java-a-deep-dive-into-the-best-programming-language-for-you-3k6o (дата звернення: 03.06.2025).

26. "Spring Boot," *Wikipedia, the free encyclopedia*. URL:https://en.wikipedia.org/wiki/Spring_Boot (дата звернення: 04.06.2025).

27. "Переваги та недоліки використання Spring Boot. Функції для рядків у Java," *JavaRush, 2023*. URL:https://javarush.com/ua/groups/posts/uk.3380.kava-breyk-75-perevagi-ta-nedolki-vikoristannja-spring-boot-funkc-dlja-rjadkv-u-java (дата звернення: 04.06.2025).

28. Дерев'янчук, В.А. Інформаційна система підтримки прийняття рішень молодшого командного складу з використанням технології чат ботів [Текст]: робота на здобуття кваліфікаційного ступеня бакалавра; спец.: 122 - комп`ютерні науки (інформатика) / В.А. Дерев'янчук; наук. керівник С.О. Петров. - Суми: СумДУ, 2021. - 56 с.

29. "Spring Framework," *Wikipedia, the free encyclopedia*. URL:https://en.wikipedia.org/wiki/Spring_Framework (дата звернення: 05.06.2025).

30. "Creating a Telegram Bot with Spring Boot," *Baeldung*, 2025. URL:https://www.baeldung.com/spring-boot-telegram-bot (дата звернення: 05.06.2025).

31. "About PostgreSQL," *PostgreSQL*. URL:https://www.postgresql.org/about/ (дата звернення: 06.06.2025).

32. Salunke, S.V.; Ouda, A. A Performance Benchmark for the PostgreSQL and MySQL Databases. Future Internet 2024, 16, 382. https://doi.org/10.3390/fi16100382

33. "PostgreSQL чи MySQL. Як обрати оптимальну базу даних для вашого проєкту," *DOU*, 2024. URL:https://dou.ua/forums/topic/51621/ (дата звернення: 06.06.2025).

34. "PostgreSQL vs MySQL - Difference Between Relational Database Management Systems (RDBMS)," *Amazon Web Services*. URL:https://aws.amazon.com/compare/the-difference-between-mysql-vs-postgresql/ (дата звернення: 06.06.2025).

35. P. Rungta. Uncovering Secrets of the Maven Repository (2023). http://resolver.tudelft.nl/uuid:b945ed27-1ad2-4815-aac1-ee5276977192