МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«До захисту допущено»

В. о. завідувач кафедри

_____ Ігор ШЕЛЕХОВ

(підпис)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 – Комп'ютерні науки,

освітньо-професійної програми «Інформатика»

на тему: «Інформаційна система тайм-менеджменту користувачів»

здобувача групи ІН-06-2 Котельви Максима Андрійовича

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

_____ Максим КОТЕЛЬВА

(підпис)

Керівник, доцент,
кандидат фізико-математичних наук,
доцент Галина ОЛЕКСІЄНКО _____

(підпис)

Суми – 2024

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

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

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерні науки, освітньо-професійної програми «Інформатика»
здобувача групи ІН-06-2 Котельви Максима Андрійовича

1. Тема роботи: «Інформаційна система тайм-менеджменту користувачів»
затверджую наказом по СумДУ від 22 квітня 2024 р. № 0414-VI
2. Термін здачі здобувачем кваліфікаційної роботи 28 травня 2024 р.
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для розробки інформаційного та програмного забезпечення системи. 3) Розробка інформаційної системм тайм-менеджменту користувачів. 4) Тестування і аналіз результатів
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «08» квітня 2024 р.

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

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапі кваліфікаційної роботи	Терміни виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*	*06.05 – 10.05*	
2	*Огляд технологій, що використовуються для розробки інформаційної системи тайм-менеджменту користувачів*	*11.05 – 14.05*	
3	*Розробка ІС тайм-менеджменту користувачів*	*15.05 – 23.05*	
4	*Тестування і аналіз отриманих результатів*	*24.05 – 25.05*	
5	*Оформлення пояснювальної записки*	*26.05 – 28.05*	

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

АНОТАЦІЯ

Записка: 45 стр., 15 рис., 3 додатки, 14 використаних джерел

Обґрунтування актуальності теми роботи – велика кількість завдань та обов'язків, як у професійному, так і в особистому житті людей робить тему цієї кваліфікаційної роботи затребуваною та актуальною.

Об’єктом дослідження – інформаційна система тайм-менеджменту.

Предметом дослідження – процеси тайм-менеджменту користувачів.

Мета роботи – створення інформаційної системи для полегшення процесу ведення особистих справ та управління часом.

Методи дослідження – пошук, огляд, аналіз та порівняння існуючих рішень, розгляд користувацького досвіду, моделювання та тестування системи.

Результати – сформульовано постановку задачі на основі огляду предметної області. Відповідно до визначених вимог обрано технології для реалізації. Спроєктовано інформаційну систему, що передбачало розробку функціональної моделі, діаграми прецедентів та ER-діаграму сховища. Реалізовано інформаційну систему тайм-менеджменту користувачів у вигляді Telegram боту.

ІНФОРМАЦІЙНА СИСТЕМА, ТАЙМ-МЕНЕДЖМЕНТ, TELEGRAM, BOT, PYTHON

ЗМІСТ

ВСТУП

Актуальність. Люди в нинішньому світі не рідко стикаються з великою кількістю завдань та обов'язків, як у професійному, так і в особистому житті. Таке зростання обсягу та складності інформації може значно ускладнювати її обробку та пріоритезацію. Також зростання віддаленої роботи та мобільності робить ще більш важливим для людей вміти ефективно самоорганізовуватися та управляти своїм часом.

Ефективне управління часом – це вміння володіти найціннішим ресурсом – своїм часом. Це запорука успіху у всіх сферах життя, будь то робота, навчання, особисті справи чи саморозвиток. Планування дає змогу чітко окреслити цілі та розробити стратегію їх досягнення. Завдяки продуманому підходу до розподілу часу між різними завданнями та пріоритетами, людина стає більш продуктивною та організованою.

Актуальність даної теми підтверджує не мала кількість існуючих сервісів, котрі створені з метою полегшити керування особистим часом, що в свою чергу свідчить про попит на рішення даної проблеми.

Об’єкт дослідження. Інформаційна система тайм-менеджменту.

Предмет дослідження. Процеси тайм-менеджменту користувачів.

Мета роботи. Створення інформаційної системи для полегшення процесу ведення особистих справ та управління часом.

Гіпотеза. Розроблена інформаційна система матиме функціонал, що може допомогти його користувачам краще організувати свій час, підвищити продуктивність та досягти своїх цілей.

Новизна. Полягає в інтеграції функціоналу інформаційної системи для управління особистими справами та часом з популярною платформою Telegram у формі боту. Цей бот пропонує такі можливості, як базові операції з завданнями, групування завдань, нагадування, а також відстеження активності користувача. Завдяки цьому користувачі отримують зручний інструмент для організації свого часу безпосередньо в улюбленому

месенджері, що підвищує ефективність та доступність управління завданнями.

Структура. Дана робота складається із вступу, інформаційного огляду, постановки задачі, методів рішення поставленої задачі, опису програмної реалізації інформаційної системи, висновків та списку використаних джерел інформації.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Огляд предметної області

Ця тема завжди була актуальною, адже організація та планування особистого часу – це ключ до кращого життя. Ще до появи електронних інструментів обробки інформації люди використовували переважно ручні методи. Для цього заводилися паперові щоденники, куди записувалися списки завдань та обов'язків, ставилися позначки про їх виконання. Ці методи, хоч і не позбавлені певної користі, потребували значних зусиль та часу для ведення обліку.

Зрозуміло, що з стрімким розвитком інформаційних технологій, котрі змінили наш світ і принесли безліч нових можливостей, дане питання вирішили в рамках нових можливостей.

С самого початку появи масових персональних комп’ютерів у 80-х роках попереднього століття люди почали використовувати їх для полегшення планування та ведення обліку. Перші програми, що можна було використовувати для створення списків завдань, були розроблені для особистих комп'ютерів, таких як IBM PC та Apple II. Ці програми, хоч і примітивні за сучасними мірками, але давали змогу виконувати необхідне.

Проте, справжній поштовх до розвитку та популяризації таких інформаційних систем належить 2000-м рокам, коли Інтернет став більш доступним для користувачів і почали з'являтися перші вебсервіси для управління часом.

Сьогодні існує безліч інструментів, які полегшують ведення особистих справ та управління часом. Після аналізу різноманітної літератури, включаючи вебресурси та рейтинги, можна скласти перелік найпопулярніших сервісів:

- Microsoft To Do;
- Todoist;
- Google Tasks;
- Any.do;

- TickTick.

Всі ці сервіси надають можливість створювати завдання. Вони різняться за функціональними можливостями, такими як можливість збагачення додатковою інформацією, включаючи структуруючі елементи, а також зовнішніми характеристиками та підтримкою різних платформ.

Серед наведених сервісів, за деякими даними, Microsoft To Do є найбільш популярним, що продемонстровано на рисунку 1.1 Початково цей сервіс був відомий як Wunderlist, але після його придбання Microsoft у 2015 році, він був переформатований у Microsoft To Do. Він є безкоштовним і пропонує базові можливості для ведення особистих справ, включаючи можливість організовувати списки та планувати завдання. Його інтерфейс простий, а додавання завдань швидке. Основна перевага – глибока інтеграція з екосистемою Microsoft.

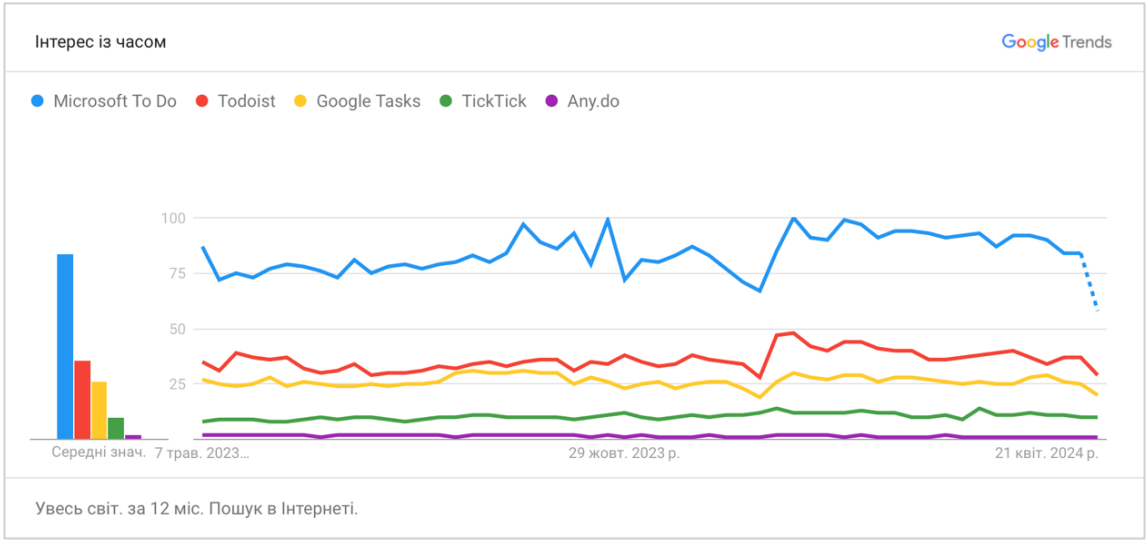

Рисунок 1.1 – Порівняння популярності сервісів в Google Trends

Todoist – це популярний частково безкоштовний сервіс, який балансує між потужністю та простотою. Цей продукт є піонером на ринку та, відповідно, одним із його лідерів. Компанія, що розробляє цей продукт, існує з 2007 року. За час свого існування компанія, не маючи значних інвестицій на

протязі довгого періоду часу, конкурує на одному з найбільш насичених ринків – програми зі списками справ [1].

Todoist пропонує швидке додавання завдань з обробкою природної мови, гнучкі функції для організації проєктів та завдань, такі як встановлення міток, пріоритетів та часу виконання, а також нагадування. Крім того, він має систему для створення розширень або інтеграцій, в раках котрих реалізовано багато нових можливостей та взаємодію з великою кількістю сервісів, таких як Gmail, Google Calendar, Microsoft Teams, Outlook та Jira. Він регулярно отримує нові функції, такі як нещодавно додані дошки Kanban та AI-помічник.

Google Tasks – це простий та безкоштовний сервіс, можливості котрого схожі з Microsoft To Do. Основною його перевагою є глибока інтеграція з сервісами компанії розробника, наприклад такими як Gmail та Google Calendar.

TickTick – це частково безкоштовний, багатофункціональний сервіс, який, подібно до Todoist, пропонує швидке додавання завдань за допомогою обробки природної мови і гнучкі інструменти для їх організації та планування. Також має власний API, але на відміну від Todoist не надає можливості створювати розширення для сервісу. Крім цього, відрізняється такими унікальними особливостями як:

- вбудований таймер;
- інструмент відстеження звичок;
- матриця Ейзенхауера для розстановки пріоритетів.

Any.do – це також частково безкоштовний сервіс, який має схожі до Todoist, TickTick можливості створення, організації та планування завдань. Серед його ключових відмінностей зручний за відгуками користувачів мобільний додаток та функція "Сплануй мій день". Він також інтегрується з календарями Google і Outlook. Серед його недоліків можна відзначити не кращий стаціонарний додаток у порівнянні з мобільним.

В таблиці 1.1 наведено спрощене порівняння вказаних сервісів [2].

Таблиця 1.1 Спрощене порівняння сервісів

Назва	Ключові можливості	Платформи	Ціна
Todoist	Обробка природної мови, інтеграції	Web, Windows, Mac, Android, iPhone, iPad	Є безкоштовна та платна (від $5/міс.) версія
TickTick	Таймер	Web, Windows, Mac, Android, iPhone, iPad	Є безкоштовна та платна (від $3,99/міс.) версія
Microsoft To Do	Глибока інтеграція з екосистемою Microsoft	Web, Windows, Mac, Android, iPhone, iPad	Безкоштовно
Google Tasks	Глибока інтеграція з сервісами Google	Web, Android, iPhone, iPad	Безкоштовно
Any.do	Фнукція "Сплануй мій день"	Web, Windows, Mac, Android, iPhone, iPad	Є безкоштовна та платна (від $2,99/міс.) версія

Ці сервіси для керування особистими справами пропонують широкий спектр функціоналу, який може задовольнити потреби більшості користувачів. Проте важливо відмітити, що жоден з них не представлений у вигляді Telegram боту. Також варто додати, що загалом кількість рішень для керування особистими справами в рамках боту дуже мала та їх функціонал є обмеженим. Це може бути проблемою для тих, хто використовує Telegram як основний засіб комунікації та бажає мати можливість керувати своїми завданнями безпосередньо через цю платформу.

Станом на весну 2024 року, без особливих зусиль можна знайти 2 функціонуючих боти для полегшення процесу ведення особистих справ та управління часом – це Taskobot та Taski Bot, що зображені на рисунку 1.2.

Перший бот пропонує базові можливості – це створення завдань та нагадувань. Другий реалізований у формі вебдодатку та не пропонує альтернативних варіантів взаємодії.

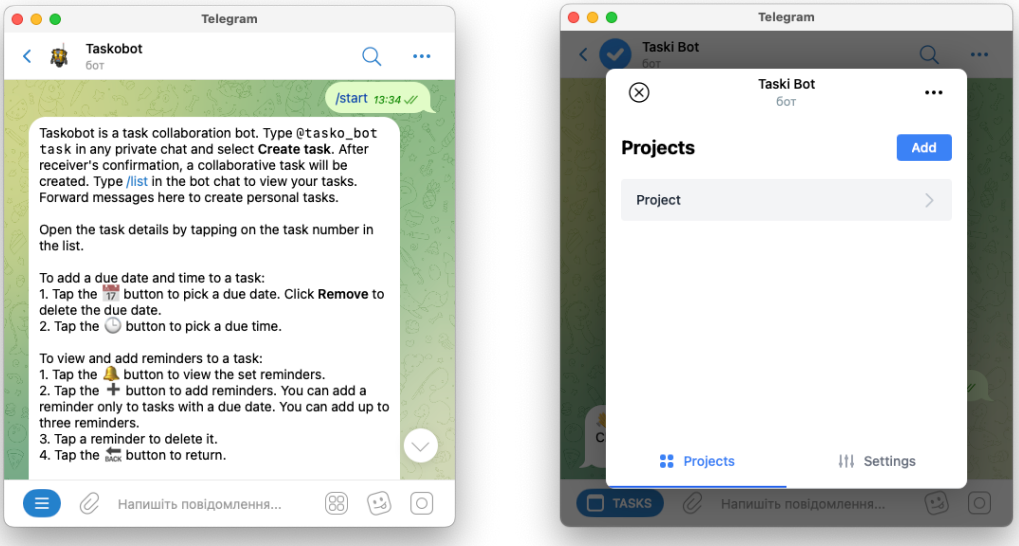

Рисунок 1.2 – Taskobot та Taski bot

Крім цього, існує бот "Telegram for Todoist", що створений ентузіастом. Як видно з назви, він забезпечує лише просту інтеграцію з Todoist і дозволяє додавати завдання до "Inbox".

1.2 Telegram

Telegram – це хмарний сервіс обміну повідомленнями для мобільних і настільних комп'ютерів, зосереджена на безпеці та швидкості [3].

Цей інструмент став важливою складовою сучасного спілкування, і його популярність зростає, особливо в контексті подій недавніх років, таких як пандемія COVID-19. Люди все частіше використовують онлайн-платформи для спілкування, роботи та навчання.

Цей месенджер вирізняється своїми широкими можливостями, високим рівнем безпеки та конфіденційності, а також зручністю використання. Telegram використовується як для особистого, так і для ділового спілкування, а також для створення тематичних груп та каналів та багато іншого.

Його щоденно використовують сотні мільйонів людей, і мати можливість організовувати та планувати свій час в тому ж додатку може бути

дуже зручним. Як уже згадувалося, для цього Telegram володіє просунутими можливостями, які дозволяють створювати в рамках цієї платформи спеціальні облікові записи – боти, що можуть стати незамінними помічниками у плануванні справ.

Боти — це спеціальні облікові записи, створені для того, щоб автоматично обробляти та відправляти повідомлення. Користувачі можуть взаємодіяти з ботами за допомогою повідомлень, що надсилаються через звичайні або групові чати. Логіка бота контролюється за допомогою HTTPS-запитів до Bot API [4].

Bot API — це інтерфейс на основі протоколу HTTPS, який дозволяє розробникам створювати Telegram ботів. Він надає доступ до різних функцій Telegram, таких як відправлення та отримання повідомлень, запуск команд, доступ до інформації про користувачів та групи тощо [5].

Для створення бота достатньо написати користувачу "@BotFather" і дотримуватися його інструкцій. Він створить бота та видасть ключ, або ж токен, авторизації, який дозволить керувати ним.

В рамках бота достатньо різноманітних засобів комунікації для реалізації різноманітних інформаційних систем. Взаємодіяти з ними користувачі можуть не лише текстовими повідомленнями, а й надсилати їм файли, геолокації, наклейки та голосові повідомлення, роблячи комунікацію більш різноманітною та зручною. Також боти ще пропонують багато інших інструментів для створення гнучких інтерфейсів, адаптованих до конкретних потреб користувачів. Наприклад, клавіатури з попередньо визначеними варіантами відповідей (рисунок 1.3) або кнопки, котрі відображаються біля повідомлень (рисунок 1.4) від бота, що спрощують навігацію та взаємодію.

Також, окрім надсилання команд і повідомлень у чат із ботом, є кілька способів взаємодії з ними, не відкриваючи жодного конкретного чату чи групи. Наприклад, "inline-режим" дозволяє надсилати запити ботам прямо з поля введення.

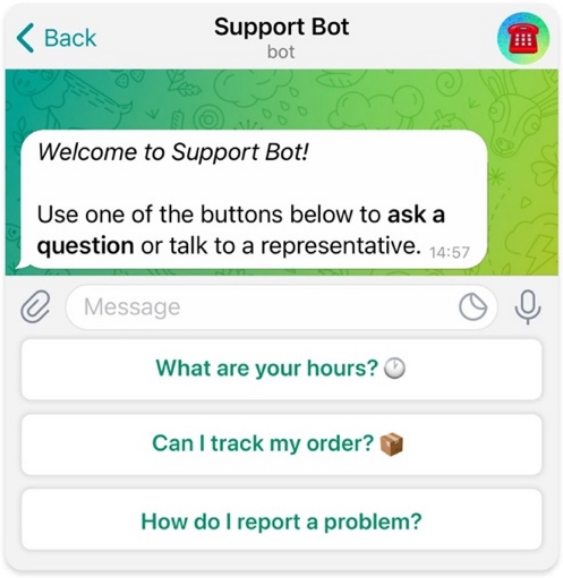

Рисунок 1.3 – Клавіатура з варіантами відповідей

Крім цього, боти за необхідністю можуть обробляти складні вхідні дані будь-якого типу через веб-програми. За допомогою цієї функції є можливість розробити будь-яку кількість гнучких, оптимізованих інтерфейсів з мовою програмування JavaScript [6].

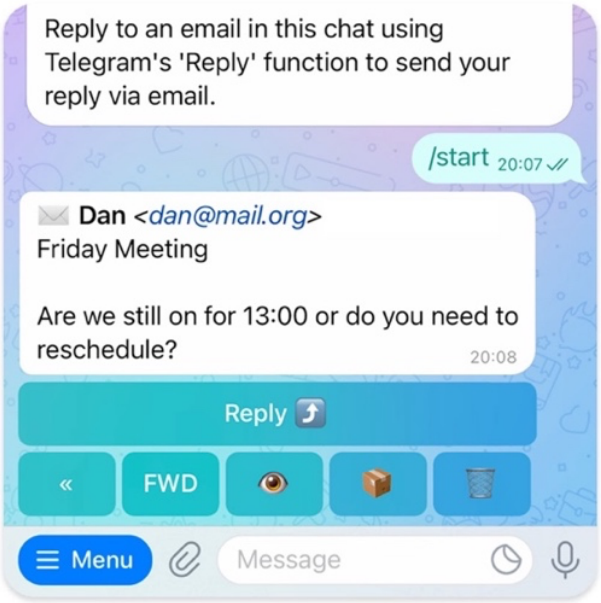

Рисунок 1.4 – Inline-keyboard

1.3 Постановка задачі

Отже, для досягнення поставленої мети необхідно розробити Telegram бота для керування завданнями. Для цього потрібно:

- спроєктувати інформаційну систему – розробити функціональну модель, ER-діаграму та змоделювати поведінку системи за допомогою діаграми прецедентів.
- обрати технології для реалізації – мову програмування, бібліотеки та СУБД;
- технічно реалізувати;
- перевірити якість та працездатність.

Для створення зручного інструменту в рамка боту необхідно реалізувати:

- базові операції з завданнями – їх створення, редагування та видалення;
- можливість групування завдань, що дозволить краще організувати свої справи;
- нагадування про завдання у визначений час.

Додатковою корисною можливістю може стати система відстеження часу виконання завдань для аналізу особистої продуктивності. Для цього варто реалізувати зручний механізм для введення даних про час виконання та генерації звітів на основі цих даних.

Реалізація функціоналу повинна бути простою та зручною для взаємодії з ботом, а саме використовуючи повідомлення, команди та різні типи клавіатури.

2 ВИБІР МЕТОДІВ РОЗВ’ЯЗАННЯ ЗАДАЧІ

2.1 Функціональне моделювання

Для кращого розуміння системи та її складових елементів можна використати методологію функціонального моделювання і графічного опису процесів IDEF0, Integrated Definition Function 0. Вона успішно застосовується в різних галузях, показавши себе ефективним засобом аналізу. Як правило, моделювання засобами IDEF0 стає першим етапом проєктування будь-якої системи.

IDEF0 призначена для формалізації та опису бізнес-процесів. Її особливістю є акцент на ієрархічне представлення об'єктів, що значно полегшує розуміння предметної області [7, 8].

Функціональна модель, або графічна діаграма, IDEF0 складається з блоків, кожен з яких представляє собою "чорний ящик", відображаючий процес або завдання. Цей блок має входи, виходи, механізми та способи управління.

IDEF0-моделі зазвичай складаються з декількох графічних діаграм. Перша діаграма в ієрархії завжди зображує функціонування системи в цілому. Такі діаграми називаються контекстними.

В рамках інформаційної системи для полегшення керування особистими справами основним завданням, або ж процесом, є обробка запитів від користувачів для створення, отримання, оновлення та видалення завдань або інших даних. Контекстна діаграма, що описує даний процес, наведена на рисунку 2.1.

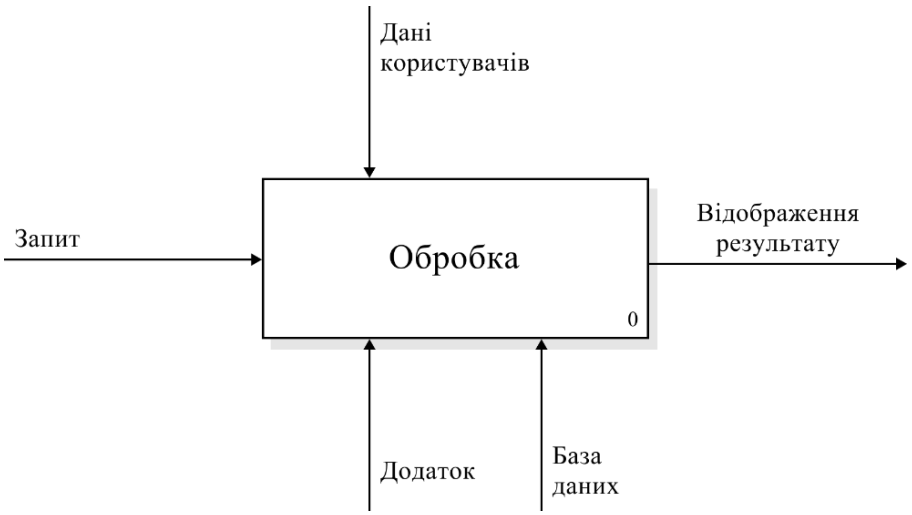

Рисунок 2.1 – Контекстна діаграма

Далі контекстна діаграма може бути деталізована, зображаючи внутрішні процеси. Такий поділ на структурні елементи називається декомпозицією. Відповідна діаграма зображена на рисунку 2.2.

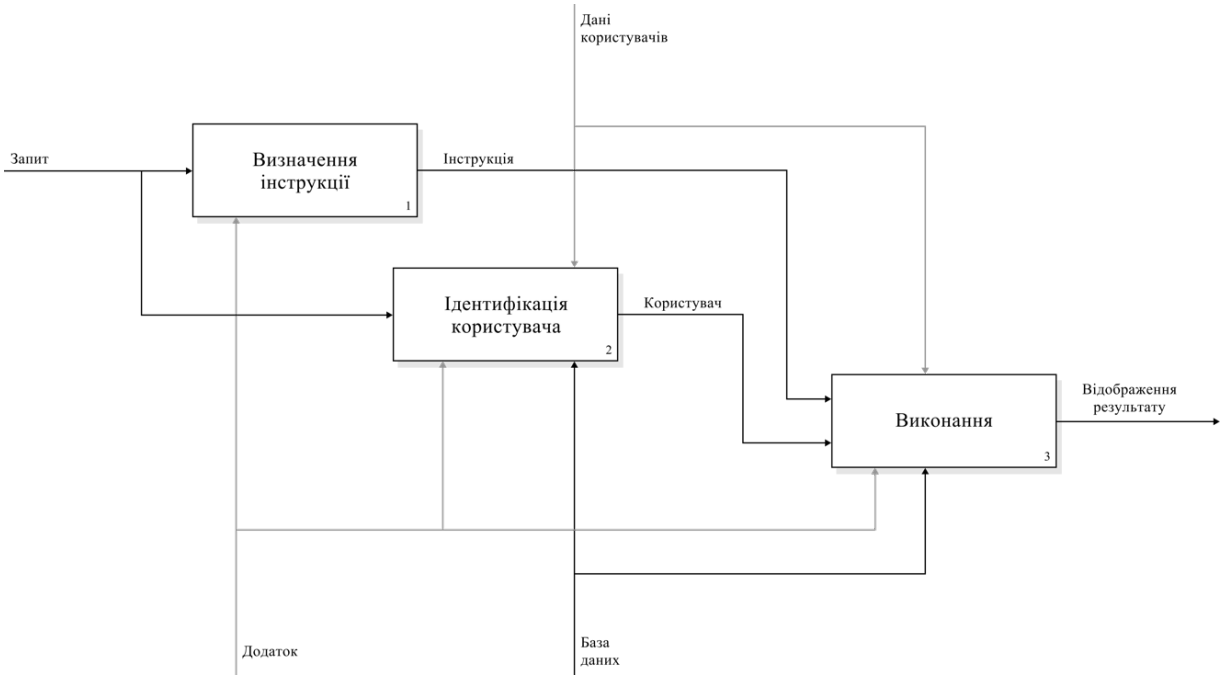

Рисунок 2.2 – Деталізована діаграма

2.2 Моделювання поведінки системи

Чітке визначення функціоналу, вимог та поведінки системи з точки зору користувача є ключовим фактором успішної розробки програмного забезпечення.

Для цього використовують діаграму прецедентів, яка є частиною мови UML (Unified Modeling Language). Діаграма наочно демонструє різні сценарії використання та типи користувачів системи. Вона має вигляд графу, що включає акторів, прецеденти, межі системи, асоціації між ними та взаємозв'язки. Ця діаграма допомагає чітко окреслити, як система буде використовуватися та як вона буде взаємодіяти з користувачами.

На рисунку 2.3 представлена діаграма ключових прецедентів для інформаційної системи тайм-менеджменту. Вона показує, які основні дії можуть виконувати користувачі системи.

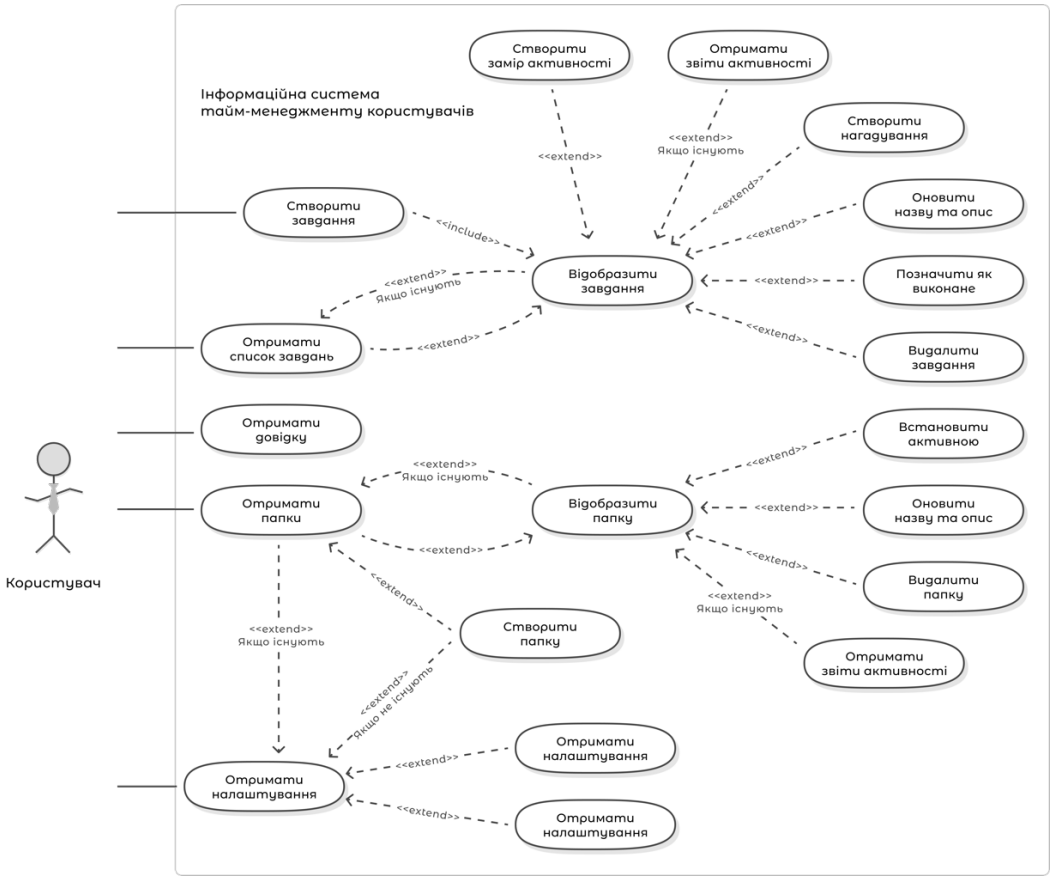

Рисунок 2.3 – Діаграма прецедентів

2.3 Вибір мови програмування

Telegram боти можуть бути написані на різних мовах програмування [9]. Перелік найпопулярніших для рішення цієї задачі наступний:

- Python – це популярний вибір завдяки своїй простоті, читабельності та великій спільноті розробників. Він має багато бібліотек, які полегшують роботу з Bot API;
- JavaScript - це ще один популярний вибір завдяки його універсальності та великій кількості фреймворків;
- PHP – це хороший вибір для розробників, який дуже активно використовується в web розробці та має багато бібліотек для роботи з Bot API;
- Java – це популярний, потужний та надійний вибір для створення складних додатків, для котрого також існують бібліотеки для створення бота.

Хоча всі перелічені мови програмування можуть використовуватися для створення Telegram ботів, Python має певні переваги, які вигідно відрізняють його від інших та роблять його кращим вибором для багатьох розробників:

- синтаксис Python чіткий і лаконічний, що полегшує його розуміння та обслуговування.
- має велику та активну спільноту розробників, що означає те, що можна легко знайти допомогу та ресурси, якщо це знадобиться;
- через велику популярність та відповідно велику спільноту він має багато гарних бібліотек для рішення великої кількості задач.

Python — високорівнева, інтерпретована, об'єктно-орієнтована мова програмування загального призначення з динамічною строгою типізацією та автоматичним управлінням пам'яттю, що орієнтована на підвищення продуктивності розробника, читання коду та його якості, а також на забезпечення переносимості написаних на ньому програм. Його застосовують

для роботи з великим обсягом інформації, при розробці вебсайтів, у машинному навчанні та для інших проектів.

2.4 Робота з Bot API

Є багато бібліотек для створення Telegram боту, де найпопулярніші це: python-telegram-bot, Telepot,PyTelegramBotAPI (або telebot) та aiogram.

З-поміж багатьох бібліотек Python для створення Telegram-ботів, вибір слід зупинити на aiogram. Ця бібліотека є однією з найпопулярніших та має ряд переваг, які вигідно відрізняють її від інших альтернатив.

Aiogram — це сучасний і повністю асинхронний фреймворк для розробки чат-ботів Telegram для Python 3.8 і версій новіше з використанням asyncio та aiohttp. Він пропонує широкий спектр функцій, які спрощують і прискорюють розробку телеграм-ботів [10].

Деякі з причин, чому aiogram є гарним вибором:

- має простий та інтуїтивно зрозумілий API, що робить його легким для вивчення та використання;
- має активну та залучену спільноту, яка завжди готова допомогти та надати підтримку;
- має чітку та детальну документацію, яка полегшує процес розробки та пошук відповідей на запитання;
- має вбудовану інтеграцію для використання інтернаціоналізації та локалізації GNU Gettext (або Fluent).

Окрім того, aiogram надає ефективні механізми обробки подій та має підтримку вбудованої системи для зберігання стану користувача. Це робить фреймворк ідеальним для створення не лише простих ботів, а й проєктів зі складною логікою взаємодії.

Aiogram базується на принципах асинхронного програмування, що робить його потужним інструментом для розробки високопродуктивних та масштабованих ботів.

На відміну від синхронного підходу, де інструкції виконуються послідовно, очікуючи завершення операцій вводу-виводу, асинхронне програмування (рисунок 2.4) дозволяє запускати наступні операції без блокування основного потоку виконання. Це особливо важливо для взаємодії з Bot API, оскільки ця взаємодія зазвичай потребує значної кількості мережевих запитів.

Асинхронність у Python з'явилася разом із випуском версії 3.5 восени 2015 року та була описана в PEP 492 (Python Enhancement Proposal 492). Цей PEP ввів новий синтаксис "async" та "await", а також виокремив концепт "корутин", або "співпрограм". На відміну від підпрограм, які мають одну точку входу-виходу, співпрограми можуть мати безліч таких точок.

Цей PEP також передбачає, що асинхронні завдання плануються та координуються циклом подій. Це механізм, що відповідає за керування виконанням асинхронних операцій та визначає порядок їх виконання, спрощуючи роботу з асинхронним кодом [11].

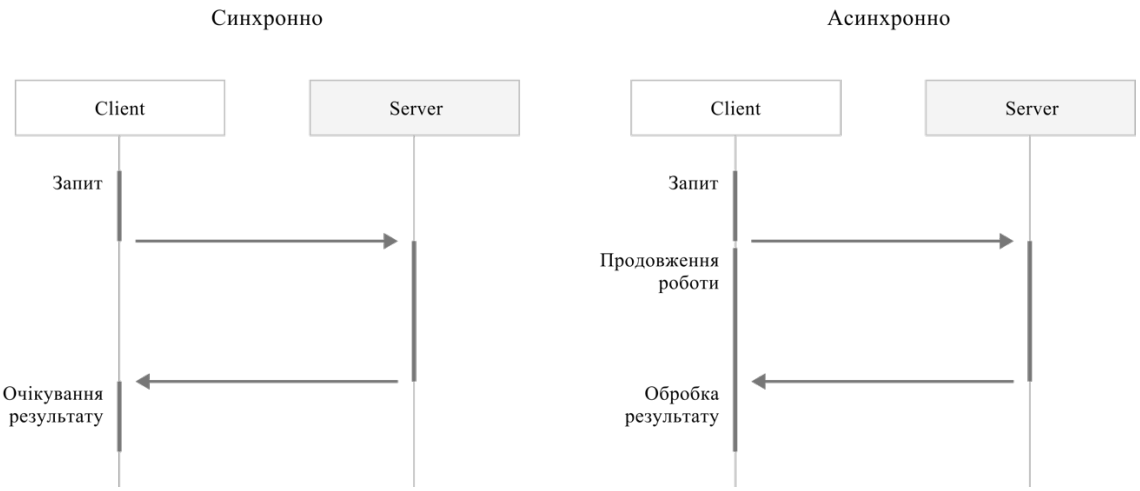

Рисунок 2.4 – Синхронне та асинхронне виконання операцій

2.5 Вибір СУБД

Розроблювана інформаційна система потребуватиме збереження даних користувачів, тому важливо визначитися з технологіями, які допоможуть це

реалізувати. Зокрема, необхідно вибрати систему управління базами даних та бібліотеку для взаємодії з нею.

Система управління базами даних (СУБД) — це набір взаємопов'язаних даних і програм для доступу до цих даних. Надає можливості створення, збереження, оновлення та пошуку інформації в базах даних з контролем доступу до даних.

Існує безліч СУБД, котрі можна обрати. СУБД бувають різні, залежно від моделі даних чи виду доступу. Через історичні причини та через широку застосовуваність і ефективність в різних областях інформаційних технологій сьогодні виокремлюють два основних типи – це реляційні та об'єктно-орієнтовані.

Реляційні в порівнянні з об'єктно-орієнтованими є більш розповсюдженими та розвиненими, тому слід обрати СУБД саме цього типу. До популярних реляційних СУБД можна віднести:

- MySQL – це безкоштовна та відкрита реляційна СУБД, яка в деяких джерелах є найбільш популярною у світі. Вона використовується в найрізноманітніших сферах, включно з веб-розробкою, системами керування контентом і бізнес-аналітикою. Сумісна з більшістю популярних мов програмування, включно з PHP, Java, Python і C++. З недоліків можна відзначити не повну підтримку SQL.
- PostgreSQL – це також безкоштовна та відкрита реляційна СКБД, яка є популярною альтернативою MySQL. Вона відрізняється більшими можливостями порівняно з MySQL і підтримує широкий спектр функцій і розширень, що дозволяють користувачам налаштовувати її під свої потреби. Проте це робить її більш складною у використанні порівняно з MySQL.
- Oracle – це комерційна, що є основним недоліком, реляційна СКБД, яка вважається однією з найпродуктивніших серед усіх і використовується у великих корпоративних додатках, системах

управління даними та аналітиці. Крім того, Oracle пропонує широкий спектр послуг підтримки, включаючи технічну підтримку, навчання та консультації.

- SQLite – це швидка та легка однофайлова реляційна СКБД, яка вбудовується в додатки і не потребує окремого сервера. Вона дозволяє зберігати всю базу даних локально на одному пристрої. Але недоліком є обмежені можливості масштабування та відсутність підтримки для складних багатокористувацьких середовищ, що робить її менш придатною для великих і критичних систем.

Система управління базами даних PostgreSQL відзначається не лише високою надійністю, ефективністю та масштабованістю, але й розширеним функціоналом, що робить її гарним вибором для різноманітних проєктів. Особливо в контексті розробки інформаційних систем з використанням Python, PostgreSQL виявляється популярним рішенням.

2.6 Взаємодія з СУБД

Для взаємодії з PostgreSQL існує чимало бібліотек, які можна використати в залежності від потреб і специфіки проєкту. Ці бібліотеки можна умовно розділити на кілька категорій, серед яких популярні:

- низькорівневі бібліотеки, такі як psycopg2, надають безпосередній доступ до функціоналу PostgreSQL і зазвичай використовуються для маніпулювання даними з використанням SQL, без проміжних абстракцій;
- ORM (Object-Relational Mapping), такі як SQLAlchemy, які надають шар абстракції над базою даних, дозволяючи працювати з об'єктами Python, як з реляційними таблицями.

Найбільш розповсюдженим типом бібліотек для взаємодії з будь-якою СУБД в Python проєктах є ORM. Їх використання не лише спрощує взаємодію з базою даних, але й надає можливість краще структурувати проєкт. Завдяки

ORM, розробники можуть визначити об'єкти своєї програми, їх взаємозв'язки та поведінку в контексті бази даних, у вигляді класів та методів.

Серед існуючих ORM в рамках даної ІС доцільно використати Peewee через її простоту.

Peewee надає базовий, але потужний функціонал ORM, який дозволяє здійснювати взаємодію з базою даних. Загалом, вибір Peewee може бути доцільним для проєктів, де простота використання, легкість інтеграції та базовий функціонал ORM вважаються ключовими критеріями [12].

Також можна використати асинхронну "обгортку" над Peewee, Peewee-async, для узгодженим з асинхронним характером бібліотеки aiogram, яка використовується для розробки Telegram бота. Однак це не є обов'язковим, оскільки це не матиме значний вплив.

У Python синхронний код для взаємодії з СУБД простий і реалізований через модуль сокетів стандартної бібліотеки, який виконаний на мові C. З іншого боку, асинхронний код у Python зазвичай виявляється складнішим, ніж синхронний. Кожен запит вимагає кількох операцій основного циклу подій, який зазвичай реалізований на Python. Це призводить до значних накладних витрат порівняно з кодом, який написаний на мові C.

Асинхронний підхід найбільш вигідний для сценаріїв з високою затримкою. У рамках даного проєкту не передбачається значних затримок в рамках взаємодії з СУДБ, оскільки матимемо просту структуру бази даних та виконання простих запитів. Тому асинхронний код може виявитися навіть трохи повільнішим за синхронний.

2.7 Додаткові технології

Для виконання ряду відкладених завдань, таких як відправка нагадувань, можна використати Celery в поєднанні з Redis.

Celery є потужним фреймворком для асинхронного виконання завдань [13]. Він дозволяє виконувати завдання в фоновому режимі, не блокуючи основний потік виконання програми. Завдяки своїй надійності та

широким можливостям, Celery є стандартом в сфері та найпопулярнішим вибором для таких сценаріїв.

У ролі посередника повідомлень (message broker) для Celery обрано рішення за замовчуванням Redis — NoSQL DB. Redis володіє високою швидкодією та легкістю конфігурації, що робить його гарним партнером для ефективного обміну повідомленнями між компонентами системи та забезпеченням стабільності в виконанні завдань [14].

Звіт про час виконання завдань доцільно створювати як часову гістограму. Для цього варто використати Matplotlib. Це популярна бібліотека для створення візуалізацій дани. Вона забезпечує потужний інструментарій для створення різноманітних типів графіків, включаючи лінійні графіки, гістограми, кругові діаграми, графіки розсіювання та багато інших.

Matplotlib є основною бібліотекою для візуалізації даних у Python і широко використовується в наукових дослідженнях, машинному навчанні, фінансовому аналізі та багатьох інших сферах.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Проєктування схеми сховища

Для функціонування ІС необхідно спроектувати сховище даних, що відповідає третій нормальній формі. Це гарантує, що дані будуть збережені без дублювання та мінімізує ризик наявності аномалій. В процесі проектування необхідно визначити всі необхідні дані для функціонування інформаційної системи, а також розробити механізми для підтримки консистентності даних.

Перш за все при проєкуванні схеми бази даних важливо спочатку визначити наявні в інформаційній системі сутності, або ж об’єкти. Виходячи з постановки задачі, можна визначити наступні сутності:

- користувач;
- завдання;
- папка.

Далі необхідно визначити їх атрибути в рамка ІС.

Користувачу для використання ІС не має потреби надавати будь-які дані. Але доречним буде додати інформацію, призначенням котрої буде покращення досвіду використання. Це бажана мова інтерфейсу та часовий пояс для відображення дат. Відповідно користувач матиме наступні атрибути:

- бажана мова інтерфейсу;
- часовий пояс.

Завдання повинно мати заголовок, або ж назву, та позначку, що вказуватиме на статус виконання. За необхідності воно також повинне мати час нагадування. Крім цього, корисним буде автоматична фіксація часу створення та можливість додати детальний опис. Відповідно атрибути завдання будуть наступними:

- назва;
- опис;
- статус виконання;
- час створення;

- час нагадування.

Також, відповідно до постановки задачі, завдання повинно мати заміри часу виконання. Для цього доречним буде виділити це як окрему сутність, що матиме відношення один-до-багатьох із завданням. Окрім тривалості часу, для зручності є сенс додати не обов’язковий атрибут опис. Атрибут тривалість часу краще визначити як два окремих – початок заміру та кінець. Це також вимагатиме встановити механізм, що забезпечуватиме відсутність аномалії коли початок більший за кінець. Відповідно атрибути сутності будуть наступними:

- час початку заміру;
- час кінця заміру;
- опис.

Завдання це сутність, що повинна належати користувачу, відповідно необхідно встановити відношення багато-до-одного із користувачем.

Папка для її ідентифікації користувачем повинна мати назву, а також для більшої зручності використання можна додати не обов’язковий опис. Оскільки назва повинна ідентифіковувати, то необхідно визначити механізм, що гарантуватиме унікальність значення даного атрибуту. Також папка відповідно до постановки задачі повинна містити або ж об’єднувати завдання. Для цього необхідно встановити відношення між папкою та завданням один-до-багатьох. Проте для зручності використання слід зробити папки не обов’язковим елементом.

Як і завдання папка це сутність, що належить користувачу, тому потрібно встановити відношення багато-до-одного із користувачем.

Крім цього, для функціонування ІС необхідно додати атрибут користувачу, що вказуватиме на активну папку. Відповідно потрібно встановити відношення один-до-одного між папкою та користувачем.

В якості первинного ключа в таблицях є сенс використати сурогатний ключ "id".

Використання сурогатного ключа, або штучного ключа, такого як "id", як первинного ключа в таблицях є досить поширеною практикою. Це відбувається з кількох причин. По-перше, використання сурогатного ключа дозволяє відокремити фізичний запис в базі даних від його семантики або значення. Це може спростити процес оновлення та управління даними, особливо в складних системах. Крім того, сурогатний ключ може бути автоматично згенерованим, що полегшує створення нових записів без необхідності ручного встановлення унікального ідентифікатора. Такий підхід також дозволяє уникнути проблем, пов'язаних з можливим вичерпанням значень природних ключів або їх зміною.

В результаті ER-діаграма, що зображена на рисунку 3.1, матиме 4 таблиці. Також додатково визначено обмеження, що гарантуватимуть цілісність даних та відсутність аномалій, а саме:

- Оскільки назва повинна бути унікальною в рамках існуючих значень користувача, то необхідно встановити обмеження унікальності на поля "name" та "account_id".
- Відношення один-до-одного між користувачем та папкою реалізовано у вигляді зіставного зовнішнього ключа. Поля "id" (первинний ключ) та "active_folder_id" таблиці "account" посилаються на поля "id", "account_id" таблиці "folder". Це також забезпечує, що користувач завжди є власником активної папки.
- Відношення багато-до-одного між завданням та папкою також реалізовано у вигляді зіставного зовнішнього ключа. Поля "folder_id", "account_id" таблиці "task" посилаються на поля "id" та "account_id" таблиці "folder". Це є частиною механізму, який забезпечує, що як папка, так і завдання належать одному й тому ж користувачу.

На рівні бази даних не передбачено механізму, який гарантував би відсутність конфліктуючих замірів виконання завдання. Тому його необхідно буде реалізувати на рівні додатку.

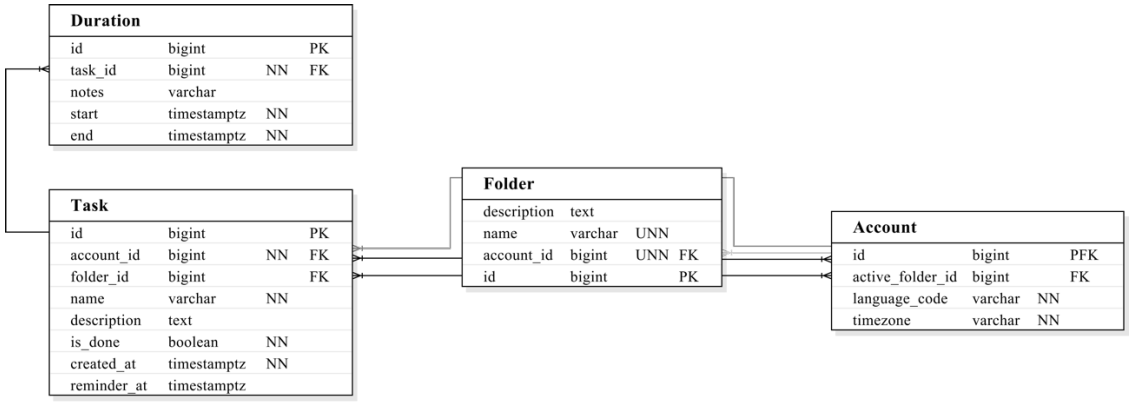

Рисунок 3.1 – ER-діаграма

3.2 Розробка бота

Розробка Telegram-бота має багато спільного з розробкою серверної частини вебсайтів. Обидва процеси передбачають роботу з даними, логікою та інтерфейсом користувача, хоча й з певними нюансами.

Структура розроблюваного бота схожа на серверну частину вебсайту, яка розроблюється з використанням патерна проєктування MVC – містить моделі та контролери. Проте відсутній рівень представлення.

Для ізоляції залежностей проєкту було створено віртуальне середовище за допомогою команди "python -m venv .venv". Щоб полегшити процес розгортання в майбутньому, відповідні залежності проєкту були зафіксовані в файлі `requirements.txt` командою "pip freeze > requirements.txt".

Основним модулем та вхідною точкою проєкту є файл main.py, наведений у додатку А. Він містить функції webhook та polling для запуску бота у відповідному режимі. Ці функції використовують стандартні можливості бібліотеки Aiogram:

- для режиму polling, режим опитування, використовується запуск у циклі подій стандартного модуля asyncio співпрограми, отриманої викликом методу start_polling об'єкта типу Dispatcher.

- для режиму webhook використовується нативна інтеграція з бібліотекою Aiohttp, робота в даному режимі передбачається через проксі сервер, такий як Nginx.

Окрім цього, даний модуль містить допоміжні функції:

- дві співпрограми для зворотного виклику в момент завершення ініціалізації компонентів проєкту, що встановлюють необхідні для функціонування зовнішні налаштування;
- функція setup для ініціалізації основних об'єктів проєкту типу Bot та Dispatcher. Головна роль останнього – бути маршрутизатором, тобто визначати, який саме обробник має обробляти запит від користувача.

Основний модуль та деякі інші використовують для своєї роботи модуль налаштувань — "settings.py". Він має містити такі параметри:

- токен для керування ботом;
- дані для доступу до бази даних;
- налаштування для запуску в режимі webhook (HTTP сервер), зокрема локальний хост, порт та зовнішня URL адреса;
- мовні налаштування;
- адресу черги завдань.

Важливим модулем проєкту є "models.py", наведений у додатку Б. У рамках цього ПЗ він реалізує рівень моделей патерну MVC. Його вміст складається з опису моделей — типів об'єктів інформаційної системи, які містять описи полів, зав’язків та обмежень. Крім того, деякі моделі визначають додаткову поведінку для себе або своїх об'єктів.

Кожен з цих типів наслідується від типу Model бібліотеки Peewee, який реалізує логіку роботи з базою даних, серіалізацію даних та опис структури реляційних таблиць. Розробники бібліотеки Peewee створювали її під впливом Django ORM, тому процес опису моделей дуже схожий.

Крім опису моделей, модуль "models.py" містить об'єкт для керування з'єднанням з базою даних, допоміжну функцію для визначення поточного UTC часу та функцію для ініціалізації сховища.

Для виконання операцій, що необхідні при обробці кожного з запиту, визначено 3 Middleware:

- AccountMiddleware – додає до контексту обробки запиту екземпляр моделі поточного користувача, тобто того, хто виконує запит;
- LanguageMiddleware – визначає мову поточного запиту, він обов’язково потребує виконання попереднього;
- CreateTaskOuterMiddleware – обробляє запити, для котрих не визначено обробника, як створення нового завдання.

Безпосередньо обробники вхідних запитів описані в пакеті модулів handlers. У рамках проєкту обробник — це співпрограма, яка першим позиційним аргументом приймає об'єкт події, а інші необхідні параметри контексту обробки запиту — як іменовані аргументи.

Для певної категорії обробників також визначено Middleware, яке збагачує контекст ресурсом запиту. Наприклад, при обробці дій з завданням, воно створює відповідний екземпляр або повертає відповідь, що вказує на те, що даний ресурс не знайдено.

Функції для створення звітів розміщені в пакеті модулів reports. Вони схожі між собою. Спершу виконується запит до бази даних, використовуючи моделі. Параметри функцій визначають вибірку даних. Після отримання даних здійснюється підготовка зручних структур даних, а потім створення звіту за допомогою бібліотеки Matplotlib.

Нагадування реалізовано модулем "reminder.py", що наведений в додатку В. Його основою є функція run. Дана функція модифікована декоратором об’єкту типу Celery, що дозволяє виконувати спеціальний виклик для надсилання задачі в чергу, що зберігається в Redis. Для виконання даної функції необхідно окремо від процесів обробки запитів користувачів

мати працюючий процес Celery. Даний процес прийматиме завдання з черги та виконуватиме його.

3.3 Опис взаємодії з ботом

Першою взаємодією з ботом завжди є команда "/start", на яку повертається привітання, опис його можливостей та клавіатура з варіантами повідомлень.

Для створення завдання можна використати клавіатуру або команду "/create" на яку бот відправить повідомлення з вказівками. Після їх виконання у користувача з’явиться нове завдання, котре відразу буде доступне для взаємодії.

Для отримання переліку не виконаних завдань користувач може використати:

- команду "/list";
- відповідне повідомлення з клавіатури;
- клавіша біля завдання, що змінить це повідомлення на список завдань в активній папці.

Повідомлення з переліком завдань міститеме клавіши для створення нового та за необхідності елементи пагінації. Натиснувши на відповідний номер завдання, воно стає доступним для взаємодії через зміну повідомлення. Створення завдання та їх перелік наведено на рисунку 3.2.

Для зміни параметрів користувач (рисунок 3.3) може використати команду "/settings" або відповідну клавішу. Повідомлення міститеме елементи для визначення нових параметрів.

Рисунок 3.2 – Створення завдання та їх перелік

Перелік папок можна отримати використавши команду "/folders" або натиснувши відповідну клавішу. Повідомлення з папками матиме схожі до переліку завдань елементи взаємодії. Окрему папку можна видалити, оновити або зробити активною. Також відповідне повідомлення міститеме клавішу для отримання переліку. Окрема папка та їх перелік наведено на рисунку 3.4.

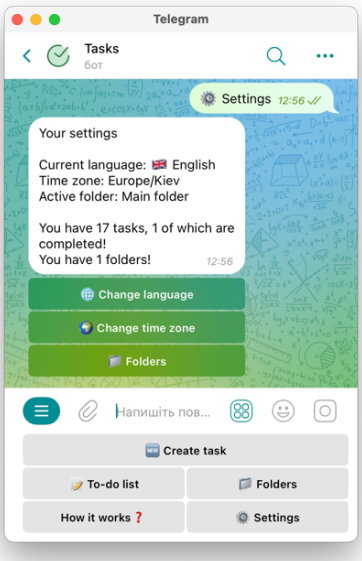

Рисунок 3.3 – Налаштування користувача

Звіти можна отримати як для одного завдання, так і для завдань з обраної папки за певний період:

- за останній день;
- за останні 7 днів;
- за останні 30 днів.

Отримання звіту наведено на рисунку 3.4, а приклад звіту наведено на рисунку 3.5.

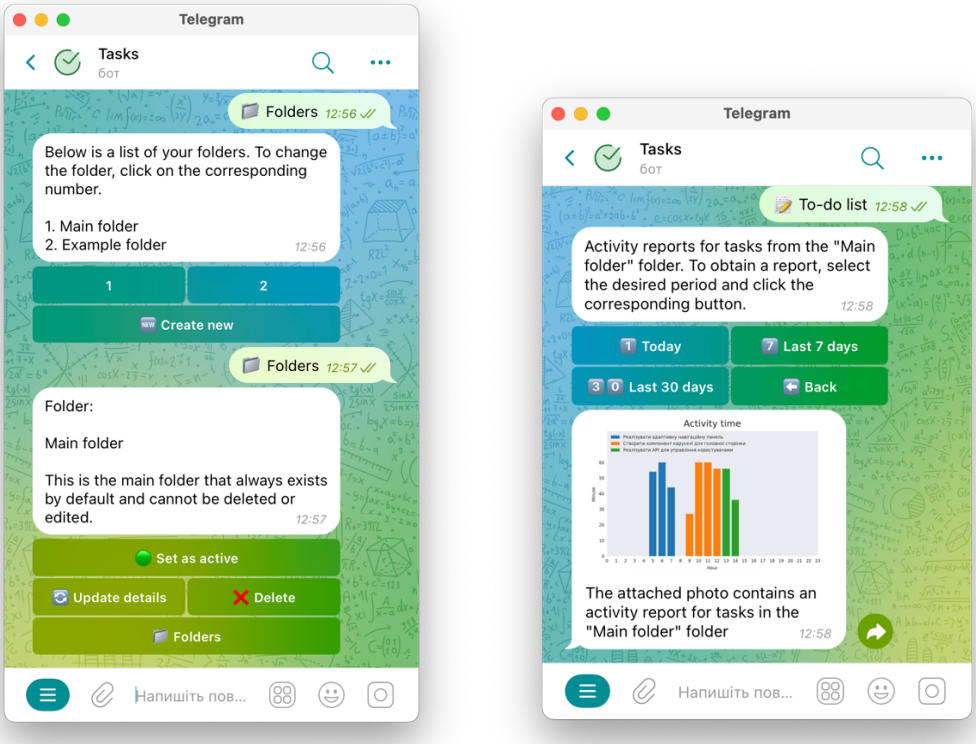

Рисунок 3.4 – Окрема папка, їх перелік та отримання звіту по паці

Рисунок 3.5 – Звіт за останній день по завданням з папки

Повідомлення із завданням також містить кнопки, що дозволяють створювати нагадування та заміри виконання. При натисканні кнопки для створення заміру користувач отримає повідомлення з кнопкою для фіксації заміру. Натиснувши кнопку для створення нагадування, користувач повинен буде вказати дату та час у визначеному форматі. Це продемонстровано на рисунку 3.6.

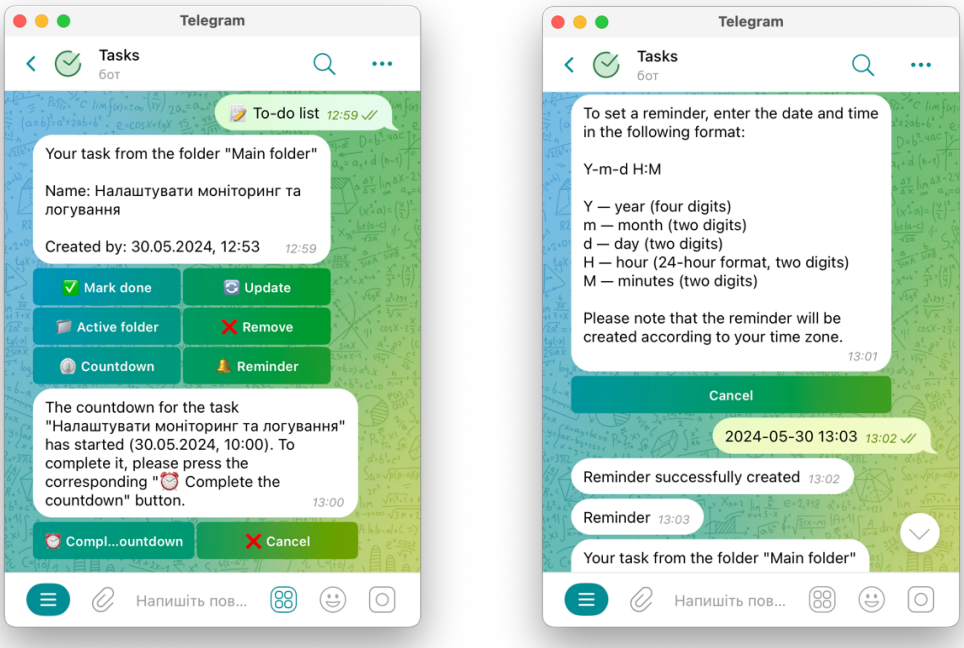

Рисунок 3.6 – Замір виконання та нагадування

ВИСНОВОКИ

Під час проведеного дослідження було виявлено, що наразі такі існуючі рішення для керування особистими справами та управління часом як: Microsoft To Do, Todoist, Google Tasks, Any.do та TickTick не забезпечують повноцінної інтеграції з популярною платформою Telegram у формі боту. Також виявлено, що наявні рішення для керування особистими справами у форматі боту обмежені.

На основі отриманих даних було сформульовано завдання для розробки інформаційної системи у вигляді Telegram боту з функціоналом, що включає базові операції з завданнями, групування завдань, нагадування про них та можливість відстеження активності користувача.

Для реалізації було вибрано методи, включаючи розробку контекстної діаграми, її деталізованої версії, діаграми прецедентів. Після аналізу вимог інформаційної системи були обрані технології, такі як мова програмування Python, система управління базами даних PostgreSQL, noSQL сховище Redis, а також бібліотеки Aiogram, Peewee, Celery та Matplotlib.

Вдало спроєктовано схему сховища даних.

У результаті вдалої реалізації було створено інформаційну систему у вигляді Telegram боту, працездатність та якість якої перевірена в рамках ручного тестування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. How to Become a Market Leader in 10 Years [Електронний ресурс] – Режим доступу до ресурсу: https://blog.doist.com/how-to-become-a-market-leader-in-10-years/ (дата звернення 18.04.2024)
2. The 7 best to do list apps in 2024 [Електронний ресурс] – Режим доступу до ресурсу: https://zapier.com/blog/best-todo-list-apps/ (дата звернення 18.04.2024)
3. Telegram Messanger [Електронний ресурс] – Режим доступу до ресурсу: https://telegram.org/ (дата звернення 19.04.2024)
4. Bots: An introduction for developers [Електронний ресурс] – Режим доступу до ресурсу: https://core.telegram.org/bots (дата зверення 19.04.2024)
5. Telegram Bot API [Електронний ресурс] – Режим доступу до ресурсу: https://core.telegram.org/bots/api (дата звернення 19.04.2024)
6. Telegram Bot Features [Електронний ресурс] – Режим доступу до ресурсу: https://core.telegram.org/bots/features (дата звернення 19.04.2024)
7. Методології моделювання. Сімейство IDEF [Електронний ресурс] – Режим доступу до ресурсу: https://elib.lntu.edu.ua/sites/default/files/elib_upload/Кондіус%202%20готова/page5.html (дата звернення 21.04.2024)
8. Нотація IDEF0 [Електронний ресурс] – Режим доступу до ресурсу: https://elib.lntu.edu.ua/sites/default/files/elib_upload/Кондіус%202%20готова/page9.html (дата звернення 21.04.2024)
9. Bot API Libary Examples [Електронний ресурс] – Режим доступу до ресурсу: https://core.telegram.org/bots/samples (дата звернення 24.04.2024)
10. Aiogram 3.1.1 documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.aiogram.dev/en/v3.1.1/ (дата звернення 24.04.2024)

11. PEP 492 – Coroutines with async and await syntax [Електронний ресурс] – Режим доступу до ресурсу: https://peps.python.org/pep-0492/ (дата звернення 25.04.2024)
12. Peewee - peewee 3.15.3 documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.peewee-orm.com/en/3.15.3/ (дата звернення 26.04.2024)
13. Celery - Distributed Task Queue — Celery 5.4.0 documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.celeryq.dev/en/v5.4.0/ (дата звернення 27.04.2024)
14. Redis documentation [Електронний ресурс] – Режим доступу до ресурсу: https://redis.io/docs/latest/ (дата звернення 27.04.2024)

ДОДАТОК А. ГОЛОВНИЙ МОДУЛЬ

```
import sys
import asyncio
import settings
import middleware
from handlers import router
from aiohttp import web
from aiogram import Bot, Dispatcher
from aiogram.utils.i18n import I18n
from aiogram.webhook.aiohttp_server import (
    SimpleRequestHandler,
    setup_application
)


def setup() -> tuple[Bot, Dispatcher]:
    '''
    Sets up and returns instances of Bot and Dispatcher for use
    '''
    i18n = I18n(path=settings.LOCALES_PATH,
                default_locale=settings.LANGUAGE_CODE)
    bot = Bot(settings.TOKEN)

    dispatcher = Dispatcher()
    dispatcher.include_router(router)
    dispatcher.update.outer_middleware(middleware.AccountMiddleware())
    dispatcher.update.outer_middleware(middleware.LanguageMiddleware(i18n))

    dispatcher.message.outer_middleware(middleware.CreateTaskOuterMiddleware())

    return bot, dispatcher


async def set_webhook(bot: Bot):
    await bot.set_webhook(
        url=settings.WEBHOOK_URL + settings.WEBHOOK_PATH,
        secret_token=settings.WEBHOOK_SECRET
    )


async def remove_webhook(bot: Bot):
    await bot.set_webhook(url='')


def webhook():
    '''
```

```
    Executes the setWebhook method of the Bot API and starts an HTTP
    server according to settings.py file to handle updates from Telegram
    '''
    bot, dispatcher = setup()
    app = web.Application()

    webhook_requests_handler = SimpleRequestHandler(
        dispatcher=dispatcher,
        bot=bot,
        secret_token=settings.WEBHOOK_SECRET,
    )

    webhook_requests_handler.register(app, path=settings.WEBHOOK_PATH)
    dispatcher.startup.register(set_webhook)

    setup_application(app, dispatcher, bot=bot)

    web.run_app(
        app, host=settings.WEB_SERVER_HOST, port=settings.WEB_SERVER_PORT
    )


def polling():
    bot, dispatcher = setup()

    dispatcher.startup.register(remove_webhook)
    coroutine = dispatcher.start_polling(bot)

    asyncio.run(coroutine)


if __name__ == '__main__':
    if '--webhook' in sys.argv:
        webhook()
    else:
        polling()
```

ДОДАТОК Б. МОДЕЛІ

```
import pytz
import peewee
import settings
from datetime import datetime, timedelta
from aiogram.utils.i18n import gettext as _


database = peewee.PostgresqlDatabase(
    settings.DB_NAME,
    user=settings.DB_USER,
    password=settings.DB_PASSWORD,
    host=settings.DB_HOST
)


def utc_now():
    return datetime.now(pytz.utc)


def init():
    with database:
        database.create_tables(
            [Account, Folder, Task, Duration]
        )
        database.execute_sql(
            'ALTER TABLE account '
            'ADD CONSTRAINT account_active_folder_fk '
            'FOREIGN KEY (active_folder_id, id) '
            'REFERENCES folder (id, account_id) '
            'ON DELETE SET NULL'
        )


class Account(peewee.Model):
    _timezones = [(tz, tz) for tz in pytz.all_timezones]

    # Id is identical to the corresponding field of a Telegram user
    id = peewee.BigIntegerField(primary_key=True)
    active_folder = peewee.DeferredForeignKey('Folder', null=True)
    timezone = peewee.CharField(choices=_timezones, default='Europe/Kiev')
    language_code = peewee.CharField(
        choices=settings.LANGUAGES,
        default=settings.LANGUAGE_CODE
    )
```

```
    id: int
    active_folder: 'Folder | None'
    timezone: str
    language_code: str

    tasks: peewee.ModelSelect
    folders: peewee.ModelSelect

    @property
    def language(self):
        return dict(settings.LANGUAGES).get(
            self.language_code, settings.LANGUAGE_CODE
        )

    class Meta:
        database = database


class Folder(peewee.Model):
    account = peewee.ForeignKeyField(
        Account,
        on_delete='CASCADE',
        backref='folders'
    )
    name = peewee.CharField(max_length=500)
    description = peewee.TextField(null=True)

    tasks: peewee.ModelSelect

    def __str__(self):
        return self.name

    class Meta:
        database = database
        constraints = [
            peewee.SQL('UNIQUE (id, account_id)'),
            peewee.SQL('UNIQUE (name, account_id)'),
        ]


class Task(peewee.Model):
    # To initialize the storage, the field
    # type must be changed to DeferredForeignKey
    folder = peewee.ForeignKeyField(
        Folder,
        null=True,
        backref='tasks'
    )
```

```
    account = peewee.ForeignKeyField(
        Account,
        on_delete='CASCADE',
        backref='tasks'
    )

    name = peewee.CharField(max_length=500)
    description = peewee.TextField(null=True)
    is_done = peewee.BooleanField(default=False)
    created_at = peewee.DateTimeField(default=utc_now)

    id: int
    folder: Folder | None
    account: Account
    name: str
    description: str
    is_done: bool
    created_at: datetime
    durations: peewee.ModelSelect

    @property
    def localized_created_at(self) -> datetime:
        try:
            timezone = pytz.timezone(self.account.timezone)
        except pytz.exceptions.UnknownTimeZoneError:
            timezone = pytz.timezone('UTC')

        return self.created_at.astimezone(timezone)

    def __str__(self):
        if self.name:
            return self.name
        return super().__str__()

    class Meta:
        database = database
        constraints = [
            peewee.SQL('UNIQUE (id, account_id)'),
            peewee.SQL(
                'FOREIGN KEY (folder_id, account_id) '
                'REFERENCES folder (id, account_id)'
                'ON DELETE CASCADE'
            )
        ]


class Duration(peewee.Model):
    task = peewee.ForeignKeyField(
```

```
    Task,
    on_delete='CASCADE',
    related_name='durations'
)

end = peewee.DateTimeField()
start = peewee.DateTimeField()
notes = peewee.TextField(null=True)

task: Task
end: datetime
start: datetime
notes: str | None

# The date format to be used when converting between
# string and datetime types for further transmission
transition_date_format = '%Y-%m-%d %H:%M:%S.%f%z'

def __str__(self) -> str:
    return str(self.end - self.start)

@classmethod
def convert_dates(cls, raw: dict):
    '''
    Takes a dict and converts the types of the start and end keys
    from string to datetime. If this is not possible, a ValueError
    will be raised.
    '''
    for attr in ('start', 'end'):
        if attr not in raw:
            raise ValueError(
                f'The key {attr} is not represented in the provided dict'
            )

        date = raw[attr]
        if not isinstance(date, datetime):
            raw[attr] = datetime.strptime(date, cls.transition_date_format)

@staticmethod
def validate(
    task: Task, start: datetime, end: datetime, notes: str | None = None
):
    if start > end:
        raise ValueError(_('Can\'t process it. Invalid data received.'))

    query = task.durations.where(
        ((start <= Duration.start) & (Duration.start <= end))
        | ((start <= Duration.end) & (Duration.end <= end))
```

```
        | ((Duration.start <= start) & (start <= Duration.end))
        | ((Duration.start <= end) & (end <= Duration.end))
    )
    overlap = query.first()  # type: ignore

    if overlap is not None:
        # TODO: Update message text
        overlap_error_message = _(
            "Sorry, I can't register this entry for the task \"%s\" "
            "because there's already another one intersecting with "
            "it - its start %s (duration %s)"
        ) % (
            task.name,
            overlap.start.strftime('%d.%m.%Y, %H:%M'),
            (overlap.end - overlap.start)
        )
        if overlap.notes:
            overlap_error_message += " and note %s" % overlap.notes

        raise ValueError(overlap_error_message)
```

```
@classmethod
def create(cls, validate: bool = True, **attributes):
    if validate:
        cls.convert_dates(attributes)
        cls.validate(**attributes)
    return super().create(**attributes)

@property
def value(self) -> timedelta:
    return self.end - self.start

class Meta:
    database = database
    constraints = [peewee.Check('"start" < "end"')]
```

ДОДАТОК В. МОДУЛЬ НАГАДУВАНЬ

```
import asyncio
import settings
from aiogram import Bot
from celery import Celery
from peewee import DoesNotExist
from models import Account, Task
from aiogram.utils.i18n import I18n
from handlers.utils import base_retrieve
from aiogram.utils.i18n import gettext as _


app = Celery('reminder', broker=settings.BROKER)


async def send(account_id: int, task_id: int):
    '''
    Coroutine for sending reminder messages.
    It will be executed asynchronously in a task queue.
    '''
    try:
        account = Account.get(id=account_id)
        task = Task.get(id=task_id)
    except DoesNotExist:
        return

    i18n = I18n(path=settings.LOCALES_PATH,
                default_locale=settings.LANGUAGE_CODE)
    bot = Bot(settings.TOKEN)

    with i18n.context(), i18n.use_locale(account.language_code):
        text, markup = base_retrieve(task)

        await bot.send_message(account.id, _('Reminder'))
        await bot.send_message(account.id, text, reply_markup=markup)

    await bot.session.close()


@app.task
def run(account_id: int, task_id: int):
    coroutine = send(account_id, task_id)
    asyncio.run(coroutine)
```