ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Цифрова система планування та автоматизації нагадувань на базі Telegram-бота»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Технології цифрового розвитку»

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

_____ Олександр ЯРЕМЕНКО
(підпис)

Виконав: здобувач(ка) вищої освіти групи ТЦР-43

Олександр ЯРЕМЕНКО

Керівник: Олексій АНАНЧЕНКО
доц. каф. к.т.н.

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Технологій цифрового розвитку

Ступінь вищої освіти Бакалавр

Спеціальність Інженерія програмного забезпечення

Освітньо-професійна програма Технології цифрового розвитку

ЗАТВЕРДЖУЮ
Завідувач кафедрою ЖЕБКА В.В_____
_____Ім'я, ПРІЗВИЩЕ
«____» _____20___р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Яременко Олександру Андрійович

1. Тема кваліфікаційної роботи: «Цифрова система автоматизації нагадувань на базі Telegram-бота"»
керівник кваліфікаційної роботи доц. каф. кан. технічних наук Олексій АНАНЧЕНКО,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року №51.

2. Строк подання кваліфікаційної роботи «18» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: мова програмування Java, фреймворк Spring Boot, Telegram Bot API, СУБД PostgreSQL та H2, технології Spring Data JPA, Hibernate, засоби обробки природної мови та алгоритми автоматичного визначення часу й пріоритету завдань.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Які існуючі системи керування завданнями та автоматизації нагадувань використовуються сьогодні, які їх переваги, недоліки та які технології доцільно використати для розробки Telegram-бота?.
2. Яким чином спроектувати архітектуру цифрової системи автоматизації нагадувань, базу даних та алгоритми обробки повідомлень користувача?

3. Програмна реалізація та опис функціонування застосунку для
4. автоматизованої побудови тематичного словника навчальної дисципліни "Робототехніка".
5. Тестування застосунку.
6. Перелік ілюстративного матеріалу: презентація
   1.Актуальність теми
   2.Діаграма діяльності системи
   3.Мета завдання та роботи
   4.Вимоги до системи
   5.Аналіз конкурентів
   6.Унікальність системи
   7.Архітектура системи
   8.Технологічний стек
   9.Принцип роботи системи
   10.Демонстрація роботи бота
   11.Апробація результатів дослідження
   12.Висновок

7. Дата видачі завдання «20» лютого 2026р

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вибір та затвердження теми кваліфікаційної роботи	20.03.2026	Виконано
2	Аналіз предметної області та наукових джерел	12.04.2026 – 14.04.2026	Виконано
3	Формування вимог та проектування архітектури системи	15.04.2026 – 19.04.2026	Виконано
4	Проектування бази даних та інтерфейсу Telegram-бота	20.04.2026 – 02.05.2026	Виконано
5	Реалізація програмної системи та модулів обробки даних	03.05.2026 – 28.05.2026	Виконано
6	Проведення тестування та аналіз результатів роботи системи	05.05.2026 – 05.06.2026	Виконано
7	Оформлення пояснювальної записки та презентації	10.07.2026 – 12.07.2026	Виконано
8	Подання кваліфікаційної роботи на кафедру та підготовка до захисту	2026 р.	В роботі

Здобувач(ка) вищої освіти _____ Олександр Яременко
(підпис)

Керівник
кваліфікаційної роботи _____ Олексій Ананченко
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 52 стор., 3 табл. 4 рис., 15 джерел.

Мета роботи – підвищення ефективності управління персональними завданнями та нагадуваннями через застосування Telegram-бота, оснащеного інтелектуальним механізмом обробки текстових повідомлень користувача.

Об’єкт дослідження – процес управління завданнями та нагадуваннями у цифровому середовищі.

Предмет дослідження – методи та засоби реалізації системи планування задач із використанням Telegram-бота.

Короткий зміст роботи – у кваліфікаційній роботі проведено аналіз сучасних систем управління завданнями та визначено їх основні переваги й недоліки. Обґрунтовано доцільність використання Telegram-бота як платформи для створення системи автоматизації нагадувань. Розроблено архітектуру програмної системи на основі Java Spring Boot, спроектовано структуру бази даних та реалізовано механізми створення, редагування і видалення задач. Реалізовано модулі автоматичного визначення часу виконання завдань та оцінки їх пріоритетності на основі текстового аналізу повідомлень користувача. Проведено тестування системи та проаналізовано результати її функціонування. Практична цінність роботи полягає у створенні зручного цифрового інструменту для автоматизації планування та нагадувань.

КЛЮЧОВІ СЛОВА: TELEGRAM-БОТ, АВТОМАТИЗАЦІЯ НАГАДУВАНЬ, СИСТЕМА ПЛАНУВАННЯ, JAVA, SPRING BOOT, POSTGRESQL, ОБРОБКА ПРИРОДНОЇ МОВИ, БАЗА ДАНИХ, TIMEPARSER, PRIORITYPARSER.

ЗМІСТ

ВСТУП

У сучасному цифровому світі управління часом є одним із ключових факторів, що впливає на продуктивність людини. Завдяки розвитку мобільних додатків і месенджерів стало можливим автоматизувати планування завдань і встановлення нагадувань, значно полегшуючи щоденну організацію діяльності користувачів.

Серед найпопулярніших інструментів для цифрової комунікації сьогодні виділяється месенджер Telegram, який підтримує використання ботів для автоматизованої взаємодії. Telegram-боти пропонують широкий спектр функцій — від надання простих інформаційних сервісів до комплексних систем управління даними та персонального планування.

Тема набуває актуальності у зв’язку зі зростанням потреби в швидких, зручних і простих у використанні системах управління завданнями. Надзвичайно важливим є той факт, що використання Telegram як платформи забезпечує доступ до системи з будь-якого пристрою та дозволяє працювати у знайомому середовищі.

Основною метою дослідження є створення цифрової системи планування та автоматизації нагадувань на основі Telegram-бота з використанням методів інтелектуальної обробки текстових повідомлень користувачів.

Для реалізації цієї мети визначено кілька ключових завдань:

- провести аналіз існуючих систем управління завданнями й нагадуваннями;
- вивчити потенціал Telegram-ботів у контексті організації планування;
- розробити архітектуру програмного рішення;
- створити Telegram-бота з функціями додавання, редагування й видалення завдань;
- інтегрувати механізми автоматичного визначення часу виконання та пріоритетності задач;
- спроектувати структуру бази даних для зберігання інформації;
- здійснити тестування системи й оцінити її ефективність.

Об’єктом дослідження обрано процес управління цифровими завданнями й нагадуваннями.

Предметом дослідження є методи та засоби реалізації системи планування задач із використанням Telegram-бота, зокрема технології обробки природної мови, програмні засоби розробки клієнт-серверних застосунків, механізми автоматичного визначення часових параметрів і пріоритетності завдань, а також інструменти зберігання та обробки даних.

Дослідження базується на використанні системного аналізу, об’єктно-орієнтованого проектування, моделювання клієнт-серверних архітектур та інструментів аналізу текстових даних, а також на застосуванні принципів програмної інженерії для розробки й тестування програмного забезпечення.

Наукову новизну роботи становить інтеграція механізмів обробки природної мови із функціоналом Telegram-бота для визначення часових характеристик і пріоритетності задач, що дозволяє реалізовувати персональні системи планування в автоматизованому режимі.

Практичне значення дослідження полягає у створенні ефективного програмного рішення, яке суттєво спрощує процес управління завданнями, скорочує час їх введення та підвищує загальну ефективність тайм-менеджменту. Дана система може слугувати основою для розробки цифрових асистентів і корпоративних платформ планування завдань.

Апробація результатів дослідження. Основні результати кваліфікаційної роботи були представлені в доповіді «Інтелектуальна система планування та автоматизації нагадувань на базі Telegram-бота». У межах апробації було висвітлено результати аналізу сучасних систем планування завдань, особливості розробленої архітектури програмного забезпечення, реалізацію модулів автоматичного визначення часових параметрів та пріоритетності завдань, а також результати тестування розробленої системи. Отримані результати підтвердили ефективність використання Telegram-бота як інструменту автоматизації процесів планування та управління персональними завданнями.

1. Яременко О.А., Ананченко О.Є. Використання Telegram-ботів для автоматизації персонального планування та нагадувань // Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. К.: ДУІКТ, 2026.

2. Яременко О.А., Ананченко О.Є. Автоматичне визначення пріоритетності задач на основі аналізу текстових повідомлень користувача // Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. К.: ДУІКТ, 2026.

Результати дослідження мають значний потенціал практичного впровадження в галузях інформаційних технологій, автоматизованих систем, цифрових сервісів персонального планування, а також платформ для організації робочого часу.

Практична цінність роботи полягає в оптимізації часу користувачів на адміністрування задач через автоматичний аналіз тексту, інтеграцію системи з Telegram та централізоване зберігання даних у базі.

Ефективність розробленої системи була випробувана шляхом тестування її функціональних можливостей у реальних умовах використання: створення, редагування й обробка завдань відбувалися в автоматичному режимі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд та порівняння засобів для керування завданнями

Через прискорений ритм життя та інформаційне перенавантаження вміння ефективно планувати свої справи стало нагальною потребою. Сьогодні людям потрібні цифрові помічники, які допомагають структурувати завдання та стежити за виконанням у встановлені терміни. Система планування - це програмне рішення (веб застосунок, мобільний додаток або чат-бот), що дає змогу вести списки справ. Базова функціональність таких систем передбачає можливість створення та редагування нотаток, встановлення дедлайнів і нагадувань, упорядкування справ за рівнем пріоритетності, а також відстеження прогресу виконання завдань. Сучасний ринок робить ставку на максимальну зручність, тобто користувачі більше віддають перевагу сервісам з інтуїтивно зрозумілим інтерфейсом і швидким додаванням інформації [10, 11]. Для більш детального порівняння найбільш популярних цифрових сервісів планування доцільно розглянути їх основні переваги та недоліки, що наведені в таблиці 1.1.

Таблиця 1.1

Огляд засобів для керування завданнями

Система	Переваги	Недоліки
Todoist	Потужний функціонал	Складність використання
Notion	Гнучкість та універсальність	Складний старт
Google Calendar	Зручний календар	Обмежена робота із задачами
Microsoft To Do	Простий у використанні	Обмежений функціонал

Проведений аналіз демонструє, що наразі на ринку представлено багато рішень для тайм-менеджменту, кожне з яких опирається на власну методологію організації роботи. Серед найпопулярніших сервісів - Todoist, Notion, Google Calendar і Microsoft To Do.

Todoist пропонує професійний підхід із розгалуженою ієрархією (проєкти, підзадачі) та системою міток. Його можливості легко розширюються за рахунок інтеграцій, однак велика кількість функцій може ускладнити швидке опанування інструменту.

Notion будується на ідеї «все в одному» - це багатофункціональне середовище, де списки справ поєднано з базами даних і документами. Головний мінус – високий поріг входження: користувачеві доводиться витрачати чимало часу на створення зручної для себе структури.

Google Calendar зосереджується на візуалізації часу через тайм-блокінг. Він незамінний для планування зустрічей і подій, але програє спеціалізованим менеджерам завдань у гнучкості роботи з текстовими списками поза календарем.

Microsoft To Do – мінімалістичний додаток, який прийшов на зміну Wunderlist. Попри зрозумілий інтерфейс і базовий набір функцій, він часто не виправдовує очікувань досвідчених користувачів через обмежені засоби автоматизації та аналітики.

Отже, наявні продукти страждають на певний дисбаланс: вони або перенасичені можливостями, що гальмує швидку роботу, або надто примітивні, що звужує простір для налаштувань під власні потреби [10, 11].

1.2 Telegram-боти як середовище для впровадження систем планування

Використання месенджерів як фронтенд-платформи для систем керування завданнями стає одним із найактуальніших трендів. Завдяки Telegram-ботам можна швидко розгортати інтерактивні сервіси, що працюють за принципом «тонкого клієнта»: користувачеві не потрібно встановлювати додаткове програмне забезпечення, оскільки вся логіка обробки даних знаходиться на сервері.

Telegram як платформа для створення системи планування має низку важливих переваг. Насамперед месенджер забезпечує кросплатформеність, що дозволяє користувачам отримувати доступ до списку завдань та синхронізувати дані з будь-якого пристрою, на якому встановлено Telegram. Взаємодія через

звичайний текстовий чат значно спрощує процес використання системи та скорочує час на ознайомлення з її функціоналом. Додатковою перевагою є зручний інтерфейс, оскільки вбудовані елементи керування, зокрема Inline Keyboards та Reply Buttons, забезпечують швидку навігацію між розділами та зміну статусів завдань. До того ж, Telegram вирізняється технологічною гнучкістю, що проявляється у простоті інтеграції із зовнішніми базами даних та API через серверні запити. У такому середовищі бот перетворюється на повноцінного цифрового помічника, здатного чітко обробляти неструктуровані повідомлення й перетворювати їх на систематизовані записи в базі завдань [1, 15]. Основні переваги використання Telegram як платформи для реалізації системи планування та автоматизації нагадувань наведено в таблиці 1.2.

Таблиця 1.2

Переваги використання Telegram як платформи для системи планування

Характеристика	Практичне значення для системи
Кросплатформеність	Доступ до задач з будь-якого пристрою
Простота взаємодії	Робота через звичайний чат
Інтерактивність	Inline-кнопки та меню
API Telegram	Інтеграція із серверною частиною
Швидкість повідомлень	Оперативна доставка нагадувань
Безпека	Захищена передача даних

Отже, використання Telegram як основної платформи для системи планування забезпечує поєднання зручності, швидкості та функціональності взаємодії з користувачем.

1.3 Ключові виклики та обмеження сучасних планувальників

Аналіз існуючих рішень дозволив визначити основні фактори, які ускладнюють швидке та зручне управління завданнями. Однією з ключових проблем є висока трудомісткість введення даних, оскільки ручне налаштування кожної задачі потребує значних витрат часу. Перешкодою також виступає складність освоєння сучасних сервісів, адже інтерфейси багатьох популярних застосунків є надто перевантаженими та незрозумілими для пересічного користувача. Суттєвим недоліком є недостатній рівень автоматизації, що

проявляється у відсутності механізмів текстового аналізу для автоматичного визначення термінів виконання та пріоритетності завдань. Окрім цього, проблемою залишається фрагментація інструментів, коли користувачеві необхідно використовувати окремі застосунки для роботи із задачами замість взаємодії у звичному середовищі, зокрема в месенджері.

Наявність цих проблем створює передумови для розробки гнучкого інструменту, що забезпечить мінімальне тертя між виникненням думки про задачу та її фіксацією в системі.

Проведений аналіз показав, що наявні засоби планування завдань мають як сильні, так і слабкі сторони. Головна їхня вада - ускладнена взаємодія з користувачем і відсутність механізмів інтелектуального опрацювання текстових даних.

Найперспективнішим напрямком видається створення такої системи, яка б давала змогу вводити задачі у вільному текстовому вигляді, автоматично визначаючи їхні параметри. А використання Telegram-бота як інтерфейсу забезпечує зручність, простоту та доступність для кожного користувача [10].

1.4 Опис ІТ-засобів

Для забезпечення високої надійності, швидкодії та масштабованості розробленої системи було проведено аналіз сучасних інструментів розробки. У результаті було сформовано стек технологій, який є індустріальним стандартом для створення складних серверних рішень. Основні технології, використані під час розробки системи, наведено в таблиці 1.3.

Таблиця 1.3

Технологічний стек розробленої системи

Технологія	Призначення	Причина вибору
Java	Серверна логіка	Надійність та кросплатформеність
Spring Boot	Backend framework	Швидка розробка та масштабованість
PostgreSQL	Основна СУБД	Надійне зберігання даних
H2 Database	Тестова БД	Швидке прототипування
Telegram Bot API	Взаємодія з Telegram	Робота з повідомленнями
Spring Data JPA	Робота з БД	ORM та автоматизація SQL
Hibernate	ORM	Управління сутностями

Фундаментом програмного забезпечення стала мова Java [2]. Вибір саме цієї мови зумовлений її архітектурними особливостями та багаторічною історією успішного застосування в корпоративному секторі.

Завдяки застосуванню Java Virtual Machine (JVM), систему можна запускати на будь-якій операційній системі без необхідності змінювати вихідний код. Цей механізм дозволяє виявляти потенційні помилки ще на етапі компіляції, що критично важливо для стабільної обробки даних користувачів. Наявність величезної кількості стандартних та сторонніх бібліотек дозволяє ефективно вирішувати задачі будь-якої складності - від мережевої взаємодії до обробки складних структур даних.

Як основний каркас системи використано фреймворк Spring Boot. Він є еволюційним розвитком екосистеми Spring і дозволяє значно прискорити процес розробки завдяки концепції автоматичної конфігурації [3].

Використання цього модуля дозволило абстрагуватися від написання низькорівневих SQL-запитів, натомість зосередившись на роботі з об’єктами, що

значно підвищує швидкість розробки [3, 13]. Впровадження залежностей забезпечує модульність архітектури, дозволяючи легко замінювати компоненти системи без ризику порушити її цілісність., а наявність вбудованого сервера застосунків (Tomcat) спрощує процес розгортання та моніторингу системи.

Для забезпечення цілісності та довготривалого зберігання інформації було обрано реляційну модель даних.

H2 Database використовувалася на етапі розробки та прототипування як легка In-Memory база даних, що дозволяє миттєво тестувати функціонал.

PostgreSQL була обрана як основна СУБД для промислової експлуатації через її високу надійність, підтримку складних транзакцій та здатність стабільно працювати під високим навантаженням [5].

Вибір месенджера Telegram як клієнтської платформи дозволив вирішити питання доступності та кросплатформеності інтерфейсу [1].

Використання Telegram дозволяє уникнути додаткових витрат на розробку UI/UX, оскільки створення окремого мобільного застосунку або вебсайту потребувало б значних ресурсів, тоді як месенджер уже забезпечує готове та звичне для користувача середовище.

API Telegram забезпечує високий рівень безпеки завдяки надійному шифруванню даних, а також гарантує швидку доставку повідомлень, що є критично важливим для стабільної роботи системи нагадувань.

Підтримка Inline-клавіатур та інтерактивних меню дала змогу реалізувати складну логіку взаємодії з користувачем у форматі зручного та інтуїтивно зрозумілого діалогового вікна.

Використаний технологічний стек утворює цілісну екосистему, яка гарантує стабільне функціонування системи планування задач та створює фундамент для її подальшої інтеграції з іншими цифровими сервісами.

1.5 Постановка задачі

Головна концепція розробленого рішення полягає у створенні гнучкого цифрового інструментарію, який би максимально спрощував процес фіксації та систематизації особистих завдань. Для досягнення цілей у системі було впроваджено широкий спектр функцій, що сприяють спрощенню управління завданнями та автоматизації рутинних процесів для користувачів. Система розроблена таким чином, щоб дозволити вводити задачі у форматі природної мови, не вимагаючи суворого дотримання синтаксичних норм. Це дає можливість швидко фіксувати справи навіть у мобільному режимі. Один із ключових елементів системи - механізм інтелектуального оброблення тексту, який автоматично визначає часові маркери і переводить нечіткі формулювання, наприклад «після обіду» чи «через тиждень», у конкретні дати й час. Для покращення процесу планування впроваджено функцію автоматичного визначення пріоритетів завдань на основі аналізу ключових слів. Це дозволяє сортувати задачі за терміновістю без необхідності додаткових дій з боку користувача. Іншим важливим компонентом стала розумна система нагадувань, яка забезпечує своєчасну доставку повідомлень через месенджер, допомагаючи уникнути пропуску важливих подій. Додатково система оснащена механізмом оптимізації текстового контенту: після визначення часу й пріоритетів завдання автоматично видаляються зайві слова, формуючи лаконічний і зрозумілий список справ. Запроваджені функціональні рішення створили програмний продукт, який не тільки зберігає дані про задачі, але й здійснює їхню інтелектуальну обробку. Це забезпечує підвищення ефективності особистого планування та управління часом.

2 ПРОЕКТУВАННЯ СИСТЕМИ

2.1 Деталізований опис загальної архітектури та структурної організації системи

Проектування сучасної інтелектуальної системи для керування завданнями неможливе без чітко продуманої архітектури, яка забезпечувала б баланс між продуктивністю, надійністю та зручністю подальшого розширення. У цій роботі за основу взято багаторівневу клієнт-серверну модель [9]. Такий підхід вирізняється високим ступенем абстракції: логіка роботи програми, спосіб подання даних користувачеві та механізми зберігання інформації можуть розвиватися незалежно одне від одного [8]. Завдяки цьому зміни в одній частині системи не призводять до збоїв в інших, що критично важливо для довготривалих проєктів. У рамках даної розробки архітектурне рішення свідомо поділено на два великі сегменти: фронтенд, який реалізовано на базі екосистеми месенджера Telegram, та бекенд, що працює під керуванням потужного фреймворку Spring Boot [3]. Розглянемо кожну зі складових докладніше.

Клієнтський рівень та інтерфейс взаємодії у традиційних додатках для планування справ (наприклад, мобільних чи вебзастосунках) розробникам доводиться створювати окремі інтерфейси для iOS, Android, Windows тощо. Це не лише збільшує час і вартість розробки, але й ускладнює синхронізацію даних між різними пристроями. У нашому проекті роль універсального клієнта виконує Telegram - популярний месенджер із відкритим API. Таке рішення забезпечує миттєву синхронізацію даних між усіма пристроями користувача, оскільки після додавання або зміни завдання його актуальний стан одразу відображається на смартфоні, планшеті чи комп’ютері, де встановлено Telegram [1]. Додатковою перевагою є використання вбудованих механізмів захисту месенджера, зокрема шифрування каналів зв’язку та наскрізного шифрування для секретних чатів, що

усуває необхідність реалізації власних криптографічних засобів на рівні бота. Крім того, таке архітектурне рішення дозволяє мінімізувати навантаження на пристрій користувача, адже вся складна обробка даних, включаючи аналіз тексту, взаємодію з базою даних та планування нагадувань, виконується на серверній стороні, тоді як клієнтська частина фактично функціонує як звичайний чат із мінімальним споживанням оперативної пам’яті та заряду акумулятора.

Таким чином, Telegram виконує функцію «тонкого клієнта», а користувач отримує звичний та інтуїтивно зрозумілий інтерфейс у вигляді діалогу з ботом [9].

Серверна частина (Backend) на базі Java Spring Boot
Вибір мови Java та фреймворку Spring Boot для реалізації серверного ядра не є випадковим. Java багато років залишається стандартом для створення високонавантажених корпоративних систем завдяки своїй надійності, кроссплатформеності (JVM працює на будь-якій ОС) та величезній екосистемі бібліотек [2]. Spring Boot, своєю чергою, значно спрощує початкове налаштування проєкту, автоматизуючи конфігурацію багатьох компонентів [3]. Серверна частина виступає центральним координаційним вузлом, який отримує запити від бота (тобто від Telegram), обробляє їх згідно з бізнес-логікою, звертається до бази даних за потреби та відправляє відповіді назад. Ключову роль тут відіграють контейнер інверсії управління (IoC) та механізм впровадження залежностей (Dependency Injection, DI) [8]. Завдяки DI нові модулі можна додавати до системи без порушення цілісності наявного коду - достатньо описати їхні інтерфейси та зареєструвати в контексті Spring.

Далі детально розглянемо кожен із функціональних компонентів, з яких складається бекенд.

Модуль взаємодії з Telegram API (Interface Controller)

Цей компонент є «вхідними дверима» системи. Він реалізує підтримку Webhook-з’єднань: сервер надає Telegram публічну РЕЖИМ ДОСТУПУ-адресу, на яку месенджер надсилає HTTP-запити щоразу, коли користувач щось пише боту [1]. Завдяки такому підходу дані надходять у реальному часі, без періодичного опитування (polling). Модуль відповідає за приймання JSON-пакетів, які надсилає

Telegram, їхню десеріалізацію (перетворення на зручні внутрішні об'єкти - DTO) та первинну валідацію: наприклад, перевіряє, чи існує користувач у системі, чи не надто часто він надсилає запити (щоб уникнути зловживань). Також тут можуть виконуватися допоміжні дії, такі як логування вхідних повідомлень.

Ядро бізнес-логіки та обробки природної мови (Logic & Processing Unit) є однією з найскладніших і водночас найважливіших частин архітектури системи [6,7].

Цей модуль відповідає за інтелектуальний аналіз текстових повідомлень, які вводить користувач. Оскільки система позиціонується як інструмент для швидкого планування, ми прагнемо, щоб людина могла написати боту, зустріч з менеджером завтра о 14:00 або «полити квіти щодня о 9 ранку», не дотримуючись суворих команд. Саме тут відбувається:

- токенізація тексту;
- розпізнавання ключових слів (ключові фрази: «завтра», «о 14:00», «щодня», «до п'ятниці»);
- ідентифікація намірів (intent detection) - чи має на увазі користувач створення нової задачі, перегляд списку, редагування дедлайну чи, можливо, видалення запису.

Після того як намір визначено, виконується мапінг на конкретну дію. Якщо ж розпізнати намір не вдається, бот ввічливо просить уточнити повідомлення або показує коротку підказку.

Рівень персистентності та управління даними (Data Persistence Layer) Для довготривалого зберігання інформації про завдання, користувачів та налаштування система використовує реляційну базу даних (наприклад, PostgreSQL або MySQL, залежно від середовища розгортання) [5]. Взаємодія з базою реалізована через бібліотеку Spring Data JPA, яка є надбудовою над стандартом Java Persistence API [4, 13]. Завдяки JPA розробник працює зі звичайними Java-об'єктами (Entity), а фреймворк автоматично генерує необхідні SQL-запити (SELECT, INSERT, UPDATE, DELETE). Це значно пришвидшує розробку та знижує ймовірність помилок [4].

Кожна задача в базі даних зберігається у вигляді окремої сутності TaskEntity, яка містить набір обов’язкових атрибутів, зокрема унікальний числовий ідентифікатор, ідентифікатор користувача в Telegram для визначення належності завдання, текстовий опис задачі, часову мітку створення, запланований термін виконання (дедлайн), а також статус виконання, який може позначатися як активне, виконане або прострочене завдання.

Рівень персистентності також відповідає за виконання транзакцій - тобто таких груп операцій, які мають виконатися повністю або не виконатися взагалі, наприклад, створення задачі та одночасне оновлення лічильника активних справ у профілі користувача [9]. Асинхронна служба нагадувань (Notification & Scheduling Service).

Оскільки будь-яка система планування немислима без роботи з часом, у бекенді реалізовано спеціальний внутрішній планувальник (Task Scheduler). Він працює у фоновому режимі - зазвичай за допомогою анотації @Scheduled у Spring, що дозволяє запускати методи через задані інтервали (наприклад, кожні 5 хвилин) [3]. Планувальник періодично сканує базу даних у пошуках завдань, у яких дедлайн наближається (наприклад, менше ніж за 15 хвилин або щойно минув). Якщо такі завдання знайдено, сервіс нагадувань формує відповідне повідомлення та ініціює зворотний виклик до Telegram API - тобто надсилає push-повідомлення конкретному користувачеві прямо в чат. Важливо зазначити, що цей механізм працює асинхронно, не блокуючи основний потік обробки вхідних повідомлень, тому користувачі не помічають затримок навіть під час сильної завантаженості сервера.

Масштабованість та перспективи розширення. Архітектура спроектована за модульним принципом, що повністю відповідає вимогам гнучкої (Agile) розробки та подальшої підтримки. Що це означає практично? Якщо кількість активних користувачів почне зростати - наприклад, з десятків до тисяч або навіть сотень тисяч, - систему можна відносно легко перенести на хмарні платформи (Amazon Web Services, Google Cloud Platform, Microsoft Azure) [12]. Це дозволить використовувати автоматичне масштабування серверних віртуальних машин,

балансувальники навантаження та керовані бази даних [12]. Альтернативним варінтом є поділ монолітного застосунку на мікросервіси: наприклад, виділити сервіс нагадувань, сервіс обробки природної мови та основний API-шлюз як окремі незалежні процеси.

Крім того, передбачена можливість підключення зовнішніх API для розширення функціональності бота. Це можуть бути сервіси прогнозу погоди (щоб нагадати взяти парасольку), календарі (Google Calendar, Outlook), інтеграція з Trello або Asana, або навіть курси валют чи новинні стрічки. Додавання такого роду модулів не вимагатиме переписування ядра системи - достатньо реалізувати новий клас-обробник, який реєструється в контейнері Spring. Таким чином, наш бот перетворюється не просто на планувальник, а на потужного персонального асистента, здатного адаптуватися до потреб конкретного користувача.

2.2 Опис основних компонентів системи

Клас TelegramBot є центральним елементом системи, який відповідає за приймання та обробку повідомлень користувачів. У ньому реалізовано метод обробки оновлень onUpdateReceived, за допомогою якого визначається тип отриманого повідомлення та виконується відповідна логіка системи рис 1.1. Основними функціями класу є обробка текстових повідомлень і callback-запитів, керування станами користувачів, а також виклик необхідної бізнес-логіки для виконання дій у системі.

У процесі обробки повідомлень використовується система умовних операторів, що дозволяє визначити дію користувача та виконати відповідний сценарій.

Одним із найважливіших і водночас найбільш новаторських елементів розробленої системи є компонент, відповідальний за аналіз часових параметрів, - TimeParser. Його ключове призначення полягає в тому, щоб максимально спростити користувачеві взаємодію з планувальником. Замість того щоб заповнювати громіздкі форми з випадаючими календарями та полями для годин і

хвилин, людина може просто написати боту повідомлення природною мовою. Це докорінно змінює досвід використання: планування справ стає швидким, інтуїтивно зрозумілим і не вимагає додаткових кліків.

Алгоритм роботи TimeParser базується на комбінації кількох підходів, що дозволяє охопити широкий спектр формулювань часових вказівок.

Лексичний аналіз за допомогою регулярних виразів (RegEx) [7].Система сканує вхідне текстове повідомлення рядок за рядком, використовуючи попередньо налаштовані регулярні вирази для пошуку конкретних шаблонів. Наприклад, вона розпізнає такі формати, як 14:00, 14:30, 25.12, 2025-12-25, а також словесні конструкції по типу 2 години або через 15 хвилин. Завдяки цьому можна точно виділити явно задані числові та символьні позначення часу та дати.

Розпізнавання відносних часових маркерів. Користувачі рідко вводять абсолютні дати - набагато частіше вони використовують слова-орієнтири: сьогодні, завтра, післязавтра, на наступному тижні, через місяць. Для цього компонент містить словник таких ключових слів, кожному з яких зіставлено правило перерахунку відносно поточної системної дати (моменту отримання повідомлення). Наприклад, завтра – це поточна дата плюс одна доба, післязавтра - плюс дві доби, а через тиждень - плюс сім днів.

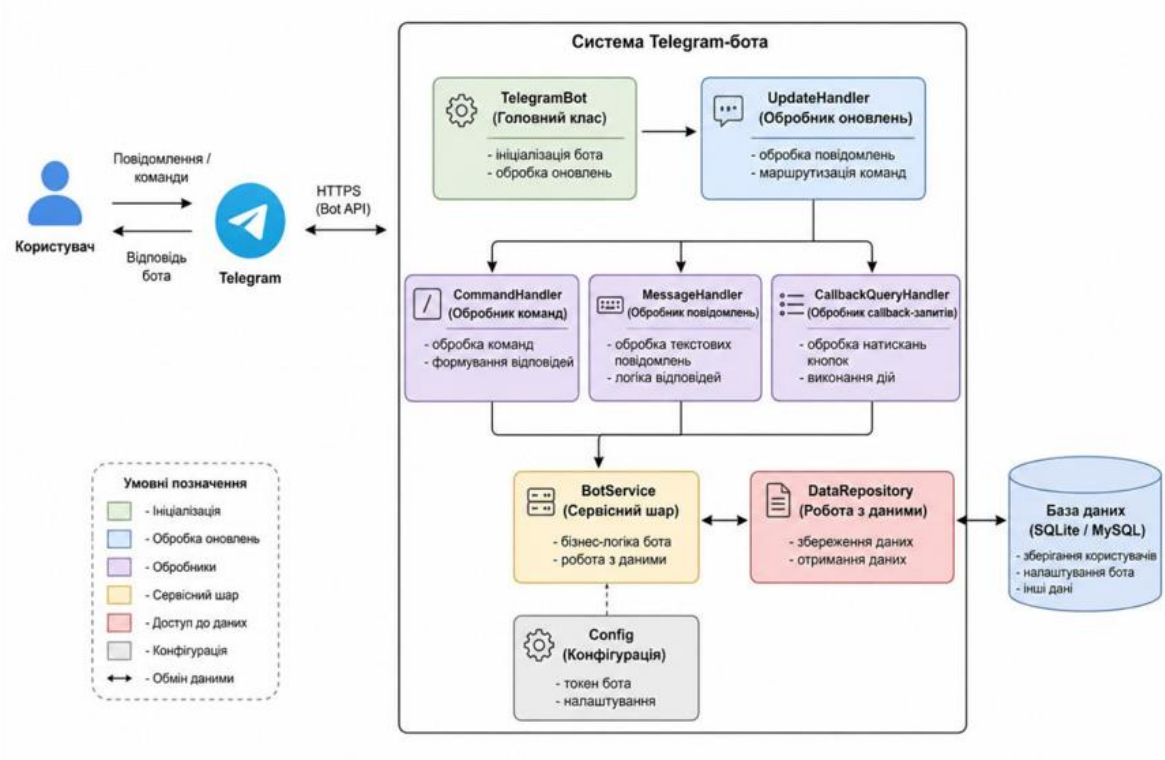


Рис. 1.1 Діаграма структури бота

Контекстуальна інтерпретація розмитих часових понять (нечітких проміжків).Людська мова часто оперує приблизними категоріями: ввечері, зранку, після обіду, опівночі. Для зручності в TimeParser реалізовано логіку, яка перетворює такі розмиті поняття на конкретні числа [6, 7]. За замовчуванням «ввечері» може означати 18:00 або 19:00 (залежно від налаштувань бота), після обіду - 15:00, рано вранці - 08:00, а опівдні - 12:00. Усі ці параметри легко змінюються в конфігураційному файлі, що дозволяє адаптувати систему під особисті вподобання.

Приклад роботи алгоритму в реальному сценарії. Нехай користувач надсилає боту повідомлення: Купити квитки завтра о 10 ранку. TimeParser виконує послідовні кроки [6, 7].

Спершу він ідентифікує ключове слово завтра, отримавши поточну системну дату, додає до неї 24 години. Так визначається календарний день виконання.

Потім алгоритм знаходить маркер 10 ранку. Оскільки «ранку» чітко вказує на години без хвилин, встановлюється значення 10:00.

Як результат, компонент формує структурований об’єкт типу LocalDateTime, який містить точну дату та час: наприклад, 2026-04-25T10:00:00. Цей об’єкт уже готовий до збереження в базі даних як дедлайн задачі [2].

Завдяки такому підходу користувачеві більше не потрібно вручну налаштовувати часові параметри - все відбувається автоматично. Це не лише економить час, але й робить процес планування майже миттєвим, що безпосередньо підвищує задоволеність користувачів і показник їхнього повернення (retention).

У будь-якій системі управління завданнями важливо мати змогу відокремлювати термінові справи від другорядних. Однак примушувати користувача щоразу обирати пріоритет через меню або кнопки - це додаткове когнітивне навантаження, яке сповільнює роботу. Саме тому було розроблено модуль PriorityParser, який автоматично визначає рівень пріоритетності задачі на основі семантичного аналізу її тексту.

Робота класифікатора не потребує втручання людини: він самостійно присвоює кожній новій задачі один із трьох статусів: High (високий), Medium (середній) або Low (низький). Алгоритм аналізує вхідний рядок, вишукуючи наперед задані маркери.

Високий пріоритет (Priority.HIGH).Система реагує на слова, що несуть емоційне забарвлення терміновості або функціональної критичності: терміново, важливо, дуже треба, горить, дедлайн скоро, негайно. Якщо хоча б одне з таких слів присутнє, завдання автоматично отримує найвищий пріоритет. Це означає, що під час виведення списку справ (наприклад, за командою показати завдання) такі записи завжди будуть у верхній частині, привертаючи увагу користувача.

Низький пріоритет (Priority.LOW).Присвоюється у випадках, коли в тексті виявлено вказівки на необов’язковість, відсутність терміновості чи відтермінування. Типові ключові слова: колись, «потім», не спішно, можливо, як буде час, факультативно. Такі завдання не турбують користувача нагадуваннями (або турбують рідше) і можуть ховатися в кінці списку.

Стандартний пріоритет (Priority.MEDIUM). Це значеннь встановлюється автоматично для переважної більшості задач, у тексті яких не знайдено жодного явного маркера ні високого, ні низького пріоритету. Такий підхід є «безпечним за замовчуванням»: користувач завжди може згодом змінити пріоритет вручну через спеціальну команду, якщо система помилилася.

Впровадження PriorityParser дозволяє системі самостійно наводити лад у робочому просторі користувача. Коли вхідних задач багато (десятки або навіть сотні), автоматичне сортування за пріоритетом допомагає не розгубитися і зосередитися на найважливішому. Таким чином, когнітивне навантаження на людину знижується, а продуктивність планування зростає.

Будь-яка практична система планування неможлива без надійного та ефективного зберігання даних. Нижній рівень архітектури - це рівень доступу до даних (Data Access Layer), центральним елементом якого є інтерфейс TaskRepository. Він реалізований із використанням технології Spring Data JPA - потужного інструменту, що надає зручну абстракцію над реляційною базою даних. Завдяки JPA об’єкти Java (які називаються Entities) автоматично відображаються на таблиці в базі, а їхні поля - на стовпці. Розглянемо ключові аспекти реалізації цього модуля детальніше.

Однією з головних переваг Spring Data JPA є те, що розробнику не потрібно писати більшість SQL-запитів вручну. Достатньо оголосити метод в інтерфейсі репозиторію, дотримуючись певної конвенції іменування (наприклад, findByUserId), і фреймворк сам згенерує необхідний код для виконання стандартних операцій створення (CREATE), читання (READ), оновлення (UPDATE) та видалення (DELETE) - так званий CRUD. Це скорочує час розробки та зменшує ймовірність синтаксичних помилок.

Окрім базових операцій, репозиторій підтримує кастомні методи пошуку, необхідні для виконання специфічної бізнес-логіки. Наприклад, метод findByUserIdAndIsCompletedFalse() дозволяє миттєво отримати список лише тих завдань певного користувача, які ще не виконані. Інші зручні методи можуть знаходити задачі за діапазоном дат, за пріоритетом або за ключовими словами в

назві. Завдяки цьому серверна логіка працює з даними дуже ефективно, без зайвих фільтрацій на стороні Java-коду.

Всі операції, які змінюють стан бази даних (додавання, оновлення, видалення), виконуються в межах транзакцій. Це означає, що якщо під час виконання такої операції станеться технічний збій (наприклад, відключення живлення сервера або помилка в мережі), база даних не опиниться в проміжному, неконсистентному стані. Транзакція або повністю фіксується, або повністю відкочується. Такий підхід гарантує, що дані користувача ніколи не будуть пошкоджені або втрачені через апаратні чи програмні помилки.

Модуль персистентності забезпечує повний цикл управління інформацією про завдання.

Первинна фіксація відбувається в момент створення користувачем нової задачі, після чого репозиторій зберігає її в таблиці tasks.

Оновлення даних передбачає запис будь-яких змін, зокрема дедлайну, статусу, тексту чи пріоритету задачі, через механізм транзакційного оновлення відповідного запису.

Архівація та видалення задач передбачає можливість позначення виконаних завдань як завершених із подальшим збереженням у базі даних для статистики або їх повне видалення за запитом користувача.

Оскільки Spring Data JPA надає незалежний від конкретної СУБД рівень абстракції, розробник може легко змінювати тип бази даних без змін у бізнес-логіці програми. Наприклад, під час розробки зручно використовувати легку вбудовувану базу H2 (яка працює в пам’яті), а для промислової експлуатації перейти на продуктивну PostgreSQL або MySQL. Для цього достатньо змінити кілька рядків у конфігураційному файлі (application.properties) - жодного рядка коду в репозиторіях або сервісах переписувати не потрібно. Це свідчить про високий рівень архітектурної гнучкості та професійний підхід до проєктування системи.

2.3 Функціональна логіка та алгоритм функціонування системи

Процес взаємодії користувача з інтелектуальною системою планування побудовано на циклічній моделі роботи з запитами. Кожне повідомлення проходить однакову послідовність етапів — від отримання даних до формування та передачі відповіді. Такий підхід забезпечує швидку реакцію системи та точну інтерпретацію команд завдяки адаптації алгоритму до характерних сценаріїв взаємодії користувачів із ботом.

Система функціонує на основі подієво-орієнтованої архітектури, де окремі дії, такі як надсилання повідомлення, натискання кнопки чи вибір команди в Telegram, вважаються подіями. Інформація передається через Telegram, який слугує посередником: отримані від користувача повідомлення направляються через захищене HTTPS-з'єднання на серверний додаток у вигляді об’єкта Update. Цей об’єкт містить текст повідомлення, ідентифікатор чату, відомості про користувача та інші службові параметри.

На серверному рівні обробкою запитів займається спеціальний контролер TelegramBot. Він аналізує вхідні дані, виявляючи ключові параметри, зокрема унікальний ідентифікатор користувача (Chat ID), тип контенту та поточний контекст діалогу. Завдяки цьому система розрізняє різні види дій: створення нового завдання, зміну існуючих записів або виконання службових команд. Наприклад, команда "купити молоко" може означати створення нового завдання або оновлення попередньої задачі залежно від контексту.

Розпізнавши контекст запиту, система обирає відповідний сценарій обробки: створення нового завдання, редагування даних чи виведення інформації. Сценарій створення використовується для додавання нових записів, редагування дозволяє змінювати дедлайни, статуси або текст завдань, а інформаційний сценарій формує список активних завдань, статичні дані або довідкову інформацію. Модульний підхід до структури системи не лише полегшує управління логікою кожного процесу, але й забезпечує ефективне масштабування.

Ключовим аспектом системи є інтелектуальна обробка текстових запитів. У ході створення нових завдань текст проходить через спеціальні аналітичні модулі, які впорядковують неструктуровану інформацію в набір параметрів. Модуль TimeParser виявляє часові маркери, визначаючи дати, часові проміжки й невизначені вирази на кшталт "завтра" або "через 2 години". Отримані дані формуються у формат LocalDateTime для точності. За відсутності часових параметрів задачі автоматично встановлюється статус без дедлайну або стандартний термін за замовчуванням.

Паралельно працює модуль PriorityParser, аналізуючи текст для встановлення пріоритетності завдань. Він використовує ключові слова для класифікації завдань як високопріоритетних ("терміново", "важливо") чи низькопріоритетних ("потім", "не спішно"). У результаті текст користувача трансформується у структурований об’єкт із усіма необхідними параметрами: описом завдання, дедлайном, пріоритетом і супутньою метаінформацією.

На завершальному етапі готовий об’єкт надсилається сервісному шару системи. Дані перевіряються на коректність та зберігаються у базі даних через TaskRepository, реалізований на Spring Data JPA. Усі операції відбуваються в межах транзакційного механізму, що гарантує цілісність і захист даних навіть за умов технічних збоїв. Це виключає можливість втрати чи некоректного збереження часткової інформації.

Після завершення обробки сервер формує відповідь для користувача та надсилає її через Telegram API.

Комбінація текстового введення та інтерактивних Inline-клавіатур формує зручне середовище для взаємодії, завдяки якому скорочується кількість рутинних дій користувача і спрощується процес планування. Розроблена система дозволяє оперативно й комфортно керувати персональними задачами, автоматизує визначення дедлайнів і пріоритетів, а також створює технологічну основу для майбутнього розширення функціоналу. Зокрема, передбачається інтеграція нейромережевих моделей для більш точного аналізу намірів користувача.

2.4 Моделювання вимог

Процес розробки системи базується на комплексному аналізі очікувань користувачів та технічних стандартів до сучасного програмного забезпечення. Усі вимоги до продукту розділені на дві основні категорії: функціональні, що визначають безпосередні можливості бота, та нефункціональні, які описують якісні характеристики його роботи.

Функціональні можливості системи зосереджені навколо повного циклу управління персональними задачами. Першочерговою вимогою є забезпечення стабільного створення, редагування та видалення записів через діалогове вікно месенджера. Система повинна не просто зберігати текст, а й здійснювати його інтелектуальний аналіз для автоматичної екстракції часових маркерів та пріоритетів. Крім того, обов'язковою є наявність механізму зворотного зв'язку, що реалізується через систему активних нагадувань, які надсилаються користувачу згідно з встановленим графіком. Також передбачена можливість візуалізації списку справ за допомогою інтерактивних елементів інтерфейсу, таких як кнопки швидкої дії для відмітки про виконання.

Нефункціональні параметри визначають надійність та ефективність використання розробленого інструментарію [11]. Одним із ключових показників є швидкодія: обробка вхідних повідомлень та реакція на натискання кнопок мають відбуватися майже миттєво, щоб не переривати природний потік взаємодії. Вимога до надійності передбачає гарантоване збереження даних у реляційній базі навіть за умови технічних збоїв або перезавантаження сервера.

Особлива увага приділяється зручності використання (Usability). Інтерфейс системи має бути максимально спрощеним, щоб мінімізувати кількість дій для створення нового завдання. Завдяки використанню платформи Telegram, система автоматично стає кросплатформеною, забезпечуючи однаковий рівень комфорту як на мобільних пристроях, так і на десктопних версіях застосунку. Таким чином, сукупність цих вимог формує цілісну технічну стратегію розробки, спрямовану на створення стабільного та ефективного цифрового помічника.

2.5 Проектування Баз даних

Для забезпечення надійного зберігання інформації та підтримки цілісності даних у системі було спроєктовано реляційну модель бази даних. Центральним елементом архітектури є сутність Task (Завдання), яка акумулює в собі всі необхідні параметри для коректного функціонування планувальника та системи нагадувань.

Структура та атрибутивний склад таблиці Task:

- id (ідентифікатор). Первинний ключ (Primary Key) типу Long із стратегією автоматичної генерації значення. Він забезпечує унікальність кожного запису в системі та оптимізує швидкість пошуку операцій;
- user_id (ідентифікатор користувача). Зовнішній ключ, який пов’язує конкретне завдання з унікальним профілем користувача в месенджері Telegram. Це поле є критичним для забезпечення персоналізації даних та коректної адресації нагадувань;
- text (зміст завдання). Текстове поле, що зберігає семантичне ядро задачі. Важливо, що в цій колонці зберігається вже очищений від службових слів контент, що полегшує його візуальне сприйняття користувачем;
- deadline (термін виконання). Поле типу LocalDateTime, яке фіксує точну дату та час, коли завдання має бути виконане або коли система повинна надіслати фінальне сповіщення. Саме на основі цього атрибута працює фоновий процес (Scheduler), що перевіряє чергу нагадувань;
- priority (пріоритет). Перелічуваний тип даних (Enum), який відображає ступінь важливості задачі. Використання окремого поля для пріоритету дозволяє ефективно сортувати списки справ та реалізовувати логіку адаптивного призначення часу;
- status (статус виконання). Логічне поле (Boolean) або перерахунок, що вказує на поточний стан завдання (наприклад, «активне», «виконане» або «архівоване»). Це дозволяє системі фільтрувати актуальні записи та надавати користувачу статистику його продуктивності.

Обрана модель даних є масштабованою та дозволяє у майбутньому легко доповнювати систему новими сутностями, такими як категорії завдань, теги або налаштування профілю користувача, без порушення існуючих зв'язків. Використання технології JPA/Hibernate дозволяє автоматизувати взаємодію між об’єктною моделлю мови Java та спроєктованою структурою таблиць у PostgreSQL [2].

2.6 Проектування інтерфейсу користувача

Взаємодія користувача з програмним комплексом реалізована на базі платформи Telegram, що дозволяє використовувати звичне для більшості клієнтів середовище без необхідності встановлення додаткового програмного забезпечення.

Архітектура інтерфейсу побудована на принципах мінімалізму та оперативності, де основна увага приділяється швидкості фіксації завдань.

Взаємодія системи з користувачем реалізована через текстовий комунікаційний шлюз та інтерактивні Inline-кнопки рис 2.1.

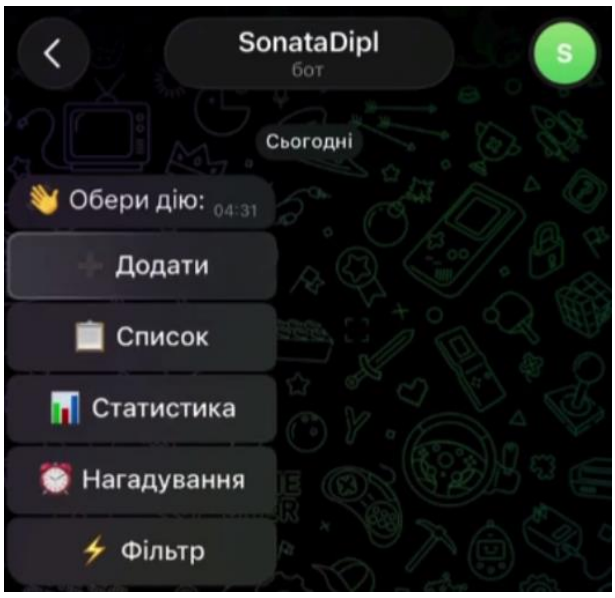


Рис. 2.1 Інтерактивні Inline-кнопки

Головним інструментом введення інформації слугують текстові повідомлення, які користувач може надсилати у довільному форматі. Використовуючи модуль обробки природної мови, система аналізує вхідні

повідомлення, автоматично виділяючи ключові атрибути для створення майбутніх завдань. Така структура дозволяє імітувати спілкування з цифровим асистентом, що покращує зручність користування та зменшує бар’єр для нових користувачів. Інтерактивні Inline-клавіатури використовуються для навігації по меню та керування існуючими записами. Вони суттєво відрізняються від класичних текстових команд, оскільки прив’язуються безпосередньо до конкретних повідомлень, що дозволяє: здійснювати миттєву зміну статусу задачі, наприклад позначення як виконано; викликати контекстні меню для редагування або видалення записів одним натисканням; забезпечувати динамічне оновлення інтерфейсу без надсилання нових повідомлень, що зберігає історію чату лаконічною.

Використання графічних елементів Telegram Bot API у поєднанні з текстовим вводом забезпечує оптимальний баланс між функціональністю та простотою. Користувач не зобов’язаний пам’ятати складний синтаксис команд, оскільки основні дії доступні через візуальні елементи керування. Крім того, такий інтерфейс є адаптивним за замовчуванням: він однаково коректно відображається на смартфонах, планшетах та десктопних версіях месенджера, забезпечуючи безперервний доступ до списку справ незалежно від пристрою.

Така структура взаємодії дозволяє досягти головної мети інтерфейсу – зробити процес планування максимально непомітним та швидким, перетворюючи рутинне управління задачами на просту діалогову взаємодію.

3 РЕАЛІЗАЦІЯ ПРОГРАМНОЇ СТИСТЕМИ

3.1 Характеристика та переваги використаних технологій

Технологічний стек обирався з огляду на надійність, масштабованість і швидкість розробки. Оскільки система має працювати в реальному часі та гарантувати цілісність даних, використано інструменти, які стали галузевими стандартами для високонавантажених серверних застосунків.

Основою системи є Java (версії 17+). Її об'єктно-орієнтована модель сприяє створенню чіткої ієрархії класів для задач і парсерів. До основних переваг використання Java належать строга типізація, яка дозволяє зменшити кількість помилок під час розробки програмного забезпечення, підтримка багатопотоковості, що забезпечує паралельне виконання модуля нагадувань та основного потоку обробки даних, а також розвинена екосистема з великою кількістю перевірених бібліотек для роботи з текстом, часовими параметрами та мережевою взаємодією.

Для серверної логіки обрано Spring Boot із принципом «угода над конфігурацією», що дало змогу зосередитись на бізнес-логіці [3].

Spring Data JPA спрощує роботу з реляційними базами даних через об'єкти замість SQL [4, 13].

Dependency Injection (DI) забезпечує гнучке керування компонентами та їхнє тестування.

Вбудований сервер (Tomcat) дозволяє запускати систему як окремий JAR-файл, полегшуючи розгортання.

Для взаємодії з Telegram у системі використовується офіційний Bot API та Java-бібліотека TelegramBots Spring Boot Starter, що забезпечує зручну інтеграцію месенджера із серверною частиною застосунку. Використання цього інструментарію дозволяє приховати низько рівневі HTTP-деталі завдяки шару абстракції, підтримує механізми Webhook і Long Polling для оптимізації отримання оновлень, а також забезпечує безпечну передачу токенів і валідацію вхідних даних.

Для збирання проєкту та керування залежностями використано Maven. Для локальної розробки застосовано базу даних H2, яка легко замінюється на PostgreSQL або MySQL у продуктивному середовищі без змін у коді [5]. Такий підхід до вибору технологій забезпечує як поточне функціонування системи, так і потенціал для масштабування на велику кількість користувачів.

Головним компонентом, що відповідає за прийом та опрацювання вхідних даних, є клас TelegramBot. Його внутрішня побудова забезпечує асинхронне опрацювання подій та розрізнення типів повідомлень, які надходять через Telegram API.

Метод onUpdateReceived виконує функцію головного диспетчера. Кожне оновлення надходить у вигляді об'єкта Update, який може містити різні дані. Алгоритм обробки включає:

Первинне розділення - перевірка на текст (hasMessage()) чи натискання кнопок (hasCallbackQuery()).

Ідентифікацію користувача - за chatId визначається запис у базі даних та поточний контекст взаємодії.

Маршрутизацію – залежно від типу оновлення викликаються спеціалізовані обробники (наприклад, handleTextMessage або handleCallbackQuery).

Для підтримки багатокрокових діалогів застосовано концепцію скінченного автомата (Finite State Machine). Це дозволяє системі запам'ятовувати етап, на якому перебуває користувач, і по-різному трактувати однакові дії залежно від контексту.

Система підтримує такі стани:

DEFAULT – очікування команди або нової задачі.

AWAITING_TASK_TEXT – очікування тексту нової задачі.

AWAITING_TIME_EDIT –очікування нового дедлайну для редагування.

AWAITING_PRIORITY_SELECTION – очікування вибору пріоритету через меню.

Така організація запобігає безладному виконанню команд і забезпечує послідовність дій.

Для роботи з Inline-клавіатурами передбачено спеціальний механізм. Поле callback_data містить структурований ідентифікатор дії (наприклад, EDIT_TASK_123 або DONE_TASK_456). Процес обробки взаємодії включає розбір команди з виділенням префікса дії та ідентифікатора відповідного об’єкта, виконання необхідної логіки шляхом виклику сервісного методу, наприклад оновлення статусу задачі, а також оновлення інтерфейсу користувача. Замість надсилання нового повідомлення бот редагує вже існуюче повідомлення, змінюючи його текст або набір кнопок, що дозволяє підтримувати порядок у чаті та забезпечує миттєвий візуальний зворотний зв’язок для користувача.

Кожен етап обробки супроводжується перевіркою коректності. Якщо користувач вводить некоректну дату або порожній рядок під час редагування, система не переходить до наступного кроку, а видає повідомлення-підказку. Це гарантує, що в базу даних потрапляють лише валідні та структуровані об'єкти. Така організація логіки спрощує розширення функціоналу: для додавання нової можливості (наприклад, групових завдань) достатньо ввести новий стан і реалізувати його обробку в диспетчері.

3.2 Програмна реалізація процесу формування завдань

Створення нового завдання є багатокроковою процедурою, орієнтованою на мінімізацію ручного введення атрибутів. Алгоритм роботи системи передбачає послідовну обробку вхідних даних, починаючи з прийому текстового повідомлення користувача у довільній формі. Після цього виконується попередній лінгвістичний аналіз, який включає токенізацію тексту, видалення шумових слів та виділення змістового ядра повідомлення. Далі паралельно запускаються модулі аналізу часу та пріоритетності, що дозволяють автоматично вилучати необхідні метадані з контексту повідомлення. Сформований об’єкт задачі тимчасово зберігається у кеші або спеціальній таблиці до моменту підтвердження користувачем. Після цього через інтерактивне меню задається тип повторюваності нагадування, зокрема одноразове, щоденне або щотижневе виконання. На завершальному етапі задача

переноситься до основної бази даних, а система автоматично планує подальше надсилання нагадувань.

TimeParser є технологічним ядром системи, поєднує детерміновані (регулярні вирази) та евристичні методи аналізу часових маркерів. До технічних особливостей модуля належить використання гнучких RegEx-шаблонів для розпізнавання різних форматів часу та дат, зокрема таких конструкцій, як «14:30» або «через 2 години» рис 3.1.

Система також використовує словникові відповідності, які забезпечують перетворення природних часових позначень, наприклад «вранці» чи «ввечері», у конкретні часові значення.

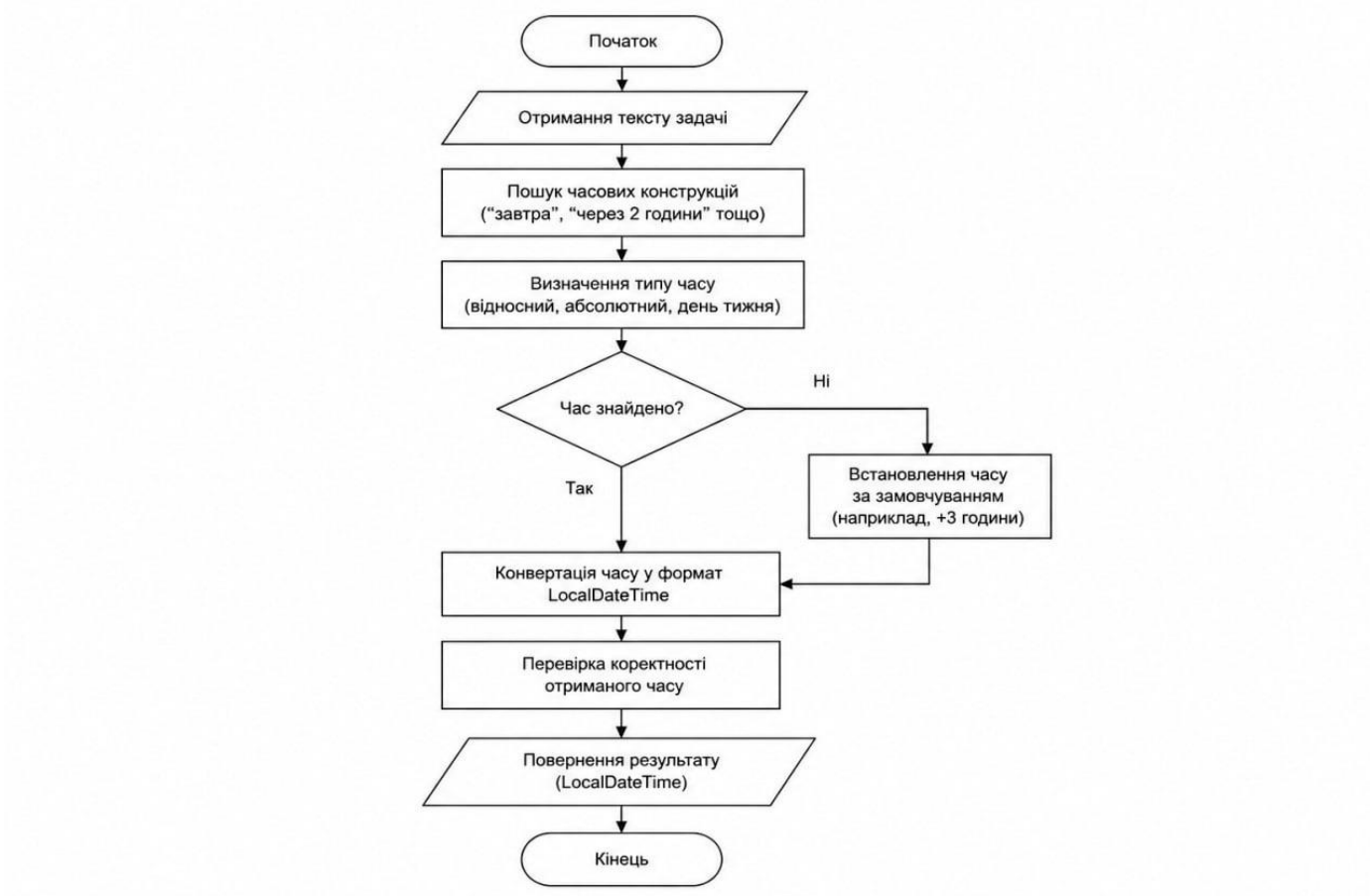


Рис 3.1 Алгоритм роботи модуля обробки часу

Окрім цього, реалізовано механізм обчислення відносних дат, таких як «завтра», «післязавтра» або «наступного вівторка», з урахуванням часової зони сервера. Завдяки такому підходу повідомлення на кшталт «Зустріч з клієнтом

післязавтра о 14:30» автоматично перетворюється на точний графік виконання без необхідності ручного вибору дати та часу.

PriorityParser працює як класифікатор на основі вагових коефіцієнтів та пошуку сигнальних маркерів у тексті. Розподіл пріоритетів:

HIGH – виявлення тригерів терміновості (терміново, важливо, deadline, негайно»). Активує посилені нагадування (наприклад, за 10 хвилин до дедлайну);

LOW – маркери відкладення (як буде час, неважливо, колись);

MEDIUM – значення за замовчуванням для звичайних записів, що забезпечує баланс у списку справ/

3.3 Функціонал управління нагадуваннями та сповіщеннями

Нагадування реалізовані як асинхронні події за допомогою Java-інструментів TaskScheduler, ScheduledExecutorService. Модуль системи реалізує механізми адаптивної взаємодії з користувачем, що підвищує зручність роботи з нагадуваннями та забезпечує гнучкість керування задачами. Зокрема, користувач має можливість змінювати запропонований системою час виконання або нагадування через інтерактивне меню без необхідності повторного створення задачі. Це дозволяє швидко коригувати графік відповідно до змін у планах користувача.

Додатково в системі реалізовано функцію Snooze, яка активується в момент надсилання нагадування. Бот автоматично пропонує набір інтерактивних кнопок для відкладення повідомлення на певний проміжок часу, наприклад на 15, 30 або 60 хвилин. Такий підхід дає змогу користувачеві не втрачати важливу задачу, навіть якщо в конкретний момент немає можливості її виконати.

Важливою особливістю модуля є також його автономність. Система автоматично контролює актуальність нагадувань та після виконання або видалення задачі самостійно очищує пов’язані записи, видаляючи застарілі дані з бази рис 3.2.

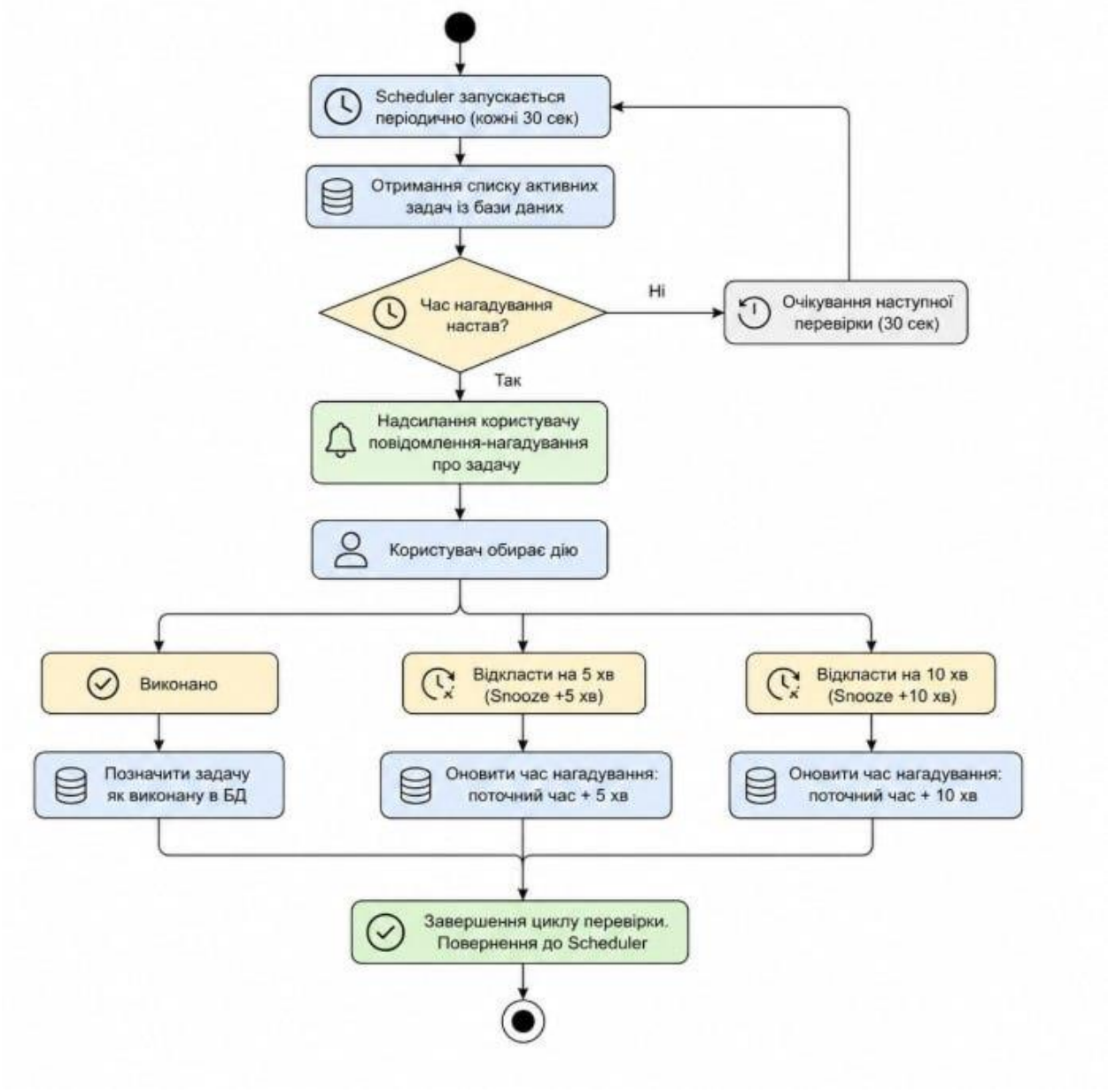


Рис. 3.2 діаграма активності процесу роботи системи нагадувань

Це дозволяє оптимізувати використання ресурсів СУБД, зменшити навантаження на систему та підтримувати високу продуктивність роботи навіть при значній кількості активних користувачів і задач.

Інтерфейс побудований на принципах мінімалізму та максимальної функціональності. Основний елемент - Inline Keyboard Markup інтерактивні кнопки в повідомленнях. Архітектура користувацького інтерфейсу побудована таким чином, щоб забезпечити максимально просту та зрозумілу взаємодію з системою. Основним елементом є головне меню, яке надає доступ до глобальних функцій,

зокрема перегляду активних задач, створення нових записів і налаштування параметрів роботи системи. Для кожної окремої задачі формується спеціальна картка, що містить текстову інформацію про завдання та набір інтерактивних кнопок для виконання основних дій, таких як позначення задачі як виконаної, редагування, зміна пріоритету або видалення.

Додатково реалізовано систему фільтрації, яка дозволяє швидко відображати лише окремі категорії завдань, наприклад термінові або заплановані на поточний день. Використання кнопок замість складних текстових команд значно спрощує взаємодію з ботом, усуває необхідність запам’ятовування команд та робить інтерфейс більш інтуїтивно зрозумілим для користувача.

У розділі детально описано технічну реалізацію системи планування задач. Висвітлено технологічний стек Java, Spring Boot, Telegram API, що забезпечує стабільність і швидкодію. Особливу увагу приділено модулям інтелектуальної обробки тексту TimeParser, PriorityParser, інтеграція яких дозволила замінити формальні запити природним спілкуванням. Описана архітектура інтерфейсу на основі inline кнопок гарантує ергономічність та високу швидкість взаємодії. Розроблена система є повноцінним інструментом автоматизації особистого планування, готовим до масштабування та додавання нових інтелектуальних функцій.

Архітектура системи побудована на принципах багаторівневої структури (Layered Architecture), що забезпечує чіткий поділ між інтерфейсом взаємодії, бізнес–логікою та рівнем доступу до даних.

Основними структурними компонентами системи є:

TelegramBot. Головний вхідний контролер системи. Його роль полягає у перехопленні вхідних оновлень (Updates) від Telegram API, первинній обробці повідомлень та передачі їх до відповідних сервісів. Він також відповідає за формування та надсилання відповідей користувачу, включаючи текстові повідомлення та інтерактивні клавіатури.

TaskService. Центральний вузол бізнес-логіки системи. Цей клас координує процеси створення та модифікації завдань, взаємодіючи з парсерами для отримання метаданих та з репозиторієм для збереження стану.

createTask() – метод ініціалізації нової задачі, який агрегує дані від користувача та результати лінгвістичного аналізу;

updateTask() – забезпечує внесення змін до існуючих об’єктів (зміна статусу, тексту або дедлайну);

deleteTask() – відповідає за коректне видалення записів із бази даних.

TaskRepository. Інтерфейс рівня доступу до даних Data Access Layer. Використовуючи можливості Spring Data JPA, він забезпечує взаємодію з PostgreSQL, виконуючи стандартні CRUD-операції та специфічні запити, наприклад пошук активних завдань для конкретного користувача.

TimeParser. Спеціалізований утилітарний компонент, відповідальний за семантичний аналіз часових маркерів.

parseTime() – основний метод, який приймає вхідний рядок та повертає об’єкт LocalDateTime. Він опрацьовує як прямі вказівки часу, так і відносні конструкції («завтр,через годину).

PriorityParser. Компонент, що відповідає за класифікацію завдань за рівнем важливості[8].

detectPriority() – метод аналізує текст на наявність лексичних тригерів терміново, важливо, «пріоритетно) та повертає відповідне значення перерахунку Priority.

Така декомпозиція дозволяє зробити систему гнучкою: наприклад, логіку розпізнавання часу в TimeParser можна вдосконалювати або замінювати на складніші моделі без необхідності переписування коду в TelegramBot або TaskRepository. Це забезпечує високу підтримуваність коду та полегшує процес тестування окремих модулів.

4 ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

4.1 Обґрунтування вибору методів тестування

Тестування – це та фаза розробки, на якій найчастіше вилазять назовні всі приховані недоліки. Тому ще на початку проектування системи я приділив цьому етапу окрему увагу. Мета – підтвердити, що написаний код дійсно робить те, для чого його створювали, а також переконатися, що програма не розвалиться в найнесподіваніший момент через некоректні дії користувача чи збій у зовнішніх сервісах.

Загалом, було вирішено не обмежуватися одним–двома методами, а побудувати багаторівневу стратегію. Нижче перерахую основні напрямки, які вдалося охопити в ході роботи.

Модульна перевірка unit–тестів. У цьому випадку найбільш відповідальними виявилися модулі TimeParser–розбір часових маркерів і PriorityParser–визначення пріоритету. Для них написані окремі тести, які проганяють різні варіанти вхідних рядків - від простих на зразок завтра о 10:00 до більш заплутаних. Це допомогло відловити кілька помилок на ранній стадії, і до того ж тепер, якщо я захочу додати нові ключові слова в словник, ризик щось зламати буде набагато меншим.

Інтеграційні випробування. Тут мова йде вже не про окремі функції, а про те, як вони спілкуються між собою та із зовнішнім світом. У моїй системі основними контрагентами виступають Telegram API через який бот приймає й надсилає повідомлення та база даних PostgreSQL. Інтеграційні тести показали, що всі ланцюжки працюють: об'єкти Java коректно зберігаються в таблицях через JPA/Hibernate, а HTTP-запити до Telegram не губляться навіть при невеликому навантаженні.

Функціональне тестування, це вже перевірка того, що бачить і відчуває сам користувач. Я просто відтворював різні життєві сценарії: пише людина зустріч через годину, – чи правильно з'являється нагадування; натискає кнопку виконано –

чи зникає завдання зі списку, і так далі. Жодних знань про внутрішній код тут не потрібно, головне – щоб система поводилася згідно з документацією.

Для перевірки коректності функціонування модуля обробки часу було протестовано алгоритми аналізу природної мови, реалізовані в класі TimeParser. Під час тестування система успішно розбирала як абсолютні часові формати, наприклад «12.08.2026 18:30», так і відносні конструкції, такі як «через 2 години» або «завтра о 09:00».

Окрему увагу приділили підтримці нечітких часових понять, таких як «ввечері», «вранці» чи «після обіду», які система автономно трансформує у конкретні часові значення. Крім того, був протестований модуль PriorityParser, призначений для автоматичного визначення пріоритету задач на основі ключових слів у тексті повідомлень. Алгоритм коректно ідентифікував високий пріоритет для задач, які включали маркери на кшталт «терміново», «дедлайн» або «зараз», а також присвоював низький пріоритет повідомленням із словосполученнями «потім» чи «неважливо».

Перевірка інтерфейсу та зручності (UI/UX). Оскільки мій бот живе всередині Telegram, а не в окремому застосунку, тестування інтерфейсу зводилося до кількох простих речей: чи швидко бот відповідає на натискання кнопок, чи коректно відображаються інлайн-клавіатури на різних пристроях (я перевіряв на телефоні й на комп'ютері), чи не виникає в людини відчуття ну і що мені з цим робити? після системних повідомлень.

Стрес-тест й робота з помилками. Окремо було перевірено систему на некоректних вхідних даних: порожні повідомлення, дати з неіснуючими числами (32.13.2025), хаотичне натискання кнопок не за порядком. У таких випадках головне, щоб програма не впала (не викинула виняток на рівні всього сервера), а правильно обробила ситуацію - наприклад, сказала не розумію або поставила значення за замовчуванням. У цілому система показала себе досить життєздатною.

Що в підсумку. Комбінація описаних методів дала досить повну картину. Я не стверджую, що помилок не залишилося взагалі - їх неможливо виловити всі. Але критичні сценарії перевірені, вузькі місця знайдено й виправлено, а сам підхід до

тестування дозволяє надалі безпечно допрацьовувати систему, не лякаючись, що одне виправлення зламає щось інше.

4.2 Розробка Текст-кейсів

Для того щоб перевірка системи була структурованою та послідовною, попередньо було сформовано набір тестових сценаріїв. Вони охоплюють типові ситуації, коли користувач взаємодіє із системою коректно і нетипові випадки що можуть виникати під час реального використання. Кожен сценарій має чітко визначену мету та очікуваний результат роботи системи.

Під час тестування основна увага була зосереджена на оцінці функціональності механізму автоматичного створення задач. Перевірялися сценарії, пов’язані зі створенням, редагуванням, видаленням та завершенням задач через Telegram-інтерфейс. Окрему увагу приділяли тестуванню системи callback-кнопок, яка забезпечує взаємодію із задачами без необхідності введення додаткових команд. Також були успішно підтверджені коректність роботи механізму автоматичного визначення часу виконання задач і алгоритму встановлення пріоритетів. Система після отримання повідомлення автоматично створювала структурований об’єкт задачі та зберігала його в базі даних.

Сценарій 1. Перевірка коректності розпізнавання дати й часу.

У межах даного сценарію користувач вводить повідомлення: Купити молоко завтра о 10:00. Очікується, що модуль TimeParser коректно визначить часові маркери, а саме визначить «завтра» як поточну дату з додаванням одного дня та встановить відповідний час. У базі даних має бути створений запис, де текст задачі очищений від часових підказок, а поле deadline містить відповідне значення, обробка конструкції відбувається за наступним кодом.

```
if (text.contains("завтра")) {
LocalTime time = extractTime(text);
return LocalDate.now().plusDays(1).atTime(time);
}
```

У результаті тестування встановлено, що система працює коректно, без затримок.

Сценарій 2. Обробка введення без зазначення часу.

Користувач вводить повідомлення без часових параметрів, наприклад: Зробити домашнє завдання. Очікується, що система коректно обробить відсутність часових маркерів: або створить задачу без дедлайну, або застосує значення за замовчуванням. При цьому не повинні виникати помилки виконання.

Сценарій 3. Автоматичне визначення пріоритету задачі.

У цьому випадку користувач вводить: Терміново здати звіт, очікується що модуль PriorityParser розпізнає ключове слово «терміново» та автоматично встановить високий рівень пріоритету. Додатково передбачається візуальне відображення цього пріоритету в інтерфейсі, обробка конструкції відбувається за наступним кодом.

```
if (text.contains("терміново")
|| text.contains("дедлайн")) {
return Priority.HIGH;
}
```

Результати тестування показали, що класифікація виконується без багів та без затримок [8].

Сценарій 4. Перегляд списку активних задач.

Під час виконання цього сценарію користувач звертається до функції перегляду списку задач. Очікується, що система здійснить запит до бази даних та відобразить усі активні задачі у структурованому вигляді. Для кожної задачі мають бути доступні елементи керування наприклад, Виконано, Редагувати, Видалити.

У результаті список формується коректно, інтерфейсні елементи працюють належним чином.

Сценарій 5. Редагування існуючих задачі.

Користувач змінює параметри вже створеної задачі, зокрема дедлайн. Очікується що система перейде у режим введення нових даних, після чого оновить відповідну інформацію в базі та повідомить користувача про збереження змін.

У процесі тестування підтверджено, що редагування виконується правильно.

Сценарій 6. Видалення задачі.

У цьому сценарії користувач видаляє задачу через відповідну кнопку. Очікується що запис буде видалено з бази даних, а також буде скасовано заплановані нагадування. Система повинна повідомити користувача про успішне виконання дії.

Результати показують що функціонал працює стабільно, повторні нагадування не надходять.

Сценарій 7. Спрацювання нагадування.

Умова: настає момент часу, визначений у полі deadline. Очікується, що фоновий процес знайде відповідну задачу та відправить користувачеві нагадування, обробка конструкції відбувається за наступним кодом.

```
@Scheduled(fixedRate = 60000)
public void checkReminders() {
List<Task> tasks =
taskRepository.findByReminderTimeBefore(
LocalDateTime.now()
);
}
```

В результаті нагадування надходять вчасно, інформація відповідає збереженим даним.

4.3 Результати тестування

Після того як усі тестові сценарії з попереднього підрозділу було виконано, настав час підбити підсумки. Загалом, система поводилася передбачувано й стабільно, хоча без кількох дрібних нюансів, звісно, не обійшлося.

Найголовніше - модулі семантичного аналізу TimeParser і PriorityParser справляються зі своїми завданнями в переважній більшості випадків. Коли користувач пише короткими, імперативними фразами на зразок нагадай про зустріч

завтра о 14:00, бот коректно витягує дату, час, а іноді навіть пріоритет, якщо є слова терміново або неважливо. Усі операції з даними - створення, редагування, видалення, архівація - виконуються з дотриманням ACID-принципів. Це означає, що база даних PostgreSQL не втрачає інформацію навіть у разі несподіваного перезапуску сервера.

Під час тестування я навмисно намагався зламати систему: надсилав порожні повідомлення, некоректні дати, хаотично натискав кнопки. Жодного разу не виникло критичної помилки, яка б призвела до аварійного завершення роботи сервера[10]. Звісно, деякі запити система не розуміла й чесно про це повідомляла, але це не збій, а обмеження логіки. Окремо варто відзначити планувальник нагадувань: він спрацьовував точно в заданий час, без помітних затримок. Інтеграція із системним часом і чергами повідомлень працює надійно.

До того ж було проведено тестування асинхронного сервісу нагадувань ReminderService була перевірена робота Scheduled-механізмів Spring Boot, які автоматично здійснюють перевірку дедлайнів через встановлені часові інтервали. Система успішно визначала завдання з активними нагадуваннями та надсилала відповідні повідомлення користувачам за допомогою Telegram Bot API.

Окремо протестували функцію відкладення нагадувань (Snooze). Після натискання inline-кнопок «+5 хв» або «+10 хв» система автоматично оновлювала параметр reminderTime і заново планувала сповіщення без необхідності повторного введення даних користувачем. Тестування також підтвердило надійність механізму управління станами користувача, реалізованого в класі TelegramBot. Система коректно обробляла різні сценарії, включаючи редагування завдань, зміну часу, встановлення пріоритетів і навігацію між меню через callback-запити.

Інтерфейс через Telegram, на мою думку, вийшов зручним. Inline-клавіатури дозволяють керувати завданнями, навіть не читаючи інструкції - інтуїтивно зрозуміло, куди натискати. Текстові відповіді бота лаконічні, без зайвої води. Час відгуку системи коливається в межах 1–2 секунди, що для месенджера цілком комфортно. Жодних зависань або довгих пауз я не помітив.

Також зустрічались граничні випадки: наприклад, якщо користувач формулює завдання надто складною метафорою або використовує нестандартний порядок слів – система може не розпізнати час або неправильно визначити пріоритет. Це особливість не самої програми, а взагалі будь-якого rule-based підходу. Але для більшості реальних користувачів, які пишуть короткими фразами, а саме так зазвичай спілкуються з ботом, ці проблеми виникають не часто. Тому я класифікував такі випадки як незначні помилки, які не блокують використання.

Результати тестування дають підстави стверджувати, що система цілком працездатна, функціонально повна й готова до реальної експлуатації, навіть при одночасній роботі з великою кількістю задач система зберігала стабільну швидкість обробки повідомлень і не демонструвала критичних помилок під час взаємодії з базою даних або Telegram API [10]. Звісно, ідеальних програм не буває, але всі критичні сценарії відпрацьовано, а дрібні огріхи не впливають на загальну картину. Інтелектуальний планувальник може використовуватися як інструмент для повсякденного тайм-менеджменту.

ВИСНОВКИ

У процесі виконання дипломної роботи було розроблено систему автоматизації нагадувань на базі Telegram-бота, яка дозволяє користувачу створювати, редагувати та контролювати власні задачі.

Під час роботи було проаналізовано сучасні інструменти для планування завдань, що дало змогу визначити їх сильні та слабкі сторони. Зокрема було встановлено що більшість існуючих рішень або перевантажені функціоналом, або навпаки не забезпечують достатнього рівня автоматизації.

На основі проведеного аналізу було спроектовано архітектуру більш зручної системи автоматизації. За базу було взято месенджер Telegram, а серверна частина реалізована з використанням Java та Spring Boot. Такий підхід дозволив забезпечити стабільність роботи, масштабованість та зручність, гнучкість використання системи.

У рамках практичної частини було реалізовано ключові функції: створення, редагування та видалення задач, автоматичне визначення часу, встановлення пріоритетів, а також систему нагадувань. Особливу увагу було приділено обробці природної мови, що дозволяє користувачу вводити задачі у довільній формі.

Результати тестування показали, що система працює стабільно, обробляє введені дані та своєчасно надсилає нагадування. Це підтверджує її відповідність поставленим вимогам.

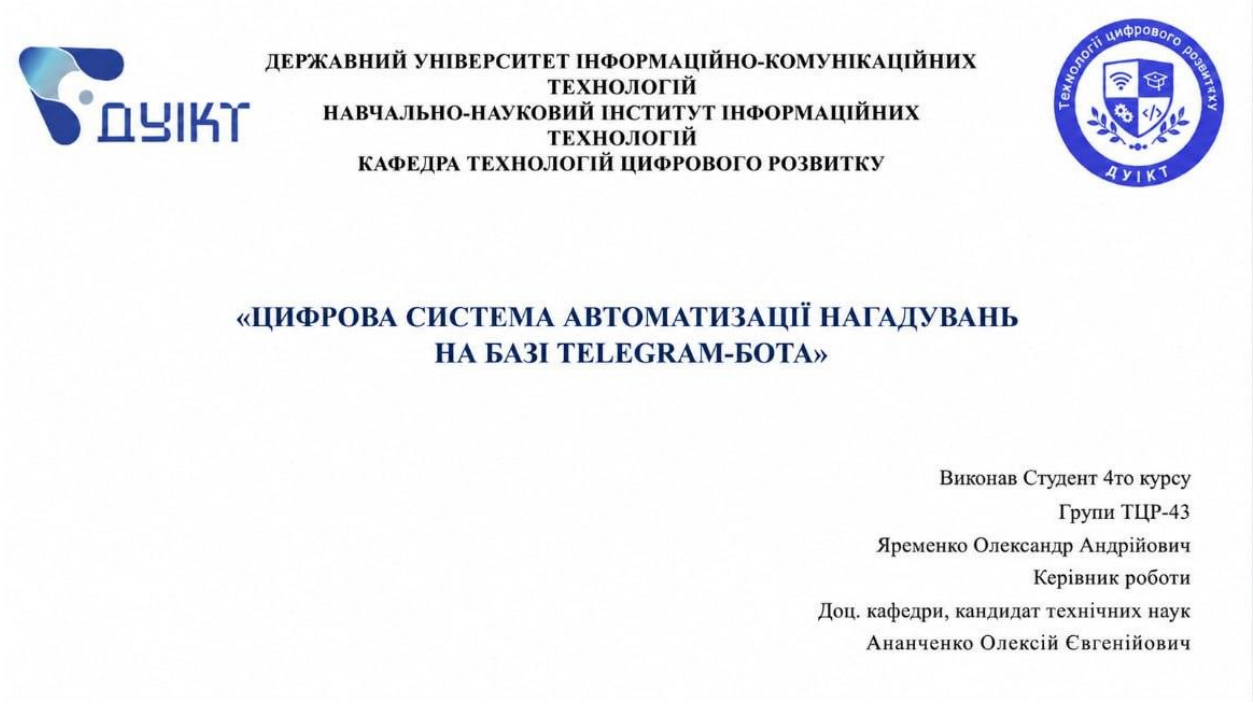
Отже, мета дипломної роботи була досягнута, а розроблений програмний продукт може бути використаний на практиці. У подальшому систему можна розширити за рахунок додавання нових функцій, зокрема інтеграції з іншими сервісами або використання більш складних алгоритмів обробки тексту.

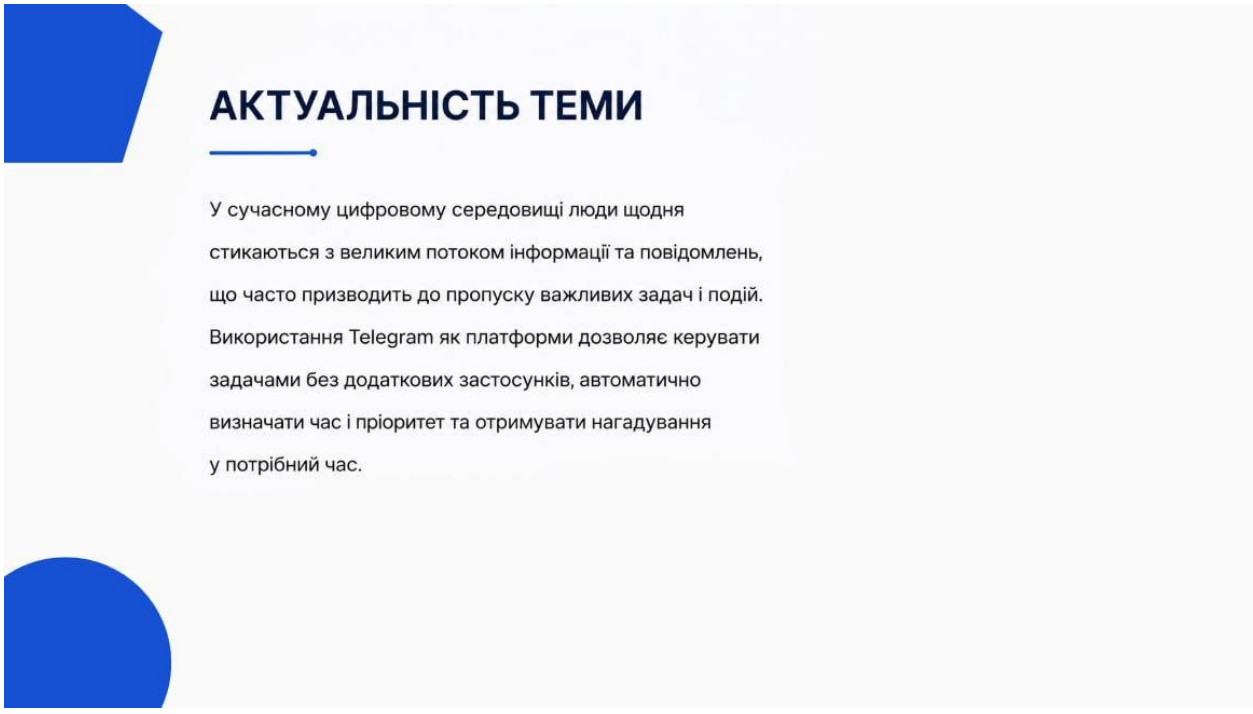
ПЕРЕЛІК ПОСИЛАНЬ

1. Telegram Bot API Documentation [Електронний ресурс]. – РЕЖИМ ДОСТУПУ: https://core.telegram.org/bots/api (дата звернення: 03.03.2026).
2. Java Platform, Standard Edition Documentation [Електронний ресурс]. – РЕЖИМ ДОСТУПУ: https://docs.oracle.com/javase/ (дата звернення: 04.03.2026).
3. Spring Boot Reference Documentation [Електронний ресурс]. – РЕЖИМ ДОСТУПУ: https://spring.io/projects/spring-boot (дата звернення: 05.03.2026).
4. Hibernate ORM Documentation [Електронний ресурс]. – РЕЖИМ ДОСТУПУ: https://hibernate.org/orm/documentation/ (дата звернення: 07.03.2026).
5. PostgreSQL Documentation [Електронний ресурс]. – РЕЖИМ ДОСТУПУ: https://www.postgresql.org/docs/ (дата звернення: 08.03.2026).
6. Jurafsky D., Martin J. H. Speech and Language Processing [Електронний ресурс]. – РЕЖИМ ДОСТУПУ: https://web.stanford.edu/~jurafsky/slp3/ (дата звернення: 10.03.2026).
7. Oracle Documentation [Електронний ресурс]. – РЕЖИМ ДОСТУПУ: https://docs.oracle.com/ (дата звернення: 11.03.2026).
8. Baeldung Java Tutorials [Електронний ресурс]. – РЕЖИМ ДОСТУПУ: https://www.baeldung.com/ (дата звернення: 13.03.2026).
9. GeeksforGeeks Programming Tutorials [Електронний ресурс]. – РЕЖИМ ДОСТУПУ: https://www.geeksforgeeks.org/ (дата звернення: 14.03.2026).
10. Fowler M. Patterns of Enterprise Application Architecture. – Boston : Addison-Wesley, 2002. – 533 p.
11. Sommerville I. Software Engineering. – 10th ed. – Boston : Pearson, 2016. – 816 p.
12. Newman S. Building Microservices. – 2nd ed. – Sebastopol : O’Reilly Media, 2021. – 616 p.
13. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner’s Approach. – 9th ed. – New York : McGraw-Hill Education, 2019. – 880 p.
14. Bird S., Klein E., Loper E. Natural Language Processing with Python. – Sebastopol : O’Reilly Media, 2009. – 504 p.
15. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. – Boston : Addison-Wesley, 1994. – 395 p.

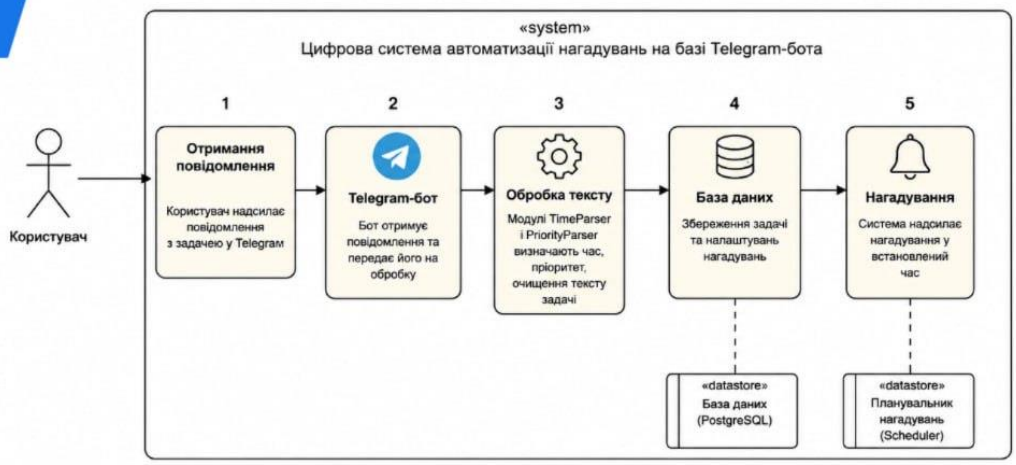
ДОДАТОК А

Демонстраційні матеріали





Діаграма діяльності системи
«Цифрова система автоматизації нагадувань на базі Telegram-бота»



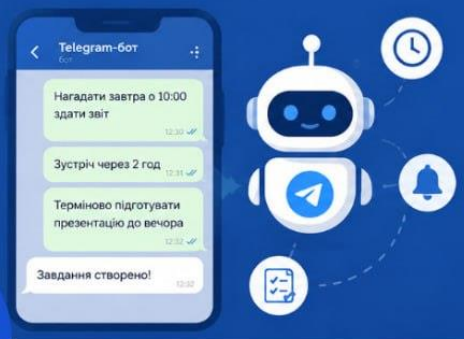
Умовні позначення (UML):

— Діяльність (Action)
— Потік керування (Control Flow)
— Актор (Actor)
— Сховище даних (Data Store)

Мета та завдання роботи

Мета роботи

Підвищити ефективність управління особистими завданнями та нагадуваннями шляхом впровадження **Telegram-бота** з **інтелектуальним аналізом текстових повідомлень користувача.**



Завдання роботи

1. Аналіз існуючих систем

Розглянуто популярні платформи та сервіси для планування задач, їх можливості, переваги та недоліки.

2. Дослідження Telegram та Bot API

Вивчено можливості Telegram-ботів та Telegram Bot API для автоматизації створення і управління задачами.

3. Проектування системи

Розроблено архітектуру системи та створено основні модулі для управління задачами та нагадуваннями.

4. Інтелектуальний аналіз тексту

Реалізовано механізми автоматичного визначення часу виконання та пріоритету задач на основі текстових повідомлень.

5. Тестування та оцінка системи

Проведено тестування системи та оцінено її функціональність, точність і зручність використання.

Вимоги до системи

Функціональні вимоги

Створення задач
Користувач може додавати задачі через Telegram-бота надсиланням текстових повідомлень.

Автоматичне визначення часу
Система автоматично розпізнає дату та час з тексту повідомлення.

Визначення пріоритету
Система аналізує текст задачі та визначає рівень важливості.

Перегляд та управління задачами
Користувач може переглядати список задач, редагувати та видаляти їх.

Нагадування
Бот надсилає нагадування про задачі у встановлений час через Telegram.

Фільтрація задач
Користувач може фільтрувати задачі за статусом, пріоритетом або датою.

Нефункціональні вимоги

Надійність
Система повинна стабільно працювати без збоїв та втрати даних.

Продуктивність
Час обробки запиту не повинен перевищувати 2 секунд при звичайному навантаженні.

Безпека
Дані користувачів повинні зберігатися захищено, з обмеженим доступом.

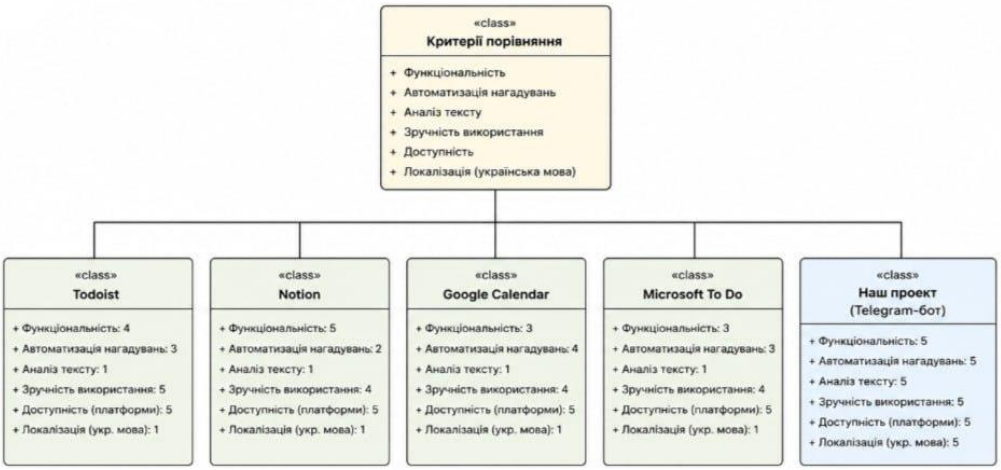
Доступність
Система повинна бути доступною 24/7 з мінімальним часом недоступності.

Локалізація
Інтерфейс та повідомлення бота повинні підтримувати українську мову.

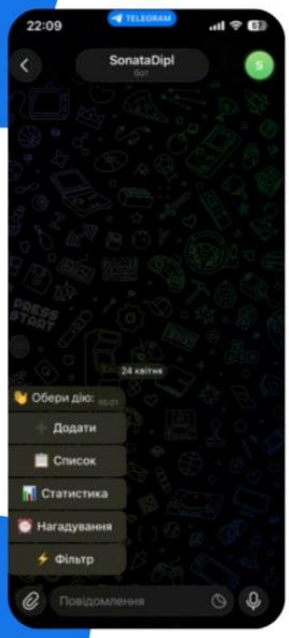
Масштабованість
Система повинна підтримувати збільшення кількості користувачів та задач без втрати продуктивності.

Аналіз конкурентів

Порівняння сервісів за ключовими показниками



Шкала оцінювання:
1 – дуже низький рівень
5 – дуже високий рівень



Унікальність системи

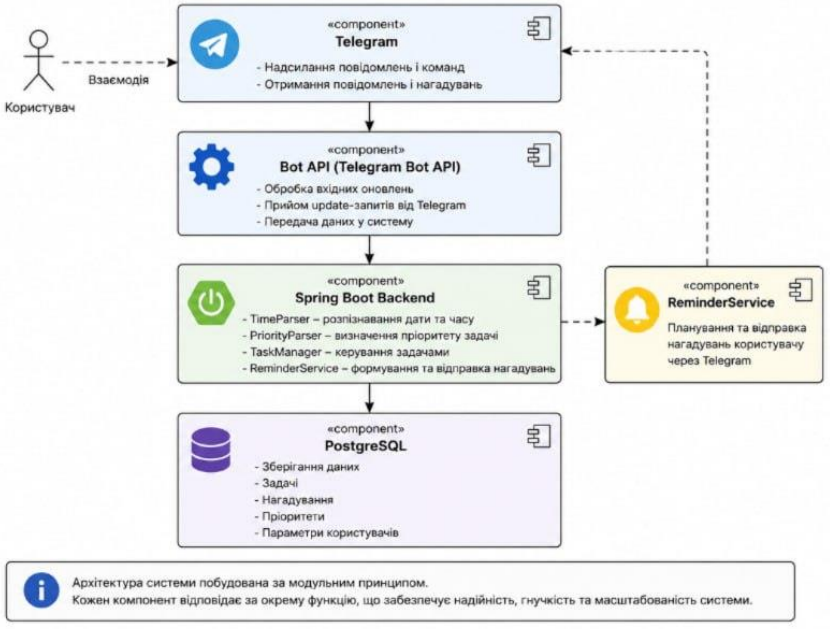
1. **Telegram-бот як єдиний інтерфейс**
Створення та управління задачами через чат Telegram

2. **Система нагадувань**
Автоматичне надсилання нагадувань користувачу

3. **Автоматичне визначення пріоритету**
Фільтрація задач за важливістю

4. **Інтелектуальна обробка тексту**
Аналіз повідомлень користувача через TimeParser і PriorityParser

Архітектура системи



Архітектура системи побудована за модульним принципом.
Кожен компонент відповідає за окрему функцію, що забезпечує надійність, гнучкість та масштабованість системи.

Технологічний стек

Java
Основна мова реалізації backend-логіки

Spring Boot
Фреймворк для побудови REST-сервісів

PostgreSQL
Реляційна БД для зберігання задач

Hibernate / JPA
ORM-маппінг та робота з даними

Telegram Bot API
Інтеграція з месенджером Telegram

Принцип роботи системи

Цифрова система автоматизації нагадувань на базі Telegram-бота

ПОСЛІДОВНІСТЬ РОБОТИ СИСТЕМИ

1. **Користувач**
   Надсилає повідомлення з задачею у Telegram.
2. **Telegram-бот**
   Отримує повідомлення та передає текст для обробки.
3. **Аналіз тексту**
   Визначає дату та час задачі, а також її пріоритет.
4. **Збереження задачі**
   Задача, час виконання та пріоритет зберігаються у базі даних.
5. **ReminderService**
   Регулярно перевіряє базу даних та надсилає нагадування у потрібний час.
6. **Надсилання нагадування**
   Бот надсилає нагадування користувачу через Telegram у встановлений час.

ОСНОВНІ КОМПОНЕНТИ СИСТЕМИ

TimeParser
Автоматично розпізнає часові вирази природної мови та визначає дату і час виконання задачі.

PriorityParser
Класифікує задачу за рівнем важливості на основі ключових слів у тексті: високий, середній, низький.

PostgreSQL
Надійне зберігання задач, часу виконання, пріоритетів та даних користувачів.

ReminderService
Регулярно перевіряє базу даних та надсилає нагадування користувачу через Telegram у потрібний час.

ПЕРЕВАГИ СИСТЕМИ

- Швидке створення задач у звичному чаті
- Автоматичне визначення часу та пріоритету
- Надійне зберігання та керування даними
- Точні та своєчасні нагадування

Демонстрація роботи бота

Відео демонстрація роботи бота

Апробаційні матеріали

1. Яременко О.А., Ананченко О.Є. Використання Telegram-ботів для автоматизації персонального планування та нагадувань // Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. К.: ДУІКТ, 2026.

1. Яременко О.А., Ананченко О.Є. Автоматичне визначення пріоритетності задач на основі аналізу текстових повідомлень користувача // Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. К.: ДУІКТ, 2026.

Висновок

Автоматизація управління задачами

Розроблена система дозволяє створювати та керувати задачами через Telegram-бота без використання окремого застосунку.

Інтелектуальна обробка тексту

Реалізовано механізми автоматичного визначення дати, часу виконання та пріоритету задач на основі текстових повідомлень.

Система нагадувань

ReminderService забезпечує автоматичне надсилання нагадувань користувачу у встановлений час.

Зберігання данних

Для зберігання задач, нагадувань і параметрів користувача використовується база даних PostgreSQL.

Тестування системи

Проведено модульне та інтеграційне тестування системи. Результати тестування підтвердили коректність роботи всіх ключових функцій.

У результаті виконання роботи було **розроблено цифрову систему автоматизації нагадувань на базі Telegram-бота.**

Запропоноване рішення дозволяє спростити процес створення та управління задачами, зменшити складність взаємодії користувача із системою планування та скоротити час на введення інформації.

Підтвердження якості

Система пройшла всі етапи тестування та відповідає встановленим вимогам до функціональності, надійності та безпеки даних.

ДОДАТОК Б

```
package com.example.sonatadiplom;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

import org.telegram.telegrambots.bots.TelegramLongPollingBot;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.methods.updatingmessages.EditMessageText;
import org.telegram.telegrambots.meta.api.objects.Message;
import org.telegram.telegrambots.meta.api.objects.Update;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.InlineKeyboardMarkup;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.InlineKeyboardButton;
import org.telegram.telegrambots.meta.api.methods.updatingmessages.DeleteMessage;


import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import java.time.LocalDateTime;



@Component
public class TelegramBot extends TelegramLongPollingBot {

    @Value("${telegram.bot.username}")
    private String username;

    @Value("${telegram.bot.token}")
    private String token;

    @Autowired
    private TaskRepository taskRepository;

    @Autowired
    private TimeParser timeParser;

    @Autowired
    private PriorityParser priorityParser;

    private Map<Long, Boolean> waitingForTask = new HashMap<>();
    private Map<Long, Long> editingTask = new HashMap<>();
    private Map<Long, Long> waitingForReminderTask = new HashMap<>();
    private Map<Long, Boolean> waitingForTimeInput = new HashMap<>();
    private Map<Long, Integer> mainMessageId = new HashMap<>();
    private Map<Long, Task> tempTask = new HashMap<>();
    private Map<Long, Boolean> waitingForReminderTime = new HashMap<>();
    private Map<Long, String> filter = new HashMap<>();
    private Map<Long, String> state = new HashMap<>();


    @Override
    public String getBotUsername() {
        return username;
    }

    @Override
    public String getBotToken() {
        return token;
    }

    @Override
    public void onUpdateReceived(Update update) {

        // ================ CALLBACK ================
        if (update.hasCallbackQuery()) {
            handleCallback(update);
            return;
        }
```

```
// ================ TEXT ================
if (!update.hasMessage() || !update.getMessage().hasText()) return;

String text = update.getMessage().getText();
Long chatId = update.getMessage().getChatId();
Integer userMessageId = update.getMessage().getMessageId();

String currentState = state.get(chatId);

if ("EDIT_TEXT".equals(currentState)) {

    Long id = editingTask.get(chatId);
    Task task = taskRepository.findById(id).orElse(null);

    if (task != null) {
        task.setText(text);
        taskRepository.save(task);
    }

    state.remove(chatId);
    editingTask.remove(chatId);

    updateMainMessage(chatId, "✅ Текст оновлено", null);
    sendTasksInline(chatId);
    return;
}

if ("EDIT_TIME".equals(currentState)) {

    try {
        Long id = editingTask.get(chatId);
        Task task = taskRepository.findById(id).orElse(null);

        if (task != null) {
            task.setReminderTime(timeParser.parse(text));
            taskRepository.save(task);
        }

        updateMainMessage(chatId, "✅ Час оновлено", null);

    } catch (Exception e) {
        updateMainMessage(chatId, "❌ Невірний формат часу", null);
        return;
    }

    state.remove(chatId);
    editingTask.remove(chatId);

    sendTasksInline(chatId);
    return;
}

// start
if (text.equals("/start")) {
    showMainMenu(chatId);
    deleteMessageLater(chatId, userMessageId, 5);
    return;
}


// додавання
if ("ADD_TASK".equals(state.get(chatId))) {



    Task task = new Task();

    String originalText = text;

    // AI время
    LocalDateTime reminderTime = timeParser.parseSmartTime(originalText);
    System.out.println("AI detected time: " + reminderTime);

    // чистка
    String cleanText = timeParser.cleanTaskText(originalText);

    task.setText(cleanText);
```

```
    task.setChatId(chatId);

    if (reminderTime != null) {
        task.setReminderTime(reminderTime);
    }

    // AI приорітет
    task.setPriority(priorityParser.detectPriority(originalText));
    System.out.println("AI detected priority: " + task.getPriority());

    // if користувач не написав час ставимо автоматично
    if (task.getReminderTime() == null) {

        switch (task.getPriority()) {

            case HIGH:
                task.setReminderTime(LocalDateTime.now().plusMinutes(30));
                break;

            case MEDIUM:
                task.setReminderTime(LocalDateTime.now().plusHours(3));
                break;

            case LOW:
                task.setReminderTime(LocalDateTime.now().plusDays(1));
                break;
        }
    }



    tempTask.put(chatId, task);

    InlineKeyboardMarkup kb = new InlineKeyboardMarkup();

    InlineKeyboardButton none = new InlineKeyboardButton();
    none.setText("❌ Без повтору");
    none.setCallbackData("repeat_NONE");

    InlineKeyboardButton daily = new InlineKeyboardButton();
    daily.setText("🔁 Щодня");
    daily.setCallbackData("repeat_DAILY");

    InlineKeyboardButton weekly = new InlineKeyboardButton();
    weekly.setText("📅 Щотижня");
    weekly.setCallbackData("repeat_WEEKLY");

    InlineKeyboardButton cancel = new InlineKeyboardButton();
    cancel.setText("❌ Скасувати");
    cancel.setCallbackData("cancel");

    InlineKeyboardButton done = new InlineKeyboardButton();
    done.setText("✅ Створити");
    done.setCallbackData("create_task");

    kb.setKeyboard(List.of(
            List.of(none),
            List.of(daily),
            List.of(weekly),
            List.of(done, cancel)
    ));

    updateMainMessage(chatId, "🔁 Обери повтор задачі:", kb);

    waitingForTask.put(chatId, false);

    deleteMessageLater(chatId, userMessageId, 2);

    return;
}


// час
if (waitingForTimeInput.getOrDefault(chatId, false)) {

    try {
        Long id = waitingForReminderTask.get(chatId);
```

```
                Task task = taskRepository.findById(id).orElse(null);

                if (task != null) {
                    task.setReminderTime(timeParser.parse(text));
                    taskRepository.save(task);

                    updateMainMessage(chatId, "✅ Нагадування встановлено", null);
                }

            } catch (Exception e) {
                updateMainMessage(chatId,
                        "❌ Не зрозумів час\n\nСпробуй:\n" +
                                "• через 10 хв\n" +
                                "• через 2 год\n" +
                                "• завтра 09:30\n" +
                                "• 12.08.2026 18:30",
                        null);
                return;
            }

            waitingForTimeInput.put(chatId, false);
            waitingForReminderTask.remove(chatId);

            sendTasksInline(chatId);
            return;
        }

        sendMessage(chatId, "❗ Використовуй кнопки");
        deleteMessageLater(chatId, userMessageId, 3);
    }

// =============== UI ===============

    private void showMainMenu(Long chatId) {

        InlineKeyboardMarkup kb = new InlineKeyboardMarkup();



        InlineKeyboardButton add = new InlineKeyboardButton("➕ Додати");
        add.setCallbackData("menu_add");

        InlineKeyboardButton list = new InlineKeyboardButton("📋 Список");
        list.setCallbackData("menu_list");

        InlineKeyboardButton stats = new InlineKeyboardButton("📊 Статистика");
        stats.setCallbackData("menu_stats");

        InlineKeyboardButton reminders = new InlineKeyboardButton("⏰ Нагадування");
        reminders.setCallbackData("menu_reminders");

        InlineKeyboardButton filterBtn = new InlineKeyboardButton("⚡ Фільтр");
        filterBtn.setCallbackData("menu_filter");


        kb.setKeyboard(List.of(
                List.of(add),
                List.of(list),
                List.of(stats),
                List.of(reminders),
                List.of(filterBtn)
        ));

        updateMainMessage(chatId, "👏 Обери дію:", kb);
    }

    private void sendTasksInline(Long chatId) {

        List<Task> tasks = taskRepository.findByChatId(chatId);

        tasks = tasks.stream()
                .sorted((a, b) -> {
                    boolean aOverdue = a.getReminderTime() != null &&
a.getReminderTime().isBefore(LocalDateTime.now());
                    boolean bOverdue = b.getReminderTime() != null &&
b.getReminderTime().isBefore(LocalDateTime.now());
```

```
                if (aOverdue && !bOverdue) return -1;
                if (!aOverdue && bOverdue) return 1;

                return b.getPriority().ordinal() - a.getPriority().ordinal();
            })
            .toList();

    String f = filter.get(chatId);

    if (f != null && !"ALL".equals(f)) {
        tasks = tasks.stream()
                .filter(t -> t.getPriority().name().equals(f))
                .toList();
    }

    if (tasks.isEmpty()) {

        InlineKeyboardMarkup kb = new InlineKeyboardMarkup();

        InlineKeyboardButton back = new InlineKeyboardButton();
        back.setText("⬅ Назад");
        back.setCallbackData("menu");

        kb.setKeyboard(List.of(List.of(back)));

        updateMainMessage(chatId, "📭 Список задач пустий", kb);
        return;
    }

    InlineKeyboardMarkup kb = new InlineKeyboardMarkup();
    List<List<InlineKeyboardButton>> rows = new ArrayList<>();

    for (Task t : tasks) {

        InlineKeyboardButton btn = new InlineKeyboardButton();

        String priorityEmoji = switch (t.getPriority()) {
            case HIGH -> "🔴";
            case LOW -> "🟢";
            default -> "🟡";
        };

        String text = priorityEmoji + (t.isCompleted() ? "✔ " : "") + t.getText();

        if (t.getReminderTime() != null && t.getReminderTime().isBefore(LocalDateTime.now())) {
            text = "⚠ " + text;
        }

        btn.setText(text);
        btn.setCallbackData("task_" + t.getId());

        InlineKeyboardButton edit = new InlineKeyboardButton();
        edit.setText("✏");
        edit.setCallbackData("edit_" + t.getId());

        rows.add(List.of(btn, edit)); // ☑ тільки це
    }

    InlineKeyboardButton back = new InlineKeyboardButton("⬅ Назад");
    back.setCallbackData("menu");



    rows.add(List.of(back));

    kb.setKeyboard(rows);

    updateMainMessage(chatId, "📋 Твої задачі:", kb);
}

private void handleCallback(Update update) {

    String data = update.getCallbackQuery().getData();
    Long chatId = update.getCallbackQuery().getMessage().getChatId();

    // назад
```

```
if (data.equals("menu")) {
    showMainMenu(chatId);

    return;
}

// список
else if (data.equals("menu_list")) {
    sendTasksInline(chatId);
    return;
}

// додати
else if (data.equals("menu_add")) {
    state.put(chatId, "ADD_TASK");
    updateMainMessage(chatId, "✏️ Введи текст задачі:", getBackKeyboard());
    return;
}



// статистика
else if (data.equals("menu_stats")) {
    showStats(chatId);
    return;
}

else if (data.equals("menu_reminders")) {
    showReminders(chatId);
    return;
}

else if (data.startsWith("delete_")) {
    deleteTask(chatId, data);
    return;
}


else if (data.equals("menu_filter")) {

    InlineKeyboardMarkup kb = new InlineKeyboardMarkup();

    kb.setKeyboard(List.of(
            List.of(btn("🔴 HIGH", "filter_HIGH")),
            List.of(btn("🟡 MEDIUM", "filter_MEDIUM")),
            List.of(btn("🟢 LOW", "filter_LOW")),
            List.of(btn("📋 Всі", "filter_ALL")),
            List.of(btn("🔙 Назад", "menu"))
    ));

    updateMainMessage(chatId, "⚡ Обери фільтр:", kb);
    return;
}

else if (data.equals("cancel")) {

    waitingForTask.put(chatId, false);
    waitingForReminderTime.put(chatId, false);
    waitingForReminderTask.remove(chatId);
    tempTask.remove(chatId);

    updateMainMessage(chatId, "❌ Скасовано", null);
    sendTasksInline(chatId);

    return;
}

else if (data.equals("edit_text")) {
    state.put(chatId, "EDIT_TEXT");
    updateMainMessage(chatId, "✏️ Введи новий текст:", getBackKeyboard());
    return;
}

else if (data.equals("edit_time")) {
    state.put(chatId, "EDIT_TIME");
    updateMainMessage(chatId, "⏰ Введи новий час:", getBackKeyboard());
```

```
    return;
}

else if (data.equals("edit_priority")) {

    InlineKeyboardMarkup kb = new InlineKeyboardMarkup();

    kb.setKeyboard(List.of(
            List.of(btn("🔴 HIGH", "priority_HIGH")),
            List.of(btn("🟡 MEDIUM", "priority_MEDIUM")),
            List.of(btn("🟢 LOW", "priority_LOW")),
            List.of(btn("⬅️ Назад", "menu"))
    ));

    updateMainMessage(chatId, "🔥 Обери пріоритет:", kb);

    return;
}

else if (data.equals("create_task")) {

    Task task = tempTask.get(chatId);

    if (task != null) {
        taskRepository.save(task);
        tempTask.remove(chatId);

        state.remove(chatId);

        updateMainMessage(chatId, "✅ Задача створена", null);
        sendTasksInline(chatId);
    }

    return;
}

else if (data.startsWith("reminder_")) {

    Long id = Long.parseLong(data.replace("reminder_", ""));
    Task task = taskRepository.findById(id).orElse(null);

    if (task == null) return;

    InlineKeyboardMarkup kb = new InlineKeyboardMarkup();

    InlineKeyboardButton edit = new InlineKeyboardButton();
    edit.setText("✏️ Змінити час");
    edit.setCallbackData("edit_reminder_" + id);

    InlineKeyboardButton delete = new InlineKeyboardButton();
    delete.setText("❌ Видалити нагадування");
    delete.setCallbackData("remind_delete_" + id);

    InlineKeyboardButton back = new InlineKeyboardButton();
    back.setText("⬅️ Назад");
    back.setCallbackData("menu_reminders");

    kb.setKeyboard(List.of(
            List.of(edit),
            List.of(delete),
            List.of(back)
    ));

    updateMainMessage(chatId,
            "⏰ " + task.getText() + "\n\n" +
                    "Поточний час: " + task.getReminderTime(),
            kb
    );

    return;
}

else if (data.startsWith("priority_")) {

    String priorityStr = data.replace("priority_", "");
```

```
    Priority priority = Priority.valueOf(priorityStr);

    Long id = editingTask.get(chatId);
    Task task = taskRepository.findById(id).orElse(null);



    if (task != null) {
        task.setPriority(priority);
        taskRepository.save(task);

        updateMainMessage(chatId, "🔥 Пріоритет оновлено!", null);
        sendTasksInline(chatId);
    }

    state.remove(chatId);
    editingTask.remove(chatId);

    return;
}

else if (data.startsWith("filter_")) {

    String f = data.replace("filter_", "");
    filter.put(chatId, f);

    sendTasksInline(chatId);
    return;
}

else if (data.startsWith("set_priority_")) {

    String p = data.replace("set_priority_", "");

    Priority priority = Priority.valueOf(p);

    Long taskId = editingTask.get(chatId);
    Task task = taskRepository.findById(taskId).orElse(null);

    if (task != null) {
        task.setPriority(priority);
        taskRepository.save(task);
    }

    updateMainMessage(chatId, "✅ Пріоритет оновлено", null);
    sendTasksInline(chatId);

    return;
}

else if (data.startsWith("edit_")) {

    Long taskId = Long.parseLong(data.replace("edit_", ""));
    editingTask.put(chatId, taskId);
    state.put(chatId, "EDIT_MENU");

    InlineKeyboardMarkup kb = new InlineKeyboardMarkup();

    InlineKeyboardButton text = new InlineKeyboardButton();
    text.setText("✏️ Текст");
    text.setCallbackData("edit_text");

    InlineKeyboardButton time = new InlineKeyboardButton();
    time.setText("⏰ Час");
    time.setCallbackData("edit_time");

    InlineKeyboardButton priority = new InlineKeyboardButton();
    priority.setText("⚡ Пріоритет");
    priority.setCallbackData("edit_priority");

    InlineKeyboardButton cancel = new InlineKeyboardButton();
    cancel.setText("⬅️ Назад");
    cancel.setCallbackData("cancel");

    kb.setKeyboard(List.of(
            List.of(text),
            List.of(time),
```

```
                List.of(priority),
                List.of(cancel)
        ));

        updateMainMessage(chatId, "✏️ Що редагуємо?", kb);

        return;
    }

    else if (data.startsWith("edit_reminder_")) {

        Long id = Long.parseLong(data.replace("edit_reminder_", ""));

        waitingForReminderTask.put(chatId, id);
        waitingForTimeInput.put(chatId, true);

        updateMainMessage(chatId, "⏰ Введи новий час:", null);
        return;
    }

    else if (data.startsWith("repeat_")) {

        String repeat = data.replace("repeat_", "");

        Task task = tempTask.get(chatId);

        if (task != null) {
            task.setRepeatType(repeat);
            tempTask.put(chatId, task);

            updateMainMessage(chatId,
                    "🔁 Повтор: " + repeat + "\n\nНатисни 'Створити'",
                    getRepeatKeyboard()
            );
        }

        return;
    }

    else if (data.startsWith("snooze_")) {

        String[] parts = data.split("_");
        int minutes = Integer.parseInt(parts[1]);
        Long id = Long.parseLong(parts[2]);

        Task task = taskRepository.findById(id).orElse(null);

        if (task != null) {
            task.setReminderTime(LocalDateTime.now().plusMinutes(minutes));
            taskRepository.save(task);

            updateMainMessage(chatId,
                    "⏰ Відкладено на " + minutes + " хв",
                    null
            );
        }

        return;
    }

    // открить задач
    else if (data.startsWith("task_")) {
        openTask(chatId, data);
        return;
    }



    // ремайнд
    if (data.startsWith("remind_")) {



        Long id = Long.parseLong(data.replace("remind_", ""));
        waitingForReminderTask.put(chatId, id);
        waitingForTimeInput.put(chatId, true);

        updateMainMessage(chatId, "⏰ Введи час:", null);
```

```
            return;
        }

        if (data.startsWith("remind_delete_")) {

            Long id = Long.parseLong(data.replace("remind_delete_", ""));
            Task task = taskRepository.findById(id).orElse(null);

            if (task != null) {
                task.setReminderTime(null);
                taskRepository.save(task);
            }

            showReminders(chatId);
            return;
        }


        else if (data.startsWith("done_")) {
            toggleTask(chatId, data);
            return;
        }

        return;
    }


// =============== HELPERS ================

    private void sendMessage(Long chatId, String text) {

        SendMessage msg = new SendMessage();
        msg.setChatId(chatId.toString());
        msg.setText(text);

        try {
            execute(msg);
        } catch (Exception ignored) {}
    }

    private void updateMainMessage(Long chatId, String text, InlineKeyboardMarkup kb) {

        try {

            if (!mainMessageId.containsKey(chatId)) {

                SendMessage msg = new SendMessage();
                msg.setChatId(chatId.toString());
                msg.setText(text);
                msg.setReplyMarkup(kb);

                Message sent = execute(msg);
                mainMessageId.put(chatId, sent.getMessageId());

            } else {

                EditMessageText edit = new EditMessageText();
                edit.setChatId(chatId.toString());
                edit.setMessageId(mainMessageId.get(chatId));
                edit.setText(text);
                edit.setReplyMarkup(kb);

                execute(edit);
            }

        } catch (Exception e) {
            if (!e.getMessage().contains("message is not modified")) {
                e.printStackTrace();
            }
        }
    }


    private void showReminders(Long chatId) {

        List<Task> tasks = taskRepository.findByChatId(chatId);

        List<Task> reminders = tasks.stream()
                .filter(t -> t.getReminderTime() != null)
```

```
                .toList();

        if (reminders.isEmpty()) {

            InlineKeyboardMarkup kb = new InlineKeyboardMarkup();

            InlineKeyboardButton back = new InlineKeyboardButton();
            back.setText("⬅ Назад");
            back.setCallbackData("menu");

            kb.setKeyboard(List.of(List.of(back)));

            updateMainMessage(chatId, "📭 Немає нагадувань", kb);
            return;
        }
        List<List<InlineKeyboardButton>> rows = new ArrayList<>();

        for (Task t : reminders) {

            InlineKeyboardButton btn = new InlineKeyboardButton();
            btn.setText("⏰ " + t.getText() + "\n🕒 " + formatTime(t.getReminderTime()));
            btn.setCallbackData("task_" + t.getId());

            rows.add(List.of(btn));
        }

        InlineKeyboardButton back = new InlineKeyboardButton("⬅ Назад");
        back.setCallbackData("menu");

        rows.add(List.of(back));

        InlineKeyboardMarkup kb = new InlineKeyboardMarkup();
        kb.setKeyboard(rows);

        updateMainMessage(chatId, "⏰ Твої нагадування:", kb);
    }

    private void deleteMessageLater(Long chatId, Integer messageId, int delaySeconds) {
        new Thread(() -> {
            try {
                Thread.sleep(delaySeconds * 1000L);
                deleteMessage(chatId, messageId);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }).start();
    }

    private String formatTime(java.time.LocalDateTime time) {

        if (time == null) return "";

        java.time.LocalDate today = java.time.LocalDate.now();

        if (time.toLocalDate().equals(today)) {
            return "сьогодні о " + time.toLocalTime().withSecond(0).withNano(0);
        }



        if (time.toLocalDate().equals(today.plusDays(1))) {
            return "завтра о " + time.toLocalTime().withSecond(0).withNano(0);
        }

        return time.toLocalDate() + " о " + time.toLocalTime().withSecond(0).withNano(0);
    }

    private void showStats(Long chatId) {

        List<Task> tasks = taskRepository.findByChatId(chatId);

        int total = tasks.size();
        int completed = (int) tasks.stream().filter(Task::isCompleted).count();
        int active = total - completed;

        int withReminders = (int) tasks.stream()
                .filter(t -> t.getReminderTime() != null)
                .count();
```

```
    int overdue = (int) tasks.stream()
            .filter(t -> t.getReminderTime() != null &&
                    t.getReminderTime().isBefore(LocalDateTime.now()))
            .count();

    double percent = total == 0 ? 0 : (completed * 100.0 / total);

    String text =
            "📊 *Твоя статистика*\n\n" +
                    "📌 Всього задач: " + total + "\n" +
                    "✅ Виконано: " + completed + "\n" +
                    "🕒 Активні: " + active + "\n\n" +
                    "⏰ З нагадуваннями: " + withReminders + "\n" +
                    "⚠️ Прострочені: " + overdue + "\n\n" +
                    "📈 Прогрес: " + String.format("%.1f", percent) + "%";

    InlineKeyboardMarkup kb = new InlineKeyboardMarkup();
    InlineKeyboardButton back = new InlineKeyboardButton();
    back.setText("⬅️ Назад");
    back.setCallbackData("menu");

    kb.setKeyboard(List.of(List.of(back)));

    updateMainMessage(chatId, text, kb);
}

public void sendExternalMessageWithKeyboard(Long chatId, String text, InlineKeyboardMarkup kb) {
    SendMessage message = new SendMessage();
    message.setChatId(chatId.toString());
    message.setText(text);
    message.setReplyMarkup(kb);

    try {
        execute(message);
    } catch (Exception e) {
        e.printStackTrace();
    }
}


private InlineKeyboardButton btn(String text, String data) {
    InlineKeyboardButton b = new InlineKeyboardButton();
    b.setText(text);
    b.setCallbackData(data);
    return b;
}

private void toggleTask(Long chatId, String data) {

    Long id = Long.parseLong(data.replace("done_", ""));
    Task task = taskRepository.findById(id).orElse(null);

    if (task != null) {
        task.setCompleted(!task.isCompleted());
        taskRepository.save(task);
    }

    sendTasksInline(chatId);
}

private void deleteTask(Long chatId, String data) {

    Long id = Long.parseLong(data.replace("delete_", ""));
    taskRepository.deleteById(id);

    updateMainMessage(chatId, "🗑 Видалено", null);
    sendTasksInline(chatId);
}

private void openTask(Long chatId, String data) {

    Long id = Long.parseLong(data.replace("task_", ""));
    Task task = taskRepository.findById(id).orElse(null);

    if (task == null) return;
```

```
        InlineKeyboardMarkup kb = new InlineKeyboardMarkup();

        kb.setKeyboard(List.of(
                List.of(btn(task.isCompleted() ? "↩️ Скасувати" : "✔️ Виконано", "done_" + id)),
                List.of(btn("⏰ Нагадати", "remind_" + id)),
                List.of(btn("✏️ Редагувати", "edit_" + id)),
                List.of(btn("❌ Видалити", "delete_" + id)),
                List.of(btn("⬅️ Назад", "menu_list"))
        ));

        updateMainMessage(chatId, "📌 " + task.getText(), kb);
    }

    private InlineKeyboardMarkup getRepeatKeyboard() {

        InlineKeyboardMarkup kb = new InlineKeyboardMarkup();

        InlineKeyboardButton none = new InlineKeyboardButton();
        none.setText("❌ Без повтору");
        none.setCallbackData("repeat_NONE");

        InlineKeyboardButton daily = new InlineKeyboardButton();
        daily.setText("🔁 Щодня");
        daily.setCallbackData("repeat_DAILY");



        InlineKeyboardButton weekly = new InlineKeyboardButton();
        weekly.setText("📆 Щотижня");
        weekly.setCallbackData("repeat_WEEKLY");

        InlineKeyboardButton cancel = new InlineKeyboardButton();
        cancel.setText("❌ Скасувати");
        cancel.setCallbackData("cancel");

        InlineKeyboardButton done = new InlineKeyboardButton();
        done.setText("☑️ Створити");
        done.setCallbackData("create_task");

        kb.setKeyboard(List.of(
                List.of(none),
                List.of(daily),
                List.of(weekly),
                List.of(done, cancel)
        ));

        return kb;
    }

    private void deleteMessage(Long chatId, Integer messageId) {
        try {
            DeleteMessage delete = new DeleteMessage();
            delete.setChatId(chatId.toString());
            delete.setMessageId(messageId);
            execute(delete);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private InlineKeyboardMarkup getBackKeyboard() {

        InlineKeyboardMarkup kb = new InlineKeyboardMarkup();

        InlineKeyboardButton back = new InlineKeyboardButton();
        back.setText("⬅️ Назад");
        back.setCallbackData("menu");

        kb.setKeyboard(List.of(
                List.of(back)
        ));

        return kb;
    }

}
```