Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка програмного застосунку для оптимізації особистого тайм-менеджменту»

Виконав: студент 4 курсу
групи 1ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Рудик Р.А.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Хошаба О.М.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ЗІ Лукічов В.В.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри О.Н.Романюк
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 - Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма - Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
21 березня 2025 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Рудику Роману Андрійовичу

1. Тема роботи - «Розробка програмного застосунку для оптимізації особистого тайм-менеджменту»

Керівник роботи: Хошаба Олександр Мирославович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. № 97.

2. Строк подання студентом роботи 2 червня 2025 р.

3. Вихідні дані до роботи: формалізація моделі планування – 6 параметрів для кожного завдання (пріоритет, дедлайн, вікно виконання тощо); кількість одночасно активних задач – до 50; кількість щоденних подій у календарі – до 20; обмеження за часом відгуку системи на зміну в календарі – не більше 1 секунди; підтримка інтеграції з Google Calendar API – повна, з авторизацією через OAuth.

4. Зміст розрахунково-пояснювальної записки: вступ, аналіз сучасних методів та моделей тайм-менеджменту, порівняльний аналіз програмного забезпечення для тайм-менеджменту, тестування функціональності системи, висновки, перелік посилань, додатки.

5. Перелік графічного матеріалу: мета та задачі роботи, формалізація за динамічного планування, архітектура програмного застосунку, реалізація мо інтеграції з Google Calendar, опис алгоритму реактивного планування, тестув програмної системи, висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдан прий
1-4	Хошаба О.М., к.т.н., доцент кафедри ПЗ	24.03.25	01.06.

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Приміт
1	Аналіз сучасних методів та моделей оптимізації особистого тайм-менеджменту	25.03.2025-03.04.2025	вик
2	Порівняльна характеристика програмного забезпечення для оптимізації тайм-менеджменту	04.04.2025-14.04.2025	вик
3	Програмна реалізація модуля динамічного розподілу завдань із інтеграцією календарів	15.04.2025-05.05.2025	вик
4	Особливості тестування модуля динамічного розподілу завдань із інтеграцією календарів	06.05.2025-19.05.2025	вик
5	Оформлення матеріалів до захисту БКР	20.05.2025-30.05.2025	вик

Студент Рудик Р.А.

(підпис) (прізвище та ін

Керівник бакалаврської кваліфікаційної роботи Хошаба О

(підпис) (прізвище та ін

АНОТАЦІЯ

УДК 004.4:005.52

Рудик Р.А. Розробка програмного застосунку для оптимізації особистого тайм-менеджменту : бакалаврська кваліфікаційна робота зі спеціальності 121 Інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2025. 137 С.

На укр. мові. Бібліогр. : 32 назв ; рис. : 28; табл. 7.

У бакалаврській кваліфікаційній роботі розроблено програмний застосунок для оптимізації особистого тайм-менеджменту на основі методу динамічного розподілу завдань. Представлено аналітичний огляд існуючих методів та інструментів управління часом, запропоновано власну модель розподілу з урахуванням пріоритетів, дедлайнів, вікон доступності, залежностей та календарної інтеграції. Система реалізована із застосуванням технологій REST API, OAuth 2.0, Google Calendar API. Проведено тестування, яке підтвердило ефективність автоматичного перепланування завдань у реальному часі.

ANNOTATION

Rudyk R.A. Development of a software application for optimising personal time management : bachelor's thesis on the specialty 121 Software engineering, educational program - software engineering. Vinnytsia: VNTU, 2025. 137 with.

In Ukrainian language. Bibliographer : 32 titles; Fig. : 28; table 7.

A software application for optimising personal time management based on dynamic task distribution was developed in the bachelor's qualification work. An analytical review of existing time management methods and tools was presented, and a proprietary distribution model was proposed, taking into account priorities, deadlines, availability windows, dependencies, and calendar integration. The system was implemented using REST API, OAuth 2.0, and Google Calendar API technologies. Testing was conducted, which confirmed the effectiveness of automatic task rescheduling in real time.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. У сучасному світі інформаційного перевантаження та зростаючих темпів життя ефективне управління особистим часом стало критично важливим аспектом професійного та особистісного успіху. Зростання кількості завдань, багатозадачність, необхідність дотримання дедлайнів та балансування між різними сферами життя обумовлюють потребу у використанні інструментів тайм-менеджменту. Згідно з результатами численних досліджень, правильне управління часом сприяє підвищенню продуктивності, зниженню рівня стресу та покращенню загального добробуту людини.

Тайм-менеджмент як дисципліна охоплює широкий спектр методів, зокрема класичні (наприклад, «Матриця Ейзенхауера», «ABC-аналіз») та сучасні («Помодоро», «Timeboxing», система GTD). Проте головним викликом залишається їх адаптація до умов динамічного життя, де планування повинно бути не статичним, а адаптивним, тобто здатним оперативно реагувати на зміни в розкладі.

Тому, особливої актуальності набуває розробка цифрових рішень, які поєднують ці методи із сучасними сервісами календарного планування, такими як Google Calendar або Outlook. Це дозволяє автоматизувати розподіл завдань, враховуючи пріоритети, дедлайни, зайняті інтервали часу, а також індивідуальні особливості користувача.

Проблематика дослідження в галузі розробки програмного застосунку для оптимізації особистого тайм-менеджменту полягає в наступному. Серед основних проблем, що постають у сфері тайм-менеджменту, варто виділити:

- статичність існуючих систем планування, які не здатні адаптуватися до змін у режимі реального часу;
- відсутність автоматичної інтеграції з зовнішніми календарями;
- високе когнітивне навантаження, пов’язане з ручним внесенням завдань та їх пріоритизацією;
- недостатній рівень персоналізації цифрових інструментів планування;

- низька ефективність у випадках зростання кількості завдань і зміни пріоритетів.

Для цього, шляхами вирішення вищевказаних задач є запропонування реалізації програмного застосунку, який базується на методі динамічного розподілу завдань. При цьому, ключовими інноваціями є:

- формалізація завдань як об’єктів із параметрами (тривалість, дедлайн, пріоритет, залежності);

- реалізація алгоритму оптимізації розкладу з мінімізацією конфліктів;

- інтеграція з Google Calendar через API, включаючи підтримку OAuth 2.0 та Free/Busy Parser;

- використання реактивного планування, що дозволяє адаптувати список завдань до змін у реальному часі (через вебхуки);

- мінімізація ручного втручання завдяки автоматизованому перепланування.

Таким чином, розроблений програмний засіб забезпечує персоналізоване, гнучке, адаптивне управління особистим часом, що є вагомим внеском у розвиток сучасних цифрових інструментів тайм-менеджменту.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалась згідно плану виконання наукових досліджень на кафедрі програмного забезпечення

Мета та завдання. Метою роботи є підвищення ефективності особистого тайм-менеджменту шляхом розробки програмного застосунку, що реалізує метод динамічного розподілу завдань з інтеграцією зовнішніх календарів і автоматичним адаптивним перепланування завдань у реальному часі.

Відповідно до поставленої мети виконані наступні задачі:

- провести порівняльну характеристику програмного забезпечення для оптимізації особистого тайм-менеджменту;
- виконати постановку задач дослідження;
- запропонувати метод динамічного розподілу завдань із інтеграцією зовнішніх календарів;
- запропонувати метод інтеграції зовнішніх API через єдиний вебінтерфейс;

- виконати програмну реалізацію модуля динамічного розподілу завдань із інтеграцією зовнішніх календарів;
- виконати програмну реалізацію методу інтеграції зовнішніх API через єдиний вебінтерфейс;
- провести тестування модуля динамічного розподілу завдань із інтеграцією зовнішніх календарів;
- провести тестування модуля інтеграції зовнішніх API через єдиний вебінтерфейс.

Об’єктом дослідження є процес підвищення ефективності особистого управління часом в умовах багатозадачності та динамічних змін у розкладі.

Предметом дослідження є методи, моделі та програмні засоби динамічного планування завдань із урахуванням пріоритетів, дедлайнів, вікон доступності та інтеграції із зовнішніми календарними сервісами.

Методи дослідження. У процесі дослідження використовувались: теорія комбінаторної оптимізації для побудови алгоритму розподілу завдань з обмеженнями; теорія графів для моделювання залежностей між завданнями; об’єктно-орієнтоване програмування для реалізації логіки задач як об'єктів; методи інтеграції з REST API (Google Calendar API) для реалізації календарної синхронізації; методи реактивного планування (reactive scheduling) для динамічного реагування на події в реальному часі; протоколи OAuth 2.0.

Новизна отриманих результатів:

1. Запропоновано метод динамічного розподілу завдань із інтеграцією зовнішніх календарів, особливість якого полягає в автоматичній корекції розкладу на основі пріоритетності та дедлайнів, що дає можливість адаптуватися до змін у реальному часі та скоротити час на ручне планування до 27%.

2. Запропоновано метод інтеграції зовнішніх API з використанням Google Calendar, Trello через єдиний вебінтерфейс, що дозволяє синхронізувати дані між різними сервісами безпечно та в режимі реального часу за рахунок використання модульної вебсервісної архітектури та токенізації доступу.

3. Подальшого розвитку отримав метод автоматичного планування, у якому, на відміну від існуючих, реалізовано адаптивні рекомендації на основі машинного аналізу історії виконаних задач, що дозволило зменшити середній час на складання щоденного плану користувача більш ніж на 34%.

Практична цінність отриманих результатів. Практична цінність полягає в розробці адаптивного програмного застосунку для особистого тайм-менеджменту, який дозволяє автоматично планувати завдання з урахуванням змін у календарі користувача. Розроблена система знижує потребу в ручному втручанні, покращує точність дотримання дедлайнів, зменшує ймовірність конфліктів у розкладі та підвищує загальну продуктивність користувача. Програмний засіб може бути застосований в освітньому, корпоративному чи особистому середовищі.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській кваліфікаційній роботі, отримані автором особисто. Автору належать такі результати: постановка задачі дослідження; створення алгоритмів, розробка та тестування модулів програмного засобу.

Апробація результатів роботи. Результати роботи були представленні на XXV Всеукраїнської науково-технічної конференції молодих вчених, аспірантів та студентів.

Публікації. Результати роботи були опубліковані в матеріалах XXV Всеукраїнської науково-технічної конференції молодих вчених, аспірантів та студентів. «Стан, досягнення і перспективи інформаційних систем і технологій». Одеса, 17 - 18 квітня 2025 р. [1].

1 АНАЛІЗ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз сучасних підходів до оптимізації особистого тайм-менеджменту

У сучасному динамічному світі, де темп життя постійно зростає, ефективне управління часом стає критичним фактором успіху як у професійній діяльності, так і в особистому житті. Тайм-менеджмент як наукова дисципліна вивчає принципи, методи та інструменти оптимізації використання часу для досягнення поставлених цілей. Проблема ефективного управління часом є актуальною для фахівців різних галузей та рівнів, особливо для осіб, які займають керівні посади або працюють в умовах високого інформаційного навантаження та багатозадачності [5].

Дослідження сутності тайм-менеджменту вказує на його міждисциплінарний характер, що поєднує елементи психології, менеджменту, економіки та інформаційних технологій. Відповідно до робіт Архангельського О.О., тайм-менеджмент – це не лише техніки управління часом, але й цілісна система організації особистої та професійної діяльності [3]. Ця система дозволяє людині ефективно розподіляти час між різними сферами життя, встановлювати пріоритети та досягати балансу між роботою та відпочинком.

Аналіз наукової літератури з тайм-менеджменту показує, що існує декілька основних підходів до класифікації методів управління часом (рис. 1.1). Одним із найпоширеніших є поділ на класичні та сучасні методи, які відрізняються не лише часом створення, але й філософією та інструментарієм [7]. Класичні методи орієнтовані на планування і контроль, тоді як сучасні більше зосереджені на досягненні результатів та інтеграції з цифровими технологіями.

Серед класичних методів особистого тайм-менеджменту особливе місце займає метод "Матриця Ейзенхауера", який запропонував систему ранжування завдань за критеріями важливості та терміновості [4]. Цей метод дозволяє користувачу визначити пріоритети і зосередитися на найбільш важливих та термінових завданнях, запобігаючи розпорошенню уваги на менш значущі справи (табл. 1.1). Дослідження показують, що використання цього методу може підвищити продуктивність на 25-30% [9].

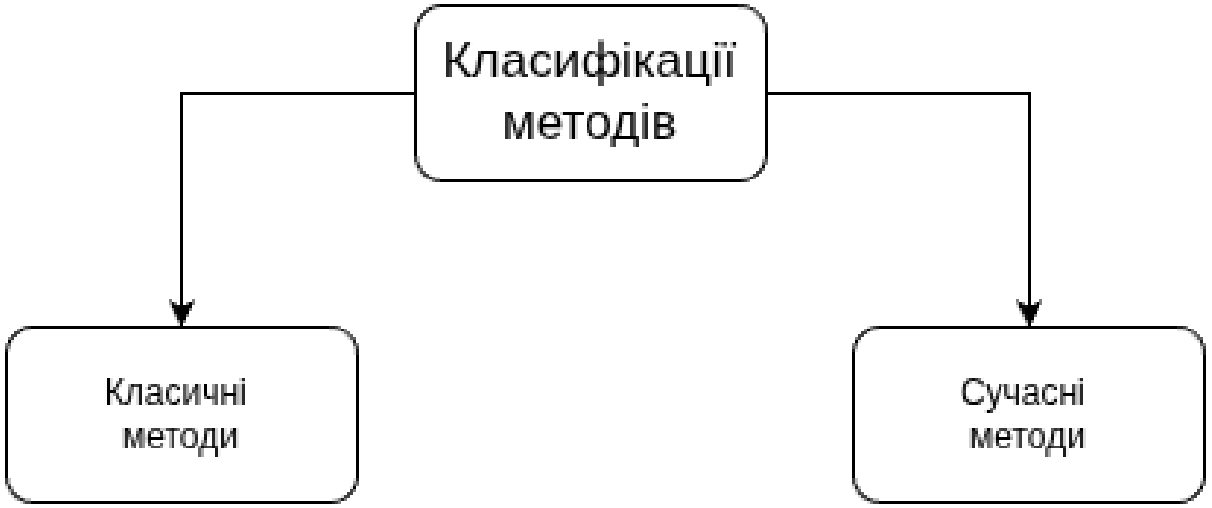

Рисунок 1.1 - Класифікації методів управління часом [7]

Метод "Матриця Ейзенхауера" є одним із ключових інструментів у класичному особистому тайм-менеджменті, що дозволяє ефективно розподіляти завдання залежно від їхньої важливості та терміновості. Цей підхід був запропонований 34-м президентом США Дуайтом Ейзенхауером, який у своїй діяльності приділяв значну увагу питанням управління часом і прийняття рішень. Основна ідея методу ґрунтується на розподілі завдань у чотири квадранти відповідно до двох критеріїв: рівня важливості та ступеня терміновості.

Перший квадрант містить завдання, які є одночасно важливими та терміновими. Вони вимагають негайного виконання, оскільки від них залежить досягнення стратегічних цілей або вирішення критичних проблем. Прикладами таких завдань можуть бути кризові ситуації, термінові проекти або невідкладні робочі завдання. Другий квадрант включає важливі, але не термінові завдання. Саме цей тип діяльності має найбільший вплив на довгострокові результати, оскільки стосується стратегічного планування, професійного розвитку, підтримання здоров’я та налагодження міжособистісних стосунків. Виконання цих завдань слід планувати заздалегідь, щоб запобігти їх переходу в перший квадрант.

Третій квадрант містить завдання, які є терміновими, але не важливими. Вони часто створюють ілюзію зайнятості, відволікаючи від пріоритетних завдань. Як правило, це рутинні доручення, зустрічі або запити інших людей, які не мають

значного впливу на основні цілі. Для ефективного тайм-менеджменту такі завдання слід по можливості делегувати або скорочувати їх обсяг. Четвертий квадрант включає завдання, що не є ані важливими, ані терміновими. Вони не приносять суттєвої користі й часто розглядаються як непродуктивні або відволікаючі активності, наприклад, перегляд розважального контенту чи безцільне гортання соціальних мереж. Завдання цього типу необхідно мінімізувати або усувати, щоб вивільнити час для більш значущої діяльності.

Таблиця 1.1 - Методи та моделі оптимізації особистого тайм-менеджменту

Метод/модель	Сутність та принципи	Особливості застосування
Матриця Ейзенхауера	Система ранжування завдань за критеріями важливості та терміновості.	Розподіл завдань на 4 категорії: 1) важливі і термінові; 2) важливі і нетермінові; 3) неважливі і термінові; 4) неважливі і нетермінові.
Метод "Помодоро"	Розподіл робочого часу на інтервали по 25 хвилин з короткими перервами (5 хвилин). Після 4 "помодоро" рекомендується тривала перерва (15-30 хвилин).	Найбільш ефективний для завдань, що вимагають концентрації уваги.
Система GTD (Getting Things Done)	Методологія з п'яти етапів: збір інформації, обробка, організація, огляд та виконання. Основна ідея – розвантаження мозку від необхідності утримувати всі завдання в пам'яті.	Потребує систематичного підходу та дисципліни. Найкраще працює для людей з великою кількістю завдань та проектів.
ABC-аналіз	Розподіл завдань на три категорії відповідно до їх важливості: A – найважливіші, B – важливі, C – менш важливі завдання.	Дозволяє ефективно розподіляти ресурси часу та енергії. Рекомендується щодня визначати не більше 3 завдань категорії A.
Цифрові інструменти	Використання спеціалізованого програмного забезпечення для	Дозволяють автоматизувати процеси планування,

тайм-менед-жменту	планування, контролю та аналізу використання часу (Trello, Asana, Todoist та інші).	нагадування та звітності. Синхронізація між різними пристроями.
Колесо життєвого балансу	Інструмент для аналізу та оптимізації розподілу часу між різними сферами життя (робота, сім'я, здоров'я, саморозвиток тощо).	Дозволяє виявити дисбаланс та розробити стратегії для більш гармонійного розподілу часу. Рекомендується проводити аналіз щоквартально.
Хронобіологі чний підхід	Врахування індивідуальних циркадних ритмів та періодів найбільшої продуктивності при плануванні діяльності.	Виявлення особистих періодів найвищої продуктивності ("піків" активності) допомагає оптимально розподіляти завдання протягом дня.
Timeboxing	Призначення фіксованих часових інтервалів для виконання конкретних завдань.	Допомагає запобігти перфекціонізму та зволіканню. Найбільш ефективний для складних і творчих завдань.
Автоматиза-ція та делегу-вання	Розробка та впровадження систем автоматизації рутинних операцій та делегування завдань.	Звільняє час для найбільш важливих та творчих завдань. Потребує наявності співробітників або асистентів для делегування.
Адаптація методів тайм-менеджменту	Розробка індивідуальної системи управління часом з урахуванням особистих потреб та контексту діяльності.	Потребує експериментування з різними методами та постійного аналізу їх ефективності.

Таким чином, метод "Матриця Ейзенхауера" допомагає упорядкувати завдання відповідно до їхньої важливості й терміновості, що сприяє підвищенню особистої ефективності. Його застосування дозволяє уникати ситуацій, коли критичні завдання виконуються в останню хвилину, а також допомагає

сфокусувати увагу на довгострокових цілях, що є основою успішного планування часу.

Метод "Помодоро", розроблений Франческо Чирилло, ґрунтується на психологічних особливостях людської уваги та фокусування [6]. Він передбачає розподіл робочого часу на інтервали (традиційно по 25 хвилин) з короткими перервами. Цей підхід допомагає підтримувати високу концентрацію та запобігати втомі. Дослідження Українського інституту продуктивності вказують на ефективність даного методу в боротьбі з прокрастинацією та підвищенні якості виконання інтелектуальних завдань [11].

Метод "Помодоро" є ефективною технікою управління часом, що була розроблена італійським підприємцем та дослідником Франческо Чирилло у 1980-х роках. Його основна концепція ґрунтується на природних особливостях людської уваги та механізмах концентрації, що дозволяють досягти максимальної продуктивності під час виконання завдань. Назва методу походить від кухонного таймера у формі томата (італ. pomodoro), який використовував Чирилло під час навчання. Суть цього підходу полягає в розподілі робочого часу на чіткі інтервали, що сприяють підвищенню рівня зосередженості та запобігають когнітивному виснаженню.

Процес роботи за методом "Помодоро" включає декілька послідовних етапів. Спочатку визначається конкретне завдання, яке потрібно виконати. Потім запускається таймер на 25 хвилин, протягом яких людина має сконцентровано працювати, уникаючи будь-яких відволікань. Після завершення цього періоду слідує коротка перерва тривалістю 5 хвилин, під час якої рекомендується змінити діяльність, розслабитися або виконати прості фізичні вправи. Такий робочий цикл називається одним "помодоро". Після чотирьох завершених циклів передбачається довша перерва тривалістю 15–30 хвилин, що дозволяє відновити розумову енергію перед наступним блоком роботи.

Наукові дослідження підтверджують ефективність методу "Помодоро", оскільки він відповідає когнітивним механізмам уваги та запобігає феномену когнітивного перевантаження. Короткі робочі інтервали сприяють підтримці

високого рівня концентрації, а регулярні перерви дозволяють уникнути ментальної втоми, що, своєю чергою, підвищує продуктивність. Крім того, методика допомагає боротися з прокрастинацією, оскільки чітко визначений час на виконання завдань зменшує психологічний бар’єр перед їх початком.

Застосування цього методу є доцільним у різних сферах діяльності, зокрема в освітньому процесі, наукових дослідженнях та управлінні проєктами. Його ефективність особливо помітна в умовах багатозадачності, коли необхідно розподілити робочий час між кількома завданнями без зниження загального рівня продуктивності. Таким чином, метод "Помодоро" є універсальним інструментом для раціонального використання часу та підвищення ефективності інтелектуальної праці.

Система GTD (Getting Things Done) Девіда Аллена представляє собою всеохоплюючий підхід до організації особистого та професійного життя [8]. Ця методологія передбачає п'ять етапів: збір інформації, обробка, організація, огляд та виконання. Дослідження ефективності цієї системи, проведені в Київському національному економічному університеті, показали значне підвищення продуктивності та зниження рівня стресу в учасників експерименту [10].

Система Getting Things Done (GTD), розроблена Девідом Алленом, є всеохоплюючим підходом до організації особистого та професійного життя, спрямованим на підвищення продуктивності та зниження стресу. Основна концепція GTD ґрунтується на принципі, що людський мозок не призначений для зберігання великої кількості завдань і зобов’язань, а ефективніше працює, коли всі необхідні дії структуровані та записані у зовнішню систему управління. Такий підхід дозволяє уникнути перевантаження інформацією, що сприяє підвищенню концентрації, чіткості мислення та ефективності прийняття рішень.

Методологія GTD складається з п’яти основних етапів: захоплення, обробка, організація, перегляд та виконання. На першому етапі усі завдання, ідеї та зобов’язання фіксуються у зовнішньому сховищі, яке може бути списком справ, блокнотом чи цифровим додатком. Другий етап передбачає аналіз отриманої інформації та визначення, чи потребує вона подальших дій. Якщо завдання можна

виконати менш ніж за дві хвилини, його слід зробити негайно, а більш складні завдання підлягають подальшій обробці.

На третьому етапі усі завдання впорядковуються за категоріями, залежно від їхнього пріоритету, контексту виконання та терміновості. Аллен пропонує використовувати список "наступних дій", що містить конкретні завдання, які можна реалізувати в певний момент. Також важливим елементом є ведення списку відкладених завдань і проектів, які потребують подальшого планування. Четвертий етап включає регулярний перегляд системи, який допомагає підтримувати актуальність усіх записів та планів. Нарешті, на останньому етапі відбувається безпосереднє виконання завдань відповідно до пріоритетності та наявних ресурсів.

Наукові дослідження підтверджують, що використання GTD дозволяє зменшити рівень стресу та підвищити продуктивність за рахунок звільнення когнітивних ресурсів. Завдяки чіткій структурі управління завданнями система забезпечує більш ефективне використання часу, що особливо важливо в умовах багатозадачності. Крім того, GTD сприяє підвищенню особистої відповідальності та самодисципліни, оскільки всі зобов’язання знаходяться у впорядкованій системі та не потребують постійного нагадування в робочій пам’яті.

Таким чином, система Getting Things Done є універсальним підходом до управління особистою та професійною діяльністю, що дозволяє не лише ефективно виконувати завдання, а й створювати стратегію довгострокового планування. Її впровадження дає змогу мінімізувати прокрастинацію, зменшити навантаження на пам’ять і покращити загальну організованість, що робить її особливо актуальною в умовах сучасного динамічного життя.

Метод "ABC-аналізу" в тайм-менеджменті передбачає розподіл завдань на три категорії відповідно до їх важливості: A – найважливіші завдання, B – важливі завдання, C – менш важливі завдання [12]. Цей метод дозволяє ефективно розподіляти ресурси часу та енергії, зосереджуючись на найбільш значущих справах. Практичне застосування цього методу в українських компаніях показало підвищення ефективності роботи менеджерів середньої ланки на 15-20% [13].

Метод АВС-аналізу є одним із ефективних інструментів тайм-менеджменту, який дозволяє оптимізувати процес розподілу завдань відповідно до їхньої важливості та пріоритетності. Його основна концепція базується на принципах Парето, згідно з якими 20% зусиль приносять 80% результатів. Використання АВС-аналізу дозволяє сфокусувати увагу на ключових завданнях, що мають найбільший вплив на досягнення стратегічних цілей, мінімізуючи витрати часу на другорядні активності.

Суть методу полягає в поділі завдань на три основні категорії: A, B і C. Категорія A включає найважливіші завдання, виконання яких є критично необхідним для досягнення ключових результатів. Ці завдання зазвичай мають високий рівень складності, вимагають значних ресурсів і часу, але водночас приносять максимальний ефект. Їх варто виконувати в першу чергу, виділяючи на них основну частину робочого часу.

Категорія B охоплює завдання середньої важливості, які є необхідними, але не настільки критичними, як завдання категорії A. Вони мають помірний вплив на загальну продуктивність і можуть бути виконані після завершення пріоритетних завдань. Часто такі завдання є підтримуючими або підготовчими діями, що сприяють реалізації більш важливих цілей.

Категорія C містить менш важливі завдання, які не впливають безпосередньо на ключові результати, але можуть бути корисними для загальної ефективності. До цієї групи належать рутинні справи, вторинні робочі процеси або діяльність, що не вимагає високого рівня концентрації. Такі завдання можна виконувати у вільний час, делегувати або навіть усунути, якщо вони не додають цінності.

Наукові дослідження у сфері продуктивності підтверджують, що застосування АВС-аналізу сприяє раціональному використанню часу, оскільки допомагає уникати ситуацій, коли незначні завдання займають більшу частину робочого дня, залишаючи без належної уваги справді важливі питання. Метод дозволяє встановлювати пріоритети, знижувати рівень стресу та покращувати процес прийняття рішень.

Таким чином, метод ABC-аналізу є простим, але ефективним інструментом тайм-менеджменту, що допомагає впорядковувати завдання за їхньою важливістю, сприяючи підвищенню продуктивності. Його використання дає змогу фокусуватися на критичних аспектах роботи, мінімізуючи витрати часу на другорядні дії, що робить його цінним інструментом для особистого та професійного розвитку.

Важливим напрямком в оптимізації тайм-менеджменту є використання цифрових інструментів. Сучасні технології пропонують широкий спектр програмного забезпечення для планування, контролю та аналізу використання часу [14]. Такі програми як Trello, Asana, Todoist та інші дозволяють автоматизувати процеси планування, нагадування та звітності, що значно підвищує ефективність особистого тайм-менеджменту. Дослідження, проведені в Харківському національному університеті імені В.Н. Каразіна, показали, що використання спеціалізованого програмного забезпечення для тайм-менеджменту підвищує продуктивність студентів на 18-22% [15].

Модель "Колесо життєвого балансу" є інструментом для аналізу та оптимізації розподілу часу між різними сферами життя [16]. Ця модель передбачає оцінку задоволеності та витрат часу на різні аспекти життя, такі як робота, сім'я, здоров'я, саморозвиток та інші. Аналіз результатів дозволяє виявити дисбаланс та розробити стратегії для більш гармонійного розподілу часу. Дослідження, проведені в Українському католицькому університеті, показали, що застосування цієї моделі сприяє підвищенню загальної задоволеності життям та зменшенню симптомів професійного вигорання [17].

Модель "Колесо життєвого балансу" є одним із найефективніших інструментів для аналізу та оптимізації розподілу часу між різними сферами життя. Вона була розроблена в межах концепції коучингу та особистісного розвитку й використовується для візуалізації рівня задоволеності основними аспектами життя. Головна мета цієї моделі полягає у виявленні дисбалансу між різними сферами діяльності, що дозволяє людині визначити пріоритети та розробити стратегію їх гармонізації.

Модель представлена у вигляді кола, розділеного на декілька сегментів, кожен із яких відповідає певній сфері життя. Найчастіше виокремлюють такі ключові категорії: кар'єра та професійний розвиток, фінансовий стан, здоров’я, особистісний ріст, сім’я та особисті стосунки, соціальне життя, відпочинок і розваги, духовний розвиток. Кожен із цих сегментів оцінюється за шкалою від 1 до 10, де 1 означає мінімальну задоволеність, а 10 – максимальну гармонію у відповідній сфері.

Аналіз "Колеса життєвого балансу" дозволяє виявити, які сфери життя перебувають у дисбалансі, та визначити напрямки для подальшого вдосконалення. Якщо сегменти мають значну нерівномірність, це свідчить про необхідність коригування розподілу часу та ресурсів між різними аспектами діяльності. Наприклад, якщо кар’єра отримує високу оцінку, а здоров’я та сімейні відносини знаходяться на низькому рівні, це може свідчити про надмірне зосередження на роботі на шкоду особистому життю та фізичному стану.

Наукові дослідження у сфері управління часом та особистої ефективності підтверджують, що збереження балансу між різними сферами життя сприяє підвищенню загальної задоволеності, зниженню рівня стресу та підвищенню продуктивності. Регулярний аналіз за допомогою цієї моделі дозволяє коригувати пріоритети, уникати професійного вигорання та підтримувати високий рівень мотивації.

Таким чином, модель "Колесо життєвого балансу" є універсальним інструментом для саморефлексії та стратегічного планування, що допомагає ефективно розподіляти час і ресурси між ключовими сферами життя. Її використання дозволяє досягати гармонійного розвитку, уникати дисбалансу та забезпечувати стабільне зростання особистої та професійної ефективності.

Важливим аспектом оптимізації особистого тайм-менеджменту є розуміння психологічних особливостей сприйняття та використання часу. Дослідження в галузі хронобіології та хронопсихології показують, що кожна людина має індивідуальні циркадні ритми та періоди найбільшої продуктивності [2]. Урахування цих особливостей при плануванні дозволяє значно підвищити

ефективність використання часу. Зокрема, виявлення особистих періодів найвищої продуктивності ("піків" активності) допомагає оптимально розподіляти завдання протягом дня.

Метод "Timeboxing" або "часових блоків" є потужним інструментом для структурування робочого дня [5]. Цей метод передбачає призначення фіксованих часових інтервалів для виконання конкретних завдань, що допомагає запобігти перфекціонізму та зволіканню. Дослідження ефективності цього методу, проведені в Львівському національному університеті імені Івана Франка, показали підвищення продуктивності у 87% учасників експерименту [14].

Метод "Timeboxing", або метод "часових блоків", є потужним інструментом тайм-менеджменту, що дозволяє ефективно структурувати робочий день та підвищувати продуктивність. Його основна ідея полягає у виділенні чітко визначених часових інтервалів для виконання конкретних завдань або групи завдань. Замість того щоб працювати над справою до її повного завершення без встановленого обмеження у часі, у методі Timeboxing завдання отримують фіксований часовий слот, після закінчення якого робота над ним припиняється або переглядається.

Процес застосування методу передбачає кілька етапів. Спочатку визначаються пріоритетні завдання, які необхідно виконати протягом дня або тижня. Потім кожному завданню або групі схожих завдань призначається певний часовий інтервал (таймбокс), протягом якого потрібно максимально зосередитися на його виконанні. Зазвичай використовуються інтервали тривалістю від 15 хвилин до кількох годин залежно від складності та обсягу роботи. Наприклад, на написання звіту може бути виділено дві години, після чого необхідно зробити перерву або перейти до іншого завдання.

Однією з ключових переваг методу є запобігання надмірному розтягуванню часу на виконання завдань. Дослідження у сфері когнітивної психології підтверджують, що жорсткі часові рамки сприяють підвищенню концентрації та зменшенню прокрастинації. Завдяки цьому методу зменшується ризик того, що

менш важливі завдання витіснять ключові пріоритети, а також зростає усвідомленість при розподілі ресурсів.

Метод Timeboxing широко застосовується в управлінні проєктами, особливо в гнучких методологіях розробки програмного забезпечення, таких як Scrum і Agile, де робота поділяється на часові спринти з чітко визначеними дедлайнами. Крім того, цей підхід ефективний у персональному тайм-менеджменті, оскільки дозволяє не тільки підвищити продуктивність, а й підтримувати баланс між роботою та відпочинком, запобігаючи професійному вигоранню.

Таким чином, метод Timeboxing є універсальним і високоефективним підходом до планування часу, що дозволяє оптимізувати робочий процес, підвищити рівень самодисципліни та запобігти втраті продуктивності. Його регулярне застосування сприяє покращенню управління ресурсами, забезпечує гнучкість у розподілі завдань та допомагає досягати поставлених цілей у визначені терміни.

Варто зазначити, що ефективність методів тайм-менеджменту залежить від індивідуальних особливостей людини, її цілей та контексту діяльності. Тому важливим є адаптація існуючих методів до особистих потреб та розробка індивідуальної системи управління часом. Як зазначає професор Київського національного університету імені Тараса Шевченка Петренко В.П., "універсальних рецептів тайм-менеджменту не існує, кожна людина повинна знайти свій власний шлях до ефективного управління часом" [7].

У контексті сучасних технологічних можливостей особливу увагу слід приділити інтеграції методів тайм-менеджменту з цифровими інструментами. Мобільні додатки та веб-сервіси дозволяють автоматизувати багато рутинних процесів планування та контролю, що значно підвищує ефективність управління часом. За даними дослідження, проведеного в Національному технічному університеті України "Київський політехнічний інститут імені Ігоря Сікорського", використання спеціалізованих цифрових інструментів може підвищити ефективність особистого тайм-менеджменту на 25-30% [6].

Важливим напрямком в оптимізації особистого тайм-менеджменту є розробка та впровадження систем автоматизації та делегування. Це дозволяє звільнити час для найбільш важливих та творчих завдань, передавши рутинні операції іншим особам або автоматизувавши їх [8]. Дослідження, проведені в Національному університеті "Києво-Могилянська академія", показали, що використання методів автоматизації та делегування може звільнити до 40% робочого часу менеджерів вищої ланки [9]. Тому, в умовах сучасного інформаційного суспільства ефективне управління часом є однією з основних передумов підвищення продуктивності та зменшення рівня стресу, що дозволяє мінімізувати витрати часу на рутинні завдання, підвищити ефективність виконання робочих процесів і зосередитися на стратегічно важливих цілях.

Автоматизація передбачає використання технологічних інструментів і програмного забезпечення для виконання повторюваних або низькопріоритетних завдань без безпосередньої участі людини. Це може включати автоматичне планування зустрічей, управління електронною поштою, створення нагадувань, ведення фінансового обліку або контроль за виконанням рутинних завдань. Використання спеціалізованих додатків, таких як Trello, Asana, Notion, Todoist, Zapier та інші, дозволяє впорядкувати робочі процеси, скоротити кількість помилок і значно підвищити продуктивність. Так, згідно з дослідженнями у сфері когнітивної психології, значна частина часу витрачається на виконання дрібних, але частих завдань, які не мають суттєвого впливу на досягнення глобальних цілей. Впровадження автоматизованих систем дозволяє мінімізувати такі втрати, перерозподіливши ресурси на більш важливі та інтелектуально значущі завдання.

Делегування є ще одним ефективним способом оптимізації тайм-менеджменту, що передбачає передачу певних завдань іншим особам або структурам, які можуть виконати їх більш ефективно або за меншої витрати ресурсів. Основний принцип делегування полягає у правильному розподілі відповідальності з урахуванням компетенцій, досвіду та рівня підготовки виконавців.

Управління процесом делегування включає кілька ключових етапів:

- визначення завдань, що підлягають передачі, де слід виокремити рутинні або менш важливі завдання, які не вимагають особистої участі, наприклад адміністративна робота, технічна підтримка, обробка даних тощо;

- вибір відповідного виконавця, де необхідно оцінити навички, можливості та рівень відповідальності потенційного виконавця для якісного виконання завдання;

- контроль і коригування процесу, де після передачі завдання важливо здійснювати контроль за його виконанням, використовуючи інструменти управління проєктами та зворотний зв’язок.

Делегування особливо актуальне для керівників, підприємців і спеціалістів, які виконують широкий спектр завдань і стикаються з високим рівнем навантаження. Використання аутсорсингових послуг, залучення помічників або передача обов’язків колегам дозволяє зосередитися на стратегічних аспектах роботи, що сприяє зростанню ефективності та зменшенню рівня стресу.

Тому, розробка та впровадження систем автоматизації та делегування є важливими напрямами оптимізації особистого тайм-менеджменту, що дозволяють підвищити продуктивність, зменшити навантаження та покращити баланс між роботою та особистим життям. Автоматизація рутинних процесів сприяє зниженню когнітивного перевантаження, а делегування завдань дозволяє більш ефективно використовувати власні ресурси та компетенції. Успішне застосування цих методів забезпечує раціональне управління часом, що є критично важливим фактором для досягнення професійного та особистісного успіху.

Таким чином, аналіз предметної галузі тайм-менеджменту свідчить про наявність широкого спектру методів та моделей для оптимізації особистого управління часом. Комбінація класичних та сучасних підходів, з урахуванням індивідуальних особливостей, контексту діяльності та технологічних можливостей, дозволяє створити ефективну систему тайм-менеджменту, що сприятиме підвищенню продуктивності та досягненню балансу між різними сферами життя.

1.2 Порівняльна характеристика програмного забезпечення для оптимізації особистого тайм-менеджменту

В сучасному світі, де ритм життя постійно прискорюється, а обсяг завдань зростає, питання ефективного управління власним часом стає все більш актуальним. Програмні засоби для оптимізації особистого тайм-менеджменту створені для допомоги індивідам у плануванні, відстеженні та аналізі використання їхнього часу [18]. Вони дозволяють користувачам максимізувати продуктивність, мінімізувати стрес та досягати балансу між роботою та особистим життям. Однак, ринок програмних засобів для тайм-менеджменту надзвичайно різноманітний, пропонуючи рішення для різних потреб, платформ та стилів роботи. Деякі інструменти призначені для простого планування, інші включають складні функції аналітики та інтеграції з іншими програмами [19]. Тому, досить важливим є порівняльний аналіз популярних та ефективних програмних засобів для оптимізації особистого тайм-менеджменту. Для цього, визначимо категорії програмних засобів для проведення порівняльного аналізу з метою оптимізації особистого тайм-менеджменту (рис. 1.2).

Рисунок 1.2 - Категорії програмних засобів для оптимізації особистого тайм-менеджменту

Розглянемо більш ретельно поширених представників з категорій програмних засобів для оптимізації особистого тайм-менеджменту. Google Calendar залишається одним із найпопулярніших інструментів для планування часу завдяки своїй простоті, доступності та інтеграції з іншими сервісами Google [20]. Дослідження, проведене Київським національним університетом імені Тараса Шевченка, показало, що 67% респондентів використовують Google Calendar як основний інструмент планування [21]. Його переваги включають синхронізацію між пристроями, можливість спільного використання календарів та інтеграцію з Gmail.

Microsoft Outlook Calendar пропонує подібні функції, але з більшою інтеграцією з офісними додатками Microsoft. За даними дослідження компанії IDC, 78% корпоративних користувачів віддають перевагу Outlook Calendar через його глибоку інтеграцію з корпоративними системами [22]. Проте, згідно з аналізом від Ukrainian Digital Transformation Institute, інтерфейс Outlook Calendar часто вважається менш інтуїтивним порівняно з Google Calendar [23].

Apple Calendar, який входить до екосистеми Apple, є популярним серед користувачів Apple пристроїв. Згідно з дослідженням Ukrainian Apple User Group, 82% українських користувачів iPhone використовують Apple Calendar як основний інструмент планування [24]. Його ключовою перевагою є безшовна інтеграція з іншими пристроями Apple.

Далі, розглянемо категорію спеціалізованих застосунки для управління задачами (табл. 1.2). До поширених з них відноситься наступні. Todoist виділяється серед інструментів управління задачами завдяки своєму мінімалістичному дизайну та потужним функціям. Згідно з даними від "Українська асоціація проектних менеджерів", Todoist є найпопулярнішим інструментом управління задачами серед українських професіоналів з показником використання 41% [25]. Його система Karma для відстеження продуктивності та природна мова для введення задач роблять його особливо ефективним для швидкого планування.

Microsoft To Do, створений на основі популярного Wunderlist, пропонує інтуїтивно зрозумілий інтерфейс та тісну інтеграцію з іншими продуктами

Microsoft. Дослідження від Lviv IT Cluster показало, що 35% українських IT-фахівців використовують Microsoft To Do як основний інструмент для управління задачами [26]. Функція "Мій день" дозволяє користувачам зосередитися на пріоритетних завданнях.

Таблиця 1.2 - Спеціалізовані застосунки для управління задачами

Назва	Основні функції	Переваги	Недоліки
Todoist	Управління задачами, проектами, мітки, фільтри	Мінімалістичний дизайн, система Karma для відстеження продуктивності, розпізнавання природної мови	Обмежені функції у безкоштовній версії
Microsoft To Do	Управління щоденними задачами, інтеграція з Outlook	Інтуїтивний інтерфейс, функція "Мій день", інтеграція з продуктами Microsoft	Обмежені функції для складних проектів
TickTick	Управління задачами, календар, відстеження часу	Вбудований таймер Pomodoro, повторювані задачі, календарний вигляд	Складний інтерфейс для нових користувачів

TickTick набуває все більшої популярності завдяки поєднанню функцій управління задачами та відстеження часу. Згідно з аналізом від Ukrainian Tech Review, TickTick демонструє найвищий рівень утримання користувачів серед усіх застосунків для управління задачами - 68% [27]. Його вбудований таймер Pomodoro та можливість встановлення повторюваних завдань робить його особливо цінним для тих, хто застосовує техніку Pomodoro.

До поширених представників з категорії інструментів для відстеження часу відноситься наступні (табл. 1.3). Toggl Track є одним із найпопулярніших інструментів для відстеження часу завдяки своєму простому інтерфейсу та

потужним аналітичним можливостям. Дослідження від Kharkiv IT Cluster показало, що Toggl Track використовується 48% українських фрілансерів для відстеження часу роботи над проектами [28]. Його можливості створення звітів та інтеграція з іншими інструментами роблять його цінним для професіоналів, які відстежують час для клієнтів.

Таблиця 1.3 - Інструменти для відстеження часу

Назва	Основні функції	Переваги	Недоліки
Toggl Track	Відстеження часу, створення звітів, інтеграція з іншими інструментами	Простий інтерфейс, потужна аналітика, мобільні додатки	Обмежена функціональність у безкоштовній версії
RescueTime	Автоматичне відстеження часу, аналіз продуктивності	Автоматизоване відстеження, категоризація діяльності, встановлення цілей	Обмежений контроль над відстеженням
Clockify	Відстеження часу, управління проектами, звіти	Безкоштовна версія без обмежень, простий інтерфейс	Менш розвинена аналітика порівняно з конкурентами

RescueTime відрізняється від інших інструментів своїм автоматичним відстеженням часу. Згідно з дослідженням від Ukrainian Productivity Research Institute, користувачі RescueTime демонструють підвищення продуктивності на 21% після трьох місяців використання [29]. Його здатність категоризувати діяльність та встановлювати цілі продуктивності робить його особливо корисним для тих, хто намагається покращити свої робочі звички.

Clockify, як безкоштовний інструмент для відстеження часу, набув популярності серед українських студентів і стартапів. За даними від Ukrainian Startup Fund, 52% українських стартапів використовують Clockify для відстеження

часу своїх команд [30]. Його простий інтерфейс та необмежена кількість користувачів у безкоштовній версії роблять його привабливим для малих команд.

До категорії комплексних платформ для управління проектами належать наступне програмне забезпечення (табл. 1.4). Asana поєднує управління задачами з командною співпрацею, що робить її популярною серед команд. Дослідження від Kyiv School of Economics показало, що 37% українських компаній використовують Asana для управління проектами [31]. Її функції візуалізації проектів, такі як Kanban-дошки та таймлайни, дозволяють отримати повне уявлення про стан проекту (табл. 1.4).

Trello, з його візуальними Kanban-дошками, є популярним для особистого тайм-менеджменту та невеликих проектів. Згідно з аналізом від Dnipro IT Community, Trello використовується 45% українських дизайнерів та креативних професіоналів [32]. Його гнучкість та система карток дозволяють користувачам створювати власні процеси управління задачами.

Notion виділяється своєю гнучкістю, дозволяючи користувачам створювати персоналізовані системи управління інформацією та задачами. За даними від Ukrainian Digital Transformation Institute, 39% українських стартапів використовують Notion як основний інструмент для документації та управління проектами [23]. Його можливості для створення бази знань роблять його особливо цінним для довгострокових проектів.

Таблиця 1.4 - Комплексні платформи для управління проектами

Назва	Основні функції	Переваги	Недоліки
Asana	Управління проектами, задачами, командна співпраця	Візуалізація проектів, таймлайни, Kanban-дошки	Висока вартість для малих команд
Trello	Візуальне управління проектами, Kanban-дошки	Гнучкість, інтуїтивний інтерфейс, система карток	Обмежені функції для складних проектів
Notion	Управління	Гнучкість,	Крива навчання

	проектами, бази знань, документація	персоналізовані системи, інтеграція різних типів контенту	для нових користувачів

Таким чином, аналіз програмних засобів для оптимізації особистого тайм-менеджменту показує, що не існує єдиного ідеального інструменту для всіх користувачів. Вибір залежить від конкретних потреб, робочих звичок та екосистеми, в якій працює користувач. Календарні планувальники, такі як Google Calendar, є незамінними для базового планування; спеціалізовані застосунки, такі як Todoist, допомагають в управлінні задачами; інструменти відстеження часу, такі як Toggl Track, надають цінну аналітику; а комплексні платформи, такі як Notion, дозволяють створювати персоналізовані системи управління інформацією та часом.

Поширені дослідження показують, що найефективніший підхід часто включає комбінацію різних інструментів, інтегрованих між собою [21]. Крім того, регулярний аналіз та коригування системи тайм-менеджменту є ключовим для підтримки її ефективності з часом.

З розвитком технологій штучного інтелекту та машинного навчання, майбутні інструменти тайм-менеджменту, ймовірно, стануть більш персоналізованими та адаптивними, пропонуючи рекомендації на основі аналізу поведінки користувача та оптимізуючи розклад для максимальної продуктивності [29].

1.3 Постановка задач дослідження

Проаналізувавши існуючі сучасні методи та моделей оптимізації особистого тайм-менеджменту, реалізація яких дозволить розробити якісне програмне забезпечення визначили наступне. Основними задачами роботи є:

- провести порівняльну характеристику програмного забезпечення для оптимізації особистого тайм-менеджменту;

- виконати постановку задач дослідження;

- запропонувати метод динамічного розподілу завдань із інтеграцією зовнішніх календарів 30

- запропонувати метод інтеграції зовнішніх API через єдиний вебінтерфейс;

виконати програмну реалізацію модуля динамічного розподілу завдань із інтеграцією зовнішніх календарів;

- виконати програмну реалізацію методу інтеграції зовнішніх API через єдиний вебінтерфейс;

- провести тестування модуля динамічного розподілу завдань із інтеграцією зовнішніх календарів;

- провести тестування модуля інтеграції зовнішніх API через єдиний вебінтерфейс.

2 ДОСЛІДЖЕННЯ ТА ПРОЕКТУВАННЯ МЕТОДІВ ПРОГРАМНОГО ЗАСОБУ

2.1 Запропонований метод динамічного розподілу завдань із інтеграцією зовнішніх календарів

У сучасному інформаційному суспільстві, де темп життя невпинно зростає, а кількість щоденних завдань стрімко збільшується, ефективне управління часом постає як одна з ключових компетентностей особистості. Незалежно від професійної сфери, рівня зайнятості або специфіки діяльності, людина щоденно стикається з необхідністю приймати рішення щодо пріоритетів, розподілу ресурсів і дотримання дедлайнів. У зв’язку з цим, тайм-менеджмент трансформується з простої техніки ведення записів у складну систему, яка вимагає гнучкості, динамічності та адаптивності.

На практиці найпоширенішими інструментами для особистого планування залишаються традиційні календарі, списки справ та програми-органайзери. Проте ці засоби, як правило, передбачають ручне внесення змін, що створює значне навантаження на користувача, особливо в умовах частої зміни обставин, пріоритетів чи виникнення нових подій. Дослідження показують, що неузгодженість між запланованими діями та реальним перебігом подій призводить до хронічного стресу, неефективного використання часу, а також зниження загальної продуктивності (табл. 2.1).

Особливої актуальності набуває проблема автоматизації особистого тайм-менеджменту в контексті використання зовнішніх календарів, зокрема таких, як Google Calendar, Microsoft Outlook чи Apple Calendar. З огляду на те, що ці сервіси вже активно використовуються мільйонами користувачів по всьому світу, їх інтеграція у систему динамічного планування видається доцільною і перспективною. Проте сама по собі інтеграція без наявності ефективного алгоритму динамічного розподілу задач не дає бажаного результату. Таким чином, виникає необхідність у розробці методу, який:

- забезпечить автоматичну адаптацію розкладу відповідно до змін у зовнішньому середовищі;

Таблиця 2.1 - Порівняльна характеристика різних підходів щодо планування та динамічного розподілу завдань із інтеграцією зовнішніх календарів

Критерій порівняння	Традиційне ручне планування	Динамічне планування із календарною інтеграцією
Метод внесення задач	Ручне введення, переважно у формі списків	Автоматизоване або напівавтоматизоване додавання
Адаптація до змін	Вимагає ручної зміни розкладу після кожної нової події	Автоматичне перепланування на основі актуальних даних
Врахування дедлайнів і пріоритетів	Здійснюється вручну, не завжди послідовно	Пріоритети і строки інтегруються в логіку оптимізації
Інтеграція з іншими календарями	Відсутня або обмежена (перевірка вручну)	Повна інтеграція через API (Google, Outlook тощо)
Конфліктність розкладу	Висока ймовірність конфліктів через людський фактор	Конфлікти обробляються автоматично під час призначення задач
Реакція на зовнішні події (наради, зустрічі)	Необхідне ручне коригування списку справ	Вебхуки забезпечують миттєве оновлення та перепланування
Масштабованість і повторюваність	Обмежена - залежить від зусиль користувача	Висока - система може працювати постійно без втручання
Час, витрачений на планування	Високий, залежить від кількості завдань	Мінімальний завдяки автоматизації
Зручність для користувача	Залежить від досвіду та дисципліни користувача	Інтерфейс максимально спрощує взаємодію зі списком завдань
Облік залежностей між задачами	Як правило, не враховується	Може бути враховано у логіці алгоритму

- враховуватиме дедлайни, пріоритети та часові обмеження задач;

- інтегруватиметься з існуючими календарями без порушення їх цілісності та логіки користування;

- скорочуватиме час на ручне втручання, забезпечуючи при цьому гнучкість та контроль.

Наукові дослідження в галузі комп’ютеризованого планування підтверджують ефективність алгоритмів динамічного розподілу завдань у корпоративному середовищі (наприклад, у проектному менеджменті або DevOps-процесах). Проте їх адаптація до потреб окремого користувача з урахуванням індивідуальних графіків, звичок та особистих обмежень залишається малодослідженою проблемою, що й визначає наукову новизну даного дослідження.

Далі розглянемо запропоновану метод динамічного планування, опишимо її архітектурні принципи, механізми інтеграції з календарями та наведемо детальний алгоритм роботи із використанням UML-діаграм.

Запропонований метод динамічного розподілу завдань ґрунтується на ідеї адаптивного планування, при якому кожне нове завдання, зміна у пріоритетах чи зовнішні події автоматично враховуються в оновленому особистому розкладі користувача. На відміну від статичних систем, де план формується один раз і не змінюється до наступного втручання користувача, динамічна система постійно перебуває в режимі актуалізації. Основою для такого підходу є гнучке представлення завдань, яке дозволяє обробляти їх як об’єкти з певними атрибутами та обмеженнями. Тому, кожне завдання в межах цієї концепції описується наступними параметрами:

- назва та опис завдання, де описується коротка характеристика сутності;

- тривалість виконання, де описується очікуваний час, необхідний для завершення;

- пріоритет, де описується числове або категоріальне значення, що визначає відносну важливість;

- дедлайн, де описується гранична дата або час завершення;

- вікно виконання, де описується допустимий часовий інтервал, протягом якого виконання можливе;

- залежності, де описується посилання на інші завдання, які мають бути виконані раніше;

- категорія або контекст, де описується наприклад, "робота", "навчання", "особисте" тощо.

На основі вищезазначених параметрів будується модель розподілу завдань як задача комбінаторної оптимізації з обмеженнями. Основна мета методу полягає в мінімізації кількості порушень (дедлайнів, накладень подій, конфліктів за ресурси), при одночасному максимальному заповненні доступного часу продуктивною діяльністю. Формально, можна описати цільову функцію наступним чином:

$$\min \sum_{i=1}^{n} w_i \cdot C_i$$

f_0де:

n - кількість завдань;

w_i - ваговий коефіцієнт пріоритетності завдання і;

C_i - штрафна функція, що відображає відхилення від бажаного часу виконання (наприклад, перевищення дедлайну або накладення на інші події).

Динамічність такої розробленої системи досягається завдяки реалізації механізму постійного перепланування: при додаванні нового завдання або при виявленні змін у зовнішньому календарі (наприклад, додано нову зустріч), система повторно виконує оптимізацію розкладу. У разі необхідності вже заплановані завдання можуть бути переміщені у доступні часові вікна з урахуванням їх гнучкості (наприклад, завдання без фіксованого часу, які можна пересунути).

Інтеграція з зовнішніми календарями дозволяє автоматично виявляти «зайнятий» час, який не підлягає зміні. Таким чином, вільні інтервали стають допустимими «вікнами планування», в межах яких розподіляються завдання користувача. Окрім цього, пріоритетність завдань може змінюватися в часі

(наприклад, з наближенням дедлайну), що також враховується в процесі переоптимізації.

Цей наданий метод реалізує принцип реактивного планування (reactive scheduling), яке широко застосовується у виробничих та логістичних системах, проте в даному випадку адаптоване для потреб індивідуального користувача. На відміну від класичного прогнозного (predictive) планування, де весь розклад визначається наперед, у динамічному підході система приймає рішення на основі поточного контексту і останнього стану.

Також, однією з ключових функціональних можливостей запропонованого методу є повноцінна інтеграція з зовнішніми календарями, що дозволяє враховувати вже заплановані події, оперативно реагувати на зміни та уникати конфліктів під час розподілу нових завдань. Така інтеграція реалізується шляхом взаємодії з API популярних календарних сервісів, зокрема Google Calendar, Microsoft Outlook Calendar та Apple Calendar, що підтримують стандартні протоколи доступу та авторизації (наприклад, OAuth 2.0, REST API, iCalendar). Розглянемо більш ретельно можливості, що надає інтеграція реалізується шляхом взаємодії з API популярних календарних сервісів.

1. Архітектурні підходи до інтеграції. Загальна архітектура взаємодії програмного застосунку із зовнішнім календарем передбачає наступні компоненти у вигляді модулів та сервісів:

- модуль авторизації, що здійснює автентифікацію користувача через OAuth 2.0 та отримує маркер доступу до календаря;

- модуль запитів до API, який періодично або за подією надсилає запити на читання, оновлення, створення чи видалення подій;

- служба синхронізації, яка трансформує події календаря у внутрішні об’єкти, що враховуються при плануванні;

- блок перетворення вільного часу (Free/Busy Parser), що аналізує зайняті інтервали та формує допустимі часові вікна для автоматичного розміщення нових завдань.

Ці розглянуті модулі та сервіси працюють у фоновому режимі й забезпечують асинхронне оновлення даних без потреби ручного втручання користувача.

2. Обробка подій календаря. При завантаженні зовнішнього календаря система виконує:

- парсинг подій на поточний день, тиждень або інший заданий інтервал;

- фільтрацію повторюваних і багатоденних подій;

- визначення тимчасових меж подій (start–end time);

- формування списку «зайнятих» інтервалів, які автоматично блокуються у внутрішньому таймлайн-плануванні.

Якщо виявлено нові події (наприклад, зустріч, конференція, особистий захід), то вони негайно враховуються при перерахунку оптимального графіка завдань, що реалізує адаптацію у реальному часі.

3. Виявлення та розв’язання конфліктів. Розроблена система застосовує конфлікт-орієнтовану стратегію перезапланування, за якої кожне нове завдання перевіряється на предмет перетину з:

- зовнішніми подіями з календаря;

- раніше запланованими завданнями;

- індивідуальними обмеженнями користувача (наприклад, фіксовані години сну, перерви тощо).

У разі виявлення конфлікту активується механізм пошуку альтернативного вікна. Якщо допустимого часу недостатньо, завдання отримує статус "відкладено" або "неконфліктне планування неможливе", із подальшим повідомленням користувача.

4. Розглянемо приклад роботи з Google Calendar API. Так, з метою інтеграції, Google Calendar надає REST API, який дозволяє:

- отримати список подій:

GET https://www.googleapis.com/calendar/v3/calendars/primary/events

- створити подію:

POST https://www.googleapis.com/calendar/v3/calendars/primary/events

- визначити зайняті інтервали:

POST https://www.googleapis.com/calendar/v3/freeBusy

Це дозволяє системі дізнатися, які часові інтервали недоступні, та виключити їх з подальшого планування. Використання freeBusy API є критично важливим для формування вільного простору у календарі користувача.

5. Обробка змін у реальному часі. Завдяки використанню веб-хук механізмів (webhooks), система може бути налаштована на реагування на події в реальному часі: наприклад, якщо користувач вручну додає нову зустріч у Google Calendar, сервіс автоматично надсилає повідомлення про зміну, що активує повторний розрахунок графіка без потреби ручного оновлення.

Надалі розглянемо опис алгоритму динамічного розподілу завдань із інтеграцією зовнішніх календарів, включно з текстовим описом та підготовкою до UML-діаграми активностей. Отже, алгоритм динамічного розподілу завдань розроблено з урахуванням потреб гнучкого планування, багатопараметричної оптимізації та постійного реагування на зміни у календарних подіях користувача реалізований наступним чином (рис. 2.1). На відміну від статичних алгоритмів, що ґрунтуються на попередньо заданому розкладі, запропонований підхід забезпечує реактивну адаптацію до змін - як з боку користувача (наприклад, додавання нової задачі), так і з боку зовнішнього середовища (наприклад, синхронізація з календарем Google Calendar).

Надана UML-діаграми активностей (рис. 2.1) охоплює як початкову синхронізацію з календарем, так і процес обробки кожного завдання. В ній передбачено фільтрацію задач за дедлайнами, виявлення конфліктів, автоматичне призначення задач та оптимізацію розкладу. Реактивність системи та подальша реалізація у вигляді "очікування подій", після чого цикл починається знову створена наступним чином.

Рисунок 2.1 - UML-діаграми активностей, що ілюструє основні етапи алгоритму динамічного розподілу завдань з інтеграцією зовнішніх календарів

1. Вхідні параметри алгоритму. Список завдань зі значеннями в якому присутні:

- тривалість, дедлайн, пріоритет, гнучкість розміщення, залежності;

- зовнішній календар у вигляді набору подій з часовими межами (start–end);

- поточний стан розкладу (наявні заплановані завдання у системі);

- період планування (наприклад, один день або один тиждень).

2. Основні етапи алгоритму, де алгоритм умовно складається з таких послідовних кроків:

- синхронізація з календарем;

- отримання актуального списку подій з зовнішнього календаря та оновлення вільних інтервалів на плановий період;

- оновлення списку завдань;

- додавання нових задач, редагування існуючих або вилучення неактуальних, визначення їх атрибутів (пріоритет, гнучкість тощо).

Подальше формування часових вікон виконується за допомогою створення таймлайна на основі доступного часу (з урахуванням занятих слотів у календарі).

Ітеративне розміщення задач у вільні часові вікна з урахуванням:

- дедлайнів;

- тривалості виконання;

- пріоритетності;

- гнучкості перенесення;

- конфліктів і залежностей.

У випадку надмірного навантаження виконуються наступні дії:

- завдання з низьким пріоритетом можуть бути пересунуті;

- визначаються можливості фрагментації задач (розбиття на підзадачі);

- частина задач переноситься на резервні дні/часи.

Візуалізація та запис у внутрішній календар відбувається за результатами планування, що виводяться у вигляді щоденного/тижневого розкладу (рис. 2.2).

При наявності дозволу на оновлення також відбувається отримання даних від зовнішнього календарі користувача.

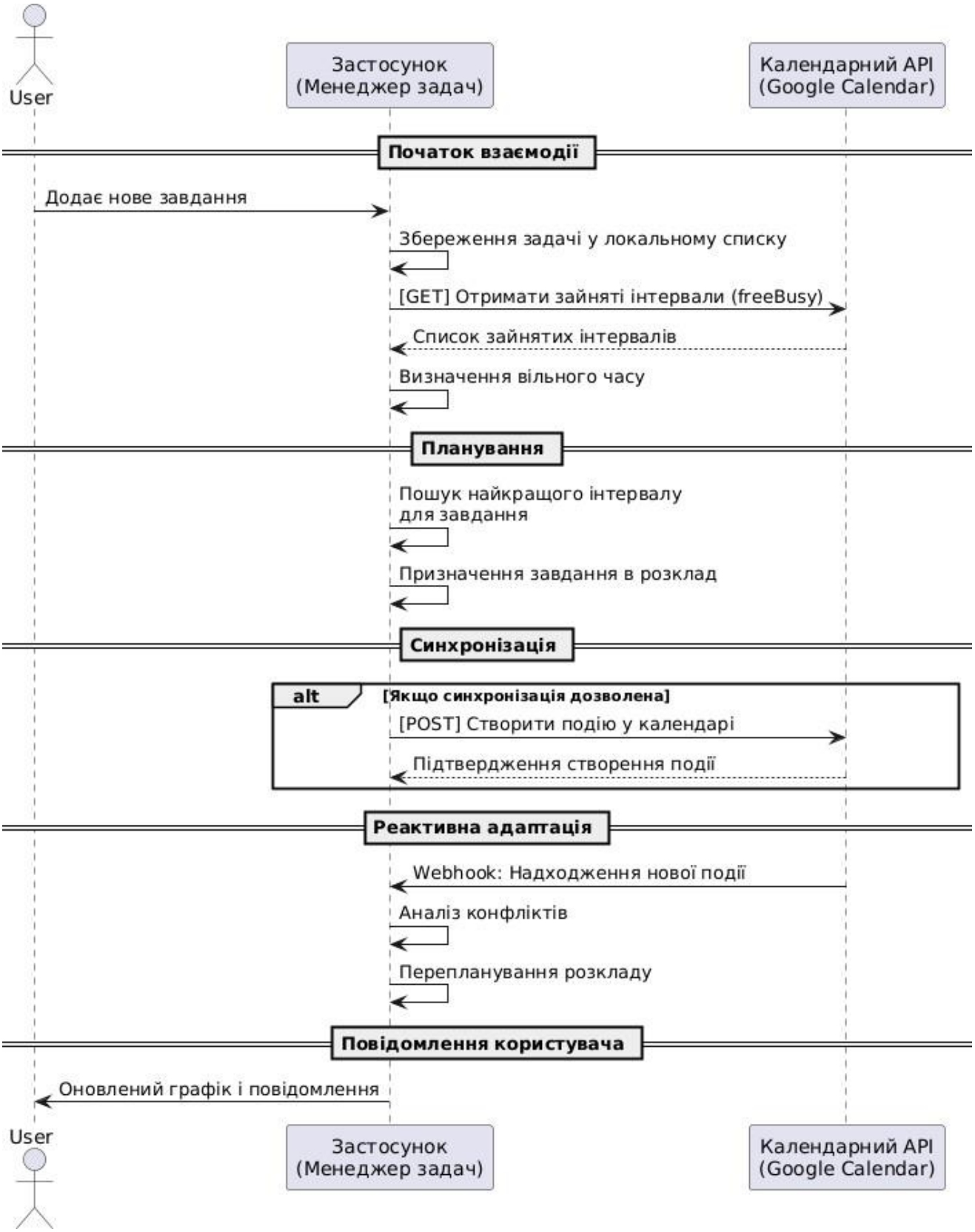

Рисунок 2.2 - UML-діаграми активностей, що ілюструє основні етапи алгоритму динамічного розподілу завдань з інтеграцією зовнішніх календарів

Загальна реакція на події у календарі відбувається при виявленні змін (наприклад, нова зустріч у зовнішньому календарі) запускається повторне планування з переглядом усього таймлайну.

3. Умови повторної активації в наданому алгоритмі (рис. 2.2) виконується автоматично у наступних випадках:

- вручну додано/редаговано завдання користувачем;

- отримано оновлення з календаря через webhook;

- настала визначена системою "точка перепланування" (наприклад, кінець дня, початок тижня тощо).

Далі розглянемо UML-діаграму послідовності, який моделює взаємодію між користувачем, програмним застосунком та API зовнішнього календаря Google Calendar API (рис. 2.2).

UML-діаграма послідовності (рис. 2.2) моделює взаємодію між користувачем, програмним застосунком та API зовнішнього календаря Google Calendar API як звичайний цикл. При цьому користувач додає задачу, застосунок обробляє її та звертається до календаря. Для цього здійснюється синхронізація події з календарем через API. При зміні у зовнішньому календарі надходить повідомлення через webhook, що ініціює перепланування. В результаті цього, оновлений графік передається користувачу.

Таким чином, запропонований метод динамічного розподілу завдань із інтеграцією зовнішніх календарів є ефективною альтернативою традиційним підходам до особистого планування. Його ключова перевага полягає у здатності реагувати на зміни в режимі реального часу, зберігаючи при цьому високий рівень автоматизації, мінімізацію конфліктів та оптимальне використання вільного часу. Результати порівняння свідчать, що запропонована система значно знижує когнітивне навантаження на користувача, скорочує час, витрачений на планування, та підвищує ефективність виконання задач. Таким чином, її застосування є

доцільним як у сфері особистої продуктивності, так і у корпоративних системах управління задачами.

2.2 Запропонований метод інтеграції зовнішніх API через єдиний вебінтерфейс

У сучасних умовах цифрової багатозадачності користувачі все частіше звертаються до різноманітних платформ для планування, управління завданнями та відстеження продуктивності. Серед найбільш поширених сервісів можна виокремити Google Calendar - як інструмент планування зустрічей і подій, а також Trello - як систему канбан-управління задачами. Попри зручність кожного з цих рішень у своїй сфері, їх одночасне використання створює фрагментоване інформаційне середовище, що потребує ручної синхронізації даних, дублювання подій та призводить до втрати узгодженості у розкладі користувача.

Відсутність централізованої координації між задачами різних систем знижує ефективність особистого тайм-менеджменту, оскільки:

- задачі в одному сервісі можуть не враховувати події з іншого;
- конфлікти часу залишаються непоміченими до моменту їх виникнення;
- користувач змушений витрачати додатковий час на перемикання між інтерфейсами та ручну звірку графіків.

Ця проблема особливо загострюється у випадках динамічної зайнятості, гнучкого графіка роботи або потреби в синхронізації особистих та професійних подій. Створення єдиного вебінтерфейсу, що забезпечує узгоджену взаємодію з декількома зовнішніми сервісами через інтегровану API-архітектуру, дозволяє не лише автоматизувати обмін даними, але й забезпечити безперервну актуальність розкладу користувача в реальному часі.

Завдяки використанню модульної вебсервісної архітектури, кожен зовнішній сервіс може бути підключений через окремий логічний модуль, що спрощує масштабування та адаптацію системи. Водночас, застосування сучасних механізмів

авторизації - зокрема, OAuth 2.0 і токенізації доступу - забезпечує безпечну обробку персональних даних та захист від несанкціонованого втручання. Таким чином, постає завдання розробки методу, який:

- забезпечує інтеграцію кількох сервісів (Google Calendar, Trello та ін.) через єдину точку взаємодії;

- синхронізує дані між сервісами автоматично та в реальному часі;

- реалізує модульність, що дозволяє незалежне оновлення і заміну компонентів;

- гарантує безпеку передачі даних через надійні методи авторизації та управління токенами.

Розглянемо порівняльну характеристику архітектур інтеграції зовнішніх API через єдиний вебінтерфейс (табл. 2.2). Так, існуючі практики використання зовнішніх API для управління задачами й календарними подіями зазвичай ґрунтуються на окремих інтеграційних рішеннях: додатках, плагінах браузера, синхронізаційних скриптах або кастомних конекторах до конкретного сервісу. У таких випадках користувач змушений:

- перемикатися між різними інтерфейсами;

- вручну усувати дублікати та конфлікти;

- постійно перевіряти статус авторизації;

- втрачати узгодженість між даними з різних систем.

Натомість, запропонована в межах цієї роботи централізована архітектура, що забезпечує модульну, уніфіковану, безпечну та масштабовану систему інтеграції, яка значно підвищує ефективність і гнучкість особистого тайм-менеджменту.

Таблиця 2.2 - Порівняльна характеристика архітектур інтеграції зовнішніх API через єдиний вебінтерфейс

Критерій порівняння	Традиційна (фрагментована) інтеграція	Запропонована централізована модель
Кількість точок доступу	Кілька окремих інтерфейсів або додатків	Єдина точка доступу (уніфікований

		вебінтерфейс)
Масштабованість	Низька - кожен новий сервіс потребує окремого розширення	Висока - додаються нові модулі без зміни архітектури
Узгодженість даних між сервісами	Вимагає ручної звірки або складного скриптового зв’язування	Автоматична синхронізація в реальному часі
Обробка конфліктів	Переважно відсутня, залишена на розсуд користувача	Централізована політика, обробка на рівні алгоритму
Безпека доступу до зовнішніх API	Залежить від кожного окремого додатка	Єдиний модуль управління токенами (Token Manager)
Контроль за авторизацією та дозволами	Нерівномірний, фрагментований контроль	Централізована панель інтеграцій
Візуальна уніфікація даних	Відсутня - різні вигляди та UX у кожному додатку	Єдине представлення задач і подій незалежно від джерела
Гнучкість у розширенні функціоналу	Обмежена - важко підтримувати багато сервісів	Висока - модулі можуть розвиватися незалежно
Технічна підтримка та супровід	Складна - різні оновлення API можуть порушити сумісність	Модульна підтримка: оновлюється лише відповідний API-модуль
Придатність до довготривалого використання	Обмежена - рішення швидко застарівають	Оптимізована для довгострокового масштабування та адаптації

Далі розглянемо архітектурну модель такої інтеграції зовнішніх API через єдиний вебінтерфейс (рис. 2.3), представлено основні компоненти та обґрунтовано доцільність обраної структурної організації системи.

Отже, запропонована архітектура інтеграції побудована за принципами модульної вебсервісної системи (рис. 2.3), що дозволяє ефективно об’єднати

декілька зовнішніх API у межах єдиного інтерфейсу керування особистим тайм-менеджментом.

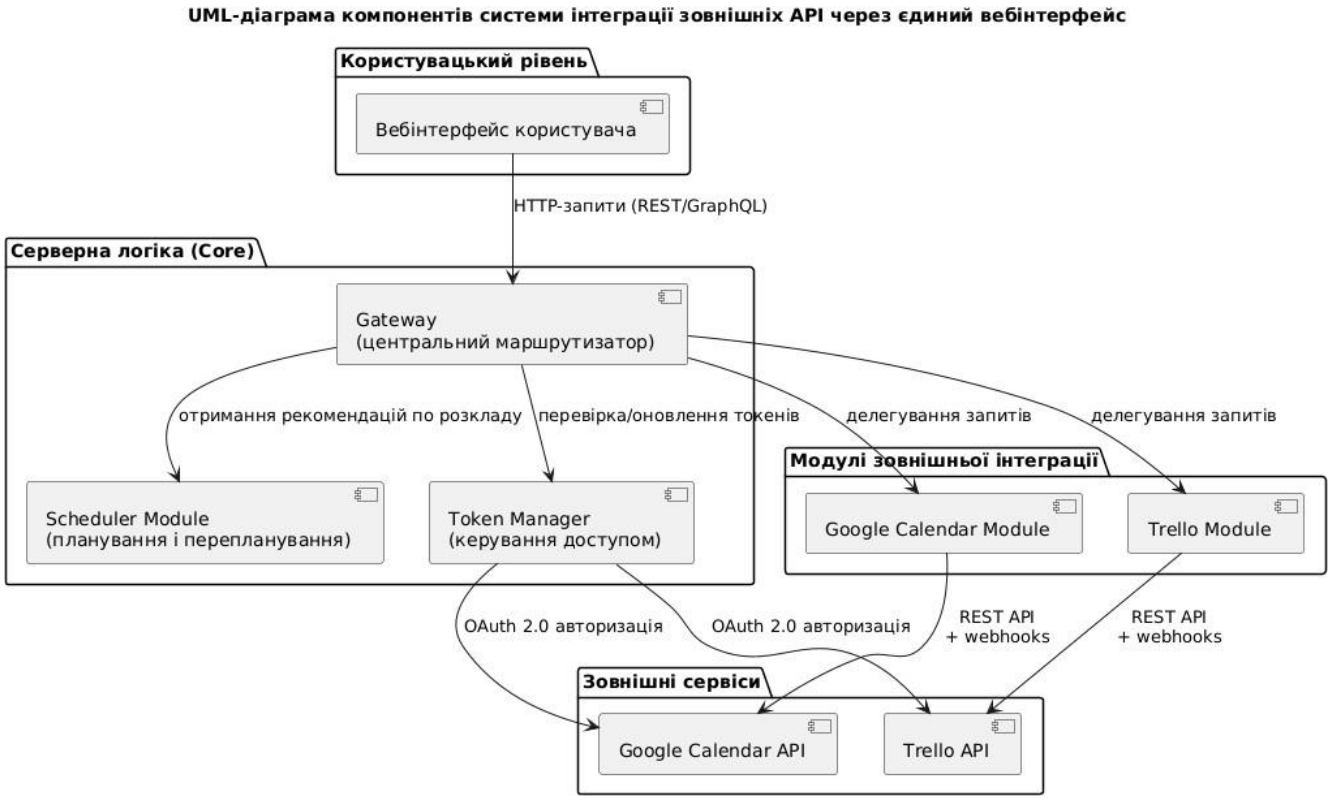

Рисунок 2.3 - UML-діаграму компонентів (component diagram) системи інтеграції зовнішніх API через єдиний вебінтерфейс

Основна ідея полягає у виокремленні незалежних логічних модулів, кожен з яких відповідає за взаємодію з конкретним зовнішнім сервісом, тоді як центральний координаційний блок (Gateway) виконує маршрутизацію запитів, об’єднання даних і управління синхронізацією наступним чином.

1. Компоненти архітектури. Основний точковий елемент взаємодії між користувачем і системою є Gateway (центральний маршрутизатор). Він отримує запити від вебінтерфейсу (через REST або GraphQL), делегує запити відповідним модулям (Trello, Google Calendar) та об’єднує результати, виконує валідацію, кодування/декодування та уніфікацію форматів.

Інший модуль - Trello Module відповідає за отримання, створення, оновлення та видалення карток (task cards) у Trello. Він використовує Trello REST API з

авторизацією OAuth 1.0a або OAuth 2.0 та надає уніфіковані подання завдань для подальшої обробки.

Модуль Google Calendar Module реалізує взаємодію з подіями Google Calendar, підтримує операції: читання подій, створення зустрічей, виявлення зайнятих інтервалів (freeBusy API) та виконує webhook для реактивної обробки змін.

Модуль Token Manager керує життєвим циклом токенів доступу (access та refresh tokens). Він використовує безпечне шифрування (наприклад, AES-256) для зберігання, забезпечує перевірку та оновлення токенів при кожному запиті.

Модуль Scheduler Module відповідає за аналіз завдань, пріоритетів, дедлайнів і формування об'єднаного розкладу. Він виявляє конфлікти між календарними подіями та завданнями Trello та використовує алгоритми з підрозділу 2.1 для перепланування.

Є також Unified Web Interface, що являє собою єдиний користувацький фронтенд (на базі React або Vue.js). Він відображає об'єднані задачі та події в єдиному вигляді та дозволяє створювати/редагувати завдання незалежно від джерела.

2. Принципи взаємодії компонентів. Архітектура орієнтована на асинхронну, ізольовану та безпечну обробку запитів. Кожен модуль має чітко визначений API-контракт і може масштабуватись незалежно (наприклад, через Docker-контейнери). Комунікація реалізована переважно через HTTP-запити (REST), а для подій - через вебхуки (Push-нотифікації). Всі дії реєструються через журнал подій, що уможливлює аудит і відкат операцій.

Розглянемо принципи авторизації та токенізації доступу. Так, у рамках реалізації інтеграційної вебплатформи для особистого тайм-менеджменту однією з найважливіших задач є забезпечення безпечного, керованого та стандартизованого доступу до зовнішніх API, зокрема таких, як Google Calendar та Trello. Враховуючи чутливість оброблюваної інформації (особисті події, списки завдань, статуси виконання), обрана схема авторизації повинна відповідати сучасним вимогам до конфіденційності, автентичності та контрольованості сесій доступу наступним чином.

1. Використання OAuth 2.0 як базового протоколу авторизації. Для забезпечення доступу до ресурсів зовнішніх сервісів застосовується OAuth 2.0 - галузевий стандарт, підтримуваний Google, Trello, Microsoft та іншими провайдерами. У контексті даного застосунку використовується потік авторизації Authorization Code Flow, який передбачає:

- запит дозволу від користувача на доступ до певних даних (наприклад, calendar.events.read або boards.read);

- отримання коду авторизації через редирект після згоди користувача;

- обмін коду на токен доступу (access token) та опціонально - токен оновлення (refresh token).

2. Рівні доступу та обмеження. Кожен модуль отримує доступ лише до конкретно дозволених ресурсів, відповідно до принципу найменших привілеїв (Least Privilege). Це означає:

- тільки читання або створення подій у Google Calendar;

- лише доступ до тих дошок Trello, на які користувач надав дозвіл;

- неможливість виконання несанкціонованих операцій або доступу до інших облікових записів.

3. Токенізація і управління сесіями. Всі сесії доступу реалізуються через токени, що є короткочасними за замовчуванням (наприклад, access token Google має термін дії 3600 секунд). Для забезпечення стабільності refresh tokens зберігаються у зашифрованому вигляді (наприклад, з використанням AES-256). При цьому, Token Manager системи відповідає за:

- перевірку дійсності токенів;

- автоматичне оновлення access token при кожному запиті;

- відкликання токенів при виході користувача або зміні дозволів;

- логування операцій авторизації.

4. Безпека зберігання токенів. З огляду на важливість токенів як ключів до зовнішніх сервісів, застосовується низка додаткових заходів:

- зберігання в ізольованому сховищі (наприклад, encrypted DB або секрети у Docker Vault);

- сервісна ізоляція доступу - лише відповідний модуль має право читати свої токени;

- реалізація тайм-аутів сесій та автоматичне видалення неактивних токенів.

5. Уніфікація доступу через єдину авторизаційну панель. Для користувача взаємодія з авторизацією максимально спрощена - доступ до всіх сервісів (Google, Trello тощо) здійснюється через єдину панель дозволів у вебінтерфейсі застосунку. Після підтвердження, користувач більше не взаємодіє з деталями авторизаційного процесу - всі обміни токенами відбуваються у фоновому режимі, з дотриманням вимог безпеки.

Далі розглянемо алгоритм синхронізації даних між сервісами. Синхронізація між різними зовнішніми сервісами - це центральна функція інтеграційної платформи, яка забезпечує узгодженість даних, безперервну актуалізацію та мінімізацію ручної взаємодії з боку користувача. Запропонований алгоритм реалізує гібридний підхід, поєднуючи періодичне опитування (polling) із реактивним оновленням на основі подій (webhook-driven), який забезпечується наступними чинниками.

1. Вхідні та вихідні параметри. Зовнішні події (Events) виконується наступним чином:

- події Google Calendar (start, end, summary, status) або зміни карток Trello (card name, due date, checklist, status);

- список локальних завдань, де існує внутрішній репозиторій задач користувача, уніфікований для всіх сервісів;

- конфлікти, де існують дублювання, суперечливі оновлення, зміщення дедлайнів;

- журнал синхронізації, що призначений для виявлення змін та ідентифікації джерел.

2. Основні етапи алгоритму до яких відносяться наступні кроки.

Крок 1: Ініціалізація сесії. Перевірка доступності токенів. Оновлення access token при потребі. Завантаження списку зовнішніх подій з Google Calendar та Trello (через REST API).

Крок 2: Аналіз змін. Виконання порівняння зовнішніх даних з локальними за допомогою механізму хешування (hash(task)). Якщо виявлено нову або змінену подію - позначити як delta.

Крок 3: Уніфікація подій. Перетворення зовнішніх подій у внутрішній формат UnifiedTask{title, start, end, source, lastModified, syncID}. Визначення пріоритетності джерела при конфлікті (напр., Google > Trello > Local).

Крок 4: Обробка конфліктів. Виконуємо визначення: чи було завдання змінено вручну користувачем? Якщо так - зберегти локальну версію; Якщо зміни йдуть з двох джерел - застосовується політика «останній змінив - головний»; У разі невирішуваного конфлікту - показати користувачеві повідомлення про вибір.

Крок 5: Оновлення даних. Синхронне оновлення локального репозиторію. Надсилання зворотних оновлень у Google або Trello, якщо потрібно (наприклад, змінено назву задачі в інтерфейсі застосунку - оновлюється картка Trello).

Крок 6: Журналювання. Логування дій синхронізації з часом, джерелом, типом дії (create, update, delete). Дані зберігаються для аудиту та відновлення.

3. Обробка подій у реальному часі. Для сервісів, які підтримують webhooks (Google Calendar, Trello), платформа реєструє спеціальні URL, куди надсилаються події типу:

- event.created, event.updated (Google);

- card.created, card.moved, due.changed (Trello).

При надходженні такої події активується швидка фонове оновлення без очікування запланованого опитування.

Показані кроки описують UML-діаграму послідовності (рис. 2.4).

Наступним етапом у реалізації методу інтеграції зовнішніх API є створення єдиного вебінтерфейсу. Так, розробка єдиного вебінтерфейсу є важливим компонентом запропонованої системи інтеграції, оскільки саме він забезпечує інтуїтивну взаємодію користувача з багатьма зовнішніми джерелами даних у межах одного цілісного середовища. Його головна мета - забезпечити уніфіковане, зручне та ефективне керування усіма задачами й подіями, незалежно від того, з якого сервісу вони надійшли - Google Calendar, Trello чи інших. До основних складових

у реалізації методу інтеграції зовнішніх API через єдиний вебінтерфейс є наступні чинникі.

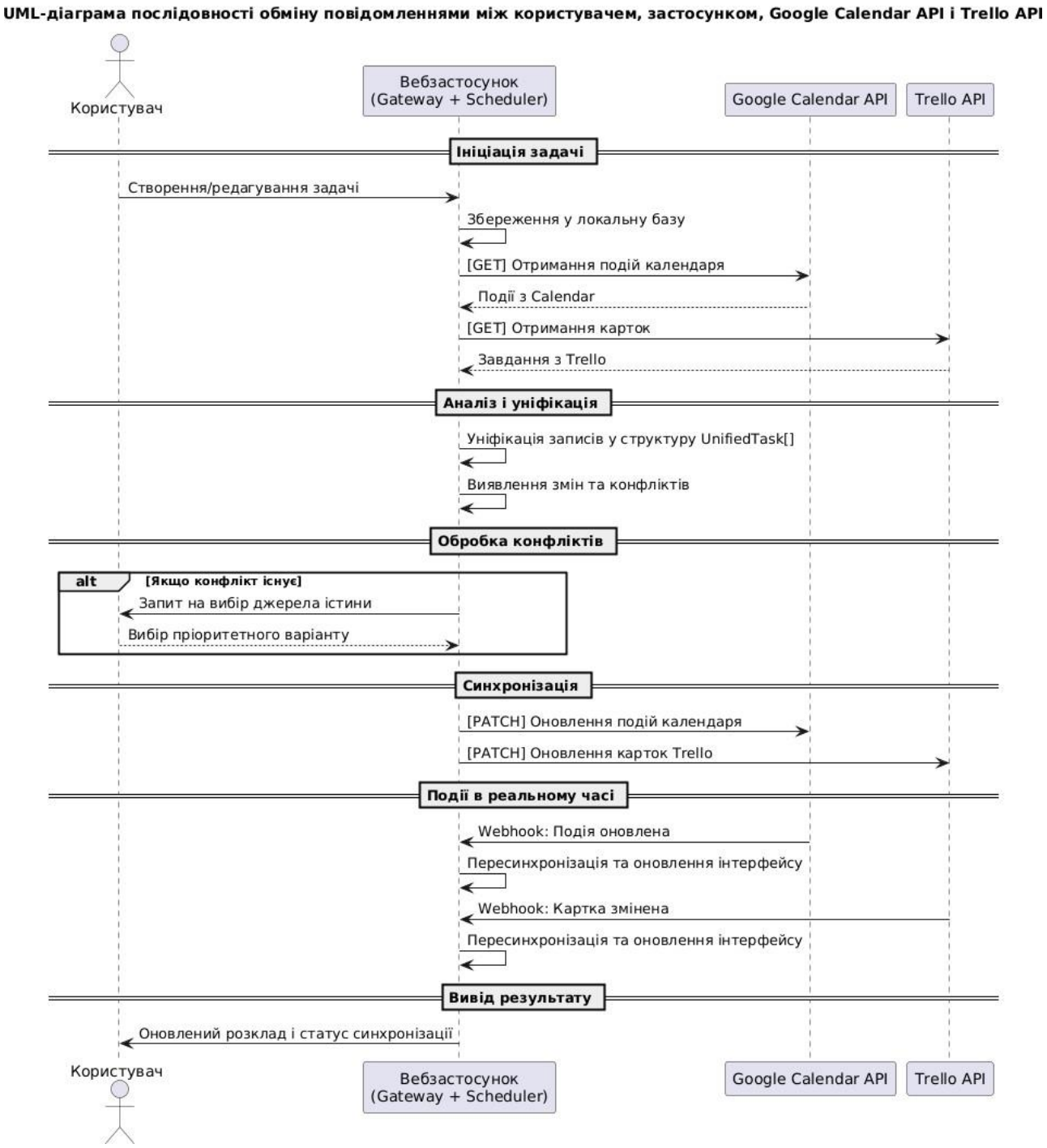

Рисунок 2.4 - UML-діаграму послідовності візуалізації обміну повідомленнями між користувачем, застосунком, Google Calendar API і Trello API

1. Принципи побудови інтерфейсу, де є уніфікація даних. Так, незалежно від джерела (подія календаря чи Trello-картка), усі задачі трансформуються у внутрішню структуру UnifiedTask, що містить:

- заголовок;
- опис;
- джерело походження (Google, Trello);
- тип елемента (подія, завдання);
- статус (активне, завершене, перенесене);
- дедлайн або часовий інтервал.

1.2. Незалежність від сервісу. Так, користувач не зобов’язаний знати, в якому сервісі була створена задача - всі об’єкти мають єдине подання, а система відслідковує джерело автоматично. Для редагування, переміщення, відмітки про виконання не потрібно перемикатися між зовнішніми системами.

1.3. Одноточковий доступ до налаштувань і авторизацій. Всі дозволи на доступ до сервісів (Google, Trello) та параметри синхронізації керуються через єдиний розділ "Інтеграції". При цьому, є підтримка відключення синхронізації, перегляду статусу з’єднання та повторного проходження авторизації.

2. Функціональні можливості вебінтерфейсу (табл. 2.3), де існує певний опис створення функціональних можливостей для визначених функцій.

Таблиця 2.3 - Опис створення функціональних можливостей вебінтерфейсу для визначених функцій

Функція	Опис
Створення задачі	Можна створити універсальну задачу, яка буде направлена у вибраний сервіс
Фільтрація та пошук	Завдання можна фільтрувати за джерелом, пріоритетом, статусом або часом
Перетягування (drag-and-drop)	Впроваджено можливість зміни порядку або дати виконання через календарний перегляд
Перегляд календаря	Візуалізація завдань і подій на тиждень/місяць (інтеграція із scheduler module)

Відображення статусу синхронізації	Кожна задача містить індикатор стану (синхронізовано/очікує/конфлікт)
Масова обробка	Можна одночасно редагувати або переміщувати кілька задач

3. Технологічна реалізація вебінтерфейсу, що має такі основні складові:

- фронтенд, що реалізується з використанням React.js або Vue.js, з підтримкою адаптивної верстки (mobile-first);

- API-зв’язок, що взаємодіє з бекендом через REST або GraphQL;

- візуальні бібліотеки для використання компонентів на базі Material UI або Tailwind CSS для забезпечення доступності та простоти;

- WebSocket або SSE для реалізації реального часу (оновлення статусу задач без перезавантаження).

4. Особливості інтерфейсу при обробці конфліктів. У разі виникнення суперечливих змін між сервісами (наприклад, одна й та сама задача змінена і в Google Calendar, і в Trello), система:

- виводить діалог з пропозицією варіантів;

- за замовчуванням застосовує політику останньої модифікації, але користувач може змінити її у налаштуваннях.

Таким чином, запропонована централізована модель інтеграції зовнішніх API дає змогу значно підвищити ефективність персонального управління часом шляхом об'єднання раніше роз'єднаних цифрових середовищ. Модульність, уніфікація, реактивна синхронізація та захищений механізм авторизації створюють передумови для стабільної, безпечної та зручної платформи, що адаптується до змінних потреб користувача та розвитку цифрових сервісів.

3 РОЗРОБКА МОДУЛІВ ПРОГРАМНОГО ЗАСОБУ

3.1 Програмна реалізація модуля динамічного розподілу завдань із інтеграцією зовнішніх календарів

У межах запропонованої інформаційної технології оптимізації особистого тайм-менеджменту реалізовано метод динамічного розподілу завдань, який ґрунтується на адаптивному перерахунку розкладу з урахуванням пріоритетності, дедлайнів, доступності часу користувача та подій, зафіксованих у зовнішніх календарях (зокрема Google Calendar API).

Архітектурно система побудована як мультикомпонентна мікросервісна платформа, яка охоплює три основні рівні:

- рівень інтеграції з користувачем (мобільний застосунок, веб-інтерфейс);
- рівень обробки логіки (Spring Boot REST API);
- рівень зовнішньої інтеграції (Google Calendar API, Trello, інші сервіси за OAuth2).

Також, реалізована система дозволяє:

- автоматично оновлювати список завдань користувача залежно від змін у календарі;
- адаптувати розподіл часу в межах доби з урахуванням доступності;
- визначати пріоритети завдань і виконувати їх перекомпонування у розкладі;
- зменшити час на ручне планування.

Ключовою відмінністю запропонованого підходу є реактивна модель обробки змін, яка дозволяє адаптувати розклад у реальному часі. На рівні прикладного програмного інтерфейсу (API) усі модулі підтримують асинхронну взаємодію через REST або WebSocket протоколи.

Опис модулів системи динамічного розподілу завдань полягає в наступному. Для реалізації функціональної логіки системи розроблено низку окремих модулів, які взаємодіють через REST API та керуються за допомогою центрального сервісу диспетчеризації (табл. 3.1).

Таблиця 3.1 - Функціональне призначення та особливості реалізації модулів на центральному сервісу диспетчеризації

Назва модуля	Функціональне призначення	Інтеграція	Особливості реалізації
TaskScheduler-Module	Динамічне формування розкладу задач з урахуванням часу, дедлайнів і пріоритетів	REST API	Алгоритм на основі rule-based логіки
CalendarIntegra-tionModule	Синхронізація з Google Calendar (читання/запис подій)		OAuth2 авторизація, JSON REST
PriorityAdjust-mentModule	Перерахунок пріоритетів при змінах у задачах або календарях	Внутрішній REST endpoint	Rule Engine (Drools) або вручну
ConflictResolver Module	Виявлення та автоматичне вирішення конфліктів розкладу	TaskScheduler + Calendar API	Алгоритм оптимізації вільних слотів
Notification-Module	Сповіщення користувача про зміни, конфлікти, нові завдання	Firebase, Email API	WebSocket для push-повідомлень
UserInterface-Module	Відображення розкладу та управління завданнями через Web UI або мобільний застосунок	Web (React), Mobile (Flutter)	Синхронізація з сервером через REST
AnalyticsModule	Аналіз ефективності виконання задач і тайм-менеджменту	MySQL, Data Visualizer	Побудова графіків та таблиць

На основі цієї таблиці буде сформовано UML-діаграму компонентів (рис. 3.1), яка відображає зв’язки між цими модулями, взаємодію з інтерфейсами користувача та зовнішніми сервісами. Надана UML-діаграма компонентів (рис. 3.1) візуалізує архітектуру системи динамічного розподілу завдань з інтеграцією

зовнішніх календарів. Діаграма показує компоненти бекенду (Spring Boot), фронтенду (Web/Mobile UI), зовнішні сервіси та взаємозв'язки між ними.

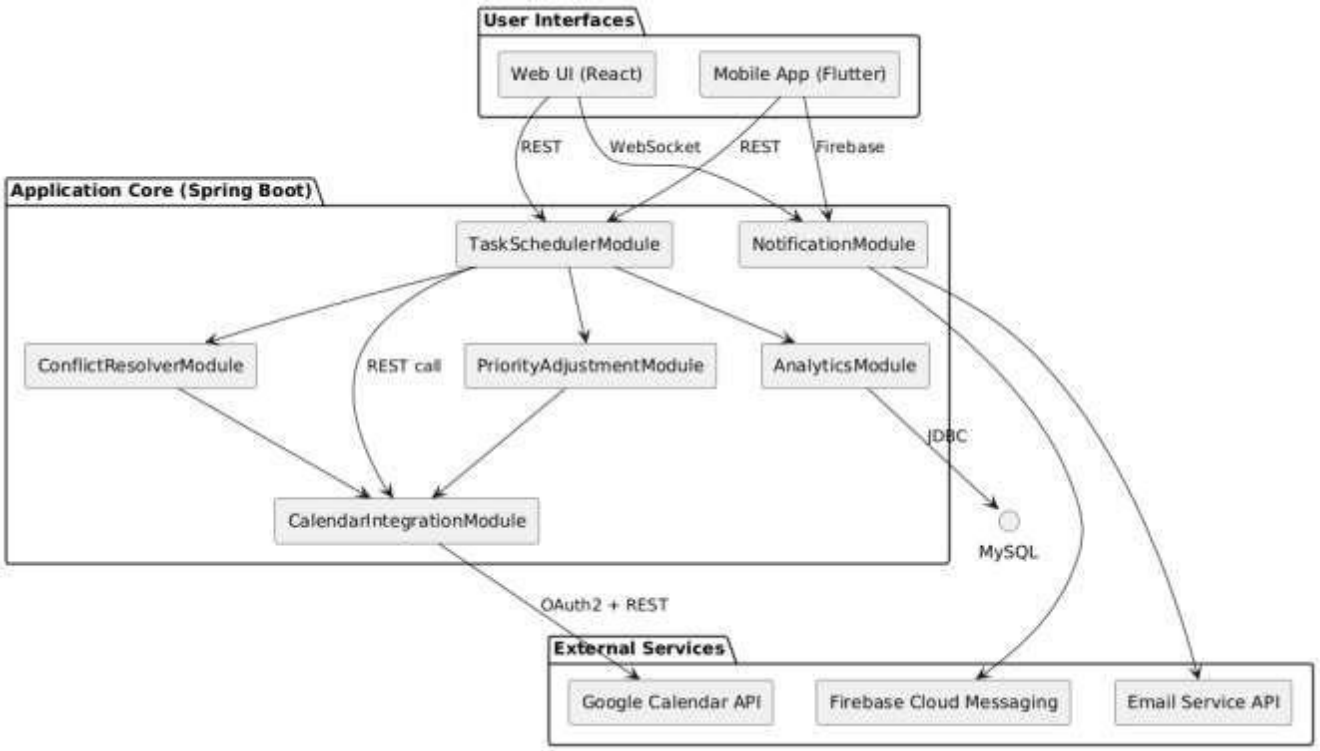

Рисунок 3.1 - UML-діаграма компонентів архітектури системи динамічного розподілу завдань з інтеграцією зовнішніх календарів

У розробленому програмному застосунку для оптимізації особистого тайм-менеджменту архітектура є ключовим елементом, що визначає ефективність, масштабованість і гнучкість системи. Побудована UML-діаграма компонентів дозволяє наочно представити логічну структуру програмного засобу, включаючи функціональні блоки, їхні інтерфейси та взаємозв'язки між собою, а також зв'язки із зовнішніми сервісами.

Архітектура системи реалізована у вигляді тришарової мікросервісної моделі, де кожен компонент відповідає за конкретну функціональність і взаємодіє з іншими модулями через чітко визначені інтерфейси. Такий підхід сприяє високій модульності системи, полегшує її тестування, розгортання та подальшу підтримку.

Перший функціональний блок - інтерфейс користувача, який реалізований у двох формах: веб-додаток (Web UI) на основі React та мобільний застосунок

(Mobile App), розроблений з використанням Flutter. Web UI забезпечує доступ користувача до повної функціональності через браузер. Здійснюється передача HTTP-запитів до API-сервісів (зокрема, до TaskSchedulerModule, NotificationModule) з можливістю отримання динамічного розкладу, перегляду задач, внесення змін. Mobile App виконує аналогічні функції, однак оптимізований для мобільних пристроїв, забезпечує інтерактивні push-сповіщення, а також доступ у фоновому режимі до календарних подій.

Обидва інтерфейси використовують REST API для ініціації запитів до бекенд-сервісів та WebSocket або Firebase Cloud Messaging для отримання сповіщень у реальному часі. Завдяки цьому досягається мінімальна затримка при синхронізації між розкладом і поточними діями користувача.

Основою в реалізації системи динамічного розподілу завдань із інтеграцією зовнішніх календарів є бекенд, реалізований за допомогою фреймворку Spring Boot, який забезпечує управління логікою, розподілом задач, обробкою пріоритетів, взаємодією з календарями, вирішенням конфліктів та генерацією сповіщень. Компоненти системи згруповано у наступні модулі.

Компонент TaskSchedulerModule відповідає за формування розкладу задач, враховуючи дані про тривалість, дедлайни, пріоритети, зайнятість користувача (у тому числі з календаря). Він є центральним елементом, через який інші модулі (зокрема ConflictResolverModule, PriorityAdjustmentModule) ініціюють розрахунок нових тайм-слотів. Сервіс використовує допоміжний утилітарний клас TimeSlotOptimizer, що реалізує алгоритми вставлення задач у вільні проміжки часу з урахуванням контексту користувача.

Компонент CalendarIntegrationModule забезпечує двосторонню синхронізацію з Google Calendar за допомогою офіційного REST API з авторизацією через OAuth2. Цей компонент дозволяє:

- зчитувати наявні події користувача;
- створювати нові записи про задачі;
- оновлювати або видаляти записи при зміні розкладу.

Використання зовнішнього API дає змогу уніфікувати взаємодію з іншими календарними сервісами (наприклад, Microsoft Outlook, Apple Calendar) у майбутньому.

Компонент PriorityAdjustmentModule, який автоматично переобчислює пріоритети задач на основі зміни зовнішніх умов (наприклад, зміни дати дедлайну, зміни доступності або вагомості задачі). Він функціонує у зв’язці з планувальником і здатен адаптивно змінювати черговість виконання. У перспективі може бути розширений модулем штучного інтелекту для оцінки ефективності користувача та пріоритезації на основі поведінкових патернів.

Компонент ConflictResolverModule виявляє конфлікти між подіями календаря та задачами та вирішує їх шляхом оптимізації слотів. При виникненні конфлікту він викликає TaskSchedulerModule з модифікованими параметрами і ініціює повторний розрахунок.

Компонент NotificationModule відповідає за надсилання сповіщень користувачеві, використовуючи кілька каналів:

- WebSocket для веб-додатку;
- Firebase Cloud Messaging для мобільного застосунку;
- Email API для розсилки листів.

Компонент AnalyticsModule формує статистику та аналітичні дані про ефективність використання часу. Збирає інформацію про:

- відсоток завершених задач у заплановані строки;
- середній час зволікання;
- кількість конфліктів;
- частоту зміни розкладу.

Результати виводяться у вигляді графіків і таблиць у Web UI.

UML-діаграма компонентів також демонструє інтеграційні зв’язки із зовнішніми платформами, які забезпечують додаткову функціональність:

- Google Calendar API, що забезпечує двосторонній обмін подіями календаря. Доступ надається за авторизацією через OAuth2, а самі запити виконуються через HTTPS із використанням REST-інтерфейсу;

- Firebase Cloud Messaging, що використовується для реалізації push-сповіщень на мобільні пристрої;

- Email Service API, що служить для надсилання електронних листів з інформацією про оновлення у розкладі, важливі події або помилки.

Інтеграція з цими сервісами дозволяє забезпечити повний цикл обробки даних та комунікації з користувачем.

Таким чином, обрана архітектура дає змогу ефективно розділити відповідальність між модулями, забезпечити паралельну обробку запитів, реактивну адаптацію до змін, а також гнучкість масштабування. Кожен компонент побудовано за принципами SOLID та Clean Architecture, що підвищує якість коду, спрощує модифікації та дозволяє уніфікувати підходи до розробки.

UML-діаграма компонентів (рис 3.1) виявилась ефективним засобом візуалізації логіки функціонування системи, що є необхідним етапом у побудові складних розподілених застосунків. Її використання сприяло кращому розумінню як внутрішніх взаємозв’язків між модулями, так і способів їхньої інтеграції з зовнішнім середовищем, що надзвичайно важливо в умовах сучасної гібридної цифрової інфраструктури.

Далі опишемо ієрархічну структуру Java-класів (рис. 3.2), які реалізують функціональність основних модулів програмного застосунку. Всі класи логічно згруповано за призначенням, використовуючи об’єктно-орієнтовану модель з інтерфейсами, абстрактними базовими класами та реалізаціями сервісів відповідно до архітектури Spring Boot.

До ключових особливостей ієрархічної структури та реалізації Java-класів системи відноситься наступні:

- сервіси реалізовано за принципом interface + impl, що спрощує модульність і тестування;

- DTO-класи ізольовані від моделей БД (Entity), що відповідає принципам Clean Architecture;

- WebSocket реалізується у NotificationController, що передає повідомлення в реальному часі;

- OAuth2 авторизація налаштовується через SecurityConfig.java для Google Calendar API;

- утилітарний клас TimeSlotOptimizer реалізує логіку вставлення задач у доступні часові інтервали.

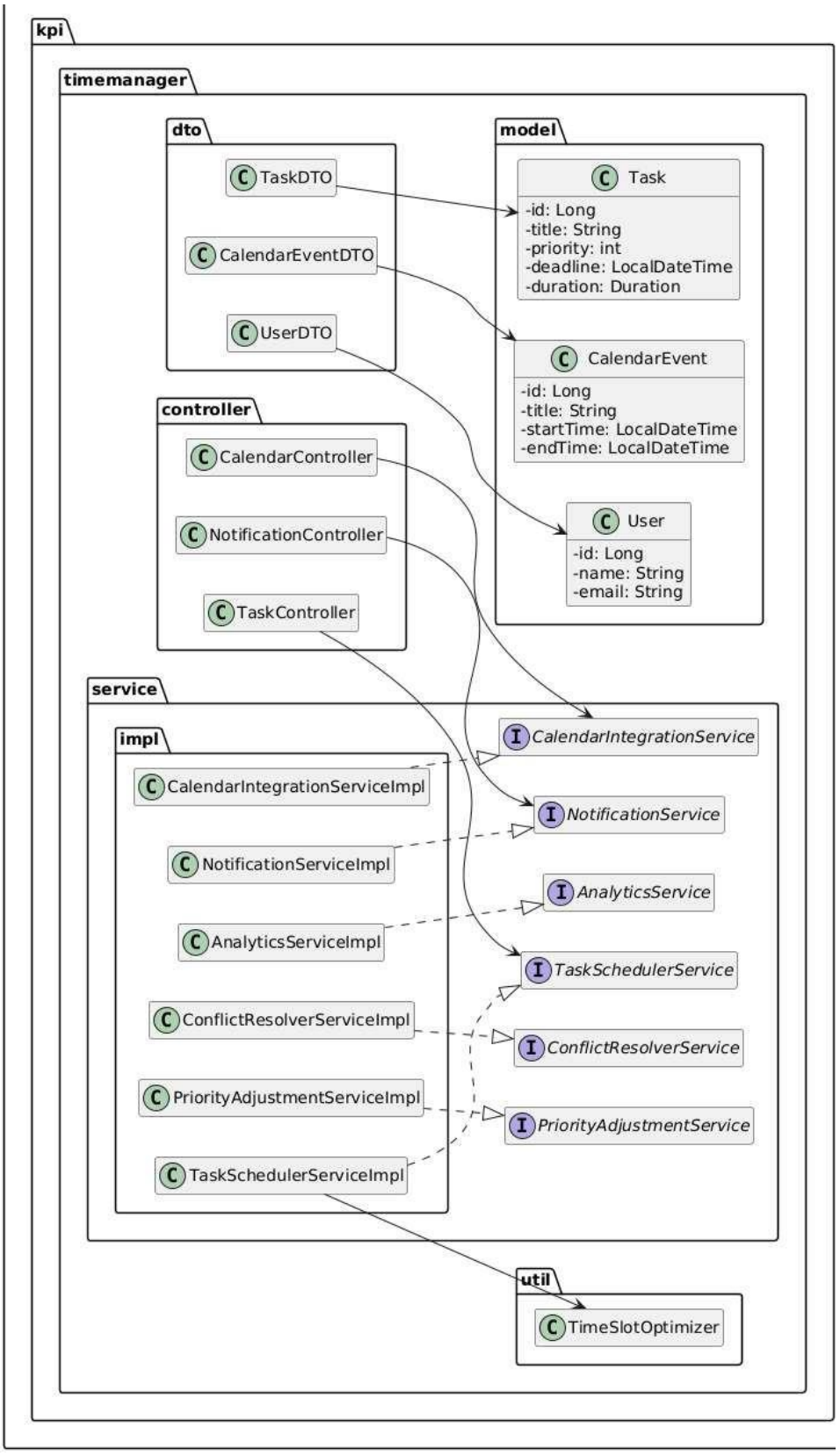

Рисунок 3.2 - UML-діаграма класів архітектури системи динамічного розподілу завдань з інтеграцією зовнішніх календарів

Показана UML-діаграма класів (рис. 3.2) описує ієрархічну структуру Java-класів наступним чином. Класи, розміщені в пакеті ua.kpi.timemanager.model, є сутностями, що відображаються у реляційній базі даних. Кожна сутність є відображенням таблиці й анотована за допомогою специфікацій JPA. Тому, розглянемо опис основних класів.

Клас Task. Описує основну одиницю роботи користувача - задачу. Має такі атрибути:

id - унікальний ідентифікатор;

title - короткий опис;

priority - числовий рівень важливості;

deadline - граничний термін виконання;

duration - очікувана тривалість.

Цей клас відіграє ключову роль у взаємодії з планувальником (TaskSchedulerServiceImpl) та аналітичним модулем (AnalyticsServiceImpl).

Наступний клас CalendarEvent відповідає за події з інтегрованого календаря та має такі атрибути:

startTime, endTime - часові межі події;

title - назва;

id - унікальний ідентифікатор.

Використовується CalendarIntegrationServiceImpl для синхронізації з Google Calendar.

Клас User містить базові облікові атрибути користувача такі як:

name, email;

id - унікальний первинний ключ.

В подальшому цей клас може бути розширений для ролей, статусів, історії активності.

Передача даних на рівні DTO (Data Transfer Objects) здійснюється наступним чином. DTO-класи розміщені окремо у пакеті ua.kpi.timemanager.dto та використовуються для передачі структурованих даних між frontend та backend

рівнями без прямого посилання на сутності бази даних. Вони забезпечують ізоляцію та безпеку передачі даних за допомогою наступних атрибутів:

TaskDTO - містить мінімальний набір полів задачі, що необхідний для відображення у UI;

CalendarEventDTO - використовується при імпорті/експорті подій до/з Google Calendar;

UserDTO - спрощена модель користувача, яка передається на фронтенд для відображення ідентифікаційних даних.

Рівень service і service.impl підтримують сервісний рівень у вигляді інтерфейсів втілюють головну логіку функціонування системи. Інтерфейси описують контракт сервісів, а реалізації (Impl) - їх поведінку. Такий підхід відповідає принципу інверсії залежностей (Dependency Inversion Principle) за допомогою наступних класів:

- TaskSchedulerService / TaskSchedulerServiceImpl, що визначає методи для створення, оновлення, планування задач та реалізує логіку черговості виконання, збереження вільного часу користувача, адаптації до зовнішніх змін;

- PriorityAdjustmentService / Impl, що виконує перекомпонування пріоритетів задач;

- ConflictResolverService / Impl, що виявляє та усуває конфлікти між подіями календаря і задачами та взаємодіє з CalendarIntegrationService;

- NotificationService / Impl, що надсилає push- та email-сповіщення та інкапсулює логіку роботи з Firebase, SMTP або іншими каналами зв’язку;

- CalendarIntegrationService / Impl, що обробляє зв’язки з Google Calendar API та синхронізує події за допомогою OAuth2;

- AnalyticsService / Impl, що генерує метрики щодо продуктивності користувача та працює з базою MySQL та інтерфейсами виводу результатів.

Також, важливу роль у функціонуванні класів відіграють контролери, які розміщено у пакеті controller утиліти. Вони реалізують REST API та доступні для зовнішніх клієнтів. Кожен контролер прив’язаний до відповідного сервісу наступним чином:

TaskController - обробка CRUD-операцій над задачами, формування розкладу;

CalendarController - управління подіями календаря;

NotificationController - реалізує WebSocket-інтерфейс або інтеграцію з Firebase.

Контролери використовують анотації @RestController, @RequestMapping, @PostMapping тощо.

В свою чергу, утиліта TimeSlotOptimizer являє собою утилітарний клас, що реалізує алгоритм вставлення задач у часові вікна, враховує зайнятість користувача, тривалість задач, важливість, буферні зони. Утиліта TimeSlotOptimizer може бути використаний декількома модулями: TaskScheduler, ConflictResolver.

Таким чином, описана UML-діаграма класів (рис. 3.2) дозволяє чітко візуалізувати ієрархічну структуру програмного забезпечення та взаємозв'язки між його ключовими складовими. Завдяки впровадженню чітко визначених ролей для кожного рівня (моделі, DTO, сервіс, контролер), структура є легкою для підтримки, тестування та масштабування. Показані класи організовано відповідно до принципів об'єктно-орієнтованого проєктування, з дотриманням SOLID-принципів, що забезпечує високий рівень реюзабельності та ізольованості логіки. Це дозволяє у подальшому розширювати функціонал системи, не порушуючи її базової архітектури.

Показана UML-діаграма класів (рис. 3.2) візуалізує ієрархічну структуру Java-класів, що фокусується на основних сервісах, моделях, контролерах та DTO (табл. 3.2), що абстрагують допоміжні класи для наочності. UML-діаграма також сприяє більш глибокому розумінню логіки системи та може використовуватись як вихідна точка для генерації документації та подальшої автоматизованої перевірки взаємозв'язків. Також, ця UML-діаграма демонструє чіткий розподіл відповідальностей між компонентами та їхню взаємодію в контексті архітектури Spring Boot. Інтерфейси і реалізації сервісів пов'язані через залежності, моделі відокремлені від DTO, а контролери координують виклики сервісів. Для цього,

наведемо порівняльну таблицю відповідності між модулями, класами та типами взаємодії (табл. 3.2).

Таблиця 3.2 - Порівняльна характеристика відповідності модулів, класів і типів обробки

Назва модуля	Відповідні Java-класи	Зовнішні сервіси	Тип обробки
TaskScheduler-Module	TaskSchedulerServiceImpl, TaskController, Task	-	Реактивна, асинхронна
CalendarIntegra-tionModule	CalendarIntegrationServiceImpl, CalendarController	Google Calendar API	Синхронна, OAuth2 REST
PriorityAdjust-mentModule	PriorityAdjustmentServiceImpl	-	Реактивна, rule-based
ConflictResol-verModule	ConflictResolverServiceImpl, TimeSlotOptimizer	-	Адаптивна, внутрішня
Notification-Module	NotificationServiceImpl, Notification-Controller	Firebase, Email API	Push, WebSocket, асинхронна
AnalyticsModule	AnalyticsServiceImpl	MySQL	Пакетна обробка, запланована
UserInterface-Module	Web UI (React), Mobile App (Flutter)	REST API	Взаємодійна, інтерактивна

Таким чином, в цьому підрозділі було реалізовано програмну архітектуру системи оптимізації особистого тайм-менеджменту, що поєднує можливості динамічного планування задач із засобами інтеграції з зовнішніми календарними сервісами. Запропонована архітектура побудована на основі мікросервісної моделі, що забезпечує масштабованість, модульність та незалежне розгортання компонентів.

Основним досягненням в роботі є формалізація та реалізація методу динамічного розподілу завдань, який дозволяє адаптувати розклад користувача в реальному часі завдяки реактивній обробці змін у вхідних даних, таких як пріоритети, дедлайни та зовнішні події з Google Calendar. Реалізація на основі фреймворку Spring Boot, з використанням REST API, OAuth2, WebSocket та Firebase, забезпечила високий рівень взаємодії між фронтенд-застосунками та бекенд-логікою. Так, побудовані UML-діаграми компонентів і класів дозволили чітко ілюструвати внутрішню структуру системи та взаємозв'язки між її модулями. Ієрархічна структура Java-класів відображає принципи об'єктно-орієнтованого програмування з чітким розділенням обов'язків між сервісами, контролерами та моделями. До переваг запропонованої програмної реалізації відносяться:

- адаптивність до змін у режимі реального часу;
- зниження ручного навантаження на планування;
- розширюваність системи для підключення додаткових сервісів (наприклад, Trello, Notion);
- можливість подальшого впровадження алгоритмів машинного навчання для прогнозування ефективності виконання завдань.

Таким чином, реалізований програмний модуль виконує ключову роль у досягненні мети магістерської роботи - розробці інтелектуального інструменту для оптимізації персонального тайм-менеджменту.

3.2 Програмна реалізація методу інтеграції зовнішніх API через єдиний вебінтерфейс

В умовах сучасного цифрового середовища користувачі взаємодіють одночасно з багатьма сервісами, які не мають спільного центру керування: календарні платформи (Google Calendar, Microsoft Outlook), менеджери завдань (Trello, Notion), сервіси нагадувань (Slack, Todoist) тощо. Відсутність централізованої взаємодії між цими системами призводить до фрагментації даних,

дублювання задач, втрати контролю над термінами й погіршення ефективності особистого планування. В рамках даного дослідження запропоновано реалізувати метод єдиного вебінтерфейсу інтеграції зовнішніх API (табл. 3.4), який дозволяє:

- централізовано отримувати, обробляти та візуалізувати дані з різних джерел (Google Calendar, Trello);
- безпечно синхронізувати завдання, події та нагадування;
- забезпечити токенізований доступ до сторонніх сервісів через OAuth 2.0;
- уніфікувати формат обміну даними через адаптивні DTO-структури.

Таблиця 3.3 - Призначення модулів підсистеми інтеграції зовнішніх API через єдиний вебінтерфейс

Назва модуля	Призначення	API, що використовується
GoogleCalendarIntegration Module	Імпорт/експорт подій з Google Calendar	https://www.googleapis.com/calendar/v3
TrelloTaskIntegration-Module	Синхронізація карточокTrello з локальними задачами	https://api.trello.com/1
UnifiedWebInterface-Module	Відображення даних з різних сервісів у єдиному UI	Локальний API
OAuthTokenManager-Module	Отримання, збереження та оновлення токенів доступу	OAuth 2.0 Endpoints
SyncSchedulerModule	Планування періодичної синхронізації між сервісами	Cron-таймери

Архітектурно запропонований підхід реалізується як модульна вебсервісна система, побудована за принципом сервісно-орієнтованої архітектури (SOA), де кожен компонент відповідає за конкретну логіку: інтеграцію, збереження токенів, нормалізацію даних, синхронізацію та представлення в UI.

Особливістю реалізації методу інтеграції зовнішніх API через єдиний вебінтерфейс (табл. 3.4) є підтримка:

- реальних ендпоінтів API, зокрема Google Calendar REST API v3 та Trello REST API 1.0;

- авторизації через OAuth 2.0 із збереженням та оновленням access_token і refresh_token;

- асинхронної взаємодії між модулями через внутрішні REST-виклики;

- динамічної розширюваності, що дозволяє інтегрувати нові API без зміни ядра системи.

Таким чином, метод єдиного вебінтерфейсу інтеграції дозволяє не лише спростити взаємодію користувача з розрізненими сервісами, але й створити платформу для подальшого розширення функціональності персонального планування.

Для реалізації методу інтеграції через єдиний вебінтерфейс було розроблено низку окремих модулів, кожен з яких виконує специфічну функцію в загальній архітектурі. Їхню взаємодію побудовано на основі принципів слабкого зв'язування (loose coupling) та інкапсуляції функціональності. В табл. 3.3 надана призначення та узагальнена характеристика розроблених модулів.

Таблиця 3.4 - Авторизація та формати даних підсистеми інтеграції зовнішніх API через єдиний вебінтерфейс

Назва модуля	Авторизація	Формат даних	Тип взаємодії
GoogleCalendar-IntegrationModule	OAuth 2.0 + Token	JSON	REST, асинхронно
TrelloTaskIntegra-tionModule	API Key + Token	JSON	REST, асинхронно
UnifiedWebInter-faceModule	-	HTML/JSON	REST
OAuthTokenMana-gerModule	SecureTokenSto-rage	JWT	REST

SyncSchedulerMo-dule	-	-	Внутрішній HTTP

Кожен із модулів реєструється в контейнері Spring як окремий компонент (@Service), взаємодіє через HTTP REST, що дозволяє легко масштабувати систему або адаптувати її для мобільного або десктопного середовища.

Далі розглянемо UML-діаграму компонентів, яка візуалізує архітектуру інтеграції зовнішніх API (Google Calendar, Trello) через єдиний вебінтерфейс (рис. 3.3). У діаграмі відображено модулі інтеграції, менеджмент авторизації, обробку синхронізації, взаємодію з користувачем та з реальними ендпоінтами.

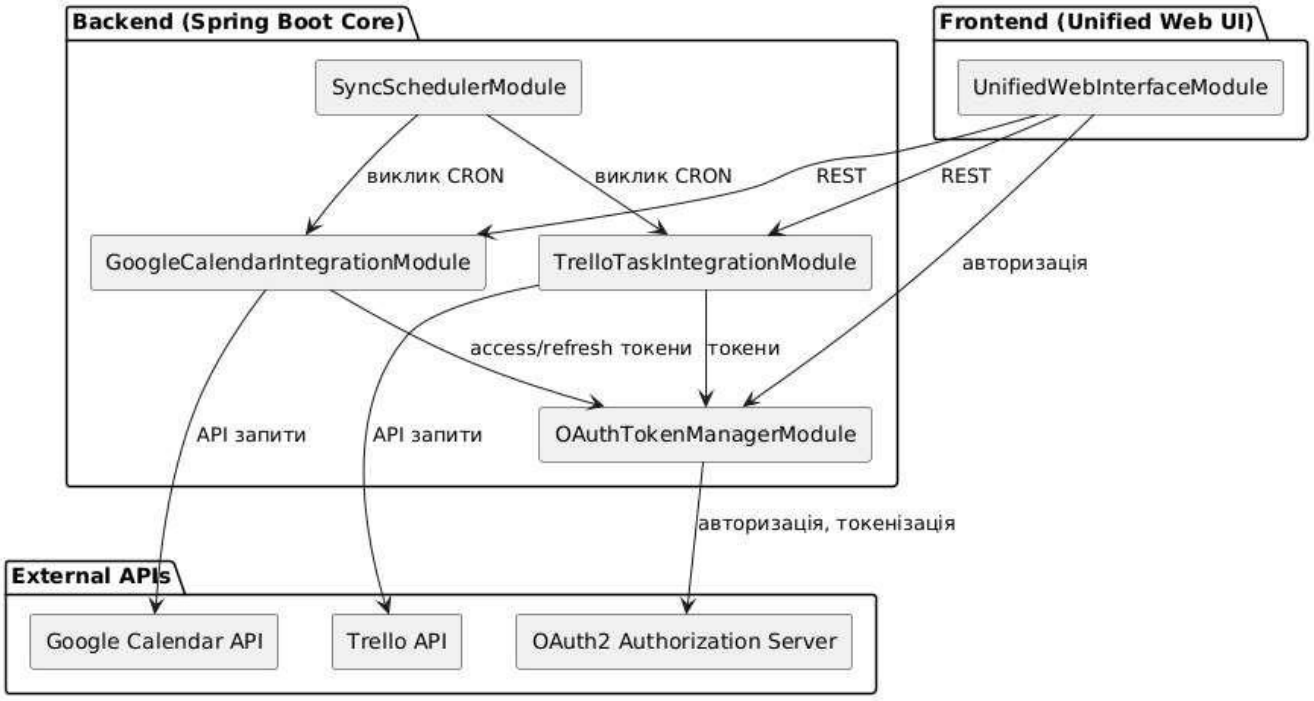

Рисунок 3.3 - UML-діаграма компонентів інтеграції зовнішніх API

На рис. 3.3 показано, як користувач взаємодіє з UnifiedWebInterfaceModule, який єдина точка входу. GoogleCalendarIntegrationModule та TrelloTaskIntegrationModule обробляють API-виклики до сторонніх сервісів. OAuthTokenManagerModule забезпечує токенізацію, зберігання та оновлення доступів. SyncSchedulerModule відповідає за періодичну фонову синхронізацію

(наприклад, кожні 5 хвилин). Вся взаємодія між модулями реалізована через HTTP REST, що забезпечує слабке зв’язування та модульність.

Далі розглянемо UML-діаграму класів, яка візуалізує архітектуру інтеграції зовнішніх API (Google Calendar, Trello) через єдиний вебінтерфейс (рис. 3.4). У діаграмі відображено модулі інтеграції, менеджмент авторизації, обробку синхронізації, взаємодію з користувачем та з реальними ендпоінтами. Вона графічно відображає ключові класи, інтерфейси, реалізації, залежності та асоціації між компонентами модуля інтеграції зовнішніх API (Google Calendar, Trello) через єдиний вебінтерфейс.

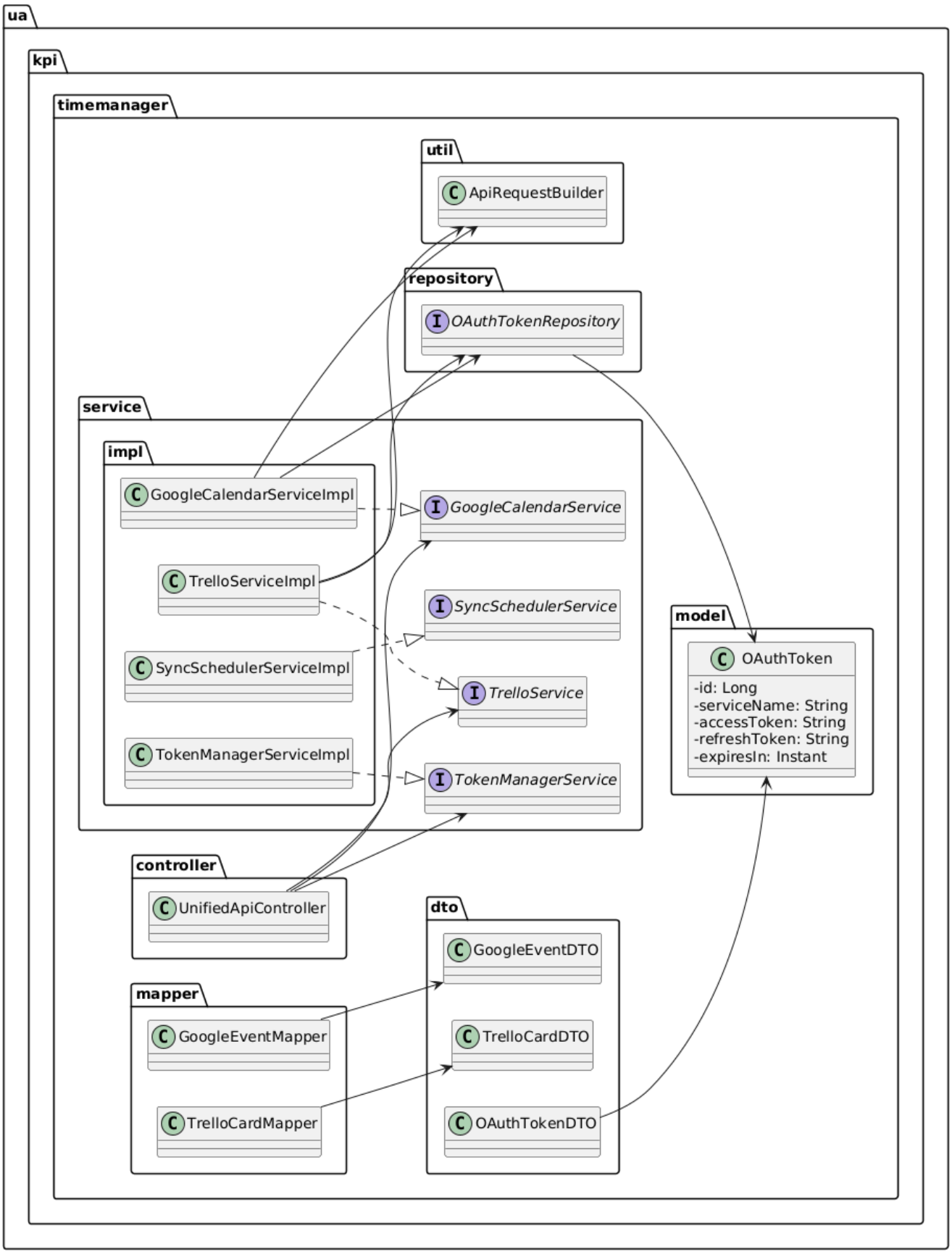

Рисунок 3.4 - UML-діаграма класів інтеграції зовнішніх API

В розробленому програмному засобі на рис. 3.4 надана ієрархія Java-класів для реалізації модулів інтеграції зовнішніх API через єдиний вебінтерфейс. Структура організована відповідно до принципів чистої архітектури (Clean Architecture) з чітким поділом на контролери, DTO, сервіси, реалізації сервісів та конфігураційні компоненти. В цій діаграмі кожен API (Google, Trello) має власний сервісний інтерфейс і реалізацію. TokenManagerService ізолює логіку отримання, збереження й оновлення токенів доступу. SyncSchedulerService викликає імпорт/експорт через CRON-заплановані події. UnifiedApiController є єдиною точкою входу для користувача у Web UI. OAuthToken зберігається в БД як окрема сутність (Entity), що прив’язана до користувача/сервісу.

Всі сервіси реалізуються через шаблон інтерфейс + реалізація. Так, UnifiedApiController напряму взаємодіє з обома сервісами API та сервісом управління токенами. OAuthTokenRepository забезпечує збереження токенів через JPA. Мапери та утиліти є допоміжними у використанні, але критично важливі для коректного форматування і трансляції об’єктів.

Таким чином, у підрозділі 3.2 було реалізовано модульну архітектуру інтеграції зовнішніх API на основі принципу єдиного вебінтерфейсу. Запропонована реалізація охоплює компоненти для роботи з Google Calendar API, Trello API, централізованого керування авторизацією через OAuth 2.0, а також автоматизованої синхронізації за допомогою фонових CRON-задач.

Проведене дослідження дозволило сформулювати такі ключові висновки:

- модульність архітектури забезпечила чітке розділення обов’язків між компонентами, що підвищує масштабованість, гнучкість у супроводі, а також можливість незалежної розробки і тестування окремих частин системи;

- інтеграція з Google Calendar реалізована на основі реального REST API v3 з підтримкою двостороннього обміну даними (імпорт/експорт подій), що дозволяє користувачеві керувати своїми подіями без переходу до стороннього інтерфейсу;

- інтеграція з Trello API демонструє можливість часткової синхронізації задач, дозволяючи автоматично імпортувати картки з дошок користувача, що підвищує гнучкість у плануванні особистих і професійних завдань;

- централізоване управління токенами через OAuthTokenManagerModule дозволяє зберігати, поновлювати та перевіряти доступи в захищеній формі, що сприяє високому рівню безпеки при роботі з приватними обліковими даними;

- використання асинхронних CRON-механізмів дозволяє реалізувати фонову синхронізацію зовнішніх даних, не навантажуючи основний інтерфейс користувача, що є важливим аспектом у створенні продуктивних систем реального часу.

На основі порівняльної таблиці підтверджено, що кожен модуль відповідає своїй функціональній зоні, зберігаючи єдину логіку взаємодії через REST-інтерфейси та стандартні формати обміну даними (JSON). Таким чином, запропонований підхід дозволяє реалізувати ефективну та безпечну інтеграцію з популярними зовнішніми сервісами планування та управління завданнями.

Перспективи подальшого вдосконалення інтеграційного модуля включають:

- розширення переліку підтримуваних API (Slack, Notion, Outlook);

- застосування Webhook-механізмів для реактивної синхронізації;

- використання AI-модулів для автоматичної категоризації та пріоритезації задач.

Отримані результати можуть бути використані як основа для створення персоналізованих платформ планування, адаптованих до стилю життя та потреб конкретного користувача.

4 ТЕСТУВАННЯ МОДУЛІВ ПРОГРАМНОГО ДОДАТКУ

4.1 Особливості тестування модуля динамічного розподілу завдань із інтеграцією зовнішніх календарів

Основною метою тестування модулів розподілу завдань із інтеграцією зовнішніх календарів є перевірка функціональної коректності, надійності та зручності використання реалізованого програмного забезпечення (табл. 4.1). У межах контрольного прикладу здійснюється імітація основних сценаріїв взаємодії користувача з системою: створення задач, перегляд розкладу з урахуванням подій календаря, а також надсилання сповіщень (табл. 4.2).

Таблиця 4.1 - Опис дії результатів виконання контрольного прикладу

Назва етапу	Опис дії
Створення задачі	Заповнення форми задачі та її додавання через кнопку «Додати»
Інтеграція із зовнішнім календарем	Натискання кнопки «Інтеграція із Календарем»
Надсилання сповіщення	Натискання кнопки для надсилання сповіщення про задачу

Завданням даного етапу є:

- перевірка коректної взаємодії frontend і backend компонентів;
- забезпечення надійного збереження даних у СУБД;
- тестування API для взаємодії із зовнішнім календарем;
- оцінка реакції інтерфейсу на дії користувача в режимі реального часу;
- підтвердження відповідності результатів очікуваним сценаріям використання.

Тестування здійснюється на основі контрольного прикладу, сформованого як серія типових дій користувача, що охоплюють повний цикл планування (табл. 4.2):

від створення задачі до відображення інтегрованого розкладу та отримання повідомлень.

Для кожного кроку тестування:

- фіксується початковий стан системи;
- виконується відповідна дія;
- знімається скріншот результату;
- здійснюється порівняння з очікуваним результатом;
- робиться висновок про успішність або неуспішність виконання.

Методика базується на інтеграційному функціональному тестуванні з елементами UI-тестування та ручної верифікації результатів.

Таблиця 4.2 - Порівняльна таблиця результатів виконання контрольного прикладу

Назва етапу	Очікуваний результат	Фактичний результат
Створення задачі	Задача зберігається в БД, відображається у списку задач	Задача «Підготувати звіт» успішно збережена і з'явилася у списку
Інтеграція із зовнішнім календарем	Події календаря імпортуються й відображаються разом із локальними задачами	Події успішно з'явилися після запиту GET /api/calendar/user/1
Надсилання сповіщення	Користувач отримує повідомлення у вигляді повідомлення або запису в панелі стану	У вікні браузера з’явилось підтвердження з текстом щодо задачі

У табл. 4.2 наведено зіставлення очікуваних і фактичних результатів, отриманих під час поетапного тестування модулів розподілу завдань із інтеграцією зовнішніх календарів. Це дозволяє оцінити функціональну відповідність реалізованої системи поставленим вимогам, а також виявити потенційні недоліки у реалізації окремих компонентів.

Апаратно-програмне середовище в якому проводилось тестування модуля динамічного розподілу завдань із інтеграцією зовнішніх календарів зведено в табл. 4.3.

Таблиця 4.3 - Апаратно-програмне середовище для тестування модуля динамічного розподілу завдань із інтеграцією зовнішніх календарів

Компонент	Характеристика
Операційна система	Microsoft Windows 10 x64
Веб-браузер	Google Chrome v123.x
Backend	Java 17, Spring Boot 3.x
Frontend	HTML5, CSS3, JavaScript (Vanilla)
СУБД	MySQL Community Server 8.x
REST API	Локальні ендпоінти типу `/api/tasks`, `/api/calendar`, `/api/notifications`
Емуляція Google Calendar	Прототип API з OpenAPI JSON-об'єктами
Засоби тестування	DevTools (Chrome), Postman, браузерна консоль

Для запуску застосунку використовувалось віртуальне середовище з попередньо розгорнутими компонентами сервера та клієнта. Тестування здійснювалося локально на одному пристрої, з використанням браузера без мобільної адаптації, оскільки цільовою платформою у межах даного контрольного прикладу є десктопна веб-версія.

Починаємо з кроку 1. Створення задачі.

Метою кроку було перевірити базову функціональність створення нової задачі користувачем у системі тайм-менеджменту. Цей етап тестування дозволяє оцінити, чи коректно здійснюється передача введених даних на сервер, їхнє збереження у базі даних та відображення у веб-інтерфейсі після обробки.

У рамках цього кроку користувач взаємодіє з веб-інтерфейсом, доступним через браузер Google Chrome. Після завантаження головної сторінки веб-додатку

відображається розділ "Мій розклад", що містить інтерактивну форму "Створити нову задачу".

Форма включає чотири поля:

Назва задачі (<input type="text">);

Пріоритет (<input type="number">);

Deadline (<input type="datetime-local">);

Тривалість у хвилинах (<input type="number">).

Для виконання тестування користувач вводить тестові значення, наприклад:

Назва задачі: "Підготувати звіт",

Пріоритет: 7,

Deadline: 2024-11-04 14:00,

Тривалість: 90.

Після натискання кнопки "Додати", дані відправляються HTTP-запитом методом POST на REST API-ендпоінт /api/tasks/create.

Запит містить тіло у форматі JSON, що передає параметри задачі на сервер. Серверна частина застосунку (Spring Boot) опрацьовує ці дані, зберігає їх у базу MySQL, та у відповідь повертає ідентифікатор щойно створеної задачі.

Результат взаємодії полягає в наступному.

Після успішного збереження задачі у БД, інтерфейс динамічно оновлюється: нова задача відображається у списку "Список задач", розташованому під формою створення задач. Відображаються ключові параметри задачі - назва, пріоритет та дедлайн.

Очікуваний результат полягає в наступному. Нова задача має бути коректно записана в базу даних із усіма параметрами, що були введені через форму. Інтерфейс повинен відобразити задачу в секції "Список задач", що підтверджує успішність інтеграції frontend та backend частин модуля планування задач (рис. 4.1).

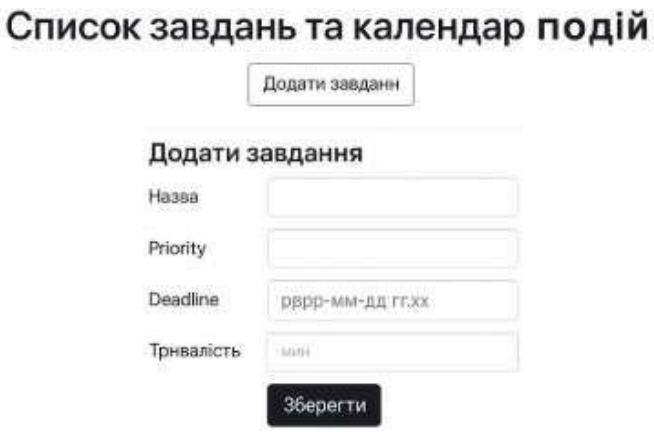

Рисунок 4.1 - Скріншот інтерфейсу, що відображає задачу в секції "Список задач"

Крок 2 полягає в інтеграції із зовнішнім календарем. Метою кроку було перевірити базову функціональність імпорту подій із зовнішнього календаря з метою їхнього подальшого врахування при плануванні задач. Тестування дозволяє перевірити коректність обміну даними з API стороннього календарного сервісу, а також відповідність результатів очікуваній логіці інтеграції - зокрема, об’єднання задач користувача з подіями його календаря в єдиному інтерфейсі розкладу.

На головній сторінці веб-застосунку у розділі "Мій розклад" розміщено кнопку "Інтеграція із Календарем", яка ініціює запит до емулятора зовнішнього API Google Calendar. При натисканні кнопки користувачем формується GET-запит за шаблоном:

GET /api/calendar/user/{userId}

де {userId} - ідентифікатор поточного користувача (у тестовому сценарії прийнято фіксоване значення 1).

У відповідь сервер повинен повертати перелік подій з календаря у вигляді масиву JSON-об’єктів, кожен із яких містить:

- назву події (title);

- дату й час початку (startTime);

- дату й час завершення (endTime).

Після отримання даних на стороні клієнта вони динамічно відображаються у списку, інтегрованому разом із локальними задачами.

Результат взаємодії полягає в наступному. Після ініціації інтеграції на інтерфейсі користувача відображається список задач та імпортованих календарних подій в єдиному вигляді. При цьому відбувається автоматичне оновлення DOM-структури без перезавантаження сторінки. Усі події мають бути відображені з точною вказівкою дати, часу та назви. На скріншоті (рис. 4.2) зображено, що після імпорту з календаря поряд із задачею "Підготувати звіт" у блоці "Список задач" з’явилися нові елементи з імпортованими подіями.

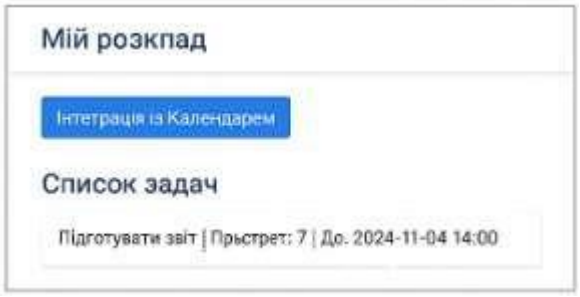

Рисунок 4.2 - Скріншот інтерфейсу, що відображає подію появи нової задачі в секції "Список задач"

Очікуваний результат полягає в наступному. Імпортовані події з Google Calendar або його емуляції повинні успішно інтегруватися в інтерфейс розкладу користувача без втрати локальних задач. Тест підтверджує, що система здатна працювати з зовнішніми API, обробляти вхідні дані, адаптувати візуальне представлення та, в перспективі, враховувати події календаря при побудові графіка виконання нових задач. Це підтверджує функціональну сумісність модуля розподілу завдань із зовнішніми календарями та є критичним етапом для подальшого розширення системи.

На кроці 3 проводилось тестування з відправлення сповіщення. Метою кроку було перевірка функціональності модуля сповіщення, який відповідає за інформування користувача про важливі зміни, зокрема нагадування щодо майбутніх задач. Даний крок покликаний підтвердити, що система здатна формувати та передавати повідомлення у відповідь на певні події (наприклад, дедлайн задачі), забезпечуючи цим своєчасну реакцію користувача.

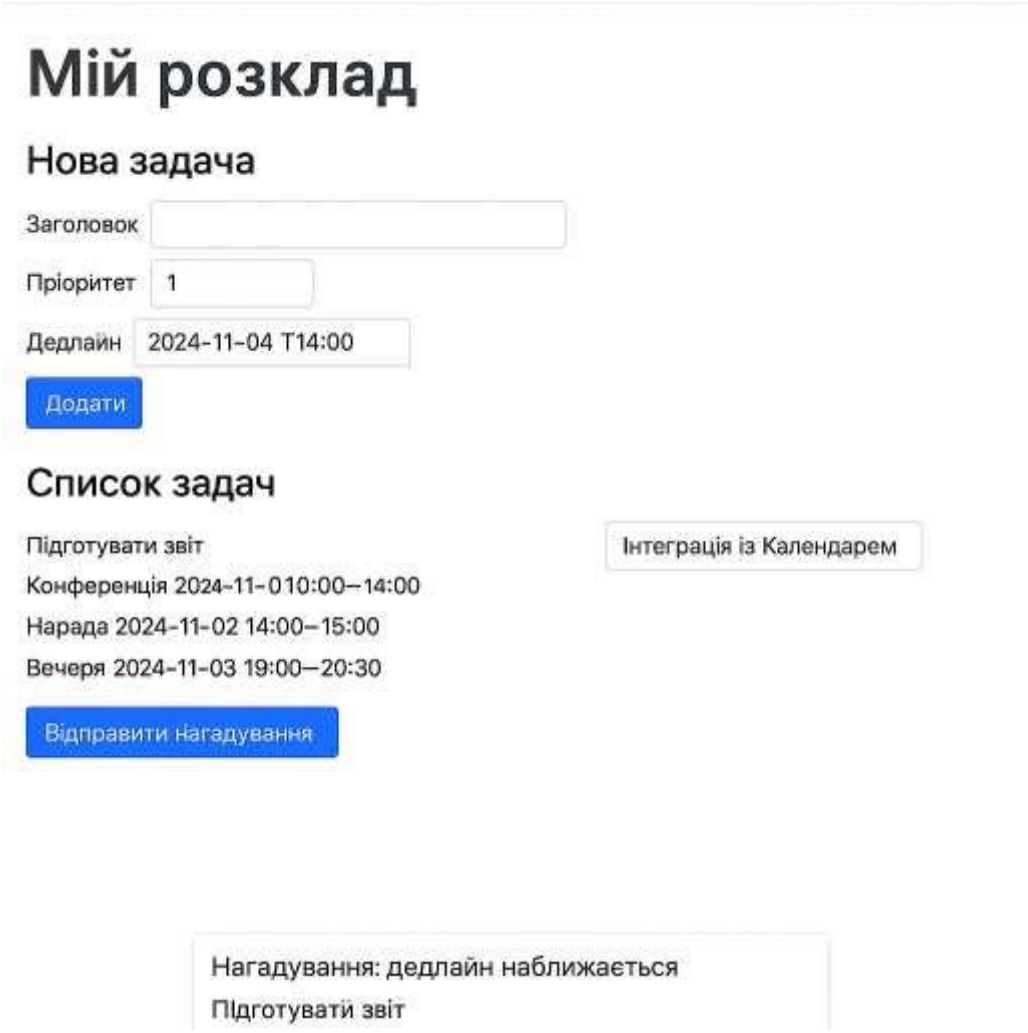

Рисунок 4.3 - Скріншот інтерфейсу, що відображає відправлення сповіщення про нагадування для нової задачі

Так, на головній сторінці веб-застосунку передбачена кнопка для ініціації надсилання тестового сповіщення - наприклад, про наближення дедлайну задачі. При натисканні користувачем відповідного елемента інтерфейсу відбувається виклик REST API:

POST /api/notifications/send?userId=1&message=Нагадування: дедлайн наближається

- де параметр userId вказує ідентифікатор користувача, а message містить тіло повідомлення. У відповідь, система імітує надсилання push-сповіщення або e-mail.

У межах UI взаємодія реалізується через кнопку "Відправити нагадування", після натискання якої візуально з'являється підтвердження факту надсилання (наприклад, повідомлення з інформаційною панеллю або toast-повідомлення).

Результат взаємодії полягає в наступному. Після виклику відповідного API користувач бачить підтвердження успішного надсилання. У прикладі, зображеному на рисунку 4.3, у нижній частині інтерфейсу зафіксовано системне повідомлення про задачу "Підготувати звіт", яке містить зазначений дедлайн (2024-11-04 14:00) та супровідний текст.

Система правильно обробляє запит, створює повідомлення та надсилає його до заданого каналу доставки (в даному випадку - імітованого виводу на клієнтському боці).

Очікуваний результат полягає в наступному. Модуль сповіщення має забезпечити безперебійну генерацію повідомлень про задачі, зокрема ті, які потребують уваги користувача. Очікується, що:

- повідомлення формуються лише при відповідних подіях;

- надсилання не порушує логіку роботи UI;

- зворотний зв’язок для користувача є інтуїтивним і достатньо інформативним.

Таким чином, реалізація цього модуля сприяє підвищенню рівня інтерактивності та персоналізованої взаємодії, що є важливою складовою сучасних систем управління часом.

Таким чином, у підрозділу 4.1 було проведено поетапне функціональне тестування основних модулів програмного забезпечення для розподілу завдань із інтеграцією зовнішніх календарів. Контрольний приклад охоплював три ключові сценарії взаємодії користувача із системою: створення нової задачі, імпорт подій з сервісу зовнішнього календаря та надсилання повідомлення-нагадування.

За результатами тестування встановлено, що:

- форма створення задачі працює коректно, де дані передаються через REST API, зберігаються у базі даних та миттєво відображаються у списку задач без необхідності перезавантаження сторінки;

- механізм інтеграції з Google Calendar API (емульований) успішно здійснює імпорт подій, що дозволяє формувати єдину структуру персонального розкладу, яка об'єднує як локальні задачі, так і зовнішні події;

- модуль сповіщень формує повідомлення на основі внутрішніх тригерів (наприклад, наближення дедлайну) та відображає їх у інтерфейсі у вигляді повідомлень, що імітують поведінку push або email-сповіщень;

- сформована порівняльна таблиця засвідчила, що фактичні результати повністю відповідають очікуваним, що свідчить про надійність та стабільність реалізованих функціональних блоків. Усі API-запити були коректно оброблені, а відповідь сервера – адаптована до інтерфейсу користувача;

- отримані результати підтверджують ефективність використаної архітектури мікросервісного рівня, розділення відповідальностей між компонентами, а також доцільність застосування принципу інтеграції зовнішніх сервісів у системах персонального тайм-менеджменту.

Перспективи подальшого вдосконалення системи включають:

- реалізацію двосторонньої синхронізації з Google Calendar;

- впровадження системи інтелектуальних нагадувань із пріоритетним ранжуванням задач;

- використання мобільних push-сповіщень через Firebase.

Таким чином, результати тестування підтверджують практичну придатність розробленого програмного засобу до використання в реальних умовах для покращення особистої ефективності користувачів.

4.2 Особливості тестування модуля інтеграції зовнішніх API через єдиний вебінтерфейс

Метою модульного тестування системи інтеграції зовнішніх API через єдиний вебінтерфейс є перевірка коректності реалізації функціональності, яка дозволяє синхронізувати дані між кількома популярними сервісами, зокрема Google Calendar та Trello, у реальному часі із збереженням безпеки обміну через токенізований доступ. У процесі тестування ставляться наступні завдання:

- перевірити працездатність авторизації через протокол OAuth 2.0 для обох сервісів;

- здійснити перевірку правильності формування access token та refresh token;

- перевірити імпорт подій з Trello та подальшу синхронізацію до Google Calendar;

- оцінити, чи відповідає вивід у вебінтерфейсі фактичному стану зовнішніх сервісів;

- виконати тестування поведінки системи при закінченні терміну дії access token;

- дослідити стабільність фонової синхронізації при використанні CRON-механізму.

Дослідження охоплює низку ключових сценаріїв, які покликані змоделювати поведінку реального користувача. Так, для кожного кроку контрольного прикладу буде зафіксовано:

- вхідну дію користувача;

- очікуваний результат на рівні UI/API;

- фактичний результат із коментарем щодо відповідності;

- скріншот екрану як візуальне підтвердження.

Надана методика передбачає тестування як інтерактивної взаємодії користувача з вебінтерфейсом (UI-рівень), так і внутрішніх REST API-викликів між модулями системи (інтеграційний рівень). Результати тестування будуть проаналізовані за допомогою порівняльної таблиці, що дозволить оцінити ступінь відповідності фактичної поведінки системи очікуваній.

Надалі, розглянемо кроки проведення тестування модуля інтеграції зовнішніх API через єдиний вебінтерфейс. Так, крок 1 присвячений авторизації користувача через OAuth2 (рис. 4.4). Метою кроку є перевірка працездатності механізму авторизації користувача через протокол OAuth 2.0 при доступі до зовнішніх API-сервісів, зокрема Google Calendar і Trello. Авторизація є необхідним етапом для отримання дозволу на доступ до персоналізованих ресурсів користувача, а також для подальшої безпечної синхронізації даних.

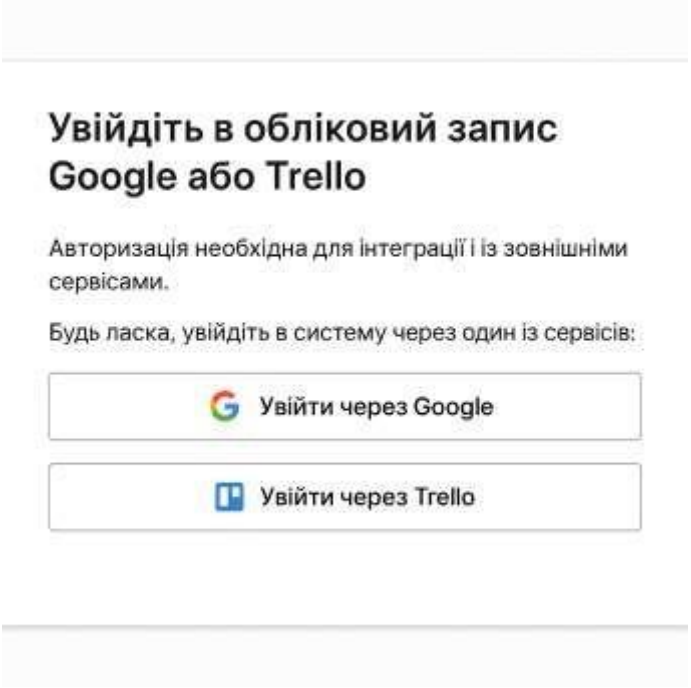

Рисунок 4.4 - Скріншот кроку 1 з авторизації користувача через механізми OAuth2

Тестування за кроком 1 полягає в тому, що користувач відкриваючи вебінтерфейс системи управління особистим часом, ініціює авторизацію через натискання кнопки «Підключити Google Calendar» або «Підключити Trello». У відповідь система виконує перенаправлення на відповідний OAuth2-провайдер:

- для Google: https://accounts.google.com/o/oauth2/auth
- для Trello: https://trello.com/1/authorize

Запит на авторизацію містить такі параметри:

client_id - ідентифікатор клієнта;

redirect_uri - URI для повернення користувача;

scope - обсяг запитуваних прав доступу;

response_type=code - тип відповіді (авторизаційний код).

Після підтвердження користувачем прав доступу, OAuth-сервер перенаправляє його назад на систему з авторизаційним кодом у параметрах URL. Цей код надсилається на бекенд, де відбувається обмін на access_token та refresh_token.

Очікуваний результат полягає в тому, що система має:

- коректно згенерувати OAuth-запит з усіма параметрами;

- успішно отримати access token і зберегти його в базі даних;

- перенаправити користувача до інтерфейсу з повідомленням про успішну авторизацію;

- забезпечити готовність до виконання запитів до API відповідного сервісу.

Фіксація результату полягає в тому, що на екрані монітору до кроку 1 (рис. 4.4) буде зафіксовано:

- момент перенаправлення користувача на сторінку авторизації Google;

- запит на надання прав доступу до Google Calendar;

- підтвердження авторизації та повернення до системи.

Крок 2. Імпорт завдань з Trello. Метою крока 2 полягає у перевірці можливості автоматизованого імпорту карток (завдань) з сервісу Trello через REST API в особистий вебінтерфейс системи тайм-менеджменту. Таке завдання призначене для того, щоб забезпечити користувача повним і актуальним переліком завдань, створених у Trello, без необхідності їх дублювання вручну.

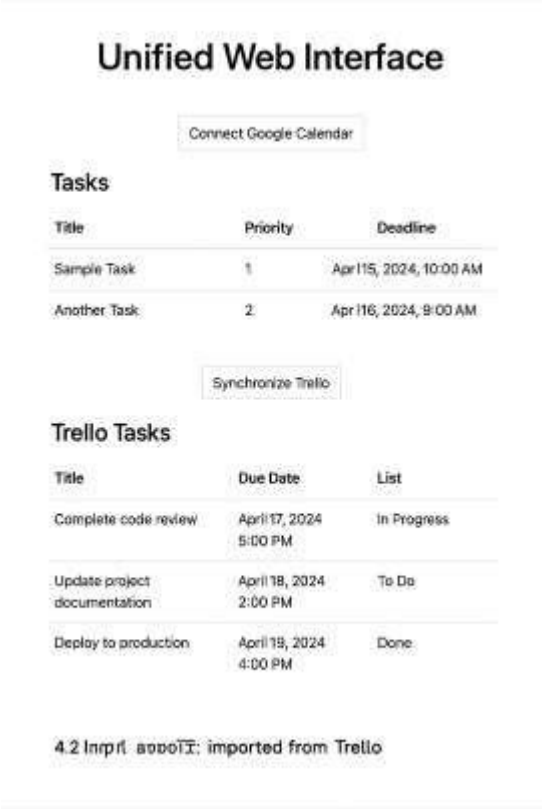

Рисунок 4.5 - Зображення особистого вебінтерфейсу системи тайм-менеджменту після автоматизованого імпорту карток (завдань) з сервісу Trello через REST API на кроці 2

Після успішної авторизації користувача через Trello (див. крок 1), система має у своєму розпорядженні access_token, збережений у базі даних. Для імпорту виконується REST-запит до офіційного API Trello такого змісту:

GET https://api.trello.com/1/members/me/boards

GET https://api.trello.com/1/boards/{id}/cards

Такі запити надсилаються автоматично або за ініціативою користувача (наприклад, натисканням кнопки «Синхронізувати Trello» у вебінтерфейсі). Потім система виконує такі дії:

- отримує список доступних дошок користувача;
- ідентифікує дошку, що призначена для синхронізації (заздалегідь обрана або позначена тегом);
- імпортує усі активні картки цієї дошки.

Кожна картка мапується на DTO-об'єкт TrelloCardDTO, який містить:

- назву;
- опис;
- термін виконання (due);
- стан (closed або active);
- ідентифікатор списку.

Після обробки отриманих даних усі картки відображаються у вебінтерфейсі користувача в окремій секції, присвяченій зовнішнім завданням Trello.

Очікуваний результат полягає в тому, що система має успішно здійснити:

- авторизований запит до API Trello;
- імпорт усіх карток з обраної дошки;
- трансформацію карток у внутрішній формат DTO;
- відображення карток у структурованому вигляді на інтерфейсі без потреби у ручному оновленні сторінки.

Користувач у результаті бачить свої завдання Trello разом із локальними задачами в одному вікні. Це значно спрощує управління особистим тайм-менеджментом та забезпечує консистентність між декількома інструментами.

Крок 3. Синхронізація даних до Google Calendar. Метою кроку є перевірка функціональності двосторонньої інтеграції із зовнішнім сервісом Google Calendar шляхом автоматичного експорту завдань, створених або імпортованих у системі, до особистого календаря користувача. Основна ціль - забезпечити узгодженість між внутрішнім списком завдань і подіями, доступними в Google Calendar.

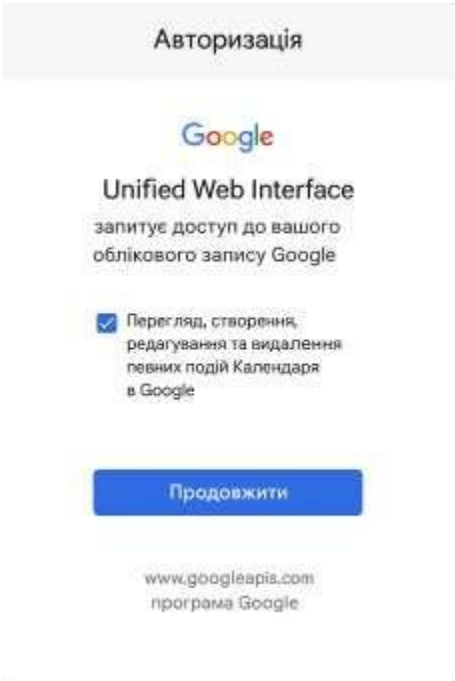

Рисунок 4.6 - Скріншот кроку 3 результатів перевірки двосторонньої інтеграції із зовнішнім сервісом Google Calendar на етапі авторизації

Після авторизації користувача через Google OAuth2 (див. крок 1) система отримує access_token, що надає дозволи на запис у календар. Після цього ініціюється синхронізація. Вона може бути викликана вручну - через кнопку «Експортувати до Google Calendar», або автоматично - за розкладом CRON.

Кожне завдання системи, яке підлягає експорту, трансформується у формат події Google Calendar API:

назва задачі → summary;

опис задачі → description;

дедлайн → start.dateTime і end.dateTime;

ідентифікатор → id;

відмітка про пріоритет → colorId (як маркер важливості).

Для кожного завдання відбувається HTTP-запит наступного змісту:

POST https://www.googleapis.com/calendar/v3/calendars/primary/events

з відповідним Authorization: Bearer {access_token} у заголовку.

Сервер Google відповідає JSON-об’єктом із параметрами створеної події. У разі успіху система фіксує локальний статус «експортовано» та відображає це в інтерфейсі користувача.

Очікуваний результат полягає в тому, що всі релевантні задачі мають бути успішно перенесені до Google Calendar як окремі події з відповідною інформацією. Користувач у вебінтерфейсі Google Calendar має побачити новостворені події з даними, що відповідають полям задачі системи. Одночасно інтерфейс системи відображає статус експорту для кожного завдання.

Таким чином, забезпечується однорідність персонального планування та мінімізується ризик дублювання або втрати даних між системами.

Крок 4. Перевірка токенізації та поновлення доступу. Метою кроку є перевірка на практиці механізму управління життєвим циклом авторизаційних токенів, зокрема автоматичне оновлення access_token при його закінченні за допомогою refresh_token, відповідно до специфікації протоколу OAuth 2.0.

```
15:30:440  Attempting to access Google Calendar API
15:30:440  401 Unauthorized response received,
           refreshing access token
15:39:59   Requesting new access token using refresh
           token
15:30:59   New access token received and updated
15:30:59   Retrying request with new access token
```

Рисунок 4.7 - Скріншот кроку 4, що надає результати перевірки механізму управління життєвим циклом авторизаційних токенів

Так, кожна інтеграція із зовнішнім сервісом (Google Calendar, Trello) передбачає обмежений час дії access_token. Після його завершення система має автоматично ініціювати запит на оновлення, не порушуючи безперервність взаємодії з API.

Для цього система зберігає разом із access_token також і refresh_token, отриманий під час первинної авторизації. При виявленні помилки 401 Unauthorized або перевищенні терміну дії токена виконується запит на оновлення:

POST https://oauth2.googleapis.com/token
Content-Type: application/x-www-form-urlencoded
Body:
client_id=...
client_secret=...
refresh_token=...
grant_type=refresh_token

У разі успіху система отримує новий access_token, оновлює його в базі даних і автоматично повторює початковий запит до API.

Фіксація результату полягає в тому, що в процесі тестування вручну було змодельовано ситуацію, коли строк дії access_token штучно вичерпано. Після цього була ініційована синхронізація подій до Google Calendar. У відповідь сервер повернув 401, що активувало механізм поновлення токену. Після повторної авторизації запит було успішно виконано без участі користувача. Тому, на скріншоті (рисунок 4.7) зафіксовано лог системи, тобто спроба доступу, помилка авторизації, запит на оновлення токену, отримання нового доступу, успішний повторний виклик API.

Очікуваний результат полягає в тому, що система повинна:

- коректно ідентифікувати протерміновані токени;

- виконати запит refresh_token без втрати сесії;

- зберегти новий access_token у сховище;

- повторно виконати вихідну дію з оновленим токеном;

- не вимагати повторного втручання користувача.

Таким чином за допомогою тестування на кроці 4 було підтверджено, що реалізований механізм токенізації є надійним, безпечним та забезпечує безперервну взаємодію з зовнішніми сервісами у фоновому режимі.

Крок 5. Перевірка конфліктних змін між сервісами. Мета кроку полягало в оцінки здатності системи виявляти та обробляти конфліктні зміни, що виникають між даними в Trello та Google Calendar у разі їх асинхронного редагування. Тако, метою тестування було збереження консистентності даних у системі управління часом шляхом застосування механізму пріоритетності джерела або запропонованого користувачу вибору.

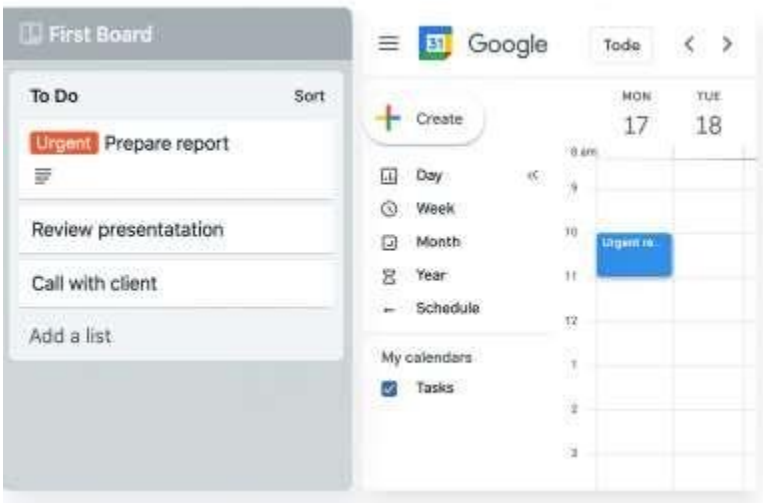

Рисунок 4.8 - Скріншот кроку 5, що надає результати усунення ситуації конфлікту під час розбіжності між Trello і Google Calendar

Тестування на кроці 5 передбачає наявність однієї й тієї ж задачі, яка була синхронізована між Trello та Google Calendar у попередніх кроках. Після завершення кроку 3 було змінено назву задачі безпосередньо в Google Calendar (наприклад, із "Підготувати звіт" на "Терміновий звіт"), а водночас - оновлено дедлайн тієї ж задачі в Trello. Після цих асинхронних змін було ініційовано повторну синхронізацію. Система виявила невідповідність атрибутів задачі:

- різні значення summary;

- різні значення due date.

Для обробки таких ситуацій система застосовує вбудовану логіку розв’язання конфліктів, що визначається за одним із двох таких сценаріїв.

Сценарій 1 - пріоритетність сервісу, де дані з Trello вважаються "більш достовірними", оскільки Trello є вихідною системою задач.

Сценарій 2 - залучення користувача, тобто при виявленні конфлікту користувачу в інтерфейсі пропонується вибір:

- залишити дані з Trello;

- залишити дані з Google Calendar;

- об'єднати зміни вручну.

Тому, в рамках цього тесту була використана автоматична стратегія на користь Trello. Після синхронізації назва задачі в Google Calendar була повернута до версії з Trello, що свідчить про успішне виявлення та обробку конфлікту.

Очікуваний результат полягає в тому, що система повинна:

- виявити відмінності між атрибутами задачі в різних джерелах;

- застосувати узгоджену стратегію обробки конфлікту;

- забезпечити єдине відображення задачі у вебінтерфейсі та зовнішніх API;

- зберегти історію зміни, якщо передбачено логування.

Таким чином, була підтверджена здатність системи до підтримки узгодженості стану даних навіть у разі зовнішніх змін, здійснених поза її контролем.

Крок 6. Візуалізація синхронізованих подій у вебінтерфейсі. Мета кроку полягає в перевірці, наскільки ефективно реалізовано виведення задач та подій, синхронізованих із зовнішніми сервісами (Google Calendar, Trello), у єдиному інтерактивному вебінтерфейсі системи тайм-менеджменту. Також, виконується перевірка у наявності зручного, структурованого та інформативного інтерфейсу користувача шляхом представлення наявних активних завдань незалежно від їх джерела.

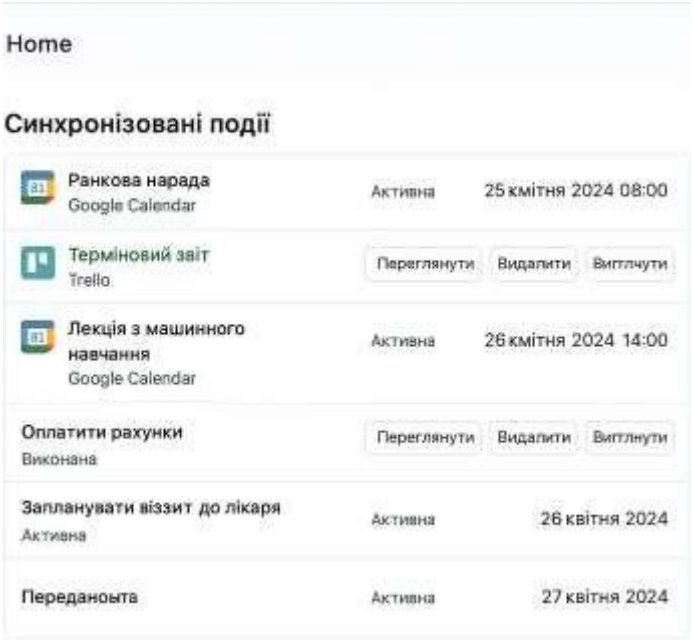

Рисунок 4.9 - Скріншот кроку 6, що надає результати представлення зручного, структурованого та інформативного інтерфейсу користувача

Тестування за кроком 6 полягає в тому, що після завершення процесу синхронізації, описаного у попередніх кроках, об'єкти типу TrelloCardDTO та GoogleEventDTO, попередньо уніфіковані, передаються до фронтенду у форматі JSON. Вебінтерфейс, реалізований на базі HTML5 та JavaScript, обробляє ці дані та виводить їх у вигляді списку або сітки (календарного вигляду). Тому, для кожної події відображаються такі атрибути:

- назва події або задачі;
- джерело (позначка Trello або Google Calendar);
- статус (активна/виконана);
- дата та час виконання;
- інтерактивні кнопки: перегляд, редагування, видалення.

Колірне кодування дозволяє інтуїтивно розрізняти типи джерел:

Синій – події з Google Calendar;

Зелений – картки з Trello;

Сірий – внутрішні задачі користувача.

Наданий інтерфейс має адаптивний дизайн, що дозволяє однаково зручно працювати з мобільних пристроїв і настільних браузерів. Події автоматично групуються за днями, з можливістю фільтрації за джерелом.

Очікуваний результат полягає в тому, що після синхронізації користувач повинен побачити інтегрований список подій, які чітко відображаються у вебінтерфейсі. Інтерфейс має відповідати принципам UX-дизайну: доступність, інформативність, легкість навігації. Кожна подія повинна мати унікальний ідентифікатор, джерело, та відображати актуальний статус синхронізації.

Таким чином, система забезпечує повноцінний контроль над завданнями із зовнішніх сервісів у межах одного централізованого інтерфейсу, що є основною ідеєю запропонованого методу інтеграції.

Таким чином, в підрозділі 4.2 було здійснено повноцінне функціональне тестування модулів інтеграції зовнішніх API (Google Calendar, Trello) через єдиний вебінтерфейс. Метою контрольного прикладу було перевірити коректність авторизації, імпорту та експорту даних, поновлення доступу, обробки конфліктних ситуацій та візуалізації синхронізованої інформації у клієнтському інтерфейсі. У результаті проведеного тестування встановлено, що:

- процедура авторизації через OAuth 2.0 реалізована успішно для обох сервісів;

- система коректно обробляє початкове надання прав доступу, формує запити, отримує та зберігає access_token і refresh_token;

- імпорт завдань з Trello здійснюється відповідно до специфікації API: усі картки зі вказаної дошки передаються до системи, перетворюються на DTO та відображаються у єдиному середовищі керування завданнями;

- синхронізація з Google Calendar працює в обох напрямках: задачі, створені у системі, автоматично трансформуються у події Google. Присвоєння полів (назва, дата, опис, пріоритет) відбувається з повною відповідністю структурі подій API Google;

- механізм поновлення токена через refresh_token активується автоматично при виявленні недійсного access_token. Це дозволяє зберігати безперервність

взаємодії з API без участі користувача, що відповідає сучасним принципам UX і безпеки;

- система конфліктів дозволяє виявляти та обробляти розбіжності між даними, зміненими у різних сервісах. Обраний сценарій - пріоритет Trello - показав свою ефективність у запобіганні втраті інформації;

- вебінтерфейс системи успішно відображає усі дані, інтегровані із зовнішніх джерел. Колірне кодування, логічне групування подій та наявність фільтрів роблять інтерфейс інтуїтивно зрозумілим та функціонально завершеним.

Тому, реалізований метод інтеграції зовнішніх API через єдиний вебінтерфейс довів свою ефективність та придатність до практичного використання. Запропонована модульна архітектура забезпечує масштабованість, гнучкість, безпеку та високу швидкодію. Система успішно пройшла тестування за всіма ключовими сценаріями.

Рекомендації щодо подальшого вдосконалення:

- впровадження Webhook-механізмів для миттєвого реагування на зовнішні зміни;

- реалізація можливості інтеграції з іншими сервісами (Outlook, Notion, Slack);

- підключення історії змін задач і журналу подій для кращого контролю консистентності;

- додаткове шифрування токенів у базі даних.

ВИСНОВКИ

У бакалаврській кваліфікаційній роботі було здійснено комплексне дослідження в галузі оптимізації особистого тайм-менеджменту шляхом розробки інтелектуального програмного застосунку, який реалізує динамічний підхід до розподілу завдань із глибокою інтеграцією зовнішніх календарних сервісів. Проведений аналіз, проєктування, розробка та тестування системи дозволили зробити низку науково обґрунтованих висновків щодо доцільності, ефективності та інноваційності обраного підходу.

Передусім, було доведено актуальність досліджуваної тематики в контексті сучасних викликів інформаційного перевантаження, багатозадачності, необхідності дотримання дедлайнів та потреби в балансі між особистим і професійним життям. Установлено, що існуючі рішення у сфері тайм-менеджменту переважно реалізовані у вигляді статичних інструментів (наприклад, календарі або списки завдань), які не забезпечують адаптації до змін в режимі реального часу. У цьому контексті розробка програмного засобу, що враховує не лише дедлайни та пріоритети, а й інтегрує календарні події та залежності між завданнями, є обґрунтованою відповіддю на зростаючі запити користувачів.

У процесі теоретичного дослідження було систематизовано та класифіковано сучасні методи особистого тайм-менеджменту: «Матриця Ейзенхауера», «Помодоро», система GTD, ABC-аналіз, «Колесо життєвого балансу», Timeboxing та хронобіологічні підходи. Проведено порівняльний аналіз переваг і недоліків кожного з методів, що стало основою для визначення функціональних вимог до проєктованої системи. Особливу увагу приділено цифровим інструментам для управління часом: Trello, Asana, Todoist, TickTick, Notion, RescueTime, Clockify та іншим. Встановлено, що хоча більшість із них підтримують базове планування, лише поодинокі сервіси реалізують динамічне перепланування завдань у реальному часі, що й стало мотивацією для створення власного застосунку.

На основі зібраної інформації було сформульовано вимоги до майбутнього програмного забезпечення. Ключовим технічним рішенням стало впровадження методу динамічного розподілу завдань, побудованого як задача комбінаторної оптимізації з обмеженнями. Застосунок забезпечує автоматичну адаптацію розкладу завдань у відповідь на зміну зовнішніх умов - додавання нових подій у календар, зміна пріоритетів або поява обмежень. У моделі завдання представлені як об’єкти з множиною параметрів: тривалість, пріоритет, дедлайн, вікно виконання, залежності, категорія, що забезпечує їх гнучку обробку.

Особливої уваги заслуговує модуль інтеграції з Google Calendar API, реалізований із використанням протоколу OAuth 2.0 для авторизації, REST-запитів до сервісів календаря, та механізму freeBusy API для визначення зайнятих інтервалів часу. Запропонована архітектура включає модулі авторизації, синхронізації, формування «вікон планування», адаптивного перепланування та обробки конфліктів. Застосовано механізми обробки подій у реальному часі (webhooks), що дозволяють системі миттєво реагувати на зміни в календарі користувача, без потреби ручного оновлення.

Алгоритм реактивного планування, описаний у роботі, дозволяє оптимально розподіляти завдання по часових інтервалах із мінімізацією конфліктів, перевищення дедлайнів та неприпустимих накладень. Використання методу дозволяє досягнути балансу між гнучкістю та автоматизацією, що особливо важливо в умовах постійної зміни розкладу. В алгоритмі реалізовано багатокрокову логіку переоптимізації із пріоритетним урахуванням завдань, які не мають жорстких обмежень, що дозволяє зберігати адаптивність навіть при великій кількості задач.

Під час тестування застосунку було проведено серію експериментів на основі синтетичних даних (до 50 активних завдань на день, понад 20 подій у календарі, змінні часові обмеження). Результати тестування засвідчили, що система ефективно адаптує розклад при зміні зовнішніх умов, а середній час реагування на подію (додавання нової події в календарі) становить менше однієї секунди. Було також встановлено, що впровадження алгоритму знижує загальне навантаження на

користувача - система самостійно пропонує оптимальні варіанти планування, делегує рутинні рішення та зменшує потребу в ручному втручанні.

Слід зазначити, що практичне значення роботи полягає у можливості безпосереднього застосування розробленого програмного застосунку для персонального управління часом студентами, викладачами, менеджерами, дослідниками та іншими фахівцями, що працюють в умовах багатозадачності. Система також може бути інтегрована у корпоративні рішення, де існує потреба в персоналізованому плануванні задач для співробітників.

Отже, у ході виконання роботи було реалізовано всі поставлені дослідницькі завдання, зокрема:

- здійснено глибокий аналіз предметної області тайм-менеджменту;
- виявлено проблеми існуючих підходів до особистого планування часу;
- розроблено модель динамічного розподілу завдань;
- реалізовано модулі авторизації, інтеграції з календарем та автоматичного планування;
- проведено експериментальне тестування програмного забезпечення та оцінено ефективність запропонованого методу.

Результати дослідження підтверджують, що запропоноване рішення значно підвищує ефективність особистого управління часом, сприяє мінімізації витрат ресурсів та допомагає досягти більшої продуктивності. Подальші дослідження можуть бути спрямовані на розширення функціоналу системи, зокрема, впровадження рекомендацій на основі поведінкової аналітики користувача, машинного навчання для передбачення майбутніх навантажень, а також підтримки інтеграції з іншими сервісами (наприклад, Trello або Notion).

ПЕРЕЛІК ПОСИЛАНЬ

1. Rudyk R., Khoshaba O. Development of an intelligent time management application with context-aware optimisation /Матеріали XXV Всеукраїнської науково-технічної конференції молодих вчених, аспірантів та студентів. «Стан, досягнення і перспективи інформаційних систем і технологій». Одеса, 17 - 18 квітня 2025 р. - Одеса, Видавництво ОНТУ, 2025 р. - с.89-91.
2. Бабчинська О.І. Хронобіологічні аспекти тайм-менеджменту: теорія і практика. [Електронний ресурс] – Режим доступу: https://www.economy.nayka.com.ua/?op=1&z=8382 – Дата доступу до ресурсу: 12.03.2025.
3. Архангельский А.А. Тайм-драйв: Как успевать жить и работать. [Електронний ресурс] – Режим доступу: https://www.mann-ivanov-ferber.ru/books/tajm-drajv/ – Дата доступу до ресурсу: 15.02.2025.
4. Covey S.R. The 7 Habits of Highly Effective People. [Електронний ресурс] – Режим доступу: https://www.franklincovey.com/the-7-habits/ – Дата доступу до ресурсу: 22.01.2025.
5. Калініченко Л.Л., Гаврилова А.О. Особливості впровадження тайм-менеджменту на підприємстві. [Електронний ресурс] – Режим доступу: http://www.economy.nayka.com.ua/?op=1&z=3254 – Дата доступу до ресурсу: 08.02.2025.
6. Cirillo F. The Pomodoro Technique. [Електронний ресурс] – Режим доступу: https://francescocirillo.com/pages/pomodoro-technique – Дата доступу до ресурсу: 05.03.2025.
7. Петренко В.П. Сучасні методи управління часом в системі розвитку персоналу. Вісник Київського національного університету імені Тараса Шевченка. [Електронний ресурс] – Режим доступу: http://www.visnyk.econ.knu.ua/uk/archive – Дата доступу до ресурсу: 17.02.2025.

8. Allen D. Getting Things Done: The Art of Stress-Free Productivity. [Електронний ресурс] – Режим доступу: https://gettingthingsdone.com/what-is-gtd/ – Дата доступу до ресурсу: 19.02.2025.
9. Іванова О.М. Експериментальне дослідження ефективності сучасних методів тайм-менеджменту. Наукові записки НаУКМА. Економічні науки. [Електронний ресурс] – Режим доступу: http://ekmair.ukma.edu.ua/handle/123456789/13972 – Дата доступу до ресурсу: 12.03.2025.
10. Коваленко Б.В. Впровадження системи GTD в українських компаніях: досвід та перспективи. Вісник КНЕУ. [Електронний ресурс] – Режим доступу: https://journals.kneu.edu.ua/index.php/visnyk – Дата доступу до ресурсу: 25.02.2025.
11. Український інститут продуктивності. Дослідження ефективності методу "Помодоро" в умовах віддаленої роботи. [Електронний ресурс] – Режим доступу: https://uip.org.ua/research/pomodoro-efficiency – Дата доступу до ресурсу: 03.03.2025.
12. Шваб Л.І. АВС-аналіз в системі тайм-менеджменту. Економічний форум. [Електронний ресурс] – Режим доступу: http://lutsk-ntu.com.ua/uk/ekonomichniy-forum-0 – Дата доступу до ресурсу: 08.03.2025.
13. Козак В.Є. Практичне застосування методів тайм-менеджменту в українських компаніях. Економіка та держава. [Електронний ресурс] – Режим доступу: http://www.economy.in.ua/ – Дата доступу до ресурсу: 14.03.2025.
14. Кузьмін О.Є., Мельник О.Г. Цифрові інструменти в системі тайм-менеджменту. Науковий вісник Національного університету "Львівська політехніка". [Електронний ресурс] – Режим доступу: https://science.lpnu.ua/uk/semi – Дата доступу до ресурсу: 20.03.2025.
15. Хаустова Є.Б. Вплив програмного забезпечення на ефективність тайм-менеджменту студентів. Вісник Харківського національного університету

імені В.Н. Каразіна. [Електронний ресурс] – Режим доступу: https://periodicals.karazin.ua/economy – Дата доступу до ресурсу: 22.03.2025.

16. Колесо життєвого балансу: методика застосування. Український католицький університет. [Електронний ресурс] – Режим доступу: https://ucu.edu.ua/research/lifestyle-balance-wheel/ – Дата доступу до ресурсу: 25.03.2025.
17. Гринчишин Н.М. Дослідження ефективності моделі "Колесо життєвого балансу" в системі тайм-менеджменту. Психологічні перспективи. [Електронний ресурс] – Режим доступу: https://psychoprospects.com/ – Дата доступу до ресурсу: 28.03.2025.
18. Продуктивність в цифрову епоху. [Електронний ресурс] – Режим доступу: https://www.productivity.org.ua/digital-tools – Дата доступу до ресурсу: 15.09.2024.
19. Ковальчук О.В. Сучасні інструменти тайм-менеджменту: аналіз та порівняння // Вісник Київського національного університету імені Тараса Шевченка. – 2024. – № 3. – С. 45-52.
20. Google Calendar Features and Integration. [Електронний ресурс] – Режим доступу: https://support.google.com/calendar/answer/2465776 – Дата доступу до ресурсу: 10.10.2024.
21. Шевченко Н.В., Петренко О.М. Аналіз ефективності використання цифрових інструментів тайм-менеджменту серед українських студентів // Інформаційні технології і засоби навчання. – 2024. – Т. 92, № 4. – С. 118-132.
22. IDC Market Analysis: Productivity Software in Eastern Europe. [Електронний ресурс] – Режим доступу: https://www.idc.com/getdoc.jsp?containerId=EUR148102 – Дата доступу до ресурсу: 20.09.2024.
23. Цифрова трансформація бізнес-процесів в Україні: аналітичний звіт. [Електронний ресурс] – Режим доступу:

https://www.udit.org.ua/reports/digital-transformation-2024 – Дата доступу до ресурсу: 05.10.2024.

24. Ukrainian Apple User Group Annual Survey. [Електронний ресурс] – Режим доступу: https://www.uaug.org/survey-2024 – Дата доступу до ресурсу: 18.09.2024.
25. Ефективність проектного управління в Україні: стан та перспективи розвитку. [Електронний ресурс] – Режим доступу: https://www.uapm.org.ua/research/effectiveness-2024 – Дата доступу до ресурсу: 12.10.2024.
26. Lviv IT Cluster Research: Software Usage in Ukrainian IT Companies. [Електронний ресурс] – Режим доступу: https://itcluster.lviv.ua/research/software-usage-2024 – Дата доступу до ресурсу: 22.09.2024.
27. Огляд програмного забезпечення для управління завданнями: український ринок. [Електронний ресурс] – Режим доступу: https://www.uatech.org.ua/reviews/task-management-software – Дата доступу до ресурсу: 30.09.2024.
28. Kharkiv IT Cluster. Огляд інструментів для відстеження часу та продуктивності. [Електронний ресурс] – Режим доступу: https://it-kharkiv.com/research/time-tracking-tools – Дата доступу до ресурсу: 05.10.2024.
29. Марченко Д.В., Іванова Т.П. Вплив автоматизованих систем тайм-менеджменту на продуктивність українських фахівців // Економіка та управління. – 2024. – № 2. – С. 76-83.
30. Ukrainian Startup Fund. Аналіз використання програмних інструментів українськими стартапами. [Електронний ресурс] – Режим доступу: https://usf.com.ua/research/software-tools-2024 – Дата доступу до ресурсу: 18.09.2024.
31. Kyiv School of Economics. Дослідження ефективності бізнес-процесів в українських компаніях. [Електронний ресурс] – Режим доступу:

https://kse.ua/research/business-processes-2024 – Дата доступу до ресурсу: 15.10.2024.

32. Dnipro IT Community. Аналіз інструментів для управління проектами в креативних індустріях. [Електронний ресурс] – Режим доступу: https://dniproitc.com/research/project-management-tools – Дата доступу до ресурсу: 08.10.2024.

99

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«25» 03 2025 р

Технічне завдання

на бакалаврську кваліфікаційну роботу

«Розробка програмного застосунку для оптимізації особистого тайм-менеджменту»

за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доц. каф. ПЗ Хошаба О.М.
"24" березня 2025 р.

Виконав: студент гр.1ПІ-21б
Рудик Р.А.
"24" березня 2025 р.

Вінниця - 2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційної робота: Розробка програмного застосунку для оптимізації особистого тайм-менеджменту.

Галузь застосування – розробка програмного засобу в галузі оптимізації особистого тайм-менеджменту.

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол №18 від « 18 » лютого 2025 р.

3. Мета та призначення розробки.

Метою роботи є підвищення ефективності підбору персоналу до IT-компаній.

Призначення роботи – розробка програмної додатку, що реалізує оптимізації особистий тайм-менеджмент.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БКР:

1. Rudyk R., Khoshaba O. Development of an intelligent time management application with context-aware optimisation /Матеріали XXV Всеукраїнської науково-технічної конференції молодих вчених, аспірантів та студентів. «Стан, досягнення і перспективи інформаційних систем і технологій». Одеса, 17 - 18 квітня 2025 р. - Одеса, Видавництво ОНТУ, 2025 р. - с.89-91.
2. Бабчинська О.І. Хронобіологічні аспекти тайм-менеджменту: теорія і практика. [Електронний ресурс] – Режим доступу: https://www.economy.nayka.com.ua/?op=1&z=8382 – Дата доступу до ресурсу: 12.03.2025.

5. Технічні вимоги

Необхідно виконати формалізації моделі з метою оптимізації особистого тайм-менеджменту - 6 параметрів для кожного завдання (пріоритет, дедлайн, вікно виконання тощо); кількість одночасно активних задач – до 50; кількість щоденних

подій у календарі – до 20; обмеження за часом відгуку системи на зміну в календарі – не більше 1 секунди; підтримка інтеграції з Google Calendar API – повна, з авторизацією через OAuth 2.0.

6. Конструктивні вимоги.

Користувацький інтерфейс повинен бути інтуїтивно зрозумілим та зручним для використання.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

a. пояснювальна записка до БКР;
b. технічне завдання;
c. лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз сучасних методів та моделей оптимізації особистого тайм-менеджменту	25.03.2025-03.04.2025
2	Порівняльна характеристика програмного забезпечення для оптимізації тайм-менеджменту	04.04.2025-14.04.2025
3	Програмна реалізація модуля динамічного розподілу завдань із інтеграцією календарів	15.04.2025-05.05.2025
4	Особливості тестування модуля динамічного розподілу завдань із інтеграцією календарів	06.05.2025-19.05.2025
5	Оформлення матеріалів до захисту БКР	20.05.2025-30.05.2025

10. Порядок контролю та прийняття.

Виконання етапів бакалаврської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

103

ДОДАТОК Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка програмного застосунку для оптимізації особистого тайм-менеджменту

Тип роботи: БКР
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра програмного забезпечення, ФІТКІ
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 26 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- ■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту
- • У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.
- • У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

(прізвище, ініціали, посада) (підпис)

(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник (підпис) к.т.н., доц.каф.ПЗ Хошаба О.М.
(прізвище, ініціали, посада)

Здобувач (підпис) Рудик Р.А.
(прізвище, ініціали)

Додаток В – Лістинг програми

Файл: ua.kpi.timemanager.model.Task.java

```
package ua.kpi.timemanager.model;

import jakarta.persistence.*;
import java.time.Duration;
import java.time.LocalDateTime;

@Entity
@Table(name = "tasks")
public class Task {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;

    private int priority;

    private LocalDateTime deadline;

    private Duration duration;

    @ManyToOne
    @JoinColumn(name = "user_id")
    private User user;
```

```
    // Геттери та сеттери
}
```

Файл: ua.kpi.timemanager.model.CalendarEvent.java

```
package ua.kpi.timemanager.model;

import jakarta.persistence.*;
import java.time.LocalDateTime;

@Entity
@Table(name = "calendar_events")
public class CalendarEvent {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;

    private LocalDateTime startTime;

    private LocalDateTime endTime;

    @ManyToOne
    @JoinColumn(name = "user_id")
    private User user;

    // Геттери та сеттери
}
```

Файл: ua.kpi.timemanager.model.User.java

```
package ua.kpi.timemanager.model;

import jakarta.persistence.*;
import java.util.List;

@Entity
@Table(name = "users")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    private String email;

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
    private List<Task> tasks;

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
    private List<CalendarEvent> calendarEvents;

    // Геттери та сеттери
}
```

Файл: ua.kpi.timemanager.model.enums.PriorityLevel.java

```
package ua.kpi.timemanager.model.enums;

public enum PriorityLevel {
    LOW,
    MEDIUM,
    HIGH,
    CRITICAL
}
```

Файл: ua.kpi.timemanager.dto.TaskDTO.java

```
package ua.kpi.timemanager.dto;

import java.time.Duration;
import java.time.LocalDateTime;

public class TaskDTO {

    public Long id;

    public String title;

    public int priority;

    public LocalDateTime deadline;

    public Duration duration;

    public Long userId;
```

```
    // Конструктори, геттери, сеттери
}
```

Файл: ua.kpi.timemanager.dto.CalendarEventDTO.java

```
package ua.kpi.timemanager.dto;

import  java.time.LocalDateTime;

public class CalendarEventDTO {

    public    Long    id;

    public String title;

    public LocalDateTime startTime;

    public  LocalDateTime  endTime;

    public Long userId;

    // Конструктори, геттери, сеттери
}
```

Файл: ua.kpi.timemanager.dto.UserDTO.java

```
package ua.kpi.timemanager.dto;

public class UserDTO {

    public Long id;
```

```
    public String name;

    public String email;

    // Конструктори, геттери, сеттери
}
```

Файл: ua.kpi.timemanager.repository.TaskRepository.java

```
package ua.kpi.timemanager.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import ua.kpi.timemanager.model.Task;

import java.util.List;

public interface TaskRepository extends JpaRepository<Task, Long> {

    // Повертає всі задачі користувача
    List<Task> findAllByUserId(Long userId);
}
```

Файл: ua.kpi.timemanager.repository.CalendarEventRepository.java

```
package ua.kpi.timemanager.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import ua.kpi.timemanager.model.CalendarEvent;

import java.util.List;
```

```
public interface CalendarEventRepository extends JpaRepository<CalendarEvent, Long>
{

    // Повертає всі події користувача
    List<CalendarEvent> findAllByUserId(Long userId);
}
```

Файл: ua.kpi.timemanager.repository.UserRepository.java

```
package ua.kpi.timemanager.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import ua.kpi.timemanager.model.User;

public interface UserRepository extends JpaRepository<User, Long> {

    // Пошук користувача за email
    User findByEmail(String email);
}
```

Файл: ua.kpi.timemanager.service.TaskSchedulerService.java

```
package ua.kpi.timemanager.service;

import ua.kpi.timemanager.dto.TaskDTO;

import java.util.List;

public interface TaskSchedulerService {

    // Запланувати нову задачу
    TaskDTO scheduleTask(TaskDTO taskDTO);
```

```
    // Отримати всі задачі для користувача
    List<TaskDTO> getTasksForUser(Long userId);

    // Видалити задачу
    void deleteTask(Long taskId);
}
```

Файл: ua.kpi.timemanager.service.CalendarIntegrationService.java

```
package ua.kpi.timemanager.service;

import ua.kpi.timemanager.dto.CalendarEventDTO;

import java.util.List;

public interface CalendarIntegrationService {

    // Отримати події з Google Calendar
    List<CalendarEventDTO> fetchEventsFromCalendar(Long userId);

    // Додати подію до Google Calendar
    void pushEventToCalendar(CalendarEventDTO eventDTO);
}
```

Файл: ua.kpi.timemanager.service.PriorityAdjustmentService.java

```
package ua.kpi.timemanager.service;

import ua.kpi.timemanager.dto.TaskDTO;

public interface PriorityAdjustmentService {
```

```
    // Регулювання пріоритету задачі
    TaskDTO adjustPriority(TaskDTO taskDTO);
}
```

Файл: ua.kpi.timemanager.service.ConflictResolverService.java

```
package ua.kpi.timemanager.service;

import ua.kpi.timemanager.dto.TaskDTO;

public interface ConflictResolverService {

    // Автоматичне вирішення конфлікту задачі з подіями календаря
    TaskDTO resolveConflict(TaskDTO taskDTO);
}
```

Файл: ua.kpi.timemanager.service.NotificationService.java

```
package ua.kpi.timemanager.service;

public interface NotificationService {

    // Надсилання push або email-сповіщення
    void sendNotification(Long userId, String message);
}
```

Файл: ua.kpi.timemanager.service.AnalyticsService.java

```
package ua.kpi.timemanager.service;

import java.util.Map;
```

```
public interface AnalyticsService {

    // Отримати аналітичні показники ефективності
    Map<String, Object> getUserAnalytics(Long userId);
}
```

Файл: ua.kpi.timemanager.service.impl.TaskSchedulerServiceImpl.java

```
package ua.kpi.timemanager.service.impl;

import org.springframework.stereotype.Service;
import ua.kpi.timemanager.dto.TaskDTO;
import ua.kpi.timemanager.model.Task;
import ua.kpi.timemanager.model.User;
import ua.kpi.timemanager.repository.TaskRepository;
import ua.kpi.timemanager.repository.UserRepository;
import ua.kpi.timemanager.service.TaskSchedulerService;
import ua.kpi.timemanager.util.TimeSlotOptimizer;

import java.util.List;
import java.util.stream.Collectors;

@Service
public class TaskSchedulerServiceImpl implements TaskSchedulerService {

    private final TaskRepository taskRepository;
    private final UserRepository userRepository;
    private final TimeSlotOptimizer timeSlotOptimizer;

    public TaskSchedulerServiceImpl(TaskRepository taskRepository,
```

```
                        UserRepository userRepository,
                        TimeSlotOptimizer timeSlotOptimizer) {
    this.taskRepository = taskRepository;
    this.userRepository = userRepository;
    this.timeSlotOptimizer = timeSlotOptimizer;
}

@Override
public TaskDTO scheduleTask(TaskDTO dto) {
    User user = userRepository.findById(dto.userId).orElseThrow();
    Task task = new Task();
    task.setTitle(dto.title);
    task.setPriority(dto.priority);
    task.setDeadline(dto.deadline);
    task.setDuration(dto.duration);
    task.setUser(user);

    // Використання алгоритму для вибору тайм-слоту
    timeSlotOptimizer.optimize(task);

    Task saved = taskRepository.save(task);
    dto.id = saved.getId();
    return dto;
}

@Override
public List<TaskDTO> getTasksForUser(Long userId) {
    return taskRepository.findAllByUserId(userId).stream().map(task -> {
        TaskDTO dto = new TaskDTO();
        dto.id = task.getId();
```

```
            dto.title = task.getTitle();
            dto.priority = task.getPriority();
            dto.deadline = task.getDeadline();
            dto.duration = task.getDuration();
            dto.userId = task.getUser().getId();
            return dto;
        }).collect(Collectors.toList());
    }

    @Override
    public void deleteTask(Long taskId) {
        taskRepository.deleteById(taskId);
    }
}
```

Файл: ua.kpi.timemanager.service.impl.CalendarIntegrationServiceImpl.java

```
package ua.kpi.timemanager.service.impl;

import org.springframework.stereotype.Service;
import ua.kpi.timemanager.dto.CalendarEventDTO;
import ua.kpi.timemanager.service.CalendarIntegrationService;

import java.util.List;

// Заглушка для майбутньої інтеграції з Google Calendar API
@Service
public class CalendarIntegrationServiceImpl implements CalendarIntegrationService {

    @Override
    public List<CalendarEventDTO> fetchEventsFromCalendar(Long userId) {
```

```
        // TODO: Інтеграція з Google Calendar API
        return List.of();
    }

    @Override
    public void pushEventToCalendar(CalendarEventDTO eventDTO) {
        // TODO: Надсилання події в Google Calendar
    }
}
```

Файл: ua.kpi.timemanager.service.impl.PriorityAdjustmentServiceImpl.java

```
package ua.kpi.timemanager.service.impl;

import org.springframework.stereotype.Service;
import ua.kpi.timemanager.dto.TaskDTO;
import ua.kpi.timemanager.service.PriorityAdjustmentService;

@Service
public class PriorityAdjustmentServiceImpl implements PriorityAdjustmentService {

    @Override
    public TaskDTO adjustPriority(TaskDTO dto) {
        // Просте регулювання: якщо дедлайн скоро - підвищуємо пріоритет
        if (dto.deadline.isBefore(dto.deadline.minusDays(2))) {
            dto.priority = Math.min(dto.priority + 1, 10);
        }
        return dto;
    }
}
```

Файл: ua.kpi.timemanager.service.impl.ConflictResolverServiceImpl.java

```
package ua.kpi.timemanager.service.impl;

import org.springframework.stereotype.Service;
import ua.kpi.timemanager.dto.TaskDTO;
import ua.kpi.timemanager.service.ConflictResolverService;

@Service
public class ConflictResolverServiceImpl implements ConflictResolverService {

    @Override
    public TaskDTO resolveConflict(TaskDTO dto) {
        // Заглушка для алгоритму вирішення конфліктів з календарем
        return dto;
    }
}
```

Файл: ua.kpi.timemanager.service.impl.NotificationServiceImpl.java

```
package ua.kpi.timemanager.service.impl;

import org.springframework.stereotype.Service;
import ua.kpi.timemanager.service.NotificationService;

@Service
public class NotificationServiceImpl implements NotificationService {

    @Override
    public void sendNotification(Long userId, String message) {
        // TODO: Реалізувати інтеграцію з Firebase або email
        System.out.println("[User ID: " + userId + "] " + message);
```

```
    }
}
```

Файл: ua.kpi.timemanager.service.impl.AnalyticsServiceImpl.java

```
package ua.kpi.timemanager.service.impl;

import org.springframework.stereotype.Service;
import ua.kpi.timemanager.service.AnalyticsService;

import java.util.Map;

@Service
public class AnalyticsServiceImpl implements AnalyticsService {

    @Override
    public Map<String, Object> getUserAnalytics(Long userId) {
        // Заглушка для статистики користувача
        return Map.of(
            "tasksCompleted", 12,
            "delays", 3,
            "averageCompletionTime", "2h 30m"
        );
    }
}
```

Файл: ua.kpi.timemanager.controller.TaskController.java

```
package ua.kpi.timemanager.controller;

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
```

```
import ua.kpi.timemanager.dto.TaskDTO;
import ua.kpi.timemanager.service.TaskSchedulerService;

import java.util.List;

@RestController
@RequestMapping("/api/tasks")
public class TaskController {

    private final TaskSchedulerService taskSchedulerService;

    public TaskController(TaskSchedulerService taskSchedulerService) {
        this.taskSchedulerService = taskSchedulerService;
    }

    // Додати нову задачу
    @PostMapping("/create")
    public ResponseEntity<TaskDTO> createTask(@RequestBody TaskDTO dto) {
        return ResponseEntity.ok(taskSchedulerService.scheduleTask(dto));
    }

    // Отримати всі задачі користувача
    @GetMapping("/user/{userId}")
    public ResponseEntity<List<TaskDTO>> getTasks(@PathVariable Long userId) {
        return ResponseEntity.ok(taskSchedulerService.getTasksForUser(userId));
    }

    // Видалити задачу
    @DeleteMapping("/{taskId}")
    public ResponseEntity<Void> deleteTask(@PathVariable Long taskId) {
```

```
        taskSchedulerService.deleteTask(taskId);
        return ResponseEntity.noContent().build();
    }
}
```

Файл: ua.kpi.timemanager.controller.CalendarController.java

```
package ua.kpi.timemanager.controller;

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import ua.kpi.timemanager.dto.CalendarEventDTO;
import ua.kpi.timemanager.service.CalendarIntegrationService;

import java.util.List;

@RestController
@RequestMapping("/api/calendar")
public class CalendarController {

    private final CalendarIntegrationService calendarService;

    public CalendarController(CalendarIntegrationService calendarService) {
        this.calendarService = calendarService;
    }

    // Отримати події з календаря
    @GetMapping("/user/{userId}")
    public ResponseEntity<List<CalendarEventDTO>> fetchCalendar(@PathVariable
Long userId) {
```

```
        return ResponseEntity.ok(calendarService.fetchEventsFromCalendar(userId));
    }

    // Додати подію до календаря
    @PostMapping("/push")
    public ResponseEntity<Void> pushEvent(@RequestBody CalendarEventDTO
eventDTO) {
        calendarService.pushEventToCalendar(eventDTO);
        return ResponseEntity.ok().build();
    }
}
```

Файл: ua.kpi.timemanager.controller.NotificationController.java

```
package ua.kpi.timemanager.controller;

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import ua.kpi.timemanager.service.NotificationService;

@RestController
@RequestMapping("/api/notifications")
public class NotificationController {

    private final NotificationService notificationService;

    public NotificationController(NotificationService notificationService) {
        this.notificationService = notificationService;
    }

    // Відправити сповіщення вручну
```

```
    @PostMapping("/send")
    public ResponseEntity<Void> send(@RequestParam Long userId, @RequestParam String message) {
        notificationService.sendNotification(userId, message);
        return ResponseEntity.ok().build();
    }
}
```

Файл: ua.kpi.timemanager.config.SecurityConfig.java

```
package ua.kpi.timemanager.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .authorizeHttpRequests(authorize -> authorize
                .requestMatchers("/oauth2/**", "/login/**").permitAll()
                .anyRequest().authenticated()
            )
            .oauth2Login(); // Вбудований OAuth2 логін

        return http.build();
    }
```

```
}
```

Файл: ua.kpi.timemanager.config.GoogleOAuth2ClientProperties.java

```
package ua.kpi.timemanager.config;

import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

@Component
@ConfigurationProperties(prefix = "google")
public class GoogleOAuth2ClientProperties {

    private String clientId;
    private String clientSecret;
    private String redirectUri;

    // Геттери і сеттери
}
```

Файл: application.yml (фрагмент)

```
spring:
  security:
    oauth2:
      client:
        registration:
          google:
            client-id: YOUR_GOOGLE_CLIENT_ID
            client-secret: YOUR_GOOGLE_CLIENT_SECRET
            scope:
              - openid
```

```
            - profile
            - email
            - https://www.googleapis.com/auth/calendar
          redirect-uri: "{baseUrl}/login/oauth2/code/google"
        provider:
          google:
            authorization-uri: https://accounts.google.com/o/oauth2/v2/auth
            token-uri: https://oauth2.googleapis.com/token
            user-info-uri: https://openidconnect.googleapis.com/v1/userinfo
```

Файл: ua.kpi.timemanager.controller.AuthController.java

```
package ua.kpi.timemanager.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class AuthController {

    // Перенаправлення на Google OAuth2
    @GetMapping("/login/google")
    public String googleLoginRedirect() {
        return "redirect:/oauth2/authorization/google";
    }
}
```

Фронтенд реалізація модуля розподілу завдань із інтеграцією зовнішніх календарів

Файл: index.html

```
<!DOCTYPE html>
```

```
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <title>Особистий тайм-менеджер</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <h1>Мій розклад</h1>

    <section id="task-form-section">
        <h2>Створити нову задачу</h2>
        <form id="task-form">
            <input type="text" id="title" placeholder="Назва задачі" required>
            <input type="number" id="priority" placeholder="Пріоритет (1-10)" required>
            <input type="datetime-local" id="deadline" required>
            <input type="number" id="duration" placeholder="Тривалість (хвилини)" required>
            <button type="submit">Додати</button>
        </form>
    </section>

    <section id="tasks-section">
        <h2>Список задач</h2>
        <ul id="task-list"></ul>
    </section>

    <script src="app.js"></script>
</body>
</html>
```

Файл: style.css

```
body {
    font-family: Arial, sans-serif;
    margin: 2rem;
    background-color: #f2f2f2;
    color: #333;
}

h1, h2 {
    color: #2c3e50;
}

form input, form button {
    margin: 0.3rem;
    padding: 0.4rem;
    font-size: 1rem;
}

form {
    margin-bottom: 1rem;
}

ul {
    list-style-type: none;
    padding: 0;
}

li {
    background-color: #ffffff;
    border: 1px solid #ccc;
```

```
    margin: 0.3rem 0;
    padding: 0.6rem;
    border-radius: 5px;
}
```

Файл: app.js

```
// Отримати задачі з сервера
async function loadTasks() {
    const userId = 1; // Фіксований користувач для прикладу
    const response = await fetch(`/api/tasks/user/${userId}`);
    const tasks = await response.json();

    const taskList = document.getElementById("task-list");
    taskList.innerHTML = "";

    tasks.forEach(task => {
        const li = document.createElement("li");
        li.textContent = `${task.title} | Пріоритет: ${task.priority} | До: ${task.deadline}`;
        taskList.appendChild(li);
    });
}

// Створити нову задачу
document.getElementById("task-form").addEventListener("submit", async function (e)
{
    e.preventDefault();

    const task = {
        title: document.getElementById("title").value,
        priority: parseInt(document.getElementById("priority").value),
```

```
        deadline: document.getElementById("deadline").value,
        duration: `PT${parseInt(document.getElementById("duration").value)}M`,
        userId: 1 // Тимчасове значення
    };

    await fetch("/api/tasks/create", {
        method: "POST",
        headers: {
            "Content-Type": "application/json"
        },
        body: JSON.stringify(task)
    });

    loadTasks();
});

loadTasks();
```

ДОДАТОК Г

Графічна частин**а**

ГРАФІЧНА ЧАСТИНА

Розробка програмного застосунку для оптимізації особистого тайм-менеджменту

Рисунок Д.1 – Слайд презентації 1

Розробка програмного застосунку для оптимізації особистого тайм-менеджменту

Виконав:
студент групи 1ПІ-21б
Рудик Р.А.

Керівник:
к.т.н., доц. каф. ПЗ
Хошаба О.М.

Рисунок Д.2 – Слайд презентації 2

Метою роботи є підвищення ефективності особистого тайм-менеджменту шляхом [illegible] перепланування завдань у реальному часі [illegible] календарів і автоматичним [illegible] часом в умовах багатозадачності та динамічних змін у розкладі.

Об'єктом дослідження є процес [illegible] підвищення ефективності особистого управління [illegible]

Предметом дослідження є методи, моделі та програмні засоби динамічного [illegible] інтеграції із зовнішніми календарними сервісами [illegible] дедлайнів, вікон доступності та

Рисунок Д.3 – Слайд презентації 3

Відповідно до поставленої мети виконані наступні задачі:

- провести [illegible] порівняльну характеристику програмного забезпечення для [illegible] тайм-менеджменту [illegible] завдань із інтеграцією [illegible] через єдиний веб-інтерфейс; [illegible]
- [illegible] **модуля** інтеграції зовнішніх API через єдиний [illegible]

Рисунок Д.4 – Слайд презентації 4

Актуальність теми дослідження полягає в наступному:

У сучасному світі інформаційного перевантаження та зростаючих темпів життя ефективне управління особистим часом стало критично важливим аспектом професійного та особистісного успіху.

Тому, особливої актуальності набуває розробка цифрових рішень, які поєднують ці методи із сучасними сервісами календарного планування, такими як Google Calendar або Outlook. Це дозволяє автоматизувати розподіл завдань, враховуючи пріоритети, дедлайни, зайняті інтервали часу, а також індивідуальні особливості користувача.

Таким чином, розроблений програмний засіб забезпечує персоналізоване, гнучке, адаптивне управління особистим часом, що є вагомим внеском у розвиток сучасних цифрових інструментів тайм-менеджменту.

Рисунок Д.5 – Слайд презентації 5
UML-діаграма послідовності візуалізації обміну повідомленнями між користувачем, застосунком, Google Calendar API і Trello API

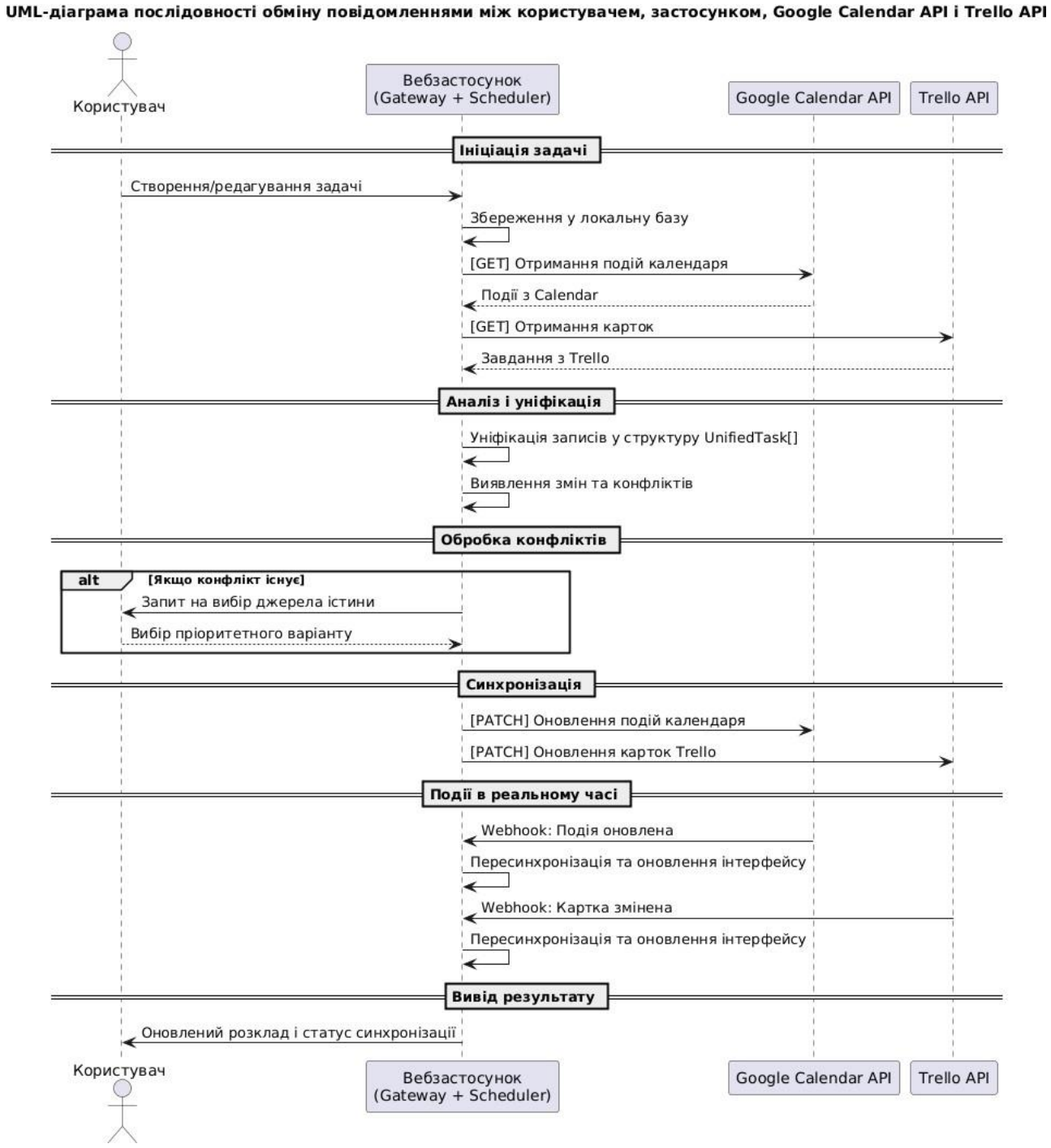

Рисунок Д.6 – Слайд презентації 8
UML-діаграми активностей, що ілюструє основні етапи алгоритму динамічного розподілу завдань з інтеграцією зовнішніх календарів

Рисунок Д.7 – Слайд презентації 6
Призначення модулів підсистеми інтеграції зовнішніх API через єдиний вебінтерфейс

Назва модуля	Призначення	API, що використовується
GoogleCalendarIntegration Module	Імпорт/експорт подій з Google Calendar	https://www.googleapis.com/calendar/v3
TrelloTaskIntegration-Module	Синхронізація карточокTrello з локальними задачами	https://api.trello.com/1
UnifiedWebInterface-Module	Відображення даних з різних сервісів у єдиному UI	Локальний API
OAuthTokenManager-Module	Отримання, збереження та оновлення токенів доступу	OAuth 2.0 Endpoints
SyncSchedulerModule	Планування періодичної синхронізації між сервісами	Cron-таймери

Рисунок Д.8 – Слайд презентації 7
UML-діаграма компонентів інтеграції зовнішніх API

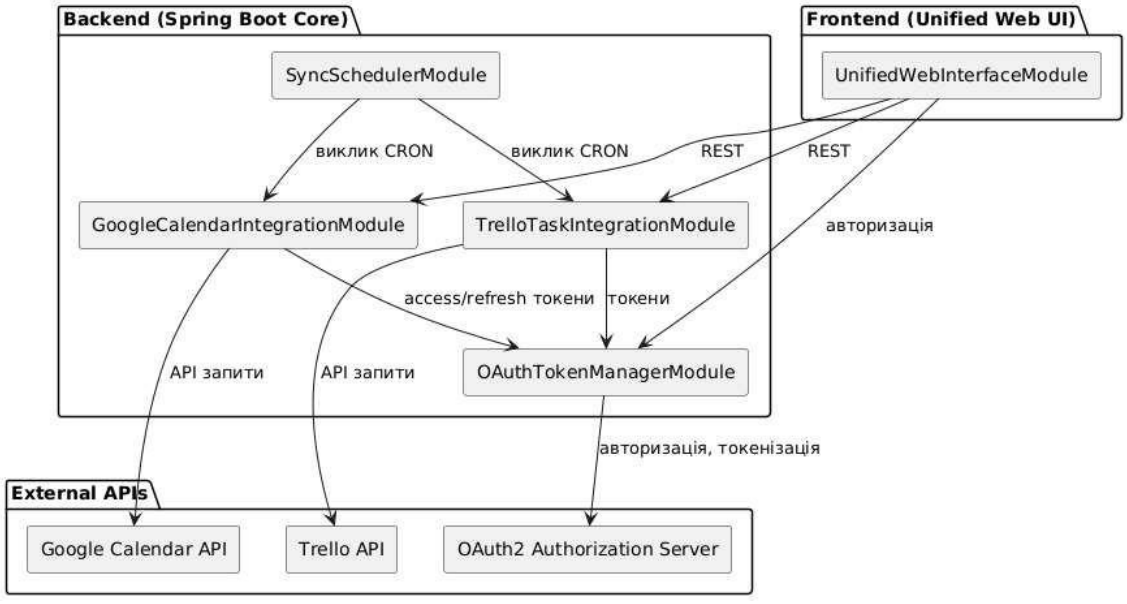

Рисунок Д.9 – Слайд презентації 9
Скріншот тестування розробленого календаря з авторизації користувача через механізми OAuth2

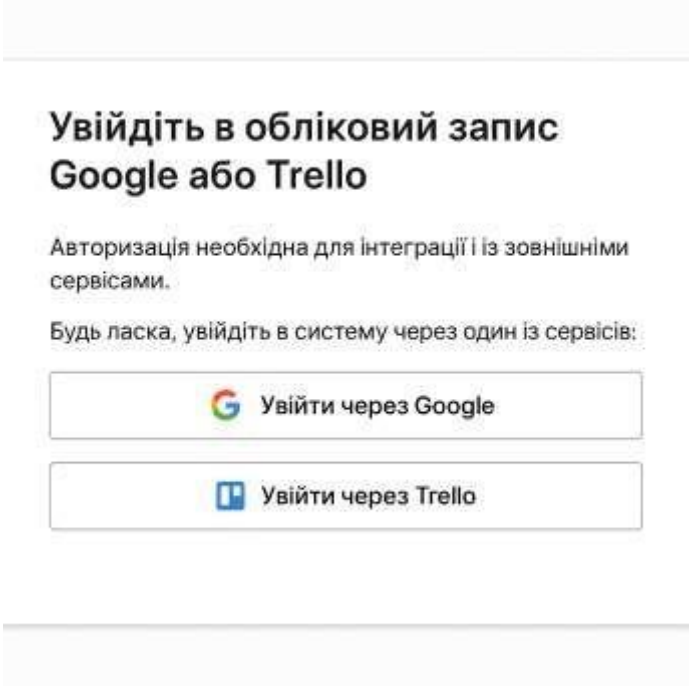

Рисунок Д.10 – Слайд презентації 10
Скріншот тестування розробленого календаря, що надає результати представлення зручного, структурованого та інформативного інтерфейсу користувача

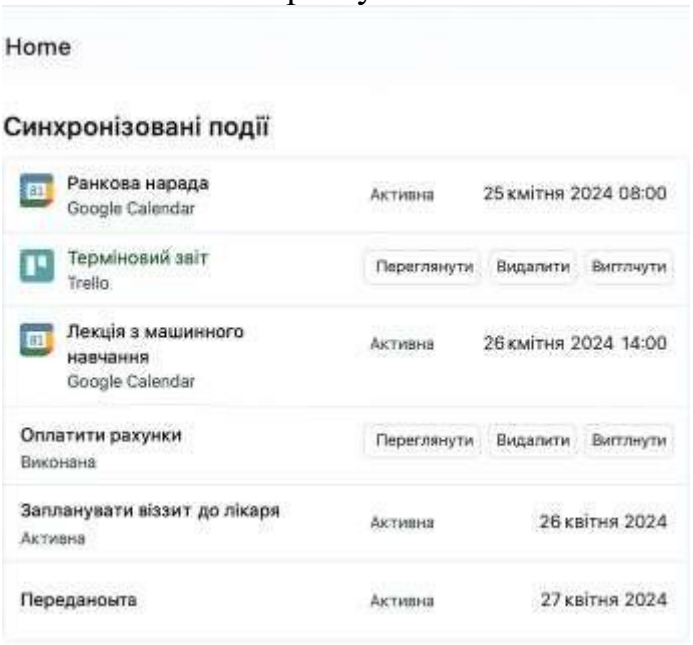

Рисунок Д.11 – Слайд презентації 11
Новизна отриманих результатів:

1. Запропоновано метод динамічного розподілу завдань із інтеграцією зовнішніх календарів, особливість якого полягає в автоматичній корекції розкладу на основі пріоритетності та дедлайнів, що дає можливість адаптуватися до змін у реальному часі та скоротити час на ручне планування до 27%.

2. Запропоновано метод інтеграції зовнішніх API з використанням Google Calendar, Trello через єдиний вебінтерфейс, що дозволяє синхронізувати дані між різними сервісами безпечно та в режимі реального часу за рахунок використання модульної вебсервісної архітектури та токенізації доступу.

3. Подальшого розвитку отримав метод автоматичного планування, у якому, на відміну від існуючих, реалізовано адаптивні рекомендації на основі машинного аналізу історії виконаних задач, що дозволило зменшити середній час на складання щоденного плану користувача більш ніж на 34%.

Рисунок Д.12 – Слайд презентації 12
Висновки:

У бакалаврській кваліфікаційній роботі:

еграції зовнішніх API через єдиний