

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Кафедра інформаційно-обчислювальних систем і управління

ПАСІЧНИК Максим Михайлович

**Програмна система відстеження діяльності користувача за
комп'ютером з наданням рекомендацій з тайм-менеджменту / A
software system for tracking user activity at the computer with
recommendations for time management**

Спеціальність 122 – Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

Кваліфікаційна робота

Виконав студент групи КН-41
М. М. Пасічник

Науковий керівник:
к.т.н., доцент П.Є. Биковий

Кваліфікаційну роботу допущено до
захисту

«___»_____ 2025 р.

В.о. завідувача кафедри
_____ Н.М. Васильків

Тернопіль – 2025

Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління
Освітній ступінь «бакалавр»
спеціальність 122 – Комп'ютерні науки
освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
Н.М. Васильків
« ____ » _____ 2024 р.

**ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
ПАСІЧНИК Максим Михайлович**
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: Програмна система відстеження діяльності користувача за комп'ютером з наданням рекомендацій з тайм-менеджменту / Software system for tracking user activity on a computer with time management recommendations

керівник роботи Биковий П.Є. к.т.н., доцент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від 20 грудня 2024 р. № 938.

2. Строк подання студентом закінченої кваліфікаційної роботи 25 травня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.

4. Основні питання, які потрібно розробити:

– проаналізувати існуючі рішення для відстеження часу користувача за комп'ютером, визначити їх переваги та обмеження;

– сформулювати концепцію побудови застосунку, який дозволить користувачеві відслідковувати проведений час за ПК, оптимізувати його та консультуватись зі інтелектуальним асистентом;

– розробити архітектуру застосунку, яка буде поєднувати продуктивність, стабільність, сучасний дизайн та можливість портування на інші системи;

– реалізувати застосунок для відстеження часу за ПК;

– протестувати застосунок в різних сценаріях взаємодії користувача з ним.

5. Перелік графічного матеріалу в роботі:

– структурна схема застосунку;

– модульна організація системи;

– схема алгоритму роботи застосунку.

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 20 грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
	Затвердження теми кваліфікаційної роботи, ознайомлення з літературними джерелами та складання плану роботи	до 01.01. 2025 р.	
	Написання 1 розділу кваліфікаційної роботи	до 01.02. 2025 р.	
	Написання 2 розділу кваліфікаційної роботи	до 01.04.2025 р.	
	Написання 3 розділу кваліфікаційної роботи	до 01.05. 2025 р.	
	Представлення попереднього варіанту кваліфікаційної роботи, перевірка та внесення змін керівником	до 15.05.2025 р.	
	Опрацювання зауважень та представлення завершеного варіанту кваліфікаційної роботи. Підготовка супроводжуючих документів	до 25.05.2025 р.	
	Перевірка кваліфікаційної роботи на оригінальність тексту	до 30.05.2025 р.	
	Оформлення кваліфікаційної роботи та отримання допуску до захисту	до 16.06.2025 р.	
	Подання кваліфікаційної роботи до захисту на засіданні атестаційної комісії	до 16.06. 2025 р.	

Студент _____ Пасічник М.М.
(підпис) (прізвище та ініціали)

Керівник кваліфікаційної роботи _____ Биковий П.Є.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота на тему «Програмна система відстеження діяльності користувача за комп'ютером з наданням рекомендацій з тайм-менеджменту» на здобуття освітнього ступеня «бакалавр» зі спеціальності 122 «Комп'ютерні науки» освітньої програми «Комп'ютерні науки» написана обсягом в 74 сторінок і містить 28 ілюстрацій, 11 таблиць, 8 додатків та 27 використаних джерел.

Метою кваліфікаційної роботи є створення додатку для відстеження часу проведеного користувачем за комп'ютером, та наданням рекомендацій з оптимізації проведеного часу за комп'ютером за допомогою ШІ-асистента.

Об'єктом дослідження є процеси моніторингу та аналізу взаємодії користувача з програмним забезпеченням комп'ютера.

Предметом дослідження є система відстеження активності користувача з функціями збору статистики, виявлення неактивності та генерації аналітичних звітів.

Розроблено програмне забезпечення для відстеження часу використання програм на комп'ютері, яке дозволяє автоматично збирати дані про активність користувача, розпізнавати періоди неактивності, генерувати детальну статистику використання програм, консультуватись з ШІ-асистентом та експортувати результати у зручному форматі.

Ключові слова: ВІДСТЕЖЕННЯ АКТИВНОСТІ, АНАЛІЗ ВИКОРИСТАННЯ ЧАСУ, ВИЯВЛЕННЯ НЕАКТИВНОСТІ, ВІЗУАЛІЗАЦІЯ ДАНИХ, PYTHON, АВТОМАТИЗАЦІЯ, ГЕНЕРАЦІЯ ЗВІТІВ.

ANNOTATION

The qualification work on the topic “ Software system for tracking user activity on a computer with time management recommendations ” for the bachelor's degree in specialty 122 “Computer Science” of the educational program “Computer Science” is written in 74 pages and contains 28 illustrations, 11 tables, 8 applications and 27 references.

The purpose of the qualification work is to create an application for tracking the time spent by the user at the computer and to provide recommendations for optimizing the time spent at the computer using an AI assistant.

The object of research is the processes of monitoring and analyzing user interaction with computer software.

The subject of the study is a user activity tracking system with the functions of collecting statistics, detecting inactivity, and generating analytical reports.

Software has been developed to track the time spent using programs on a computer, which allows you to automatically collect data on user activity, recognize periods of inactivity, generate detailed statistics on program use, and export the results in a convenient format.

Keywords: ACTIVITY TRACKING, TIME USAGE ANALYSIS, INACTIVITY DETECTION, DATA VISUALIZATION, PYTHON, AUTOMATION, REPORT GENERATION.

ЗМІСТ

Вступ.....	7
1 Аналіз предметної області.....	9
1.1 Опис предметної області.....	9
1.2 Огляд і аналіз існуючих рішень	13
1.3 Постановка задачі дослідження	18
2 Алгоритмічне та інформаційне забезпечення.....	21
2.1 Загальна структура проектованої системи.....	21
2.2 Схема алгоритму роботи застосунку BeBetter.....	23
2.3 Обґрунтування вибору технічних інструментів.....	25
2.4 Інформаційне забезпечення.....	30
3 Програмно-технологічне забезпечення прикладного програмного інтерфейсу	33
3.1 Реалізація програмного забезпечення	33
3.2 Інтерфейс користувача.....	40
3.3 Тестування розробленої програми.....	46
Висновки	54
Список використаних джерел	56
Додаток А Код для перевірки даних з БД.....	59
Додаток Б Код для отримання активного вікна.....	60
Додаток В Код для відстеження AFK.....	61
Додаток Г Код для взаємодії зі інтелектуальним асистентом.....	64
Додаток Д Інтерфейси користувача «TogglTrack» та «Timing».....	65
Додаток Е Приклад даних з БД у форматі JSON.....	66
Додаток Ж Копії опублікованих результатів.....	69

ВСТУП

У сучасному світі цифрових технологій усе більше людей пов'язують своє життя з комп'ютерами, дистанційною роботою та фрілансом [1]. З одного боку, це забезпечує гнучкість та комфорт, а з іншого — створює нові виклики, пов'язані із самодисципліною, прокрастинацією, розмитими межами між роботою та відпочинком і неконтрольованим витрачанням часу [2]. У таких умовах особливого значення набувають інструменти для відстеження цифрової активності, які дозволяють користувачеві краще зрозуміти власну продуктивність та підвищити ефективність використання робочого часу.

На ринку існує багато рішень, які частково покривають цю потребу, однак більшість з них або надто громіздкі, або потребують сторонніх вебсервісів, або ж фокусуються на бізнес-користувачах із централізованим керуванням.

У цьому контексті виникла ідея створення BeBetter — десктопного застосунку, який дозволяє користувачеві відслідковувати, як саме він проводить свій час за комп'ютером. Проєкт розроблявся з фокусом на macOS, з подальшою можливістю переносу на інші ОС. В основі BeBetter лежить зв'язка PyQt6 (для зручного та гнучкого графічного інтерфейсу) та Python (для відстеження активності, роботи з базою даних, обчислення статистики та серверної логіки). Такий підхід дозволяє поєднати UI із потужною внутрішньою логікою, яку легко розширювати, адаптувати й масштабувати.

У процесі реалізації проведено аналіз аналогічних інструментів, визначено їхні сильні та слабкі сторони, після чого сформульовано технічні й функціональні вимоги до BeBetter. Проєкт також став прикладом практичного поєднання вебтехнологій, нативної розробки, роботи з БД, API, UI/UX та кросплатформеності, що робить його цінним як з прикладної, так і з дослідницької точки зору. Для досягнення необхідної мети необхідно виконати наступні завдання:

- 1) проаналізувати існуючі рішення для відстеження часу користувача за комп'ютером, визначити їх переваги та обмеження;

2) сформувати концепцію побудови застосунку, який дозволить користувачеві відслідковувати проведений час за ПК, оптимізовувати його та консультуватись з мовною моделлю;

3) розробити архітектуру застосунку, яка буде поєднувати продуктивність, стабільність, сучасний дизайн та можливість портування на інші системи;

4) реалізувати застосунок для відстеження часу за ПК;

5) протестувати застосунок в різних сценаріях взаємодії користувача з ним.

Актуальність роботи обумовлена потребою в ефективних інструментах самоменеджменту, особливо в умовах гібридного або повністю віддаленого способу роботи.

Об'єктом дослідження є технології відстеження, збору та аналізу активності користувача на персональному комп'ютері.

Предметною областю виступає система моніторингу цифрової поведінки з використанням десктопного трекера, інтерфейсу користувача та серверної логіки для обробки даних у реальному часі.

Метою є аналіз сучасних існуючих рішень для відстеження та оптимізації проведеного за комп'ютером часу визначення їх переваг та обмежень, та розробка десктопного застосунку, що буде зручним та простим в повсякденному користуванні, визначатиме неактивність користувача, дозволить аналізувати проведений час.

Результати дослідження опубліковано в матеріалах науково-практичної конференції «Інтелектуальні інформаційні технології в прикладних дослідженнях» (ІІТАР – 2025), м. Тернопіль, 27–29 травня 2025 р. (додаток Ж).

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

У сучасних умовах цифровізації та зростання кількості дистанційної праці, особливо у сфері інформаційних технологій [27], важливим аспектом ефективної організації діяльності користувача за персональним комп'ютером є моніторинг цифрової активності. Усе більше людей стикаються з проблемою нераціонального використання робочого часу, розмитості меж між роботою та відпочинком, а також відсутністю прозорої системи контролю над власною продуктивністю [27].

Покоління Z, до якого належать особи, народжені після середини 1990-х до початку 2010-х років, виросло в епоху стрімкого розвитку цифрових технологій. Це покоління відоме своєю високою адаптивністю до нових технологій та активним використанням цифрових пристроїв у повсякденному житті. Однак, постійна присутність у цифровому середовищі може призводити до проблем з концентрацією уваги та схильності до прокрастинації [26].

Дослідження показують, що представники покоління Z витрачають значну кількість часу на соціальні мережі та інші розважальні онлайн-ресурси, що може негативно впливати на їхню продуктивність. За даними дослідження, проведеного Pew Research Center, 55% підлітків зазначили, що проводять занадто багато часу онлайн, а 41% визнають, що надмірне використання соціальних мереж відволікає їх від важливих справ [6].

Порівняно з попередніми поколіннями, такими як покоління X (народжені між 1965 та 1980 роками) та міленіали (народжені між 1981 та 1996 роками), покоління Z демонструє вищий рівень залежності від цифрових пристроїв [24]. Міленіали також активно використовують технології, але вони частіше намагаються встановлювати межі між роботою та особистим життям. Покоління X, яке виросло до ери інтернету, зазвичай менш схильне до постійного використання цифрових пристроїв і частіше віддає перевагу традиційним методам комунікації та роботи.

У сучасному суспільстві спостерігається зростання інтересу до підвищення особистої продуктивності, що призвело до популяризації різноманітних інструментів для планування та організації. Це явище іноді називають «культу продуктивності», коли прагнення до ефективності стає основною ціллю, а використання відповідних додатків перетворюється на окремий аспект життя [7].

Однак, поряд із зростанням популярності таких інструментів, виникає феномен, коли користувачі витрачають надмірну кількість часу на налаштування та оптимізацію своїх систем продуктивності, що може призводити до зниження реальної ефективності. Цей парадокс продуктивності полягає в тому, що прагнення до ідеальної організації може відволікати від виконання основних завдань.

Крім того, постійне прагнення до підвищення продуктивності може негативно впливати на психічне здоров'я. Люди можуть відчувати тривогу або почуття провини через невиконання запланованих завдань або недостатню ефективність, що, у свою чергу, може призводити до вигорання.

Зростання популярності віддаленої роботи та гнучких графіків підкреслює необхідність впровадження інструментів для моніторингу цифрової активності. Це дозволяє не лише оцінити ефективність працівників, а й виявити області для покращення та оптимізації робочих процесів. Згідно з дослідженням, проведеним компанією CleverControl, використання програмного забезпечення для моніторингу, такого як CleverControl, сприяє підвищенню продуктивності та дисципліни серед віддалених співробітників [3].

В традиційних методів обліку робочого часу є кілька проблем. Зазвичай вони базуються на ручному введенні даних, є недостатньо точними, трудомісткими та незручними у тривалому використанні. Це може призводити до неточностей та суб'єктивності в оцінці продуктивності. Якщо розглядати методи обліку часу з ручним введенням, працівник може просто маніпулювати даними, адже в цьому випадку все залежить лише від нього та його етики. Впровадження автоматизованих систем моніторингу активності користувача

дозволяє отримувати об'єктивні дані про використання робочого часу, що сприяє більш точному плануванню та управлінню часом.

Автоматизовані системи моніторингу активності користувачів надають можливість детально аналізувати, як саме витрачається робочий час. Це дозволяє виявляти неефективні процеси та оптимізувати робочі завдання. Крім того, такі системи забезпечують прозорість та об'єктивність в оцінці продуктивності, що є важливим як для звичайних користувачів, так і для корпоративного середовища.

Попри переваги, впровадження систем моніторингу активності користувачів може стикатися з певними викликами. Серед них — забезпечення конфіденційності даних, етичні аспекти спостереження за працівниками та технічні труднощі інтеграції таких систем у існуючу інфраструктуру.

Система BeBetter призначена для відстеження часу, проведеного за комп'ютером, та аналізу активності користувача. Основні процеси, які виконує система:

- відстеження активних вікон – система автоматично фіксує, які програми використовує користувач;
- виявлення періодів неактивності – BeBetter розпізнає, коли користувач відсутній за комп'ютером (AFK - Away From Keyboard);
- збір статистики – система збирає дані про використання програм за день, тиждень та місяць;
- візуалізація даних – зібрані дані представляються у зручному для аналізу вигляді через веб-інтерфейс;
- аналіз продуктивності – за допомогою інтеграції з ІІІ система може аналізувати продуктивність та надавати рекомендації.

Незважаючи на значні досягнення в області відстеження часу та продуктивності, автоматичні системи все ще стикаються з рядом викликів та проблем:

- конфіденційність даних – необхідність захисту особистої інформації користувача;

- точність визначення активності – складність розрізнення активної роботи та простого відкриття програми;
- класифікація програм – визначення, які програми є продуктивними, а які відволікаючими;
- інтеграція з різними операційними системами – забезпечення коректної роботи на різних платформах;
- аналіз та інтерпретація даних – перетворення зібраних даних у корисні поради та рекомендації.

У розробці системи BeBetter виділяються транзакційна та аналітична складові. Транзакційна охоплює відстеження активності користувача: фіксацію активних вікон, визначення періодів неактивності, збереження даних у базі. Дані про використання програм та часові проміжки зберігаються в базі даних SQLite через ORM Peewee. Аналітична складова дозволяє аналізувати кількість часу, проведеного за різними програмами, виявляти паттерни використання та надавати рекомендації щодо підвищення продуктивності. Інтеграція з OpenAI API дозволяє використовувати можливості штучного інтелекту для глибшого аналізу поведінки користувача та надання персоналізованих рекомендацій.

Інтеграція транзакційної та аналітичної складових у процеси дозволяє ефективно відстежувати використання часу та вдосконалювати продуктивність на основі аналізу даних, підвищуючи ефективність роботи користувача. Система BeBetter є потужним інструментом для підвищення продуктивності та самодисципліни, що допомагає користувачам краще розуміти свої звички використання комп'ютера та оптимізувати робочі процеси. Використання технологій штучного інтелекту для аналізу зібраних даних відкриває нові можливості для персоналізованих рекомендацій та автоматизації процесу підвищення продуктивності.

1.2 Огляд і аналіз існуючих рішень

У межах дослідження ефективності контролю цифрової активності користувачів було здійснено аналіз програмних продуктів, що реалізують функціональність обліку часу, автоматичного трекінгу додатків, відстеження періодів неактивності, генерації звітності, а також аналізу цифрових звичок користувача. Основна увага була зосереджена на вивченні технічних підходів, реалізованих у подібних системах, оцінці способів збору, обробки та візуалізації даних, а також виявленні переваг і недоліків реалізацій, що вже існують на ринку.

Зокрема, розглянуто такі продукти, як RescueTime [18], TogglTrack [19], Timing, ManicTime [20], ActivityWatch [17], а також Clockify [13]. Деякі з них використовують переважно ручний облік активностей, інші — повністю автоматизовані рішення. Принципові відмінності полягають у способі інтеграції до операційної системи, методах визначення активного вікна, механізмах класифікації додатків та підходах до побудови аналітики.

Для порівняння основних рішень було складено узагальнену таблицю 1.1

Таблиця 1.1 – Порівняння існуючих рішень

Назва продукту	Тип трекінгу	Спосіб встановлення	Локальне зберігання	Візуалізація статистики	Доступ до даних	Платформа
RescueTime	Автоматичний	Власне ПЗ + веб	Ні	Через сайт	Хмарне сховище	Windows, macOS, Linux
Toggl Track	Переважно ручний	Власне ПЗ + розширення	Частково	Через сайт + моб. додатки	Хмарне сховище	Yci OC
Timing	Автоматичний	Додаток macOS	Так	У додатку	Локально	macOS
ActivityWatch	Автоматичний	Відкритий код	Так	У веб-інтерфейсі локально	Локально	Windows, macOS, Linux
Clockify	Ручний	Веб/мобільний/десктоп	Ні	Через сайт	Хмарне сховище	Yci OC
ManicTime	Автоматичний	Десктопна програма	Так	У додатку	Локально	Windows, macOS

Аналіз засвідчив, що переважна більшість трекерів поділяє логіку на дві частини: фонову службу, яка збирає інформацію про активність користувача, та веб-інтерфейс або мобільний додаток, який використовується для перегляду статистики. Подібний підхід створює певні обмеження у швидкості отримання

зворотного зв'язку, оскільки для перегляду результатів активності користувач змушений відкривати окремий додаток або вебсайт.

Окрім цього, під час аналізу вихідного коду відкритих аналогів, зокрема ActivityWatch, було досліджено методи отримання даних про активне вікно. У більшості випадків використовується прив'язка до API операційної системи, наприклад, NSWorkspace на macOS або win32gui на Windows, що дозволяє зчитувати заголовки вікон, назви процесів та ідентифікатори програм. Частина рішень використовує додаткову інформацію з системного журналу (наприклад, System Events на macOS) для уточнення типу діяльності.

Деякі системи також використовують розширення для браузера, які передають URL-адреси та заголовки відкритих вкладок, що дозволяє точніше класифікувати активність користувача у мережі. Проте така інтеграція є обмеженою і не завжди працює стабільно на всіх платформах, особливо за умов строгої конфіденційності або обмежених прав доступу.

Варто відзначити, що більшість сучасних трекерів передбачає хмарну синхронізацію даних, що потребує постійного доступу до інтернету, і може створювати ризики для приватності. Лише деякі рішення, як ActivityWatch або ManicTime, орієнтовані на локальне зберігання та повну автономність.

Один з найпопулярніших інструментів фрилансера – «TopTracker» від компанії Tortal [11,12]. Він активно використовується не тільки у фрилансі, але й в більших компаніях, де практикується погодинна оплата праці. Додаток являє з себе невеликий застосунок з ручними відстеженням часу. В ньому можна змінювати облікові засоби, перемикатись між проєктами, а також є мінімальні налаштування. Особливої уваги заслуговують такі пункти налаштування, як «Take screenshots» та «Take camerashots». Клієнт (роботодавець) в налаштуваннях проєкту може обрати пункт, завдяки якому з певним інтервалом робитимуться знімки екрану або фото з вебкамери працівника. Зроблено це для того, щоб у будь-який момент можна було слідкувати за роботою працівника, та перевіряти, чи дійсно він працює над завданням.

Це робиться з дозволу працівника, але саме через ці дві функції, багато фрилансерів намагаються уникати роботи з TopTracker.

На рисунку 1.1 зображений інтерфейс додатку на macOS:

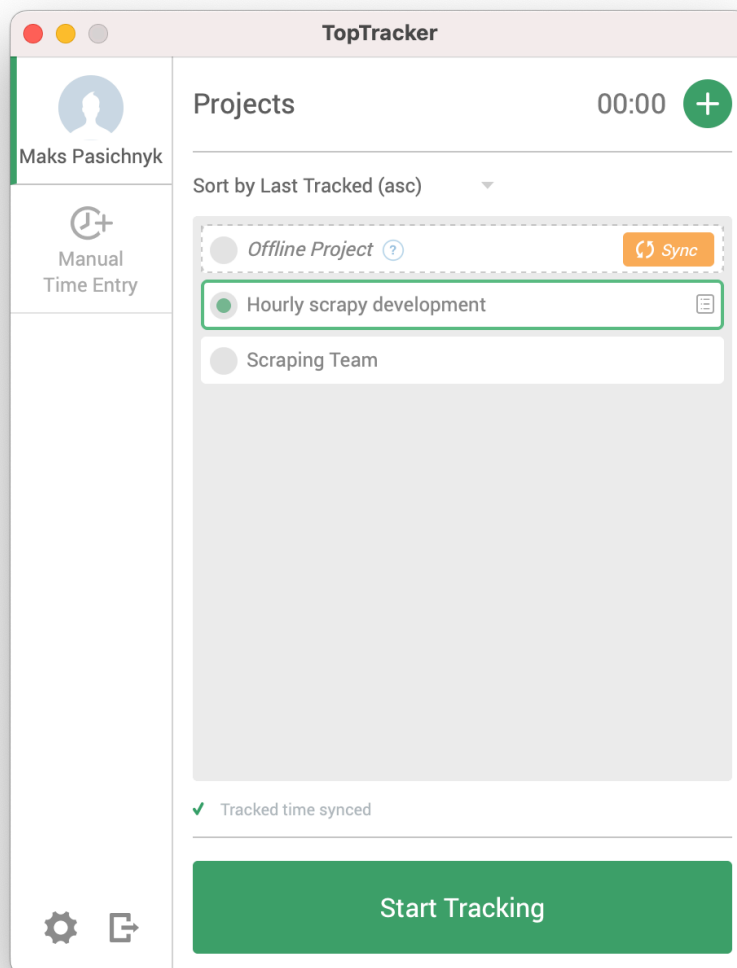


Рисунок 1.1 – Інтерфейс TopTracker [14]

Детальна статистика доступна в особистому кабінеті користувача на вебсайті застосунку. І саме завдяки своєму вебсайту, можна зрозуміти, чому TopTracker – одне з найпопулярніших рішень.

На рисунку 1.2 зображено перелік налаштувань, які пропонує додаток. В налаштуваннях проєкту, роботодавець виставляє погодинну оплату, наперед домовившись з працівником, і на сайті можна детально проаналізувати, скільки і у які дні годин було напрацьовано, і скільки зароблено. Також, можна зразу

надіслати invoice. Invoice – документ, що надається продавцем покупцеві, та містить перелік товарів і послуг, їхню кількість та суму коштів, яку покупець має за них сплатити. Це дуже спрощує процес оплати праці, та зменшує кількість роботи з документами.

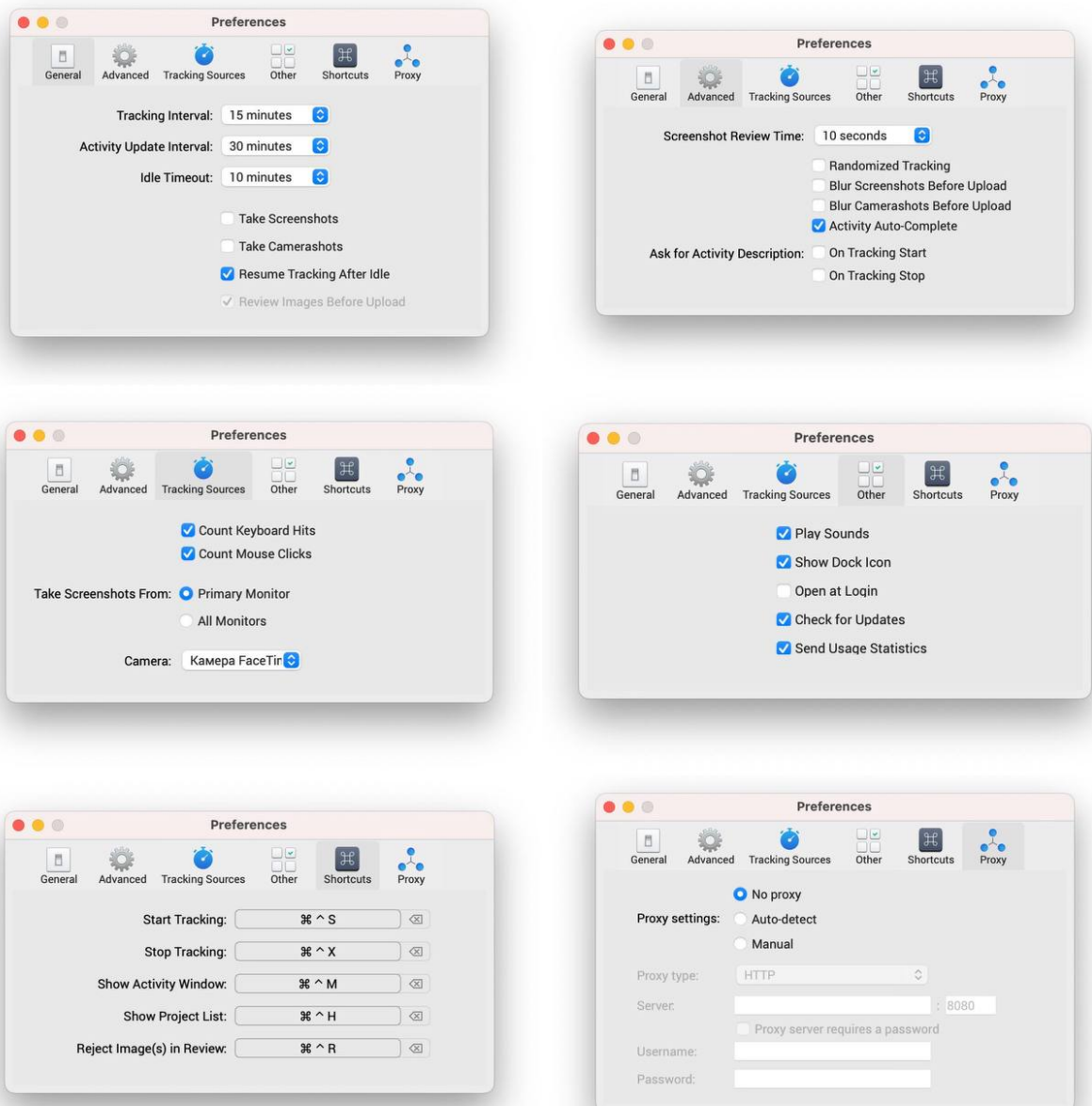


Рисунок 1.2 – Перелік налаштувань TopTracker [14]

На рисунку 1.3 зображена домашня сторінка вебсайту TopTracker. Де відображається основна інформація. Зверху назва поточного проєкту, загальна сума заробітку працівника, загальний витрачений час на роботу.

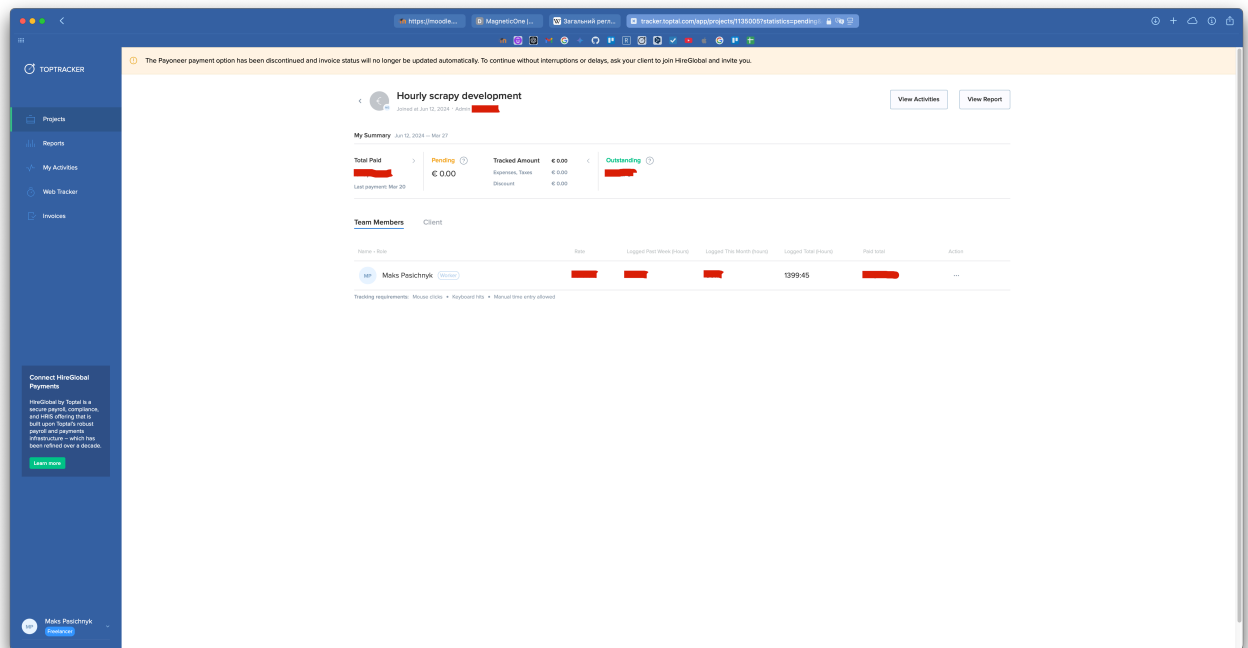


Рисунок 1.3 – Домашня сторінка TopTracker [14]

На рисунку 1.4 зображена сторінка звітів, де можна отримати детальну інформацію за певний проміжок дат, і можна дізнатись, скільки в який день напрацьовано та зароблено, а також порівняти цю інформацію з іншими днями, тижнями, місяцями.

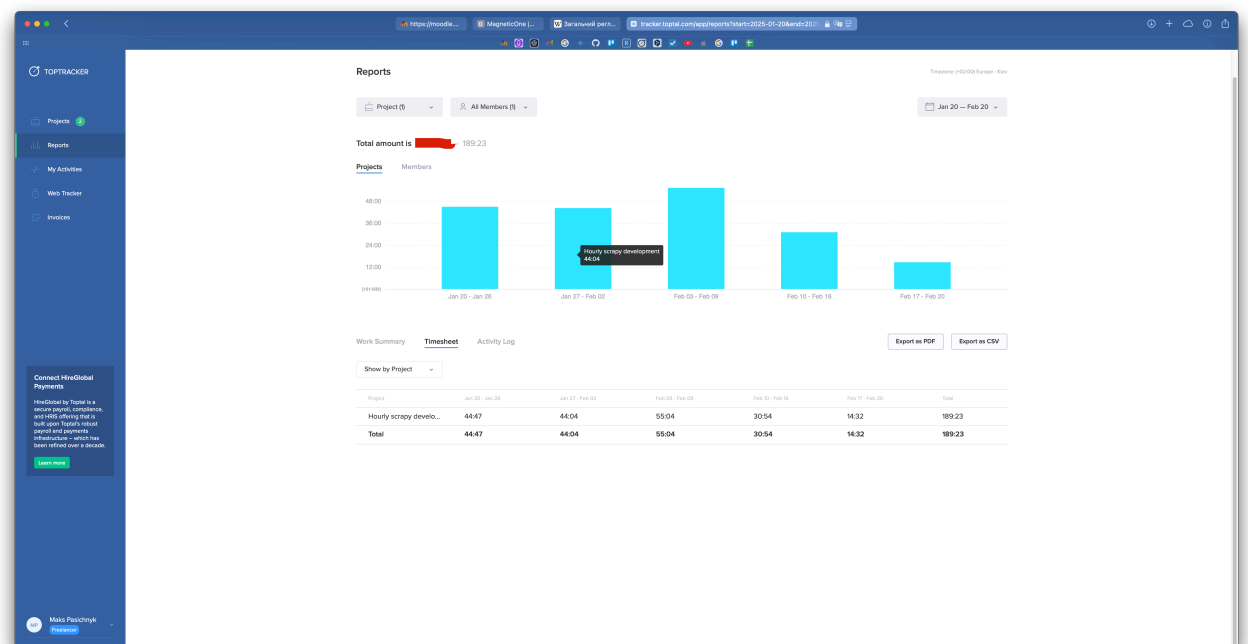


Рисунок 1.4 – Сторінка звітів [14]

Загалом, TopTracker – досить потужний та зручний інструмент як для працівника, так і для клієнта. Проте можливість клієнта в будь-який момент бачити екран та веб-камеру працівника відлякує багато фрилансерів [8][9]. До сильних переваг можна віднести можливість формування та відправки інвойсів, а також наявність детальної звітності.

У межах цього дослідження було зроблено висновок, що сучасні системи або не забезпечують достатньої гнучкості в налаштуванні, або ж розділяють трекінг і аналіз у різні додатки, що ускладнює повсякденне використання. Саме це зумовило вибір підходу, при якому і збір даних, і взаємодія з користувачем відбувається через єдиний десктопний додаток.

Розробка проєкту BeBetter передбачає реалізацію єдиної системи, що базується на поєднанні Python та HTML, що дозволяє створити кросплатформне рішення з локальним зберіганням інформації, високим ступенем інтеграції з ОС та миттєвим доступом до статистики без потреби у зовнішніх сервісах.

1.3 Постановка задачі дослідження

На основі проведеного аналізу у попередніх підрозділах було визначено низку обмежень та недоліків, які створюють незручності для користувачів та ускладнюють процес ефективного управління та відстеження робочого часу. Серед основних проблем, характерних для існуючих систем відстеження часу, можна виділити:

- а) обмежений функціонал безкоштовних версій (ліміт часу відстеження, неможливість отримання детальної статистики без оплати, обмеження на кількість програм);
- б) надмірна складність інтерфейсу, що вимагає значного часу на освоєння та налаштування системи (додаток Д);
- в) погіршення продуктивності системи під час відстеження, особливо у системах з обмеженими ресурсами;

г) відсутність зручних засобів для автоматичної інтеграції зібраних даних у власні процеси планування та аналізу продуктивності;

г) недостатня гнучкість у налаштуванні параметрів відстеження та формату вихідних даних;

д) відсутність механізмів виявлення періодів неактивності та їх аналізу.

З урахуванням зазначених проблем сформовано загальну концепцію проєктованої кваліфікаційної роботи – створення системи відстеження часу BeBetter, яка забезпечуватиме автоматичне відстеження активних програм, виявлення періодів неактивності та аналіз продуктивності використання часу. Важливою особливістю системи BeBetter стане її інтеграційна спрямованість, що дозволить користувачам легко аналізувати зібрані дані та використовувати їх для оптимізації власних робочих процесів. Вимоги до функціональності системи BeBetter:

- автоматичне відстеження активних вікон програм з мінімальним впливом на продуктивність системи;
- виявлення періодів неактивності користувача з можливістю вказати причину;
- збір та зберігання статистики використання програм за різні періоди (день, тиждень, місяць);
- візуалізація зібраних даних через зручний веб-інтерфейс;
- можливість експорту даних для подальшого аналізу;
- інтеграція з API штучного інтелекту для аналізу продуктивності.

Метою дослідження є розробка системи BeBetter – створення зручного та ефективного інструменту для автоматичного відстеження часу, який допоможе користувачам підвищити продуктивність, виявити непродуктивні звички та оптимізувати розподіл робочого часу. Особлива увага приділяється зручному та зрозумілому інтерфейсу, а також продуктивності та ефективності застосунку для його стабільної роботи на протязі робочого дня.

Для досягнення поставленої мети необхідно розв'язати такі задачі:

- проаналізувати існуючі рішення для відстеження часу користувача за

комп'ютером, визначити їх переваги та обмеження;

- сформулювати концепцію побудови застосунку, який дозволить користувачеві відслідковувати проведений час за ПК, оптимізувати його та консультуватись з мовною моделлю;

- розробити архітектуру застосунку, яка буде поєднувати продуктивність, стабільність, сучасний дизайн та можливість портування на інші системи;

- реалізувати застосунок для відстеження часу за ПК;

- протестувати застосунок в різних сценаріях взаємодії користувача з ним.

2 АЛГОРИТМІЧНЕ ТА ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Загальна структура проекрованої системи

Система BeBetter складається з десктопного додатку на PyQt6 та внутрішнього веб-інтерфейсу для відображення статистики використання часу. Система автоматично відстежує активні вікна програм, виявляє періоди неактивності користувача та генерує статистику використання часу. Основні функції системи BeBetter:

- відстеження активних вікон програм;
- виявлення періодів неактивності користувача (AFK);
- збір та аналіз статистики використання часу;
- візуалізація статистики через веб-інтерфейс;
- інтеграція з API штучного інтелекту для аналізу продуктивності.

Основними компонентами системи є:

- а) модуль відстеження часу (`time_tracker.py`) – відстеження активних вікон, виявлення неактивності;
- б) веб-інтерфейс (`web_view.py`) – візуалізація статистики використання часу;
- в) системний трей (`tray_icon.py`) – швидкий доступ до функцій програми;
- г) БД;
- д) конфігураційний файл для зміни налаштувань.

На рисунку 2.1 зображена структурна схема застосунку. Вона показує систему BeBetter для відстеження часу, поділену на функціональні блоки. Користувач працює з системним треєм та веб-інтерфейсом на PyQt6 для відстеження часу, перегляду статистики, налаштування параметрів і взаємодії з інтелектуальним асистентом. Клієнтська частина включає системний трей для швидкого доступу до функцій програми та веб-інтерфейс для візуалізації даних. Бекенд-частина складається з компонентів відстеження (модулі відстеження активного вікна та активності користувача), сервісного рівня (оркестратор процесів, аналіз даних, збір даних активності,

звіти) та інфраструктурного рівня (файлові операції, логування, обробка помилок).

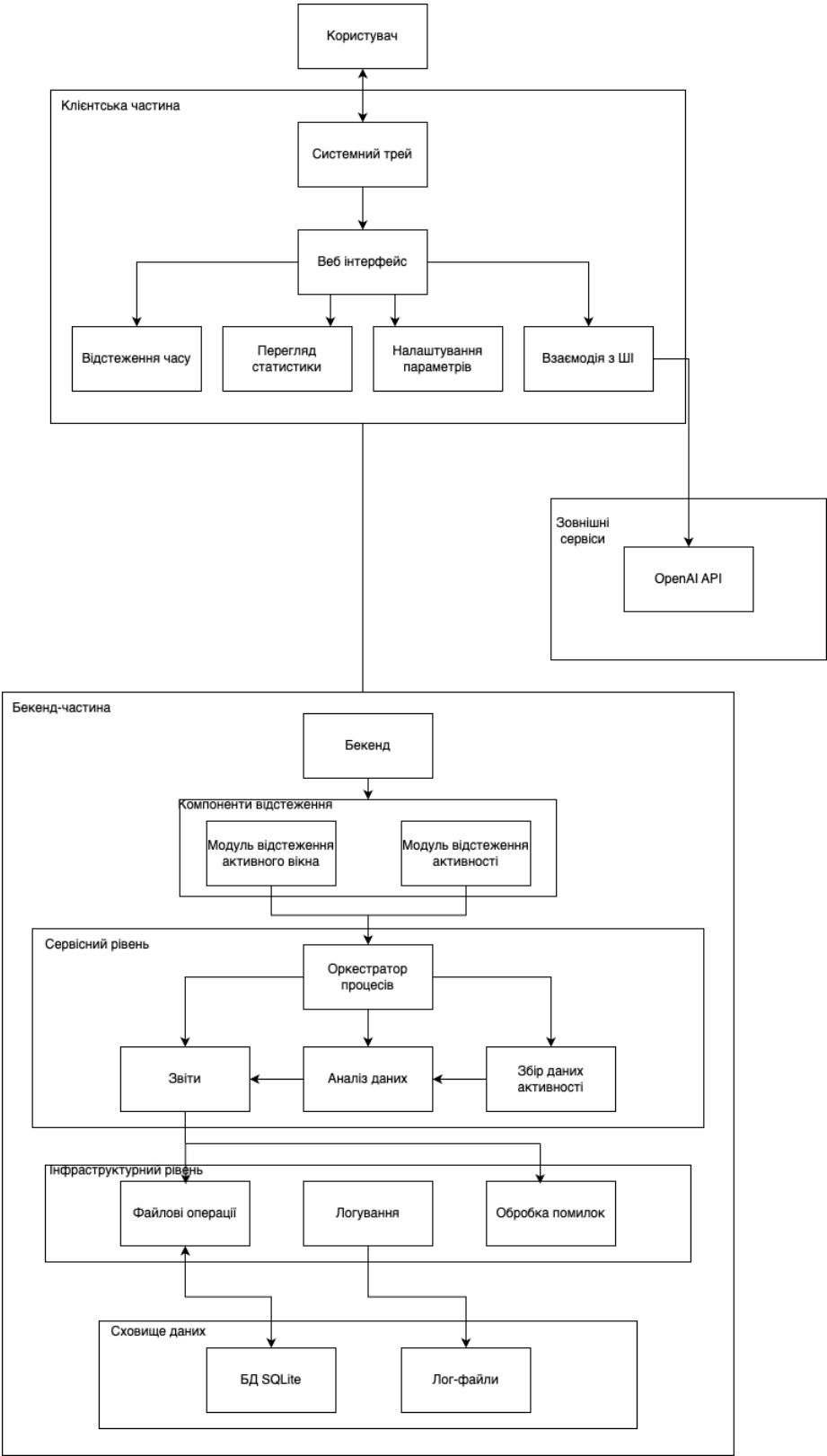


Рисунок 2.1 – Структурна схема застосунку

Дані зберігаються у БД SQLite та лог-файлах. Система відстежує активні вікна програм, виявляє періоди неактивності користувача, збирає статистику та надає рекомендації через інтеграцію з OpenAI API. Компоненти відстеження збирають дані про активність, передають їх на сервісний рівень для аналізу, результати якого відображаються у веб-інтерфейсі. Стрілки на діаграмі відображають потоки даних між компонентами. Отже, ця діаграма дає повне уявлення про структуру системи BeBetter, демонструючи її компоненти, їх взаємозв'язки та шляхи обробки даних для ефективного відстеження та аналізу використання часу користувачем.

2.2 Схема алгоритму роботи застосунку BeBetter

Щоб розпочати роботу з системою BeBetter, користувач запускає додаток, який автоматично починає відстеження часу через системний трей. Система відстежує активні вікна програм через встановлені інтервали часу (за замовчуванням 0.1 секунди) і зберігає цю інформацію в оперативній пам'яті.

При виявленні неактивності користувача (відсутність руху миші або клавіатури протягом визначеного порогу часу) система показує діалогове вікно для вказання причини відсутності. Схема алгоритму роботи наведена на рисунку 2.2. Зібрана інформація агрегується, аналізується та зберігається в базі даних SQLite. Користувач може переглядати статистику використання часу через веб-інтерфейс. Усі етапи логуються.

Схема алгоритму системи відстеження часу BeBetter описує процес обробки даних від початку до кінця. Спочатку користувач взаємодіє з системним треем для запуску або зупинки відстеження. Система постійно відстежує активні вікна програм та активність користувача. Модуль відстеження активного вікна отримує дані про поточну програму, а модуль відстеження активності контролює рух миші та клавіатури. Ці дані передаються на сервісний рівень, де оркестратор процесів координує збір та аналіз даних.

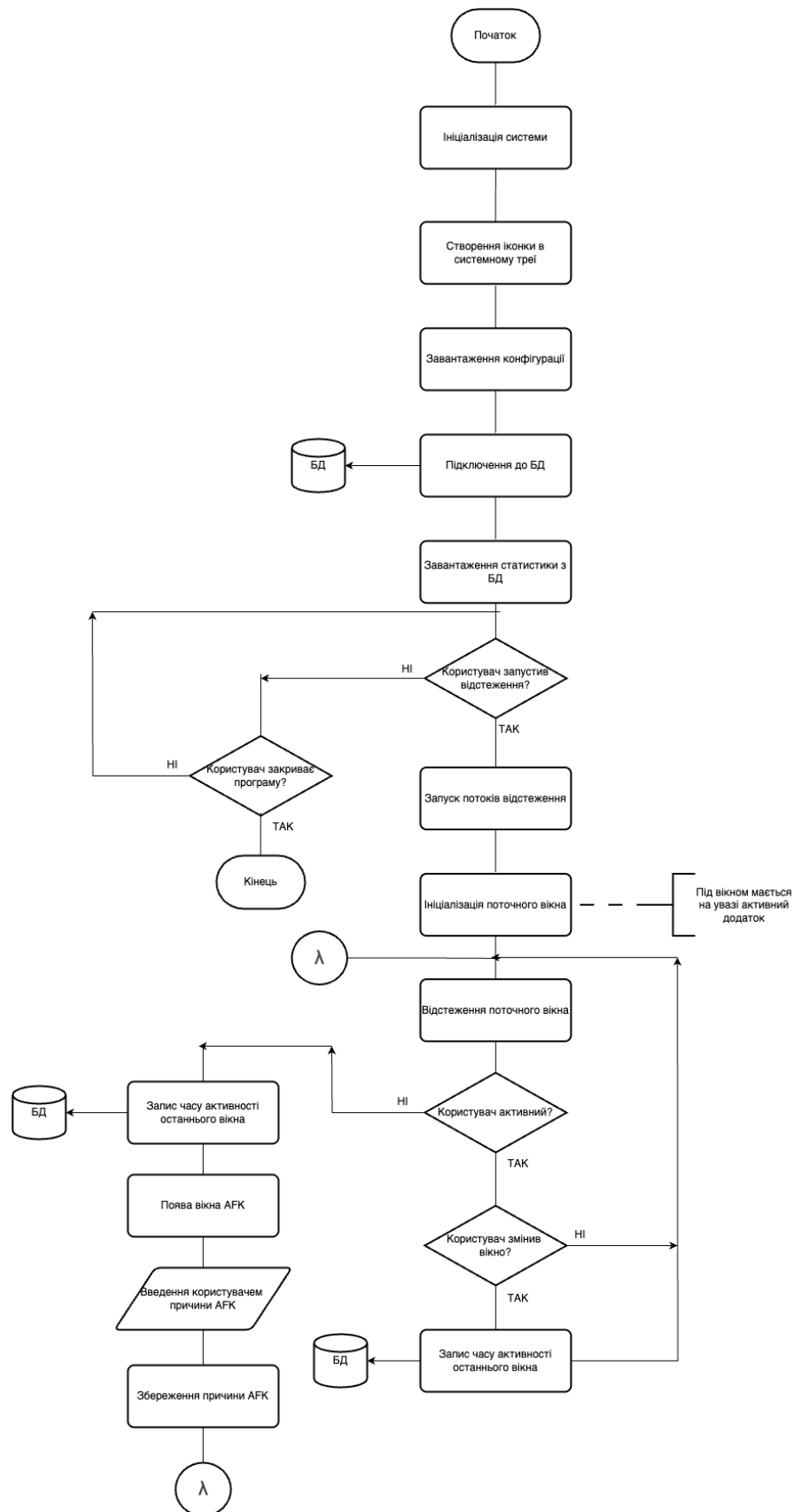


Рисунок 2.2 – Схема алгоритму роботи застосунку

При виявленні неактивності користувача система запитує причину та зберігає цю інформацію. Дані про використання програм аналізуються, агрегуються за різними періодами та візуалізуються у веб-інтерфейсі. Інтеграція з OpenAI API дозволяє отримувати рекомендації щодо підвищення продуктивності на основі зібраних даних. Уся інформація зберігається в базі даних SQLite, а системні події фіксуються в лог-файлах.

2.3 Обґрунтування вибору технічних інструментів

На основі проведеного порівняльного аналізу та з урахуванням вимог до системи BeBetter, було обрано оптимальний технологічний стек. Нижче наведено детальне обґрунтування вибору кожної технології та її переваги для реалізації системи відстеження часу.

Для розробки системи BeBetter було обрано мову програмування Python, яка має низку переваг порівняно з іншими мовами. Python забезпечує роботу програми на різних операційних системах без необхідності значної модифікації коду. Також він має розвинуту екосистему бібліотек для різних задач: від створення графічного інтерфейсу до аналізу даних та інтеграції з OpenAI API. Завдяки високорівневому синтаксису Python дозволяє швидко розробляти функціональні прототипи та кінцеві продукти, скорочуючи час від ідеї до реалізації.

Однією з ключових переваг використання PyQt6 у проєкті стало те, що ця бібліотека дозволяє створювати графічний інтерфейс, який виглядає сучасно та відповідає візуальним стандартам різних операційних систем. Особливістю PyQt6 є те, що її компоненти автоматично адаптуються до стилю системи, в якій запускається програма, завдяки чому інтерфейс виглядає нативно як на Windows, так і на macOS чи Linux. Це вигідно відрізняє PyQt6 від таких підходів, як Electron чи веб-застосунки, де зовнішній вигляд часто не узгоджується з середовищем користувача. Окрім цього, PyQt6 надає можливість вбудовувати веб-контент безпосередньо в застосунок за допомогою компонента

QWebEngineView. Це дозволяє використовувати HTML, CSS і JavaScript для створення інтерактивних елементів чи візуалізацій, не потребуючи окремого веб-сервера або запуску браузера.

Для програм, які повинні працювати у фоновому режимі, важливою функціональністю є підтримка системного трею. PyQt6 надає засоби для реалізації такого компонента через QSystemTrayIcon, що дозволяє користувачу швидко взаємодіяти з додатком, навіть якщо основне вікно закрито. Важливо також відзначити, що архітектура PyQt6 побудована на основі механізму сигналів і слотів, який значно спрощує реалізацію подієво-орієнтованої логіки. Цей підхід дозволяє ефективно реагувати на дії користувача й забезпечує чітку структуру коду. Ще однією перевагою є оптимальне використання системних ресурсів. Порівняно з Electron-додатками, PyQt6-програми працюють набагато економніше, споживаючи менше оперативної пам'яті та процесорного часу. Це особливо важливо для застосунків, які повинні працювати постійно у фоновому режимі, не створюючи зайвого навантаження на систему.

Для зберігання даних було обрано SQLite з ORM Peewee. На відміну від повноцінних серверних рішень, SQLite не потребує окремого встановлення чи налаштування — вона працює «з коробки» та зберігає дані у звичайному файлі, що суттєво спрощує як розгортання, так і подальше обслуговування програми.

Завдяки своїй компактності й оптимізації, SQLite практично не навантажує систему і показує хорошу швидкість навіть у фоновому режимі. Це особливо важливо для застосунків, які працюють постійно, але при цьому не повинні заважати користувачеві або уповільнювати комп'ютер.

Окрему увагу варто приділити тому, що SQLite підтримує повноцінну систему транзакцій з гарантією атомарності. Це означає, що всі операції запису виконуються повністю або не виконуються взагалі, завдяки чому забезпечується цілісність даних навіть у випадках неочікуваного завершення програми. Для зручної взаємодії з базою даних використовувався ORM-фреймворк Peewee, який дозволяє працювати з даними через Python-об'єкти, а не через ручне написання SQL-запитів. Це не лише покращує читабельність і підтримуваність

коду, але й пришвидшує розробку. Крім того, Reewee підтримує зберігання складних структур у вигляді JSON-поля. Це особливо зручно в контексті даного проєкту, де потрібно зберігати деталізовану інформацію про активність програм і часові інтервали їх використання. Ще однією важливою перевагою SQLite є її портативність. Оскільки вся база зберігається у вигляді одного файлу, її можна легко переносити між пристроями або створювати резервні копії без складних процедур.

Для реалізації механізму відстеження активності користувача в рамках проєкту було обрано бібліотеку `runput`. Однією з її ключових переваг є кросплатформеність: вона забезпечує стабільну роботу як на Windows, так і на macOS та Linux. Завдяки цьому додаток поводить себе однаково на різних системах, не потребуючи внесення змін у логіку моніторингу. `runput` працює на низькому рівні, що дозволяє перехоплювати системні події від клавіатури та миші навіть тоді, коли програма перебуває у фоновому режимі. Це особливо важливо для точної фіксації моментів неактивності, які важко вловити за допомогою високорівневих бібліотек. Бібліотека має простий та зручний у використанні API, що спрощує її інтеграцію з іншими частинами програми. Незважаючи на постійний моніторинг системної активності, `runput` працює ефективно та не створює помітного навантаження на систему, що критично для фонових сервісів. Окремо варто згадати про її стабільність: під час тестування вона виявилася надійнішою за деякі альтернативи, не викликала хибних спрацьовувань і не впливала на стабільність основного застосунку.

Для отримання даних про активні вікна запущених програм у цьому проєкті використовується комбінація бібліотеки `psutil` та нативних API, специфічних для операційної системи. Основна перевага `psutil` полягає в її універсальності: вона надає єдиний, уніфікований інтерфейс для доступу до інформації про процеси як у Windows, так і в Unix-подібних системах. Це дозволяє писати кросплатформенний код без необхідності створення окремої логіки для кожної ОС. Завдяки своїй ефективності `psutil` може швидко повертати актуальні дані про запущені процеси, практично не впливаючи на загальну

продуктивність системи. Крім базової інформації на кшталт імені процесу, ця бібліотека дозволяє отримувати більш розгорнуті характеристики: використання пам'яті, навантаження на CPU тощо. Це відкриває можливість для подальшого аналізу впливу окремих програм на стан комп'ютера. Оскільки psutil не надає повного доступу до інформації про вікна — зокрема, про їхній заголовок чи активність у фокусі — для доповнення функціональності застосовуються нативні API операційної системи. Наприклад, у Windows це функції WinAPI, які дозволяють визначити активне вікно і зчитати його заголовок у режимі реального часу. Така інтеграція забезпечує повнішу картину активності користувача і дає змогу точно визначати, з якою програмою він взаємодіє у певний момент часу.

Для створення візуального інтерфейсу, що відображає статистику використання часу, було використано комбінацію компонента QWebEngineView з бібліотеки PyQt6 та популярної JavaScript-бібліотеки Chart.js. Такий підхід дозволив інтегрувати сучасні веб-технології безпосередньо у десктопну програму, не вдаючись до запуску зовнішнього веб-сервера. QWebEngineView є вбудованим інструментом PyQt6, тому його використання є природним продовженням основного графічного інтерфейсу. Chart.js, у свою чергу, дає змогу будувати інформативні графіки, які виглядають сучасно та дозволяють користувачеві легко сприймати статистичні дані. Завдяки використанню HTML та JavaScript, з'явилася можливість гнучко налаштовувати зовнішній вигляд графіків, реалізовувати анімації, інтерактивні підказки та реакції на наведення.

Щоб забезпечити повноцінну взаємодію між Python-частиною додатку та вбудованим веб-інтерфейсом, використано механізм QWebChannel. Він дозволяє реалізувати двосторонній обмін даними: Python може передавати статистику в браузерну частину, а JavaScript — повідомляти про взаємодії користувача, наприклад, натискання на елемент графіка.

Такий підхід не тільки забезпечує високу якість візуального представлення, а й дозволяє легко масштабувати систему в майбутньому —

наприклад, додавати нові типи діаграм або інтерактивні панелі без зміни основної логіки програми.

Для реалізації інтелектуального асистента, був проведений аналіз доступних моделей від компанії OpenAI. Розглядались моделі GPT-4o, o4-mini, o3-mini. Для вибору оптимальної моделі був проведений аналіз результатів тестування в бенчмарках, розмір контекстного вікна та вартість використання за 1 мільйон токенів. На таблиці 2.1 наведено порівняння розмірів контекстних вікон різних моделей, де показано, що по розмірах контекстних вікон лідером є GPT-4.1 mini

Таблиця 2.1 - Порівняння розмірів контекстних вікон різних моделей

o4-mini	200K токенів
GPT-4o	128K токенів
o3-mini	200K токенів
GPT-4.1 mini	1000K токенів

На таблиці 2.2 наведено порівняння цін на використання різних моделей.

Таблиця 2.2 – порівняння цін на використання різних моделей

o4-mini	\$1.10 (вхідні)	\$4.40 (вихідні)
GPT-4o	\$2.50 (вхідні)	\$10.00 (вихідні)
o3-mini	\$1.10 (вхідні)	\$4.40 (вихідні)
GPT-4.1 mini	\$0.40 (вхідні)	\$1.60 (вихідні)

З таблиці 2.2 можна зробити висновки, що оптимальною моделлю є GPT-4.1 mini. Попри те, що розмір контекстного вікна в 1 мільйон токенів є надмірним для використання в застосунку BeBetter, ціна за її використання значно менша, ніж в o4-mini. Потужності GPT-4o надлишкові для BeBetter, а ціна занадто висока. Що стосується o3-mini – це попередник o4-mini, і при однаковій ціні з o4-mini цей варіант є ірраціональним.

2.4 Інформаційне забезпечення

Проектована система BeBetter обробляє вхідну та вихідну інформацію. До вхідної інформації можна віднести:

- а) системні події – події миші та клавіатури, інформація про активні вікна програм;
- б) користувацькі налаштування – поріг неактивності, тема інтерфейсу, мова, API-ключ;
- в) параметри відстеження – період аналізу (день/тиждень/місяць), фільтрація програм;
- г) дані про причини неактивності – введені користувачем пояснення періодів AFK.

Вихідна інформація системи BeBetter включає:

- 1) статистика використання часу – загальний час за комп'ютером, час по програмах, відсоткове співвідношення, продуктивні/непродуктивні періоди;
- 2) результати аналізу – візуалізація даних у вигляді графіків та діаграм, рекомендації щодо продуктивності;
- 3) метадані відстеження – статус (активне/призупинене/зупинене), час початку та кінця сесії, загальна тривалість;
- 4) історія та статистика – історія відстеження користувача, статистика (загальний час, середній час за день, розподіл за програмами).
- 5) Запис відстеження (TrackingRecord) – представляє період активності користувача з такими полями: ID запису (PK), час початку, час закінчення, причина неактивності, повідомлення, відстежені програми (у форматі JSON).

База даних для системи BeBetter використовує SQLite для зберігання даних відстеження часу. Для взаємодії з базою даних використовується ORM Peewee, що дозволяє абстрагуватися від прямих SQL-запитів і працювати з даними через об'єктно-орієнтовані моделі. Сутність TrackingRecord має методи для агрегації та аналізу даних за різні періоди:

- 1) `get_today_records()` – отримує всі записи за поточний день
- 2) `get_time_summary_for_today()` – розраховує підсумковий час за поточний день
- 3) `get_time_summary_for_week()` – розраховує підсумковий час за поточний тиждень
- 4) `get_time_summary_for_month()` – розраховує підсумковий час за поточний місяць
- 5) `get_time_summary_for_period()` – розраховує час за вказаний період

Дані про використання програм зберігаються в полі `programs_tracked` у форматі JSON, що дозволяє гнучко зберігати інформацію про різну кількість програм без зміни структури бази даних. Для кожної програми зберігається час використання в секундах, що потім агрегується для аналізу та візуалізації. Крім бази даних, система використовує файлову систему для зберігання:

- лог-файлів з інформацією про роботу програми
- файлів конфігурації (`config.json`) з налаштуваннями користувача
- кешованих іконок програм для відображення в інтерфейсі

Така організація інформаційного забезпечення дозволяє ефективно зберігати, аналізувати та візуалізувати дані про використання часу, забезпечуючи швидкий доступ до необхідної інформації та підтримуючи всі функціональні вимоги системи BeBetter.

На рисунку 2.3 представлена структура таблиці для запису активностей в SQLite. Оскільки база даних має просту структуру з однією основною таблицею `TrackingRecord`, діаграма буде містити лише цю сутність. На відміну від системи вашого однокористувача, яка має три пов'язані таблиці (`User`, `ProcessedFile` і `TaskStatus`), система BeBetter використовує одну таблицю для зберігання всіх даних про відстеження часу.

TrackingRecord	
PK	ID запису
	start_time end_time reason message programs_tracked

Рисунок 2.3 – Структура таблиці запису активностей

У поле "Відстежені програми" (programs_tracked) зберігається JSON-структура, яка містить інформацію про всі програми, використані під час сесії відстеження, та час, проведений у кожній з них. Це дозволяє зберігати динамічну кількість програм без зміни структури бази даних (Додаток Е).

На рисунку 2.4 представлено структуру JSON для поля programs_tracked.

```
{
  "Chrome": 3600,
  "Visual Studio Code": 1800,
  "Slack": 900,
  "Finder": 300
}
```

Рисунок 2.4 – Структура JSON для поля programs_tracked

Загалом усі компоненти формують основу системи BeBetter, яка відстежує час використання програм, виявляє періоди неактивності користувача, збирає статистику та візуалізує її через веб-інтерфейс, підвищуючи продуктивність користувача на основі аналізу даних.

3 ПРОГРАМНО-ТЕХНОЛОГІЧНЕ ЗАБЕЗПЕЧЕННЯ ПРИКЛАДНОГО ПРОГРАМНОГО ІНТЕРФЕЙСУ

3.1 Реалізація програмного забезпечення

Система VeBetter реалізована як модульний додаток з чітким розподілом відповідальності між компонентами. Основна увага при розробці приділялася ефективності відстеження, низькому споживанню ресурсів та зручності використання. Структура модульної організації системи зображена на рисунку 3.1.

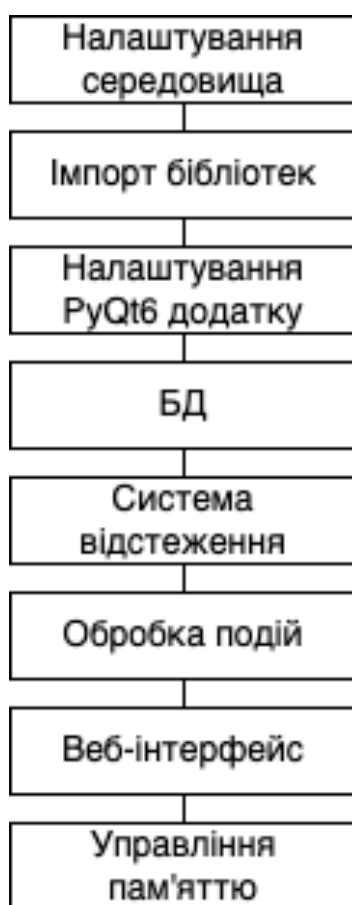


Рисунок 3.1 – Модульна організація системи

Задля безпеки та уникання конфліктів залежностей, усі процеси відбуваються в окремому середовищі. Додаток виконує модульне відстеження активності з налаштованим середовищем і PyQt6 для зручної взаємодії з користувачем. Система зберігає дані в локальній базі даних SQLite,

забезпечуючи їх цілісність та швидкий доступ. Основна логіка використовує багатопотоковість для одночасного відстеження активних вікон та активності користувача, а також інтеграцію з OpenAI API для аналізу продуктивності. Система BeBetter легко налаштовується, ефективно використовує ресурси через оптимізовані алгоритми та керує даними через структуровані записи в базі даних.

Налаштування середовища більш детально зображено на таблиці 3.1. Модуль налаштування середовища відповідає за початкову конфігурацію системи BeBetter та забезпечення її правильної роботи на різних платформах.

Таблиця 3.1 – Налаштування середовища

№	Налаштування середовища
1	Визначення шляхів до файлів
2	Завантаження налаштувань
3	Налаштування логування

При ініціалізації клас визначає шлях до конфігураційного файлу `config.json`, який знаходиться в тій же директорії, що й модуль `config.py`. Далі викликається метод `load_config()`, який завантажує налаштування з файлу або створює налаштування за замовчуванням. Важливим аспектом налаштування середовища є визначення шляху до бази даних SQLite. Це реалізовано в модулі `models.py`. Система також налаштовує логування для різних компонентів. Кожен модуль створює власний логер з відповідним ім'ям. Додатково система відключає попередження Qt для забезпечення чистого виводу в консоль.

Структура модуля «Імпорт бібліотек» детальніше зображена на таблиці 3.2. Імпорт бібліотек організований таким чином, щоб забезпечити чітку структуру коду та мінімізувати залежності між модулями. Кожен модуль системи імпортує лише ті бібліотеки, які безпосередньо необхідні для його роботи. Система BeBetter використовує переважно бібліотеки з стандартної бібліотеки Python та добре підтримувані зовнішні пакети, що

забезпечує стабільність роботи та спрощує розгортання. Всі зовнішні залежності вказані у файлі requirements.txt, що дозволяє легко встановити необхідні пакети за допомогою менеджера пакетів pip. Вибір бібліотек здійснювався з урахуванням їх кросплатформеності, що дозволяє системі BeBetter працювати на різних операційних системах без змін у коді. Це особливо важливо для PyQt6 та pynput, які забезпечують однакову функціональність на Windows, macOS та Linux.

Таблиця 3.2 – Структура модуля «імпорт бібліотек»

№	Імпорт бібліотек
1	Графічний інтерфейс (PyQt6): QObject, pyqtSignal, QEvent, QSystemTrayIcon, QMenu, QAction, QMainWindow, QDialog, QVBoxLayout, QWebEngineView, QWebChannel
2	Відстеження активності: pynput, psutil
3	Робота з даними: peewee, json
4	Системні бібліотеки: os, time, datetime, threading
5	Логування та діагностика: logging

Структура модуля «Налаштування PyQt6 додатку» детальніше зображена на таблиці 3.3. Налаштування PyQt6 додатку включає створення екземпляру QApplication для керування графічним інтерфейсом, налаштування системного трею для ненав'язливої роботи у фоновому режимі та ініціалізацію механізмів відстеження активності користувача. Це забезпечує відмінний користувацький досвід та стабільну роботу системи відстеження часу.

Таблиця 3.3 – Структура модуля «Налаштування PyQt6 додатку»

№	Налаштування PyQt6 додатку
1	Створення екземпляру QApplication
2	Ініціалізація системного трею
3	Налаштування механізмів відстеження активності

Структура модуля «БД» детальніше зображена на таблиці 3.4. Модуль бази даних відповідає за зберігання та управління даними про активність користувача. Для роботи з базою даних використовується ORM Peewee, що забезпечує зручний програмний інтерфейс для взаємодії з SQLite. Основна модель TrackingRecord зберігає інформацію про часові проміжки активності, використані програми та їх тривалість. База даних автоматично створюється при першому запуску додатку і містить всю необхідну інформацію для генерації звітів. Методи для роботи зі статистикою дозволяють гнучко отримувати дані за різні періоди: день, тиждень, місяць. Це забезпечує зручний аналіз продуктивності користувача та формування детальних звітів про використання робочого часу.

Таблиця 3.4 – Структура модуля «БД»

№	БД
1	Ініціалізація SQLite бази даних
2	Визначення моделі TrackingRecord
3	Методи для роботи зі статистикою

Структура модуля «Система відстеження активності» детальніше зображена на таблиці 3.5. Система відстеження активності є ключовим компонентом додатку BeBetter, що забезпечує автоматичний збір даних про використання програм користувачем. Система працює у фоновому режимі та складається з трьох основних компонентів. Моніторинг активного вікна здійснюється через системні API (для Windows - через win32gui, для macOS - через NSWorkspace), що дозволяє отримувати інформацію про поточну активну програму користувача. Система періодично (з налаштованим інтервалом) зчитує назву активного вікна та записує час, проведений у кожній програмі. Відстеження дій користувача реалізовано через бібліотеку rpyprut, яка відслідковує події клавіатури та миші для визначення активності. Це дозволяє системі точно визначати, коли користувач працює за комп'ютером, а коли відсутній. Модуль виявлення неактивності (AFK) автоматично виявляє

періоди, коли користувач відсутній, і дозволяє зафіксувати причину відсутності через спеціальний діалог при поверненні. Це забезпечує точність даних у звітах і дозволяє користувачу вести облік часу, витраченого на перерви та зустрічі.

Таблиця 3.5 – Структура модуля «Система відстеження активності»

№	Система відстеження активності
1	Моніторинг активного вікна
2	Відстеження дій користувача
3	Виявлення неактивності (AFK)

Структура модуля «Система обробки подій» детальніше зображена на таблиці 3.6. Система обробки подій є важливим компонентом додатку BeBetter, що забезпечує реагування на дії користувача та системні події (Додаток Б, В). Вона реалізована з використанням механізму сигналів і слотів PyQt6, що дозволяє ефективно передавати повідомлення між різними частинами програми. Сигнали та слоти PyQt6 забезпечують асинхронну взаємодію між компонентами програми.

Таблиця 3.6 – Структура модуля «Система обробки подій»

№	Система обробки подій
1	Сигнали та слоти PyQt6
2	Обробники системних подій
3	Обробка користувацької взаємодії

У системі BeBetter використовуються такі сигнали як `inactivity_detected` та `time_updated`, що дозволяють різним частинам додатку реагувати на виявлення неактивності користувача та оновлення статистики часу відповідно. Це забезпечує слабку зв'язаність компонентів і підвищує надійність системи.

Обробники системних подій відповідають за взаємодію з операційною системою. Вони обробляють події, пов'язані з системним треем, відстежують зміну активного вікна та реагують на системні повідомлення. Метод `event()` класу `TimeTracker` розширює стандартний механізм обробки подій

Qt для підтримки спеціальних подій, таких як виявлення неактивності користувача.

Обробка користувацької взаємодії реалізована через спеціалізовані обробники подій, які реагують на кліки миші по іконці в системному треї, взаємодію з контекстним меню та натискання кнопок у графічному інтерфейсі. Система підтримує анімовані переходи при відображенні меню та модальних діалогів, що підвищує зручність використання додатку.

Структура модуля «Веб-інтерфейс» детальніше зображена на таблиці 3.7. Веб-інтерфейс та графічний інтерфейс користувача у системі BeBetter реалізовані з використанням сучасних технологій і дотриманням принципів UX/UI дизайну. WebView на основі PyQt6 забезпечує відображення HTML/CSS/JavaScript контенту в нативному додатку. Це дозволяє поєднувати переваги веб-технологій (гнучкість верстки, зручність оновлень) з нативними можливостями PyQt6.

Таблиця 3.7 – Структура модуля «Веб-інтерфейс»

№	Веб-інтерфейс
1	WebView на основі PyQt6
2	Нативний інтерфейс
3	Інтеграція з ШІ та відображення статистики

Для комунікації між Python-кодом та JavaScript використовується QWebChannel, що забезпечує двосторонній обмін даними. Основні функції WebView включають відображення статистики, запуск та зупинку відстеження, а також доступ до налаштувань через верхнє меню. Сучасний нативний інтерфейс розроблений спеціально для додатку BeBetter з урахуванням естетики macOS. Він включає спеціальні компоненти, такі як RoundedButton, AppListItem та інші кастомні віджети, що забезпечують сучасний і приємний для користувача вигляд.

Інтерфейс має адаптивний дизайн, підтримує анімації та плавні переходи, а також реалізує темну і світлу теми. Модуль також забезпечує інтеграцію з ШІ

та відображення статистики через компонент ChatWidget, який надає можливість отримувати аналітичні дані та рекомендації на основі зібраної статистики використання програм. Користувач може взаємодіяти з ШІ, задавати питання щодо своєї продуктивності та отримувати корисні поради. Статистика відображається у вигляді інтерактивних елементів з використанням прогрес-барів, кругових діаграм та інших візуальних компонентів, що полегшує сприйняття інформації.

Структура модуля «Управління пам'яттю» детальніше зображено на таблиці 3.8. Система управління пам'яттю у додатку BeBetter забезпечує ефективне використання ресурсів та запобігає витокам пам'яті. Вона реалізована через декілька ключових компонентів та механізмів. Керування ресурсами через деструктори є важливим елементом системи, що гарантує коректне звільнення ресурсів при завершенні роботи.

Таблиця 3.8 – Структура модуля «Управління пам'яттю»

№	Управління пам'яттю
1	Керування ресурсами через деструктори
2	Кешування об'єктів та очищення кешу
3	Оптимізація бази даних

У класі TimeTracker реалізовано метод `__del__`, який відповідає за безпечне завершення потоків відстеження та звільнення зайнятих ресурсів. Додатково система реагує на сигнали завершення (SIGINT, SIGTERM) та подію `aboutToQuit` застосунку, що забезпечує коректне збереження даних та закриття з'єднань з базою даних перед виходом. Кешування об'єктів та очищення кешу реалізовано для оптимізації використання пам'яті при роботі з зображеннями та іконками програм. Система використовує словник `app_icons_cache` для зберігання та повторного використання іконок програм, що зменшує навантаження на диск та прискорює відображення інтерфейсу. Для файлової системи реалізовано механізм кешування іконок у окремій директорії, з можливістю їх перевикористання між сесіями.

Оптимізація бази даних забезпечує ефективну роботу з даними через управління підключеннями та транзакціями. Система відкриває з'єднання з базою даних тільки при необхідності та закриває їх після використання, що запобігає витокам ресурсів. Реалізовано також механізми періодичного збереження даних під час тривалих сесій відстеження та оптимізації запитів до бази даних для зменшення навантаження. Додатково система включає скрипт `clean_db.py` для очищення застарілих записів та оптимізації розміру бази даних.

3.2 Інтерфейс користувача

Графічний інтерфейс в додатку BeBetter в основному реалізовано за допомогою вбудованих інструментів бібліотеки PyQt6 та з підключенням CSS-стилів через метод `setStyleSheet()`. На рисунку 3.2 зображена головна сторінка застосунку.

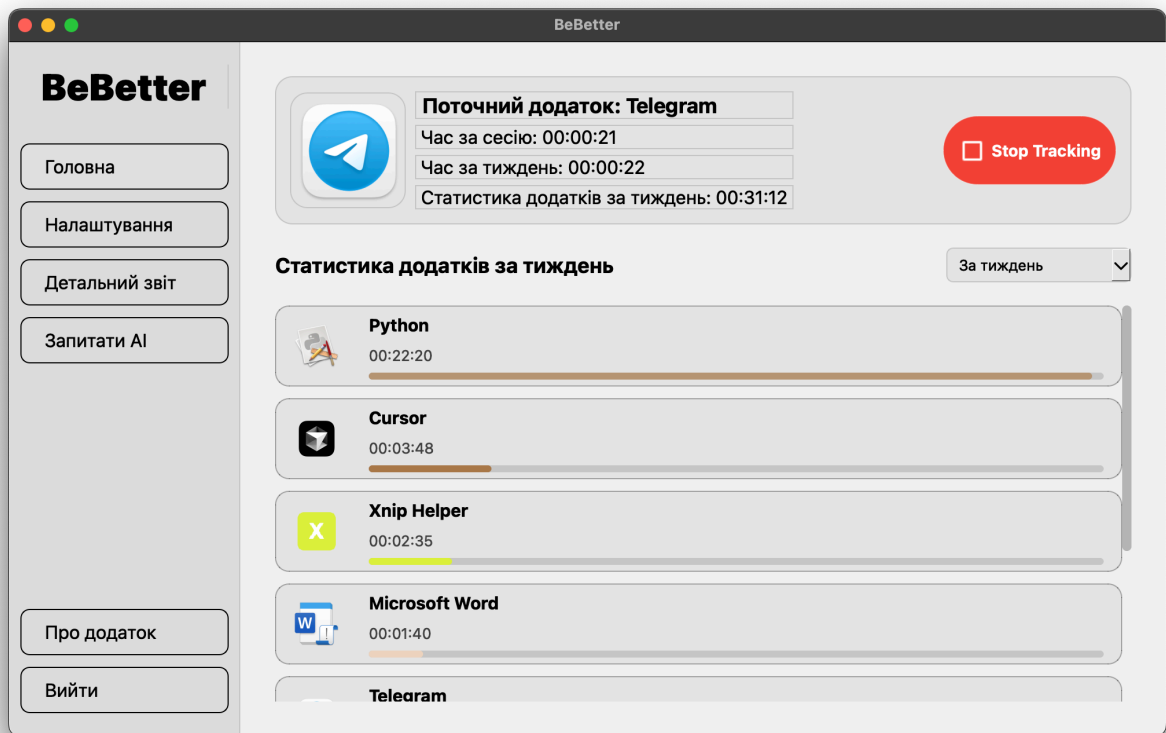


Рисунок 3.2 – Головна сторінка застосунку BeBetter

На головній сторінці розміщені основна інформація та інструменти. Основу графічного інтерфейсу складає фреймворк PyQt6, який є потужним набором інструментів для створення кросплатформних GUI-застосунків із Python. У коді використовуються різні модулі PyQt6:

- QtWidgets - для основних елементів інтерфейсу (кнопки, мітки, поля введення);
- QtCore - для системних функцій та сигналів;
- QtGui - для візуальних ефектів, шрифтів, кольорів та малювання.

Також реалізована система тем. Візуальне оформлення додатку може бути як і світлим (як на рисунку 3.2), так і темним, або ж залежати від теми системи. Всі елементи інтерфейсу мають метод `apply_theme()` для застосування відповідної теми. Головне вікно додатку BeBetter в темній темі зображене на рисунку 3.3.

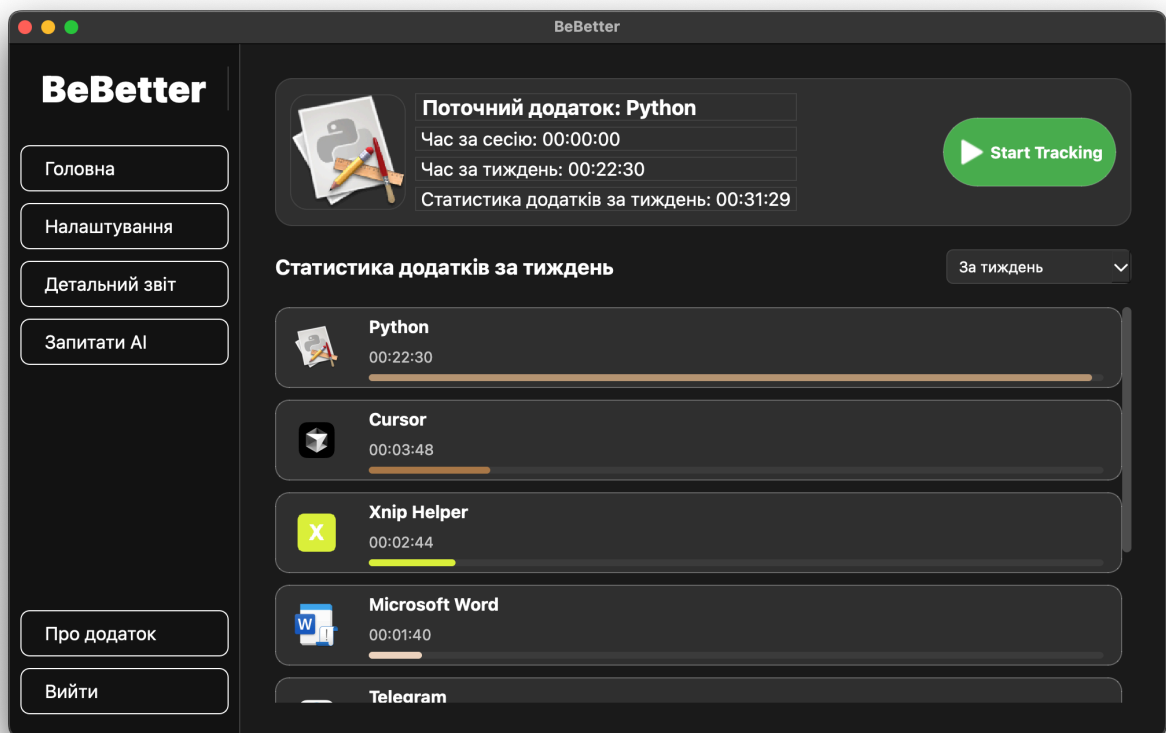


Рисунок 3.3 – Головне вікно в темній темі

Елементи інтерфейсу адаптуються до розміру вікна завдяки використанню відносних розмірів та системи компоновання (layouts). Детальніше це зображено на рисунку 3.4.

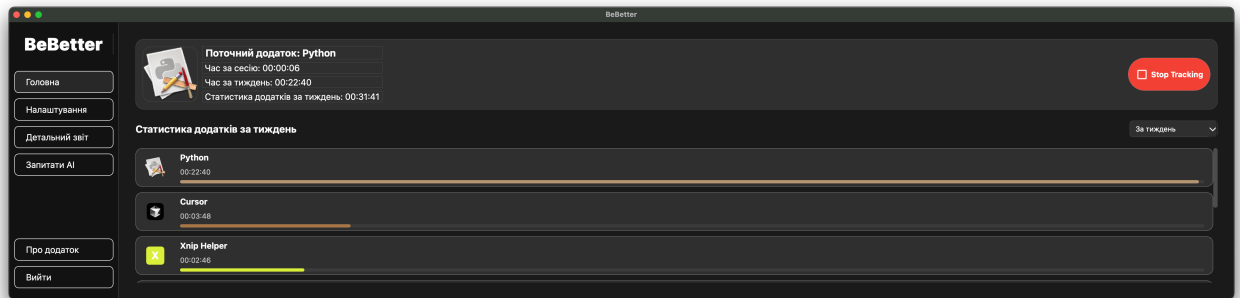


Рисунок 3.4 – Демонстрація адаптивності вікна

Після запуску додатку BeBetter, в користувача з'являється іконка в трей. Трей - це частина верхньої панелі (меню-бару), де розташовані маленькі іконки додатків. На macOS він розташований зверху справа, на Windows – знизу справа. Чорна піктограма годинника, зображеного на рисунку 3.5 – це піктограма додатку BeBetter.

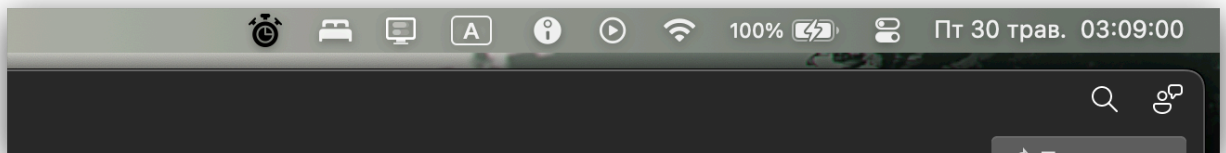


Рисунок 3.5 – Трей macOS та піктограма BeBetter

Коли користувач натискає на іконку, з'являється невеличке вікно з базовою інформацією про поточний відстежуємий додаток, та кнопки взаємодії. Детальніше це зображено на рисунку 3.6. Завдяки цьому вікну, немає необхідності займати місце в Dock ще одною іконкою. Користувач завжди може взаємодіяти з програмою через трей.

Для відображення статистики використання програм реалізовано кастомні прогрес-бари з відсотковим відображенням використаного часу. Це зроблено для наочнішого та інтуїтивного розуміння про витрачений час в певних застосунках.

За 100% довжину прогрес-бару береться застосунок, на який витрачено найбільше часу, а усі наступні адаптуються під нього. Якщо застосунок А зайняв 1 годину часу, то шкала застосунку Б, на який було витрачено 15 хвилин, буде на 75% коротшою за застосунок А. Це зображено на рисунку 3.7.

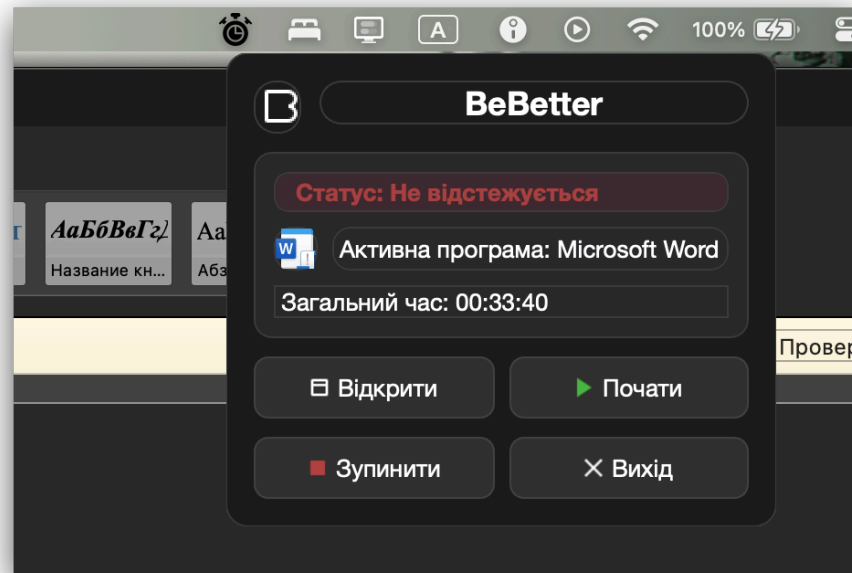


Рисунок 3.6 – Трей-вікно BeBetter.

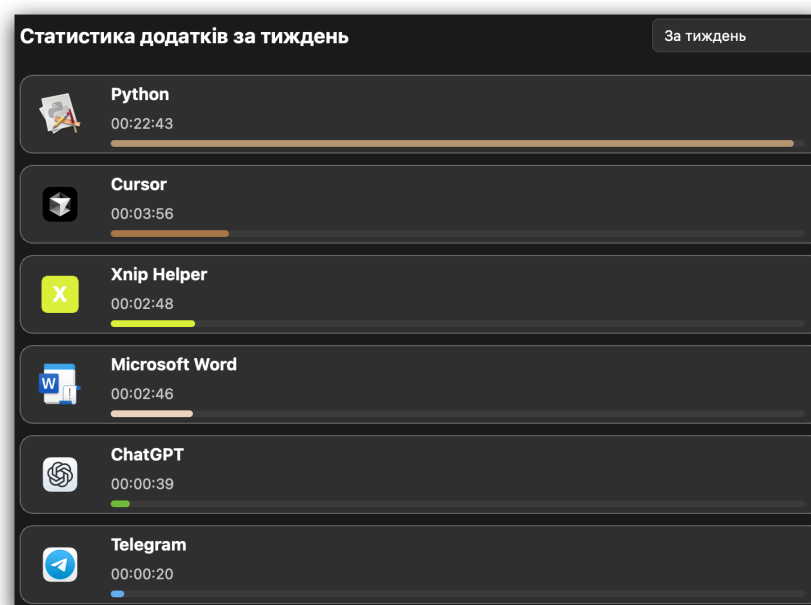


Рисунок 3.7 – Прогрес-бари додатків

Для організації переключення між різними екранами (головний екран, налаштування, AI) використовується QStackedWidget, що дозволяє ефективно керувати відображенням контенту.

На рисунку 3.8 зображено пункт «Налаштування». В налаштуваннях користувач може міняти такі параметри в застосунку, як тему (світла, темна, або системна), дозволити або заборонити програмі виявляти AFK, і якщо виявлення дозволено – обрати період, після якого програма активує модальне-вікно для того, щоб користувач ввів причину своєї відсутності. Поки користувач не підтвердить, що він повернувся – час відстеження зупинений.

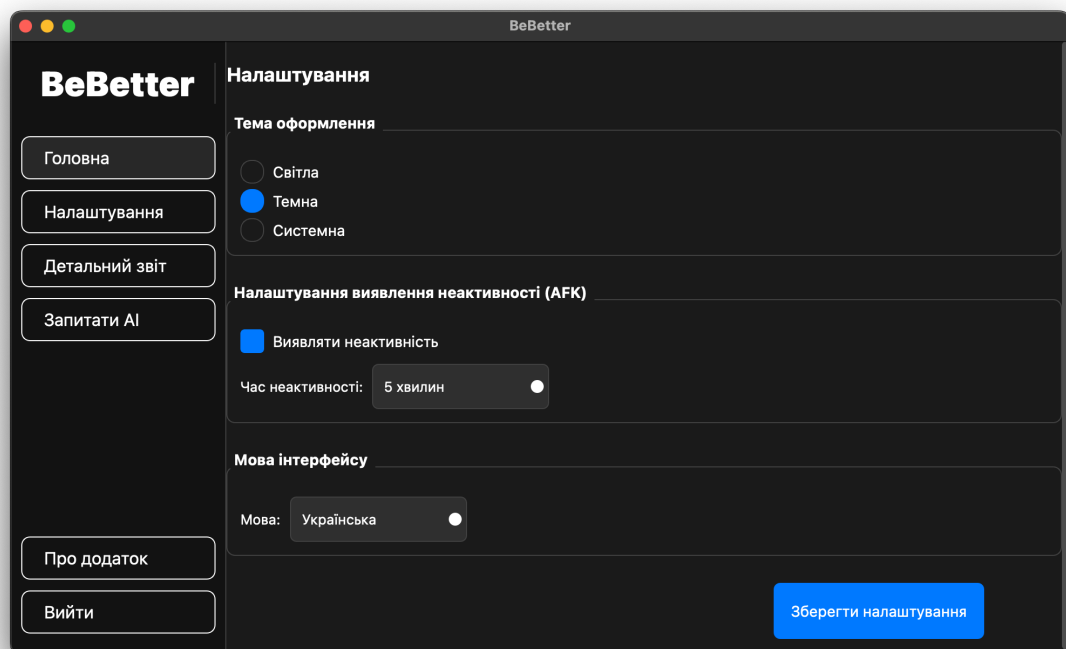


Рисунок 3.8 – Пункт налаштувань BeBetter

На рисунку 3.9 зображений інтерфейс вікна з інтелектуальним асистентом. Користувачу пропонується вибрати, дані за який період йому передати в контекст мовної моделі (Додаток Г). Також тут дублюється список програм для наочності. Сам чат з мовною моделлю, зображено на рисунках 3.10, 3.11, 3.12.

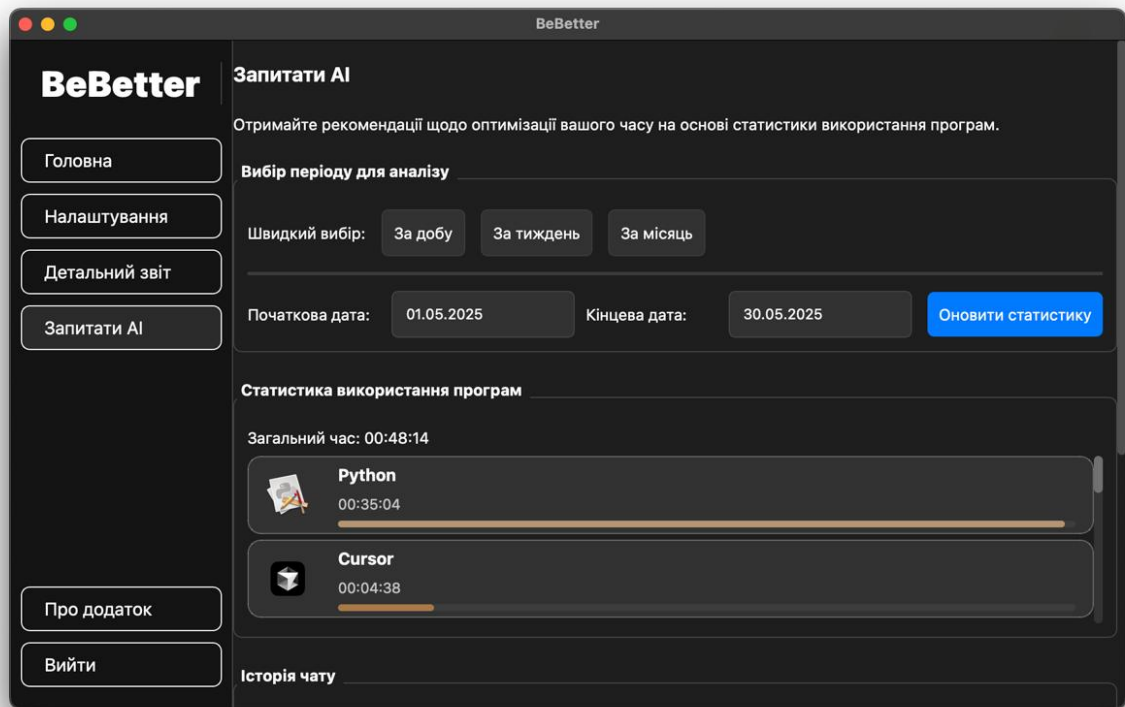


Рисунок 3.9 – Інтерфейс вікна з інтелектуальним асистентом.

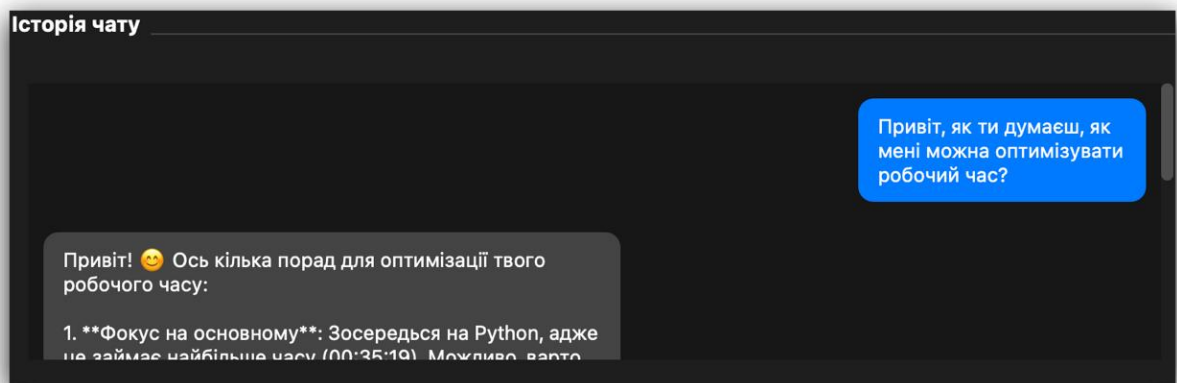
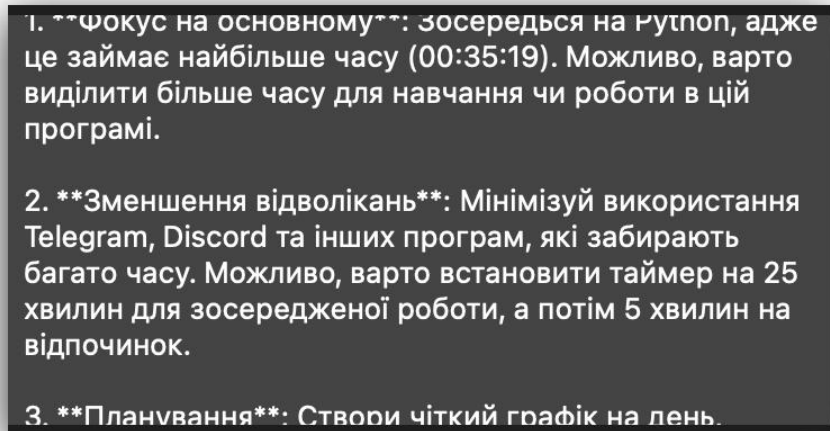


Рисунок 3.10 – Інтерфейс чату з інтелектуальним асистентом

Графічний інтерфейс додатку "BeBetter" забезпечує зручне та наочне відстеження часу використання програм. Він поєднує функціональну бокову навігацію, інформативні елементи відображення статистики та інтуїтивне керування процесом відстеження. Основними перевагами інтерфейсу є чітке представлення даних через кольорові прогрес-бари, гнучкість налаштувань та можливість отримання AI-рекомендацій щодо оптимізації часу.

Підтримка роботи у фоновому режимі через системний трей робить додаток ненав'язливим і практичним для повсякденного використання.

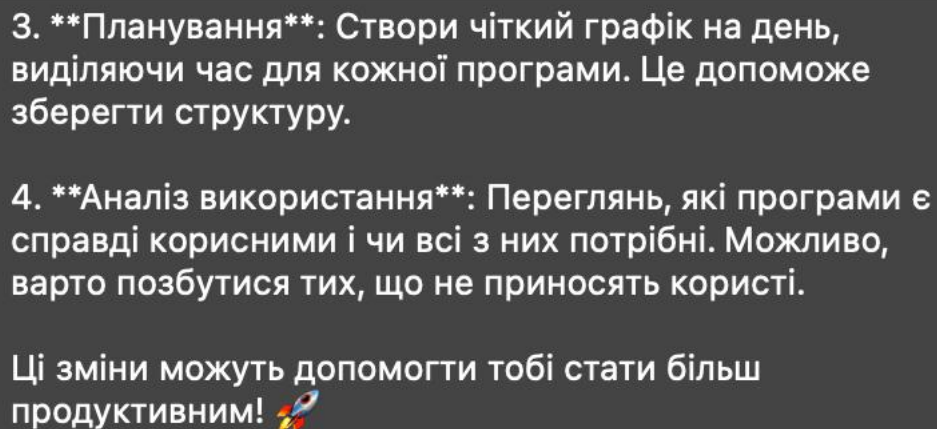


1. **Фокус на основному**: Зосередься на гуглоп, адже це займає найбільше часу (00:35:19). Можливо, варто виділити більше часу для навчання чи роботи в цій програмі.

2. **Зменшення відволікань**: Мінімізуй використання Telegram, Discord та інших програм, які забирають багато часу. Можливо, варто встановити таймер на 25 хвилин для зосередженої роботи, а потім 5 хвилин на відпочинок.

3. **Планування**: Створи чіткий графік на день.

Рисунок 3.11 – Інтерфейс чату з інтелектуальним асистентом



3. **Планування**: Створи чіткий графік на день, виділяючи час для кожної програми. Це допоможе зберегти структуру.

4. **Аналіз використання**: Переглянь, які програми є справді корисними і чи всі з них потрібні. Можливо, варто позбутися тих, що не приносять користі.

Ці зміни можуть допомогти тобі стати більш продуктивним! 🚀

Рисунок 3.12 – Інтерфейс чату з інтелектуальним асистентом

3.3 Тестування розробленої програми

Весь функціонал було перевірено за наступним функціоналом, та за наявністю тих чи інших помилок. В цей перелік увійшли наступні перевірки:

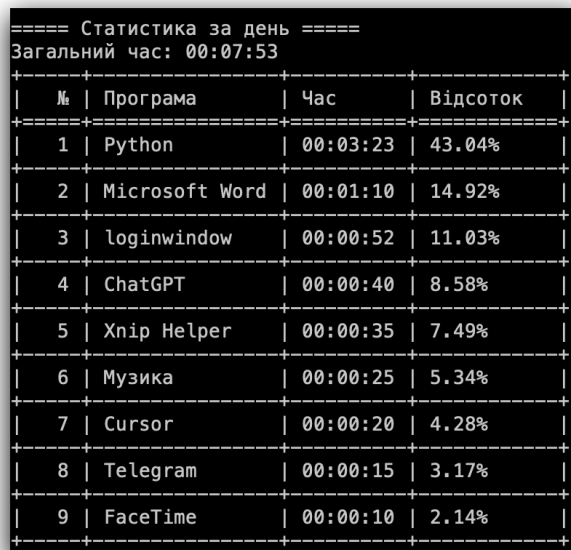
- Відповідність відображених користувачеві даних та даних в БД;

- Поведінка та працездатність елементів інтерфейсу при розтягуванні вікна;

- Спроба відстежувати додаток, та заблокувати екран;
- Спроба ввести пусту причину AFK;
- Спроба взаємодії з інтелектуальним асистентом без токена від OpenAI;
- Спроба спілкування з інтелектуальним асистентом «не по темі».

Для перевірки першого пункту, було написано невеликий скрипт (Додаток А), який брав записи з БД за день, тиждень, та місяць, окремо для кожної програми сумував всі записи, та конвертував час в формат години/хвилини/секунди.

На рисунку 3.13 зображений результат з БД за день, який повернув скрипт.



```
==== Статистика за день ====
Загальний час: 00:07:53
```

№	Програма	Час	Відсоток
1	Python	00:03:23	43.04%
2	Microsoft Word	00:01:10	14.92%
3	loginwindow	00:00:52	11.03%
4	ChatGPT	00:00:40	8.58%
5	Xnip Helper	00:00:35	7.49%
6	Музика	00:00:25	5.34%
7	Cursor	00:00:20	4.28%
8	Telegram	00:00:15	3.17%
9	FaceTime	00:00:10	2.14%

Рисунок 3.13 – Результат з БД за день

На рисунку 3.14 зображений список програм, які бачить користувач. Як видно з рисунків 3.13 та 3.14 – результати збігаються. Час за добу, та час кожної з програм ідентичний як в результаті від скрипта, так і в самій програмі.

Наступний пункт тестування – хаотична зміна розмірів вікна. Під час тестування, програма вела себе повністю справно.

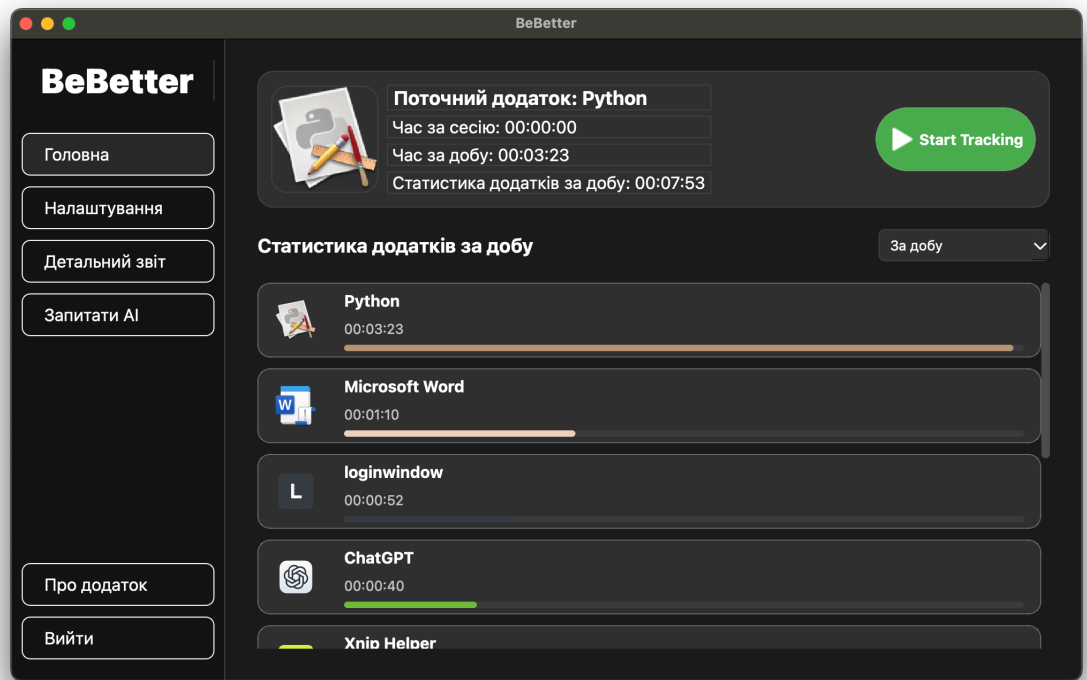


Рисунок 3.14 – Список програм, які бачить користувач

На рисунку 3.15 зображено мінімальний можливий розмір основного вікна.



Рисунок 3.15 – Мініально можливий розмір вікна

Також завдяки PyQt6, додаток ідеально працює з функціоналом traffic lights в macOS. На рисунку 3.16 зображений traffic lights, та функціонал зеленої кнопки.

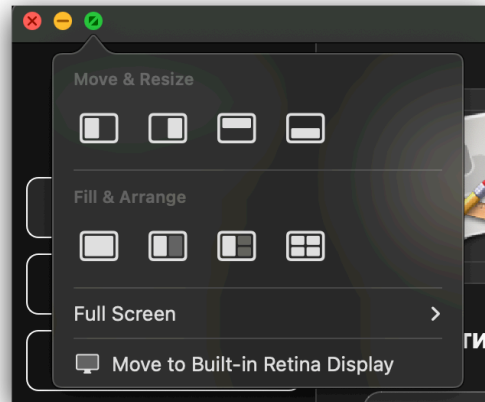


Рисунок 3.16 – Traffic lights

Наприклад, вікно BeBetter можна розмістити в будь-якому пропонованому на рисунку 3.16 положенні.

На рисунку 3.17 приклад такого розміщення вікна.

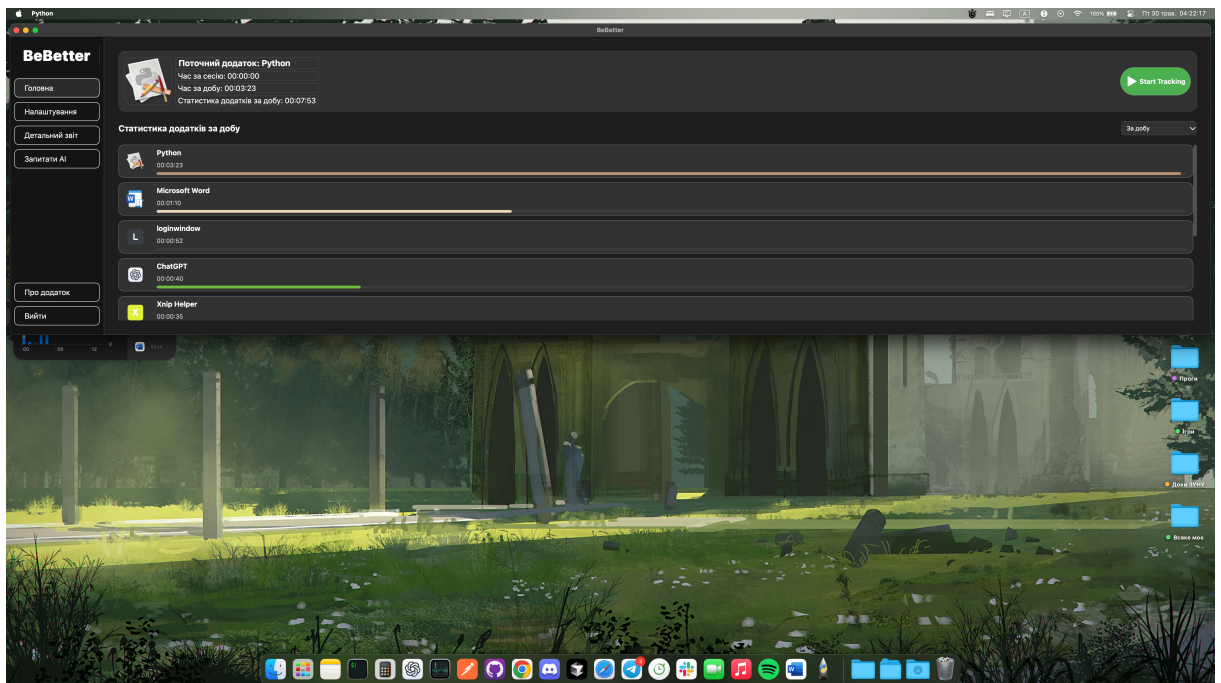


Рисунок 3.17 – Приклад швидкого розміщення вікна через traffic lights

Якщо натиснути на зелену кнопку в traffic lights – додаток стане відокремленим від робочого стола додатком, та займе собою весь екран, приховавши навіть dock. Це зображено на рисунку 3.18.

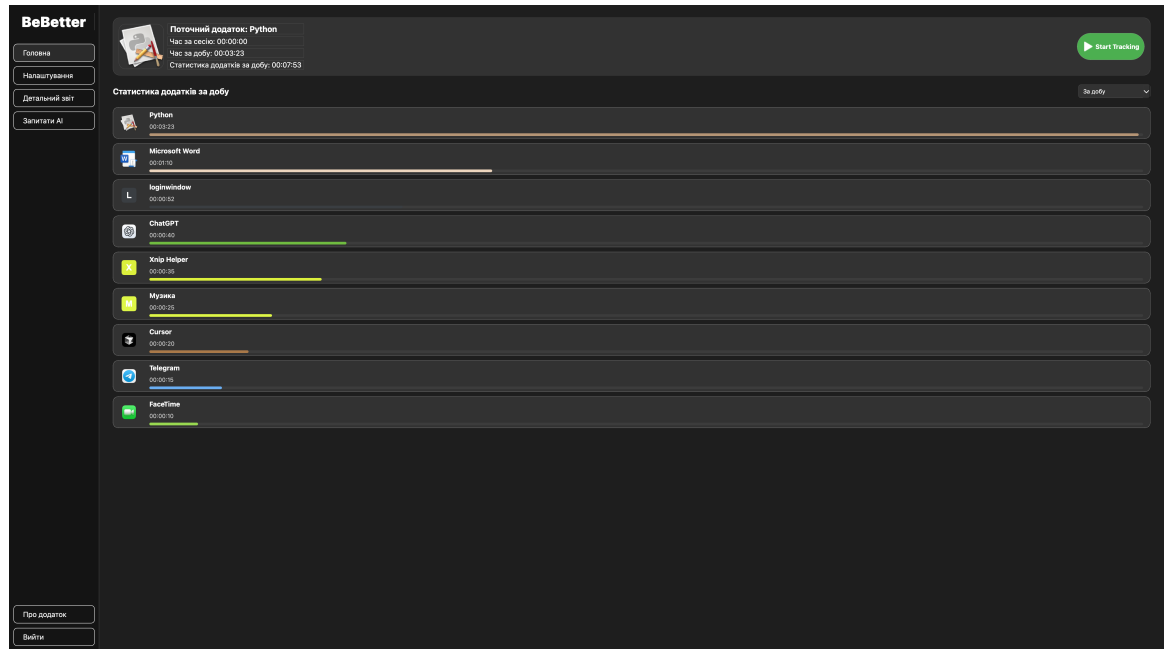


Рисунок 3.18 – BeBetter в повноекранному режимі

Наступним етапом тестування було блокування екрану під час активного відстеження. І тут знайшлась помилка, вона полягала в тому, що програма являє собою окремий Python-процес, і йому неважливо, чи екран заблокований, чи ні, і вона всерівно продовжувала записувати час використання додатку. Це було виправлено наступним чином:

- 1) Додане виявлення блокування екрану:
 - Реалізована функцію перевірки стану блокування через CoreGraphics API;
 - Створена змінна для збереження часу блокування екрану;
- 2) Реалізовано збереження стану відстеження:
 - При блокуванні екрану зберігається, чи було активне відстеження;
 - Якщо відстеження було активне, воно зупиняється його з причиною "Screen locked";
- 3) Доданий показ AFK-вікна після розблокування:

- Створений окремий слот `show_afk_after_unlock_slot`;
- Після розблокування обчислюється тривалість блокування;
- Показується AFK-вікно тільки якщо екран був заблокований під час

активного відстеження;

4) Модифікований AFK-діалог:

- Додане відображення тривалості блокування екрану у AFK-вікні;
- Автоматично заповнюється поле текстом "Блокування екрану";
- Змінений заголовок вікна на "Виявлено блокування екрану";

5) Ведеться облік часу блокування як AFK:

- Час блокування зберігається у базі даних як AFK-час;
- Для програми "loginwindow" записується тривалість блокування.

Після цих змін програма стала працювати коректно. Наступний пункт перевірки – спроба взаємодії з інтелектуальним асистентом без токена від OpenAI. Сам токен зберігається в .env-файлі, тому він надійно захищений. Для проведення тестів ключ був тимчасово видалений.

На рисунку 3.19 зображене вікно взаємодії інтелектуальним асистентом без ключа OpenAI API, де видно попередження зверху вікна, що API ключ не налаштовано.

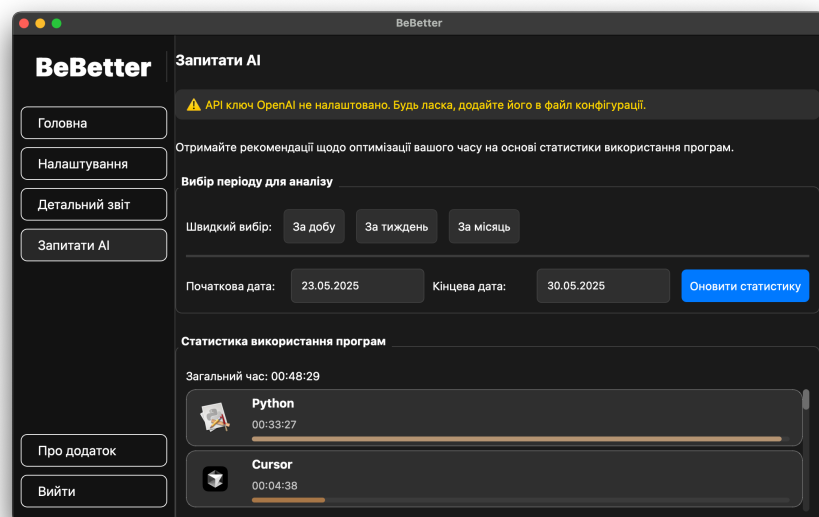


Рисунок 3.19 – Вікно взаємодії інтелектуальним асистентом без ключа OpenAI API

На рисунку 3.20 зображена кнопка «Відправити», яка неактивна.

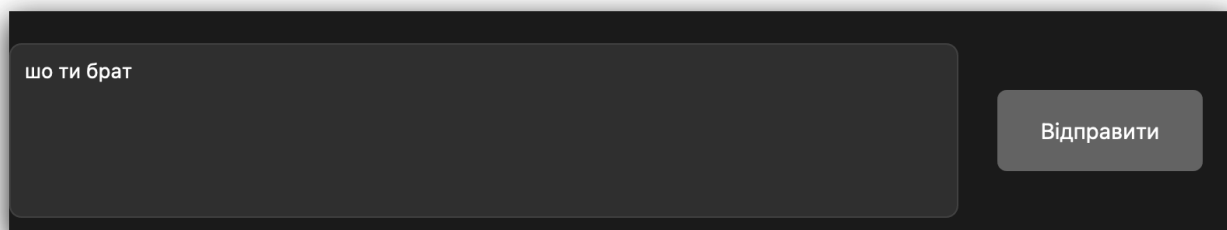


Рисунок 3.20 – Неактивна кнопка «Відправити».

Останній пункт тестування – спроба запитати інтелектуальним асистентом щось не по темі. В його контексті міститься інформація, що він чат-бот, який повинен відповідати лише на питання, зв'язані з витратою часу та його оптимізації.

На рисунку 3.21 зображена його реакція на відверто некоректне запитання. Видно, що промпт передається в контекст інтелектуальним асистентом правильно, та відповідати на невідповідні запити він не буде. На «зверхність» він відповів дружньо та тактовно, це теж одна з умов його промпту.

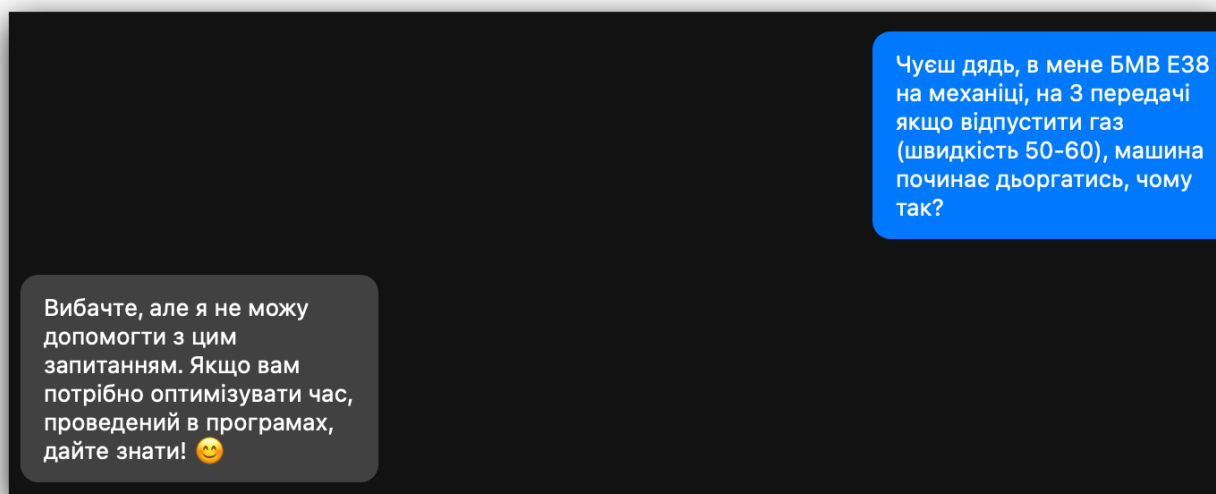


Рисунок 3.21 – Реакція інтелектуальним асистентом на некоректне запитання

З результатів усіх проведених тестів видно, що в додатку передбачені багато сценаріїв роботи користувача, та на навіть неочевидні речі додаток реагує коректно.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було виконано всі поставлені завдання:

1. Проаналізовано існуючі рішення для відстеження часу користувача за комп'ютером, що дозволило виявити їхні переваги та обмеження. Встановлено, що більшість наявних застосунків мають обмежену безкоштовну функціональність, недостатню інтеграцію з інтелектуальними системами та складний та неінтуїтивний інтерфейс користувача, в основному орієнтовані на корпоративний сегмент.

2. Сформовано концепцію побудови застосунку "BeBetter", який дозволяє користувачеві не лише відслідковувати проведений час за ПК, але й оптимізувати його завдяки консультаціям інтелектуального асистента. Розроблена концепція поєднує функціональність трекінгу часу з аналітичними можливостями та персоналізованими рекомендаціями.

3. Розроблено архітектуру застосунку, яка забезпечує продуктивність, стабільність та сучасний дизайн. Модульна структура програми дозволяє легко розширювати функціонал та портувати застосунок на інші операційні системи.

4. Реалізовано повноцінний застосунок для відстеження часу за ПК з інтуїтивним інтерфейсом, що включає: єдину кнопку початку/зупинки трекінгу з динамічною зміною кольору та тексту, автоматичне призупинення відстеження при блокуванні екрану, зручну візуалізацію статистики використання часу та інтеграцію з інтелектуальним асистентом.

5. Проведено всебічне тестування застосунку в різних сценаріях взаємодії, включаючи функціональне тестування основних компонентів, тестування інтерфейсу, тестування крайових випадків та інтеграційне тестування взаємодії всіх модулів системи.

Виконана кваліфікаційна робота демонструє ефективність обраного підходу до створення застосунку для відстеження та оптимізації часу користувача за комп'ютером. Розроблений застосунок "BeBetter" відповідає

сучасним вимогам до програмного забезпечення, поєднуючи функціональність, естетичний дизайн та інтелектуальні можливості, що робить його корисним інструментом для підвищення продуктивності користувачів ПК.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Freelance Platforms Market Report 2025-2030: Industry Growth Driven by Shift to Gig Economy and Technology Advancements. URL: <https://www.globenewswire.com/news-release/2025/04/23/3066710/28124/en/Freelance-Platforms-Market-Report-2025-2030-Industry-Growth-Driven-by-Shift-to-Gig-Economy-and-Technology-Advancements.html> (дата звернення: 18.03.2025)
2. Haris, A. Art of Delay: Procrastination in Relation with the Hybrid Work Environment and Task Performance. International Journal of Innovative Research in Science Engineering and Technology, 2023. – ISSN No:-2456-2165. (дата звернення: 19.03.2025)
3. 12 найкращих інструментів моніторингу персоналу. URL: <https://clevercontrol.com/uk/12-best-employee-monitoring-tools/> (дата звернення: 20.03.2025)
4. Peewee Documentaion. URL: <https://docs.peewee-orm.com/en/latest/> (дата звернення: 20.03.2025)
5. Загальний регламент про захист даних. URL: https://uk.wikipedia.org/wiki/Загальний_регламент_про_захист_даних
6. Apple Human Interface Guidlelines. URL: <https://developer.apple.com/design/human-interface-guidelines>
7. Social media use in 2021. URL: <https://www.pewresearch.org/internet/2021/04/07/social-media-use-in-2021/>
8. Exploring the cult of productivity. URL: <https://medium.com/@ashely.crouch/exploring-the-cult-of-productivity-part-one-778542b98789>
9. A free time tracker to share the number of hours worked in real life?. URL: https://www.reddit.com/r/freelance/comments/10gvxeq/a_free_time_tracker_to_share_the_number_of_hours/

10. Is it time to quit when your company suddenly implements the use of a time tracker?. URL: https://www.reddit.com/r/buhaydigital/comments/10wnxvg/is_it_time_to_quit_when_your_company_suddenly/
11. Why are applications, on average, increasingly requiring more storage space?. URL: https://www.reddit.com/r/explainlikeimfive/comments/10jsmtz/eli5_why_are_applications_on_average_increasingly/
12. Інвойс. URL: <https://uk.wikipedia.org/wiki/Інвойс>
13. Toptal. URL: <https://www.toptal.com>
14. Toptracker. URL: <https://www.toptal.com/tracker>
15. Clockify. URL: <https://clockify.me>
16. ActivityWatch. URL: <https://activitywatch.net/>
17. RescueTime. URL: <https://www.rescuetime.com/>
18. TogglTrack. URL: <https://toggl.com/track/>
19. ManicTime. URL: <https://www.manictime.com/>
20. macOS Sequoia. URL: <https://www.apple.com/macOS/>
21. Apple Macbook Air M1. URL: <https://support.apple.com/kb/SP825>
22. Boomers Versus Zoomers and the WFH Conundrum. URL: <https://logicalconsulting.co.nz/boomers-versus-zoomers-and-the-wfh-conundrum/>
23. Євдокимова Д., Качарова В. Прокрастинація як фактор суб'єктивного сприйняття часу “Вісник”. 2019. № 1 (39). С. 18–22. URL: <https://herald.kibit.edu.ua/index.php/visnyk/article/download/65/60/>
24. Справи на потім: що таке прокрастинація та як її здолати. URL: <https://firtka.if.ua/blog/view/spravi-na-potim-shcho-takeprokrastinatsiia-ta-iak-yiyi-zdolati>
25. Фриланс у 2024 році: тенденції та перспективи. URL: <https://etcetera.kiev.ua/uk/blog/frilans-u-2024-roczyi-tendencziyi-ta-perspektivi-oglyad-zvitu-vid-ruul/>

26. Пасічник М.М., Биковий П.Є. Програмна система відстеження діяльності користувача за комп'ютером з наданням рекомендацій з тайм-менеджменту. *«Інтелектуальні інформаційні технології в прикладних дослідженнях» (ИТАР-2025)*: збірник тез доповідей студентської науково-практичної конференції (Тернопіль, 27-29 травня 2025 р.) Тернопіль: ЗУНУ, 2025. С. 202-212.

27. Комар М.П., Саченко А.О., Васильків Н.М., Гладій Г.М., Коваль В.С., Ліп'яніна-Гончаренко Х.В. Методичні рекомендації до виконання кваліфікаційної роботи з освітньо-професійної програми «Комп'ютерні науки» спеціальності 122 «Комп'ютерні науки» за першим (бакалаврським) рівнем вищої освіти. Тернопіль: ЗУНУ, 2024. 52 с.

Додаток А

Код для перевірки даних з БД

```
def main():
    """
    Головна функція скрипта
    """
    try:
        logger.info("Запуск тестування статистики БД")

        # Отримуємо статистику за різні періоди
        day_stats = TrackingRecord.get_time_summary_for_today()
        week_stats = TrackingRecord.get_time_summary_for_week()
        month_stats = TrackingRecord.get_time_summary_for_month()

        # Виводимо дані для кожного періоду
        print_stats(day_stats, "день")
        print_stats(week_stats, "тиждень")
        print_stats(month_stats, "місяць")

        # Виводимо додаткову інформацію про часові періоди
        today = datetime.datetime.now().date()
        print("\n===== Інформація про періоди =====")
        print(f"Поточна дата: {today}")

        # Інформація про день
        print(f"День: {today}")

        # Інформація про тиждень
        start_of_week = today - datetime.timedelta(days=today.weekday())
        end_of_week = start_of_week + datetime.timedelta(days=6)
        print(f"Тиждень: {start_of_week} - {end_of_week}")

        # Інформація про місяць
        start_of_month = today.replace(day=1)
        if today.month == 12:
            next_month = today.replace(year=today.year + 1, month=1, day=1)
        else:
            next_month = today.replace(month=today.month + 1, day=1)
        end_of_month = next_month - datetime.timedelta(days=1)
        print(f"Місяць: {start_of_month} - {end_of_month}")

        logger.info("Тестування завершено успішно")

    except Exception as e:
        logger.error(f"Помилка при тестуванні: {e}")
        import traceback

        logger.error(traceback.format_exc())
        return 1

    return 0
```

Додаток Б

Код для отримання активного вікна

```
def get_active_window(self):
    """
    Get the title of the currently active window.
    """
    try:
        if platform.system() == "Windows":
            import win32gui # pyright: ignore[reportMissingImports]

            hwnd = win32gui.GetForegroundWindow()
            return win32gui.GetWindowText(hwnd)
        # macOS ?
        elif platform.system() == "Darwin":
            from AppKit import NSWorkspace # pyright: ignore[reportMissingImports]

            active_app = NSWorkspace.sharedWorkspace().frontmostApplication()
            return active_app.localizedName()
        else:
            return None
    except Exception as e:
        logger.error(f"Помилка при отриманні активного вікна: {e}")
        return None
```

Додаток В

Код для відстеження AFK

```
def detect_inactivity(self):
    """
    Detect user inactivity.
    """
    self.last_activity_time = time.time()
    self.screen_locked = False
    self.screen_lock_time = None # Час блокування екрану

    def on_activity(*args):
        """
        Called when user activity is detected
        """
        self.last_activity_time = time.time()

# Функція для перевірки блокування екрану на macOS
def is_screen_locked():
    try:
        # Використовуємо CoreGraphics для виявлення стану блокування екрану
        import CoreFoundation
        import objc
        from ApplicationServices import CoreGraphics

        # Отримуємо дані про поточну сесію
        session_dict = CoreGraphics.CGSessionCopyCurrentDictionary()

        if session_dict:
            # Ключі, які свідчать про блокування екрану
            # CGSessionScreenIsLocked – True якщо екран заблокований
            # kCGSessionOnConsoleKey – False якщо сеанс не на консолі (заблокований)
            screen_locked = session_dict.get("CGSessionScreenIsLocked", False)
            on_console = session_dict.get("kCGSessionOnConsoleKey", True)

            return screen_locked or not on_console

        return False
    except Exception as e:
        logger.error(f"Помилка при перевірці стану блокування екрану: {e}")
        return False

# Start mouse and keyboard listeners
try:
    mouse_listener = mouse.Listener(
        on_move=on_activity, on_click=on_activity, on_scroll=on_activity
    )
    keyboard_listener = keyboard.Listener(
        on_press=on_activity, on_release=on_activity
    )
    mouse_listener.daemon = True
    keyboard_listener.daemon = True
    mouse_listener.start()
    keyboard_listener.start()
```

```

while True:
    try:
        # Перевірка, чи додаток все ще працює
        if not QApplication.instance():
            logger.info(
                "Додаток закрито, зупиняємо потік виявлення неактивності"
            )
            break

        # Перевіряємо стан блокування екрану кожні 5 секунд
        current_locked_state = is_screen_locked()

        # Якщо стан змінився з розблокованого на заблокований
        if current_locked_state and not self.screen_locked:
            logger.info("Виявлено блокування екрану")
            self.screen_locked = True
            self.screen_lock_time = (
                time.time()
            ) # Запам'ятовуємо час блокування

            # Зберігаємо стан відстеження для відновлення після розблокування
            if self.tracking:
                self._was_tracking_before_afk = True
                # Зупиняємо відстеження
                self.stop_tracking(reason="Screen locked")
            else:
                self._was_tracking_before_afk = False

            # Відновлюємо відстеження при розблокуванні
            elif not current_locked_state and self.screen_locked:
                logger.info("Екран розблоковано")
                unlock_time = time.time()
                lock_duration = 0

                # Обчислюємо тривалість блокування, якщо маємо час блокування
                if self.screen_lock_time:
                    lock_duration = unlock_time - self.screen_lock_time
                    logger.info(
                        f"Екран був заблокований {lock_duration:.2f} секунд"
                    )

                self.screen_locked = False
                was_tracking = self._was_tracking_before_afk

            # Якщо екран був заблокований під час активного відстеження,
            # показуємо AFK-вікно і передаємо тривалість блокування
            if was_tracking:
                logger.info(
                    "Показуємо AFK-вікно після розблокування екрану"
                )
            # Запам'ятовуємо тривалість блокування для запису в лог AFK

```

```

        if lock_duration > 0:
            self.lock_duration = lock_duration

        # Показуємо AFK-вікно через головний потік
        QMetaObject.invokeMethod(
            self,
            "show_afk_after_unlock_slot",
            Qt.ConnectionType.QueuedConnection,
        )

    # Перевіряємо, чи ввімкнено виявлення неактивності
    if hasattr(self.config, "afk_enabled") and self.config.afk_enabled:
        current_time = time.time()
        inactivity_seconds = current_time - self.last_activity_time
        if (
            inactivity_seconds > self.config.get_inactivity_timeout()
            and self.tracking
            and not self.screen_locked # Не реагуємо на неактивність, якщо екран заблокований
        ):
            logger.info("Inactivity detected")

            # Зберігаємо стан відстеження для відновлення після AFK
            self.was_tracking_before_afk = True

            # Зупиняємо відстеження
            self.stop_tracking(reason="AFK")

            # Викликаємо сигнал AFK через QMetaObject для безпеки потоків
            QMetaObject.invokeMethod(
                self,
                "inactivity_detected_slot",
                Qt.ConnectionType.QueuedConnection,
            )

    except Exception as e:
        logger.error(f"Помилка у потоці виявлення неактивності: {e}")

        time.sleep(1)
    except Exception as e:
        logger.error(f"Критична помилка у потоці виявлення неактивності: {e}")
    finally:
        logger.info("Потік виявлення неактивності завершив роботу")

```

Додаток Г

Код для взаємодії зі інтелектуальним асистентом

```
def request_chatgpt_response(self, user_message, stats):
    """
    Відправляє запит до OpenAI API та отримує відповідь
    """
    Args:
        user_message (str): Повідомлення користувача
        stats (dict): Статистика використання програм
    """
    try:
        # Перевіряємо наявність API ключа
        if not self.config or not self.config.openai_api_key:
            self.add_message_to_chat(
                "API ключ OpenAI не налаштовано. Будь ласка, додайте його в файл конфігурації.",
                is_user=False,
            )
            self.send_button.setDisabled(False)
            self.send_button.setText("Відправити")
            return

        # Базовий системний промт для чатбота
        system_prompt = "Ти ШІ-асистент для оптимізації часу проведеного в програмах. Будь максимально щирим і ввічливим. Відповідай лише на рахунок часу і його оптимізації. Можеш використовувати емої. Не відповідай багато, лише основне."

        # Форматуємо статистику у зручний для чатбота формат
        stats_text = f"Статистика використання програм за період {stats.get('Період', 'невідомий')}: \n"
        stats_text += f"Загальний час: {stats.get('Загальний час', '00:00:00')} \n\n"
        stats_text += "Програми: \n"
        for program, time_spent in stats.get("Програми", {}).items():
            stats_text += f"- {program}: {time_spent} \n"

        # Підготуємо повідомлення для API
        messages = [
            {"role": "system", "content": system_prompt},
            {
                "role": "user",
                "content": f"Ось моя статистика використання програм: \n\n{stats_text}",
            },
        ]

        # Додаємо всі історії розмови
        for msg in self.chat_history:
            messages.append(msg)

        # Додаємо поточне повідомлення
        messages.append({"role": "user", "content": user_message})

        # Викликаємо API OpenAI
        url = "https://api.openai.com/v1/chat/completions"
        headers = {
            "Content-Type": "application/json",
            "Authorization": f"Bearer {self.config.openai_api_key}",
        }

        data = {
            "model": "gpt-4o-mini",
            "messages": messages,
            "temperature": 0.7,
            "max_tokens": 1000,
        }

        # Виконуємо запит
        response = requests.post(url, headers=headers, json=data)
        response.raise_for_status() # Перевіряємо на помилки HTTP
        result = response.json()

        # Отримуємо відповідь від API
        if "choices" in result and len(result["choices"]) > 0:
            assistant_response = result["choices"][0]["message"]["content"]
            self.add_message_to_chat(assistant_response, is_user=False)
        else:
            self.add_message_to_chat(
                "Не вдалося отримати відповідь від AI. Спробуйте пізніше.",
                is_user=False,
            )

    except requests.exceptions.RequestException as e:
        error_message = f"Помилка підключення до OpenAI API: {str(e)}"
        logger.error(error_message)
        self.add_message_to_chat(error_message, is_user=False)
    except Exception as e:
        error_message = f"Помилка при отриманні відповіді від AI: {str(e)}"
        logger.error(error_message)
        self.add_message_to_chat(error_message, is_user=False)
    finally:
        # Активуємо кнопку відправки
        self.send_button.setDisabled(False)
        self.send_button.setText("Відправити")
```

```
        data = {
            "model": "gpt-4o-mini",
            "messages": messages,
            "temperature": 0.7,
            "max_tokens": 1000,
        }

        # Виконуємо запит
        response = requests.post(url, headers=headers, json=data)
        response.raise_for_status() # Перевіряємо на помилки HTTP
        result = response.json()

        # Отримуємо відповідь від API
        if "choices" in result and len(result["choices"]) > 0:
            assistant_response = result["choices"][0]["message"]["content"]
            self.add_message_to_chat(assistant_response, is_user=False)
        else:
            self.add_message_to_chat(
                "Не вдалося отримати відповідь від AI. Спробуйте пізніше.",
                is_user=False,
            )

    except requests.exceptions.RequestException as e:
        error_message = f"Помилка підключення до OpenAI API: {str(e)}"
        logger.error(error_message)
        self.add_message_to_chat(error_message, is_user=False)
    except Exception as e:
        error_message = f"Помилка при отриманні відповіді від AI: {str(e)}"
        logger.error(error_message)
        self.add_message_to_chat(error_message, is_user=False)
    finally:
        # Активуємо кнопку відправки
        self.send_button.setDisabled(False)
        self.send_button.setText("Відправити")
```

Додаток Д

Інтерфейси користувача «TogglTrack» та «Timing»

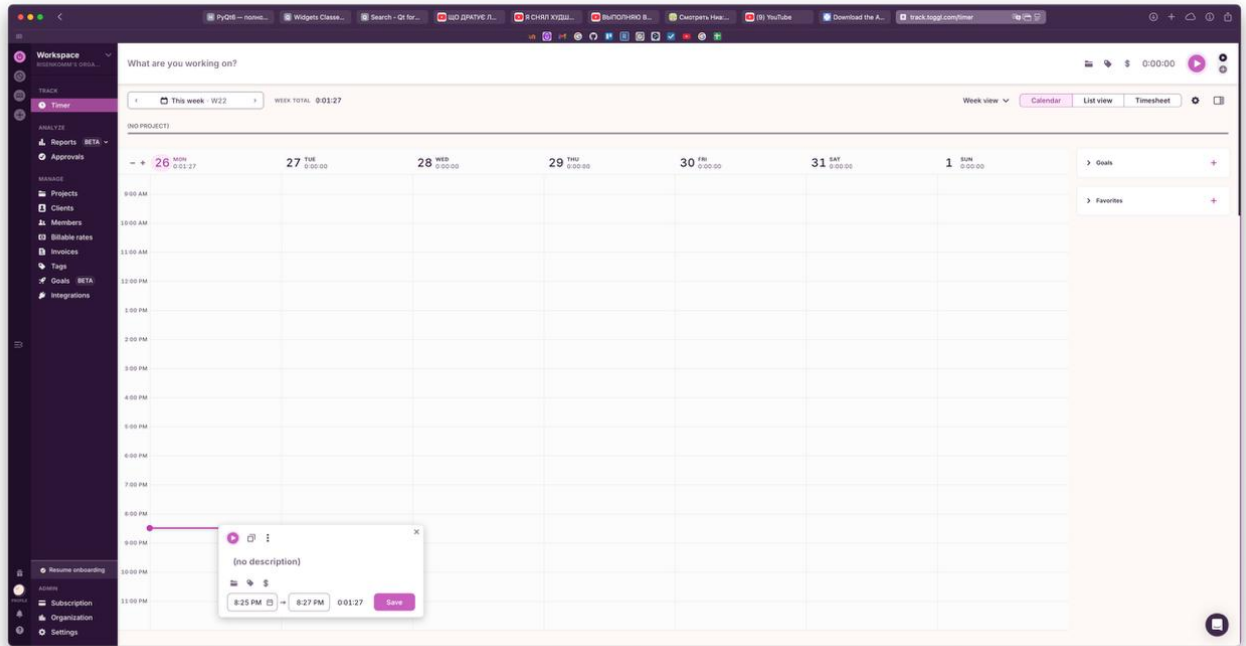


Рисунок Д.1 – Інтерфейс користувача застосунку «TogglTrack»

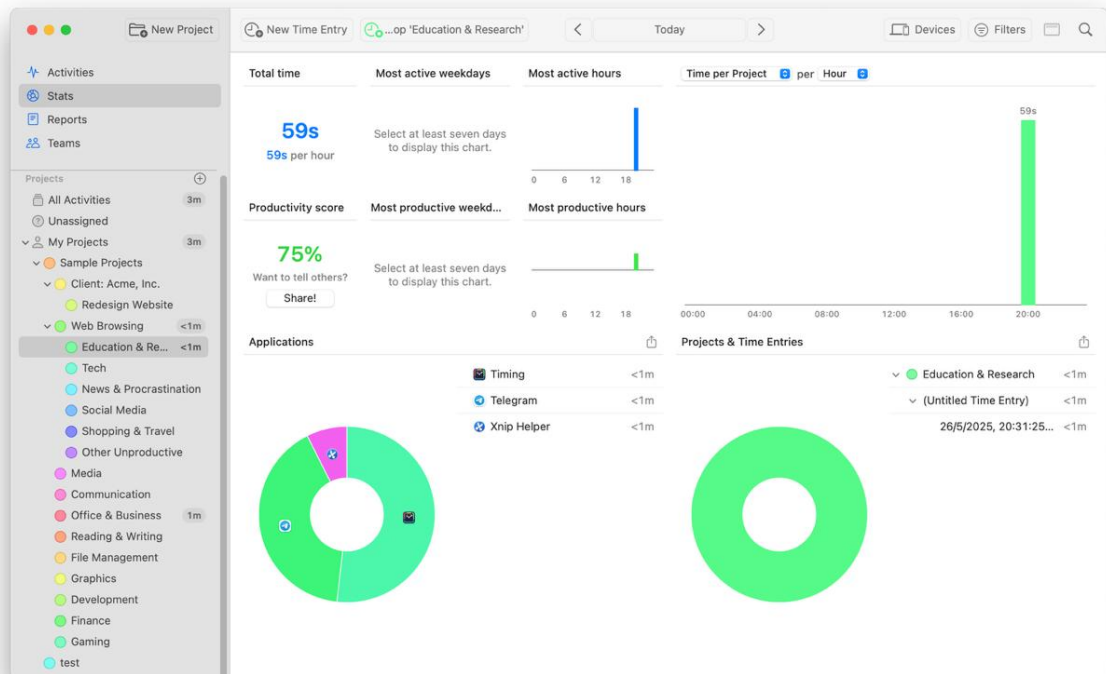


Рисунок Д.2 – Інтерфейс користувача застосунку «Timing»

Додаток Е

Приклад даних з БД у форматі JSON

```
{
  "id": 1146,
  "start_time": "2025-06-09T01:24:24.230988",
  "end_time": "2025-06-09T01:24:29.239054",
  "duration_seconds": 5.008066,
  "reason": "Продовження сесії",
  "message": null,
  "programs_tracked": {
    "Cursor": 5.009738922119141
  }
},
{
  "id": 1145,
  "start_time": "2025-06-09T01:24:20.775130",
  "end_time": "2025-06-09T01:24:25.812549",
  "duration_seconds": 5.037419,
  "reason": "Продовження сесії",
  "message": null,
  "programs_tracked": {
    "Python": 5.038904905319214
  }
},
{
  "id": 1144,
  "start_time": "2025-06-09T01:24:19.216514",
  "end_time": "2025-06-09T01:24:24.229313",
  "duration_seconds": 5.012799,
  "reason": "Продовження сесії",
  "message": null,
  "programs_tracked": {
    "Python": 5.01488733291626
  }
}
```



```

},
{
  "id": 1143,
  "start_time": "2025-06-09T01:24:15.725502",
  "end_time": "2025-06-09T01:24:20.773648",
  "duration_seconds": 5.048146,
  "reason": "Продовження сесії",
  "message": null,
  "programs_tracked": {
    "Python": 5.052229166030884
  }
},
{
  "id": 1142,
  "start_time": "2025-06-09T01:24:14.111401",
  "end_time": "2025-06-09T01:24:19.214429",
  "duration_seconds": 5.103028,
  "reason": "Продовження сесії",
  "message": null,
  "programs_tracked": {
    "Python": 5.104350805282593
  }
},
{
  "id": 1141,
  "start_time": "2025-06-09T01:24:10.712089",
  "end_time": "2025-06-09T01:24:15.721428",
  "duration_seconds": 5.009339,
  "reason": "Продовження сесії",
  "message": null,
  "programs_tracked": {
    "Python": 5.011446952819824
  }
},
{
  "id": 1140,

```

```
"start_time": "2025-06-09T01:24:09.106739",  
"end_time": "2025-06-09T01:24:14.110073",  
"duration_seconds": 5.003334,  
"reason": "Продовження сесії",  
"message": null,  
"programs_tracked": {  
    "Python": 5.006594181060791  
}  
}
```

Додаток Ж
Копії опублікованих результатів

Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління



ЗБІРНИК ТЕЗ ДОПОВІДЕЙ

Студентської науково-практичної конференції
ІНТЕЛЕКТУАЛЬНІ ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ В ПРИКЛАДНИХ
ДОСЛІДЖЕННЯХ
(ІТAR-2025)

27-29 травня 2025 року

Тернопіль
2025

Збірник тез доповідей студентської науково-практичної конференції «Інтелектуальні інформаційні технології в прикладних дослідженнях» (ІТАР-2025). (Тернопіль, 27-29 травня 2025 року). Тернопіль: ЗУНУ, 2025. 372 с.

До збірника увійшли тези доповідей учасників студентської науково-практичної конференції «Інтелектуальні інформаційні технології в прикладних дослідженнях» (ІТАР – 2025), що відбувалась у рамках AI Week на базі кафедри інформаційно-обчислювальних систем і управління Західноукраїнського національного університету. Мета конференції — об'єднати науковців, освітян, представників IT-бізнесу та здобувачів освіти для обміну досвідом, презентації сучасних досліджень і впровадження інноваційних рішень у сфері комп'ютерних наук, штучного інтелекту та суміжних галузей.

За зміст наукових праць та достовірність наведених фактологічних і статистичних матеріалів відповідальність несуть автори публікацій та їхні наукові керівники. У збірнику зберігається стилістика та орфографія авторів матеріалів.

Склад організаційного комітету конференції

Керівництво оргкомітету

Мирослав Комар, д.т.н., професор, професор кафедри інформаційно-обчислювальних систем і управління

Христина Ліп'яніна-Гончаренко, д.т.н., доцент, доцент кафедри інформаційно-обчислювальних систем і управління

Надія Васильків, к.т.н., в.о. завідувача кафедри інформаційно-обчислювальних систем і управління

Члени програмного комітету

Василь Коваль, к.т.н., доцент, доцент кафедри інформаційно-обчислювальних систем і управління, заступник декана факультету комп'ютерних інформаційних технологій

Олександр Осолінський, к.т.н., доцент, доцент кафедри інформаційно-обчислювальних систем і управління

Павло Биковий, к.т.н., доцент, доцент кафедри інформаційно-обчислювальних систем і управління

Діана Загородня, к.т.н., доцент, доцент кафедри інформаційно-обчислювальних систем і управління

Ігор Майків, к.т.н., доцент кафедри інформаційно-обчислювальних систем і управління

Члени оргкомітету

Андрій Івасечко, викладач кафедри інформаційно-обчислювальних систем і управління

Дмитро Дюг, викладач кафедри інформаційно-обчислювальних систем і управління

Христина Юрків, студентка ФКІТ ЗУНУ, технік лабораторії з проблем інформаційних технологій

Мар'яна Соя, студентка ФКІТ ЗУНУ, лаборант кафедри інформаційно-обчислювальних систем і управління

Микола Телька, студент ФКІТ ЗУНУ, лаборант кафедри інформаційно-обчислювальних систем і управління, студентський декан факультету комп'ютерних інформаційних технологій

© Кафедра інформаційно-обчислювальних систем і управління ЗУНУ

Насреддін Діана, Лип'яніна-Гончаренко Христина	197
МОДУЛЬ КОНТЕНТНОГО ФІЛЬТРУВАННЯ ДЛЯ СИСТЕМИ РЕКОМЕНДАЦІЙ ІЖІ З ВИКОРИСТАННЯМ TF-IDF І КОСИНУСНОЇ ПОДІБНОСТІ	197
Олексішин Євген, Лип'яніна-Гончаренко Христина	202
МОДУЛЬ ОЗВУЧЕННЯ ТЕКСТУ МАЛЬОПИСІВ	202
Паньчишин Андрій, Дорош Віталій	206
МОДУЛЬ КЛАСИФІКАЦІЇ НОВИННИХ СТАТЕЙ ЗА КАТЕГОРІЯМИ НА ОСНОВІ FastText	206
Пасічник Максим, Биковий Павло	208
ПРОГРАМНА СИСТЕМА ВІДСТЕЖЕННЯ ДІЯЛЬНОСТІ КОРИСТУВАЧА ЗА КОМП'ЮТЕРОМ З НАДАННЯМ РЕКОМЕНДАЦІЙ З ТАЙМ-МЕНЕДЖМЕНТУ	208
Питчак Остап	212
КОНЦЕПЦІЯ ПОБУДОВИ ПРОГРАМНОГО МОДУЛЯ ДЛЯ РЕКОМЕНДАЦІЙ КУРСІВ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ	212
Політичка Андрій, Лип'яніна-Гончаренко Христина	214
ІНТЕЛЕКТУАЛЬНИЙ МЕТОД ПЕРСОНАЛІЗАЦІЇ ВИВЧЕННЯ ІНОЗЕМНОЇ МОВИ	214
Проців Денис, Гладій Григорій	218
ПРОГРАМНИЙ МОДУЛЬ ДЛЯ ПОБУДОВИ ІНТЕРАКТИВНИХ ГЕНЕАЛОГІЧНИХ ДЕРЕВ У ВЕБ-СЕРЕДОВИЩІ	218
Раківський Роман, Лип'яніна-Гончаренко Христина	220
СИСТЕМА МОНІТОРИНГУ ВІДГУКІВ НА ВЕБСАЙТІ НА ОСНОВІ МАШИННОГО НАВЧАННЯ	220
Савчук Дмитро, Биковий Павло	225
AR-КВЕСТ ПО УНІВЕРСИТЕТУ З ВИКОРИСТАННЯМ ГЕОЛОКАЦІЇ	225
Самборович Олександра	228
ВИКОРИСТАННЯ СЕРЕДОВИЩ З ШІ, GAMMA ТА CANVA – В ОСВІТНЬОМУ ПРОЦЕСІ: НАВЧАННЯ, ТА ЗАХИСТУ(ПРЕДСТАВЛЕННЯ) ДИПЛОМНИХ РОБІТ ЗДОБУВАЧАМИ ОСВІТИ	228
Собчук Юлія, Лип'яніна-Гончаренко Христина	230
ІНТЕЛЕКТУАЛЬНА СИСТЕМА ВИЯВЛЕННЯ ФЕЙКОВИХ НОВИН В УКРАЇНСЬКОМУ МЕДІАПРОСТОРІ НА ОСНОВІ NLP	230
Созанський Андрій, Лип'яніна-Гончаренко Христина	235
ІНТЕЛЕКТУАЛЬНИЙ МОДУЛЬ ПРОГНОЗУВАННЯ ФІНАНСОВИХ ДАНИХ ІЗ ВИКОРИСТАННЯМ LSTM І ЗАСОБІВ WPF	235

Пасічник Максим
студент групи КН-41
risenkomm@gmail.com

Биковий Павло
к.т.н., доцент
pb@wunu.edu.ua

Західноукраїнський національний університет
Тернопіль, Україна

ПРОГРАМНА СИСТЕМА ВІДСТЕЖЕННЯ ДІЯЛЬНОСТІ КОРИСТУВАЧА ЗА КОМП'ЮТЕРОМ З НАДАННЯМ РЕКОМЕНДАЦІЙ З ТАЙМ-МЕНЕДЖМЕНТУ

Сучасна вища освіта вимагає від студентів ефективної самостійної роботи [1]. Зростання дистанційного навчання посилює потребу в тайм-менеджменті як ключовій навичці результативного навчання [1, 3]. Електронні інструменти тайм-менеджменту сприяють розвитку дисципліни, планування та самоорганізації [1, 4]. Дослідження показують, що цифрові трекери часу допомагають завершувати завдання вчасно, оцінювати внесок команди та формувати самодисципліну [1, 2]. Візуалізація витраченого часу підвищує усвідомленість студентів щодо власної продуктивності [1].

Більшість існуючих рішень орієнтовані на бізнес або потребують підключення до зовнішніх сервісів, що ускладнює їхнє використання для індивідуальних користувачів. У цій роботі розроблено кросплатформений таймтрекер для локального використання, який дозволяє користувачу ефективно відстежувати час за комп'ютером і усуває недоліки наявних систем.

Метою статті є аналіз впливу використання інструментів тайм-трекінгу на самоорганізацію та продуктивність студентів під час виконання навчальних проєктів. Для досягнення цієї мети здійснено огляд сучасних таймтрекерів та розроблено власний застосунок під назвою BeBetter.

На етапі аналізу було розглянуто можливості сучасних програм для відстеження часу, що сприяють самоорганізації студентів [1]. Зокрема, досліджено як потужні корпоративні системи обліку робочих годин, так і простіші персональні трекери, придатні для індивідуального використання студентами [1]. Наприклад, Time Doctor (орієнтований на командне планування в проєктах) та сервіс TrackingTime для відстеження задач пропонують схожий базовий функціонал – сповіщення, розклад, списки завдань, звіти – але відрізняються можливостями співпраці та управління ресурсами (Time Doctor підтримує інтегровану командну роботу, тоді як TrackingTime більше фокусується на індивідуальному трекінгу) [1]. Водночас особисті продуктивні застосунки на кшталт RescueTime, Toggl чи Pomodoro-таймерів автоматично фіксують, скільки часу користувач присвячує різним програмам і сайтам, надаючи детальні звіти та діаграми щодо використання робочого часу [1]. Такі



Рисунок 1 – Алгоритм роботи модуля відстеження активності

Розроблений застосунок BeBetter демонструє практичну реалізацію концепції локального таймтрекера як засобу підвищення самоорганізації студентів. Відкрите програмне забезпечення надає вагомому перевагу – можливість гнучкого допрацювання та інтеграції нового функціоналу відповідно до потреб користувача чи навчального закладу. На відміну від комерційних продуктів, котрі можуть вимагати платної підписки або надсилати особисті дані на сервер, локальне рішення з відкритим кодом гарантує конфіденційність: всі дані про активність залишаються на пристрої студента. Це особливо актуально з огляду на академічне середовище, де питання захисту персональних даних є критично важливими, а також є потреба адаптувати інструменти під специфіку навчального процесу. Крім того, відсутність залежності від інтернет-з'єднання дозволяє студентам використовувати трекер у будь-який час і в будь-якому місці

(наприклад, під час офлайн-підготовки до проєкту). З точки зору самоорганізації, такі автономні рішення розвивають у молоді почуття відповідальності за власний час: студент сам контролює та аналізує свій прогрес, що психологічно стимулює його дотримуватися плану роботи. Важливо зазначити, що таймтрекери не є панацеєю, але їх грамотне використання в навчальному процесі здатне створити середовище, в якому студенти навчаються більш ефективно розподіляти час між завданнями та менше прокрастинують.

Список використаних джерел

1. Sorokina O.A., Odarich I.N., Vaganova O.I., Bulaeva M.N., Lapshova A.V. (2020). *Timetracker capabilities in student project activities*. Propósitos y Representaciones, 9(SPE1), e819. DOI: 10.20511/pyr2021.v9nSPE1.819.
2. Koshechko N.V. (2018). *Innovations from educational discipline "Pedagogical conflictology" in professional preparation of students*. Scientific Vector of the Balkans, 1, 59–63.
3. Oros I.I. (2018). *The role of international connections in the development of the adult education system*. Humanitarian Balkan Research, 1, 57–59.
4. Osadchenko I.I. (2019). *Key concepts of situational training technology in preparing future teachers*. Scientific Vector of the Balkans, 1(3), 46–49.
5. Electron. URL: <https://www.electronjs.org/>
6. FastAPI Documentation. URL: <https://fastapi-admin-docs.long2ice.io/>
7. Toptal. URL: <https://www.toptal.com>
8. TopTracker. URL: <https://www.toptal.com/tracker>
9. Clockify. URL: <https://clockify.me>
10. ActivityWatch. URL: <https://activitywatch.net/>
11. RescueTime. URL: <https://www.rescuetime.com/>
12. TogglTrack. URL: <https://toggl.com/track/>
13. ManicTime. URL: <https://www.manictime.com/>