МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інтелектуальних
інформаційних систем, д-р техн. наук, проф.
_____Ю. П. Кондратенко
«____»_____2023 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

СИСТЕМА МОНІТОРИНГУ ТА АНАЛІЗУ ДІЯЛЬНОСТІ КОРИСТУВАЧА ПК

Спеціальність 122 «Комп'ютерні науки»

122 – БКР – 401.21910120

Виконав студент 4–го курсу, групи 401
_____В. А. Поріцький
«___» червня 2023 р.

Керівник: канд. пед. наук, доцент
_____Н. М. Болюбаш
«___» червня 2023 р.

Миколаїв – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Рівень вищої освіти **бакалавр**
Спеціальність **122 «Комп'ютерні науки»**
(шифр і назва)
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних
інформаційних систем, д-р. техн. наук, проф.
_____ Ю. П. Кондратенко
«____» _____2022 р.

З А В Д А Н Н Я

на виконання кваліфікаційної роботи

Видано студенту групи 401 факультету комп'ютерних наук Поріцькому Владиславу Анатолійовичу.

1.Тема кваліфікаційної роботи «Система моніторингу та аналізу діяльності користувача ПК».

Керівник роботи Болюбаш Надія Миколаївна, канд. пед. наук, доцент.

Затв. наказом Ректора ЧНУ ім. Петра Могили від «__» ____ 202_ р. № _____

2. Строк представлення кваліфікаційної роботи студентом «__» _____ 2023 р.

3. Вхідні (початкові) дані до роботи: предметна сфера моніторингу діяльності користувачів ПК, набір даних про дії користувача за комп'ютером.

Очікуваний результат: інформаційна система моніторингу та аналізу діяльності користувача ПК.

4. Перелік питань, що підлягають розробці (зміст пояснювальної записки):

- дослідження теоретичних засад відслідковування діяльності користувача за комп'ютером та аналізу існуючих програмних засобів забезпечення моніторингу;

- обґрунтування вибору технологій і засобів розробки системи моніторингу та аналізу діяльності користувача ПК;
- проєктування, розробка і здійснення програмної реалізації системи моніторингу й аналізу діяльності за комп'ютером індивідуальних користувачів та співробітників компаній.

5. Перелік графічного матеріалу: презентація

6. Завдання до спеціальної частини: «Оцінка ризиків пов'язаних з роботою на ПК та заходи їх зниження».

7. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис
Спеціальна частина з охорони праці	канд. тенх. наук, доцент А. О. Алексєєва	

Керівник роботи __канд. пед. наук, доцент Болюбаш Н. М._____
(наук. ступінь, вчене звання, прізвище та ініціали)

_____
(підпис)

Завдання прийнято до виконання ___Поріцький В. А._____
(прізвище та ініціали)

_____
(підпис)

Дата видачі завдання «_____» _____ 2022 р.

КАЛЕНДАРНИЙ ПЛАН
виконання бакалаврської кваліфікаційної роботи

Тема: Система моніторингу та аналізу діяльності користувача ПК

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Визначення керівника і теми БКР. Подання заяви на затвердження теми БКР	01.09.2022	1.11.2022	Виконано
2.	Отримання завдання на виконання БКР	25.11.2022	25.11.2022	Виконано
3.	Складання календарного плану	26.11.2022	10.12.2022	Виконано
4.	Огляд літератури за темою дослідження. Аналіз предметної сфери моніторингу діяльності користувачів ПК та активності персоналу компаній	11.12.2021	31.12.2021	Виконано
5.	Вибір технологій та інструментальних засобів розробки системи	1.01.2023	31.01.2023	Виконано
6.	Створення дизайну, проєктування та програмна реалізація, тестування системи	1.02.2023	15.04.2023	Виконано
7.	Робота над розділами фахової частини БКР	16.04.2023	31.04.2023	Виконано
8.	Проходження переддипломної практики, збір та аналіз матеріалів, остаточне оформлення розділів фахової частини БКР	1.05.2023	14.05.2023	Виконано
9.	Розробка спеціальної частини з охорони праці	15.05.2023	25.05.2023	Виконано
10.	Обговорення отриманих результатів з керівником та попередній захист БКР	29.05.2023	31.05.2023	Виконано
11.	Корегування роботи за результатами попереднього захисту	2.06.2023	6.06.2023	Виконано
12.	Остаточне оформлення пояснювальної записки та слайдів доповіді до захисту	7.06.2023	11.06.2023	Виконано
13.	Подання рецензенту та рецензування БКР	15.06.2023	18.06.2023	Виконано
14.	Подання БКР, її електронної копії та інших документів (відгуку, рецензії) до захисту	19.06.2023	22.06.2023	Виконано
15.	Захист БКР перед ЕК	29.06.2023	29.06.2023	Виконано

Розробив студент _____Поріцький В. А._____ _____
(прізвище та ініціали) *(підпис)*

Керівник _____канд. пед. наук, доцент Болюбаш Н. М._____ _____
(наук. ступінь, вчене звання, прізвище та ініціали) *(підпис)*

«____» _____2022 р.

АНОТАЦІЯ

бакалаврської кваліфікаційної роботи студента групи 401 ЧНУ ім. Петра Могили

Поріцького Владислава Анатолійовича

Тема: «Система моніторингу та аналізу діяльності користувача ПК»

Бакалаврська кваліфікаційна робота присвячена розробці та здійсненню програмної реалізації системи моніторингу та аналізу діяльності користувачів ПК, що є актуальним в умовах цифровізації усіх сфер діяльності суспільства, оскільки дозволяє налагодити робочі процеси як індивідуальних користувачів, так і співробітників компаній та забезпечити безпеку конфіденційної інформації.

Об'єкт роботи – процес діяльності користувача за комп'ютером.

Предмет роботи – програмні засоби та методи моніторингу й аналізу діяльності користувача ПК.

Мета роботи – підвищення ефективності моніторингу активності користувача за комп'ютером із метою оптимального планування діяльності шляхом розробки автоматизованої системи моніторингу та аналізу діяльності користувача ПК.

Бакалаврська кваліфікаційна робота складається з фахової та спеціальної частини з охорони праці. Пояснювальна записка фахової частини складається зі вступу, чотирьох розділів, висновків та додатків. У першому розділі розкрито теоретичні засади відслідковування діяльності користувача ПК та здійснено аналіз існуючих програмних засобів забезпечення моніторингу. У другому розділі обґрунтовано вибір технологій та засобів розробки системи. У третьому розділі описано проєктування, у четвертому – розробку та програмну реалізацію системи моніторингу й аналізу діяльності за комп'ютером індивідуальних користувачів та співробітників компаній.

Бакалаврська кваліфікаційна робота містить 66 сторінок (без додатків), 29 рисунків, 28 джерел, 11 додатків.

Ключові слова: моніторинг дій користувача, система моніторингу та аналізу, вебмоніторинг, десктоп-моніторинг.

ABSTRACT

for bachelor's qualification work of a student of 401 group at Petro Mohyla Black Sea National University
Poritskoho Vladyslava Anatoliiovycha

Theme: «PC user activity monitoring and analysis system»

The bachelor's qualification work is devoted to the development and implementation of the software implementation of the system for monitoring and analyzing the activities of PC users. What is relevant in the conditions of high rates of digitization of all spheres of society, as it allows you to adjust the work processes of both individual users and company employees and ensure the security of confidential information.

Object of work – the process of user activity at the computer.

Subject of work – software tools and methods of monitoring and analyzing PC user activity.

The purpose of the work is to increasing the effectiveness of monitoring the user's activity at the computer with the aim of optimal activity planning by developing an automated system for monitoring and analyzing the PC user's activity.

The bachelor's qualification work consists of a professional and special part on labor protection. The explanatory note of the professional part consists of an introduction, four sections, conclusions and appendices. In the first chapter, the theoretical principles of monitoring the PC user's activity are revealed and the analysis of the existing monitoring software tools is carried out. The second section substantiates the choice of technologies and system development tools. The third section describes the design, and the fourth section describes the development and software implementation of the computer activity monitoring and analysis system of individual users and company employees.

The bachelor's thesis contains 65 page (without appendices), 29 figures, 28 sources, 11 appendixes.

Keywords: user activity monitoring, monitoring and analysis system, web monitoring, desktop monitoring.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API – application programming interface

IDE – integrated development environment

UI – user interface

БД – база даних

ПЗ – програмне забезпечення

ПК – персональний комп’ютер

СУБД – система управління базами даних

ВСТУП

Актуальність. Розвиток цифрових технологій на сучасному етапі охоплює усі сфери діяльності суспільства, що обумовлює необхідність використання інноваційних підходів до моніторингу діяльності персоналу компаній та індивідуальних користувачів. В Україні популярність використання програмних засобів моніторингу діяльності користувачів ПК різко зросла під час карантину, пов'язаного з пандемією Covid–19, та в умовах введення воєнного стану, оскільки працівникам багатьох організацій доводиться працювати віддалено. Перспективним напрямком підвищення ефективності налагодження робочих процесів як індивідуальних користувачів, так і співробітників компаній та забезпечення безпеки конфіденційної інформації є застосування автоматизованих систем аналізу та моніторингу діяльності користувачів за комп'ютером.

До найбільш популярних ресурсів, які представлені на ринку програмного забезпечення для відслідковувати дій користувача за комп'ютером, відносять такі системи моніторингу, як Teramind, ObserveIT, ActivTrak, Veriato Cerebral, Time Doctor, InterGuard, RescueTime та інші. Наявні програмні засоби мають широку функціональність, яка дозволяє відстежувати відкриті вікна застосунків та web–активність, робити скріншоти екранів з певною періодичністю, відстежувати електронну пошту, блокувати певні вебсайти, включати відеонагляд та аудіомоніторинг. Однак такі системи моніторингу є комерційними, тому є потреба у створенні автоматизованої системи моніторингу та аналізу діяльності користувача ПК, яка за рахунок засобів контролю дій та сформованих звітів забезпечує планування роботи персоналу компаній та окремих користувачів.

Це обумовило **мету роботи**, яка полягає у підвищенні ефективності моніторингу активності користувача за комп'ютером із метою оптимального планування діяльності шляхом розробки автоматизованої системи моніторингу та аналізу діяльності користувача ПК.

Відповідно до поставленої мети було сформульовано **завдання**:

1) дослідити теоретичні засади відслідковування діяльності користувача за комп'ютером та провести аналіз існуючих програмних засобів забезпечення моніторингу;

2) обґрунтувати вибір технологій і засобів розробки системи моніторингу та аналізу діяльності користувача ПК;

3) здійснити проєктування, розробку та програмну реалізацію системи моніторингу й аналізу діяльності за комп'ютером індивідуальних користувачів та співробітників компаній.

Об'єкт роботи – процес діяльності користувача за комп'ютером.

Предмет роботи – програмні засоби та методи моніторингу й аналізу діяльності користувача ПК.

Методологічною основою дослідження є загальнонаукові та аналітичні методи, які дозволили комплексно вивчити предмет та об'єкт дослідження, дослідити основні підходи до побудови системи моніторингу та аналізу діяльності користувача за комп'ютером.

Практичне значення отриманих результатів полягає в тому, що використання розробленої системи дозволить підвищити ефективність планування діяльності за комп'ютером індивідуальних користувачів та співробітників компаній.

Структура бакалаврської кваліфікаційної роботи. Відповідно до мети, завдань і предмета дослідження, бакалаврська робота містить основну та спеціальну частини. Основна частина роботи складається із вступу, чотирьох розділів, висновку, списку використаних джерел та 11 додатків. Загальний обсяг роботи – 93 сторінок, із них основного тексту основної частини – 65 сторінок, спеціальної – 15 сторінок. Кількість використаних джерел – 28.

1 ТЕОРЕТИЧНІ ЗАСАДИ МОНІТОРИНГУ ДІЯЛЬНОСТІ КОРИСТУВАЧА ЗА КОМП'ЮТЕРОМ

1.1 Предметна сфера моніторингу активності користувачів ПК

Система моніторингу діяльності користувача ПК – це програмний засіб, який дозволяє відслідковувати дії користувача за комп'ютером. Предметна сфера системи моніторингу та аналізу діяльності користувача ПК охоплює широкий діапазон аспектів, які включають взаємодію користувача з комп'ютером, аналіз даних про активність користувача, а також захист та конфіденційність даних. Система моніторингу користувачів комп'ютера може використовуватися як роботодавцями, які бажають відслідковувати продуктивність і контролювати роботу своїх співробітників, так і індивідуальними користувачами та батьками, які хочуть контролювати діяльність своїх дітей за комп'ютером.

Звернувши увагу на дані світових трендів бачимо, що після 2020 року усе більше людей знаходять для себе віддалену роботу більш приємною. А кількість людей, які добровільно чи вимушено працюють вдома, різко зросла [1]. Крім того, незважаючи на мінливу загрозу пандемії сьогодні та ведення бойових дій у нашій країні, дистанційна робота, здається, стає все більш популярною.

За результатами проведеного аналізу можна виділити наступні факти (див. рис. 1.1):

1) на сьогоднішній день в IT секторі кількість підприємств, які повідомили про перехід на постійну модель віддаленої роботи збільшилась на 23%;

2) у середньому компанії заощаджують 11 000 доларів на рік по кожному співробітнику, який працює віддалено 2,5 дні на тиждень;

3) 40% працівників вважають, що під час пандемії вони були більш продуктивними, працюючи вдома, ніж в офісі;

4) зараз віддалена робота становить 15% можливостей роботи у світі.

Аналізуючи проблематику поставленого завдання, можна впевнено сказати, що динаміка актуальності систем з контролю дій користувачів при роботі за ПК щороку зростає. Сьогоднішні реалії призвели широкого застосування таких систем, впровадження нового функціоналу, поліпшення методів та інструментів, якими користуються як роботодавці і працівники, так і індивідуальні користувачі.

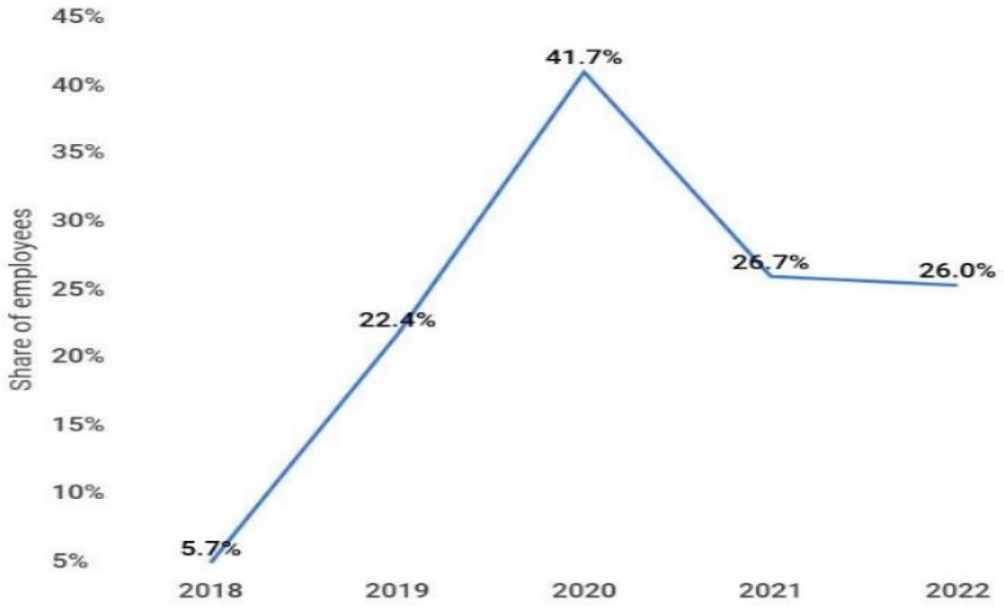


Рисунок 1.1 – Співвідношення частини людства у світі, що працює віддалено

Моніторингове програмне забезпечення для співробітників детально стежить за всіма діями на комп’ютері. При цьому, більш розвинені системи моніторингу не обмежуються простим збором даних, вони також генерують деталізовані звіти про продуктивність кожного співробітника, що полегшує аналіз та економить час.

Досить важливим застосуванням такого роду систем є оцінка продуктивності співробітників. Які ресурси витрачаються на реальну роботу, а які – на дещо інше. Як добре співробітники зосереджені на роботі, чи переривають вони її через кожні 10 хвилин? Чи вони шукають інформацію, яка стосується їхньої роботи, або, можливо, використовують робочий час для власних потреб? На всі ці питання можуть бути дані відповіді за допомогою моніторингового програмного забезпечення [2].

Здійснений аналіз дозволив виявити основні функціональні можливості моніторингових систем [3]. Розглянемо їх більш детально.

Відслідковування активності: основною функцією системи є відслідковування активності користувача ПК. Це включає в себе різні види активності, такі як відкриття та закриття програм, використання браузера, зроблені пошукові запити, використання соціальних мереж, відправлення та отримання електронних листів тощо.

Аналіз даних: виявлені дані про дії користувача потім аналізуються для виявлення цілей використання комп’ютера. Наприклад, програм, що часто відкриваються, загального часу використання комп'ютера, частоти використання певних вебсайтів, а також аналізу тексту, що був введений користувачем у певній програмі.

Звітність: система повинна мати можливість надавати звіти, які демонструють активність користувача за певний період часу. Це може включати в себе графіки активності, зведені таблиці, а також детальні звіти про конкретну активність.

Конфіденційність та безпека: у зв’язку з великим обсягом даних, які збираються і обробляються, система моніторингу та аналізу діяльності користувача ПК повинна бути обладнана відповідними механізмами захисту даних. Це може включати в себе шифрування даних, автентифікацію та авторизацію користувачів, захист від несанкціонованого доступу, а також відповідність вимогам з конфіденційності та приватності.

Персоналізація: система може надавати можливості для персоналізації параметрів моніторингу та аналізу, залежно від потреб конкретного користувача або організації. Наприклад, можливість налаштування часу активного моніторингу, виділення специфічних програм для слідкування, встановлення рівнів сповіщень тощо.

Загалом, предметна сфера системи моніторингу та аналізу діяльності користувача ПК є багатогранною і потребує глибокого розуміння від користувача або організації, яка планує використовувати таку систему.

Виділивши основні загальні вимоги до моніторингових систем можна зупинитися на більш детальному описі функціональності.

Прослуховування клавіатури. Запис натискань клавіш – одна з найбільш суперечливих функцій. Насамперед вона є досить жорстким методом контролю за людиною. Кейлоггер може випадково записати конфіденційну інформацію, таку як паролі, банківські реквізити або особисті розмови. Не всі застосунки надають таку можливість, а якщо і надають то напевно мають розробити відключення даної функції в окремих випадках.

Скріншоти. Досить зручна функція, що насамперед не потребує великого обсягу пам’яті для зберігання на відміну від запису екрана. Може виконуватися в рандомний час, та повністю без аналізу показує чим займається людина в той чи інший момент часу.

Моніторинг вебактивності. Моніторинг вебактивності – це важливий інструмент для оцінки продуктивності співробітників. Однак, як і всякий інструмент, він має свої плюси і мінуси. З плюсів можна відзначити покращення продуктивності, коли співробітники витрачають надмірну кількість часу на непродуктивні вебсайти, їх активність помітна. Захист даних, у випадках коли моніторинг вебактивності може допомогти виявити можливі витоки даних або інші дії, що ставлять в небезпеку безпеку даних, наприклад, компанії.

Із мінусів моніторингу вебактивності відзначаємо приватність. Моніторинг вебактивності сприяє створенню почуття, що хтось стежить, що може вплинути на моральний стан співробітників. Ресурси є ключовим мінусом, бо даний функціонал може вимагати значних затрат, включаючи час на аналіз зібраної інформації. Відзначимо також правові ризики, які в залежності від регулятивних норм у деяких країнах може бути обмежений законодавством [4].

Моніторинг заявок. Цей набір функцій покаже, які програми встановлені на контрольованому пристрої, які програми запущені і скільки часу користувач витрачає на їх використання. Крім того, буде відомо, чи не встановлює співробітник небажане програмне забезпечення на пристрій, що належить компанії.

Блокувальник додатків. Як і блокувальник сайтів, блокувальник додатків усуває відволікаючі фактори на робочому місці. Можна у разі потреби заблокувати програми обміну повідомленнями, ігри або інші небажані програми, якщо вони впливають на продуктивність роботи команди.

Відстеження портативних носіїв інформації. Включає контроль за тим, як працівники копіюють або передають дані через зовнішні пристрої, такі як USB–накопичувачі. Це важливий аспект безпеки, оскільки він запобігає несанкціонованому витоку важливих даних поза компанію.

Пряма трансляція з екрану. Дає можливість відстежувати, що саме робить співробітник у реальному часі, наче ви дивитеся його екран. Деякі програми навіть дають змогу записувати екран, що дозволяє переглянути робочий процес пізніше. Але як було зазначено, пам'ять, що використовує застосунок може бути обмежена, тож зберігання такого роду інформації може бути ресурсозатратним.

*Спостереження за допомогою веб камери та мікрофону***.** Якщо комп’ютери оснащені веб камерами та мікрофонами, це дозволяє вам вести відео та аудіо записи співробітників, що може сприяти підвищенню продуктивності, а також запобіганню неетичної поведінки, крадіжок або незаконного доступу до конфіденційної інформації. Як і кейлоггер, цей пункт може викликати доволі сильний дискомфорт у співробітника. Але якщо сфера діяльності компанії побудована на відео та аудіо спілкуванні з людьми – це є необхідною функцією для контролю не тільки самого робітника, а й якості його роботи.

Іншими, не досить популярними функціями систем моніторингу є *відстеження портативних носіїв інформації*. Відстеження контролю за тим, як працівники копіюють або передають дані через зовнішні пристрої, такі як USB–

накопичувачі. Це важливий аспект безпеки, оскільки він запобігає несанкціонованому витоку важливих даних поза компанію.

Моніторинг електронної пошти. Процес відстеження електронних листів, що надсилаються та отримуються співробітниками. Це не лише запобігає витоку даних, але й може служити засобом контролю за якістю, особливо у відділах обслуговування клієнтів.

1.2 Програмні засоби моніторингу діяльності співробітників

На ринку програмного забезпечення для моніторингу та аналізу дій користувачів з метою контролю та відстеження передачі конфіденційної інформації по різних каналах зв’язку використовують багато відомих систем моніторингу. Такі як Teramind, ObserveIT, ActivTrak, Veriato Cerebral, Time Doctor, InterGuard, RescueTime та інші. Розглянемо їх більш детально [5].

Система *Teramind* є одним з найбільш популярних інструментів для слідкування за діяльністю користувачів. Вона включає широкий спектр функцій (див. рис. 1.2). Одна з особливостей Teramind – це здатність визначати «аномальну поведінку», засновану на попередньо аналізованому паттерні дій користувача. Цей продукт інтегрується з різноманітними системами і має широкий спектр функцій, які спрямовані на попередження внутрішніх та зовнішніх загроз, а також підвищення продуктивності роботи.

Опишемо основні характеристик даної системи.

Відстеження активності: моніторинг вебсайтів, програм, клавіатурних натискань, файлів, пошти, чатів, соціальних медіа, та навіть друкованих документів. Ця функція допомагає виявляти можливі порушення та забезпечує прозорість діяльності користувача.

Аналіз поведінки: використовує алгоритми машинного навчання для аналізу діяльності користувача та визначення аномальної поведінки.

Моніторинг продуктивності: може генерувати детальні звіти про використання часу, активність на вебсайтах та програмах, а також аналізувати робочий час.

Контроль доступу до даних: контроль доступу до важливої інформації, а також встановлювати правила та політики для захисту даних.

В цілому, Teramind є потужним інструментом для моніторингу та аналізу діяльності користувача. Він не лише допомагає організаціям підвищити продуктивність та виявити можливі загрози, але й дозволяє захистити важливі дані та дотримуватися нормативних вимог.

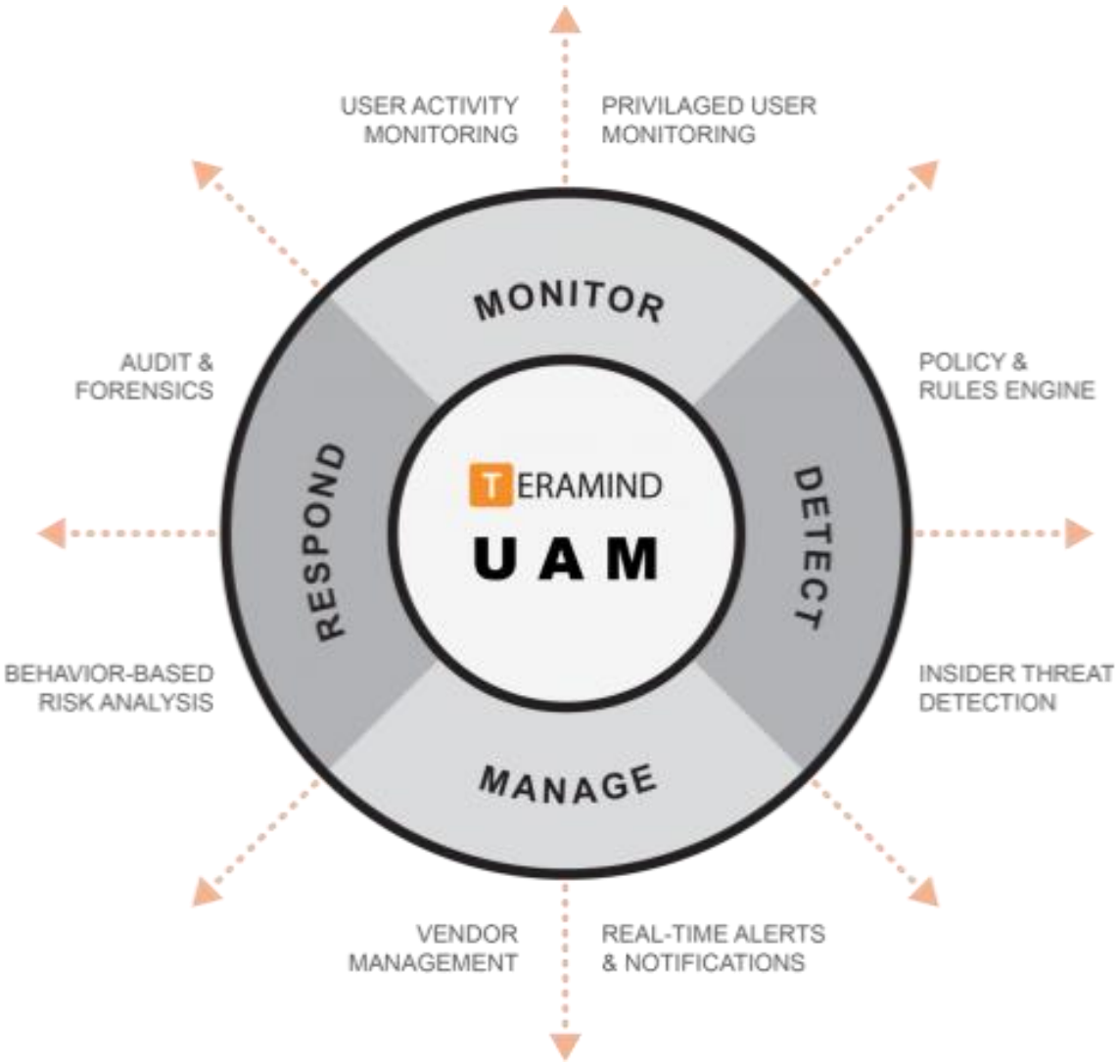


Рисунок 1.2 – Функціонал системи моніторингу від компанії Teramind

Система *Time Doctor* – це ще один потужний інструмент для моніторингу активності користувачів (див. рис. 1.3). Він розроблений з основною метою підвищення продуктивності, пропонуючи функції відстеження часу, скріншотів екрана, відстеження вебсайтів та додатків, а також детальні звіти про продуктивність.

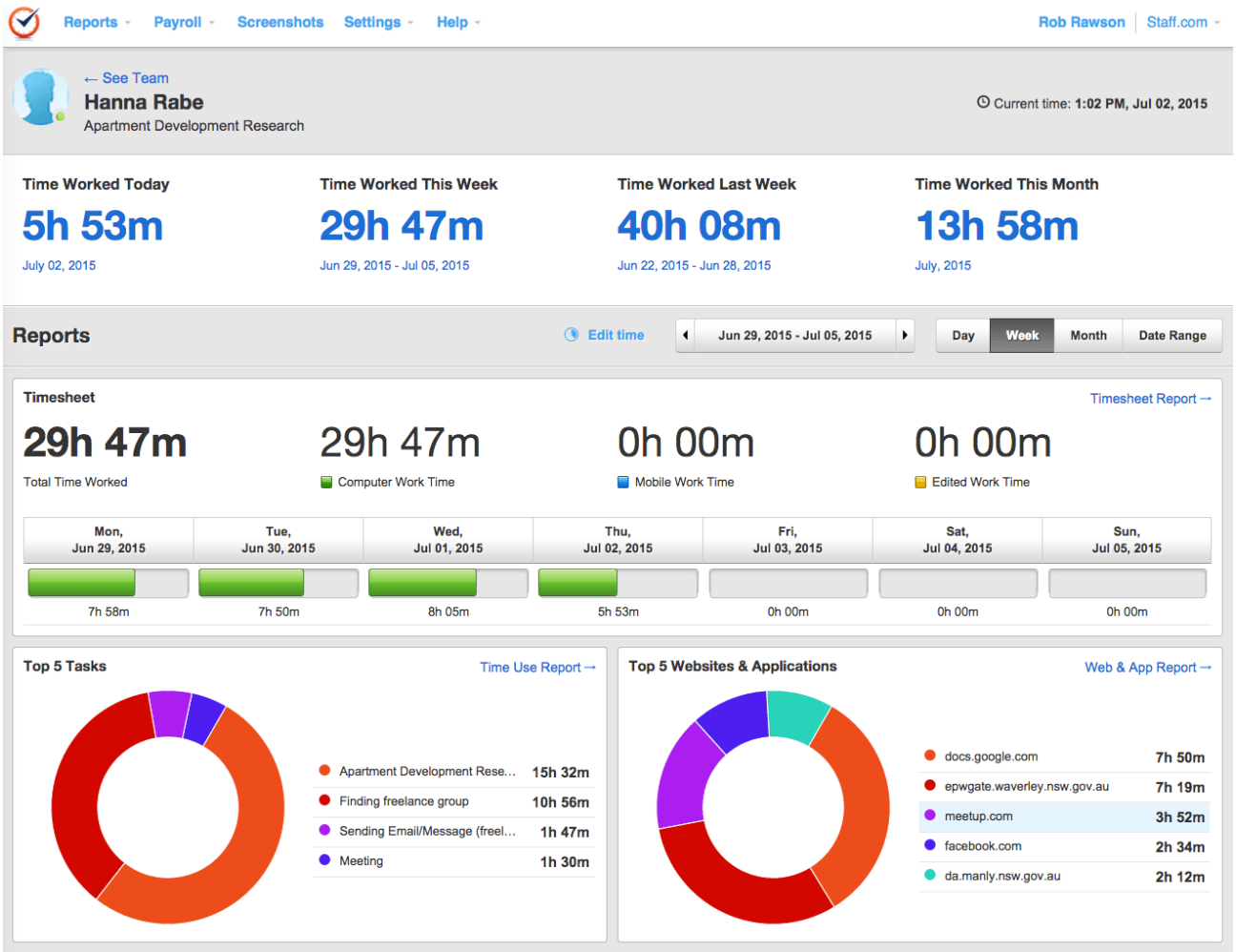


Рисунок 1.3 – UI застосунку Time Doctor

Ключові можливості Time Doctor являють собою відстеження часу, який вони витрачають на виконання завдань. Вони можуть вибирати завдання, які потрібно відстежувати, і програма автоматично фіксує час, проведений над ними. Моніторинг активності, що відстежує відвідувані вебсайти, використовувані програми та активність клавіатури і миші, скріншоти екрана, і нарешті, звіти та

аналіз, який генерує детальні звіти про відстеження часу та активності. Звіти можуть бути використані для покращення процесів роботи, оцінки продуктивності та прогнозування трудових ресурсів.

InterGuard є ще одним потужним засобом для моніторингу та аналізу діяльності користувача ПК. Ця система надає розгорнутий набір інструментів для контролю за діяльністю користувачів, забезпечення безпеки даних та підвищення продуктивності (див. рис. 1.4).

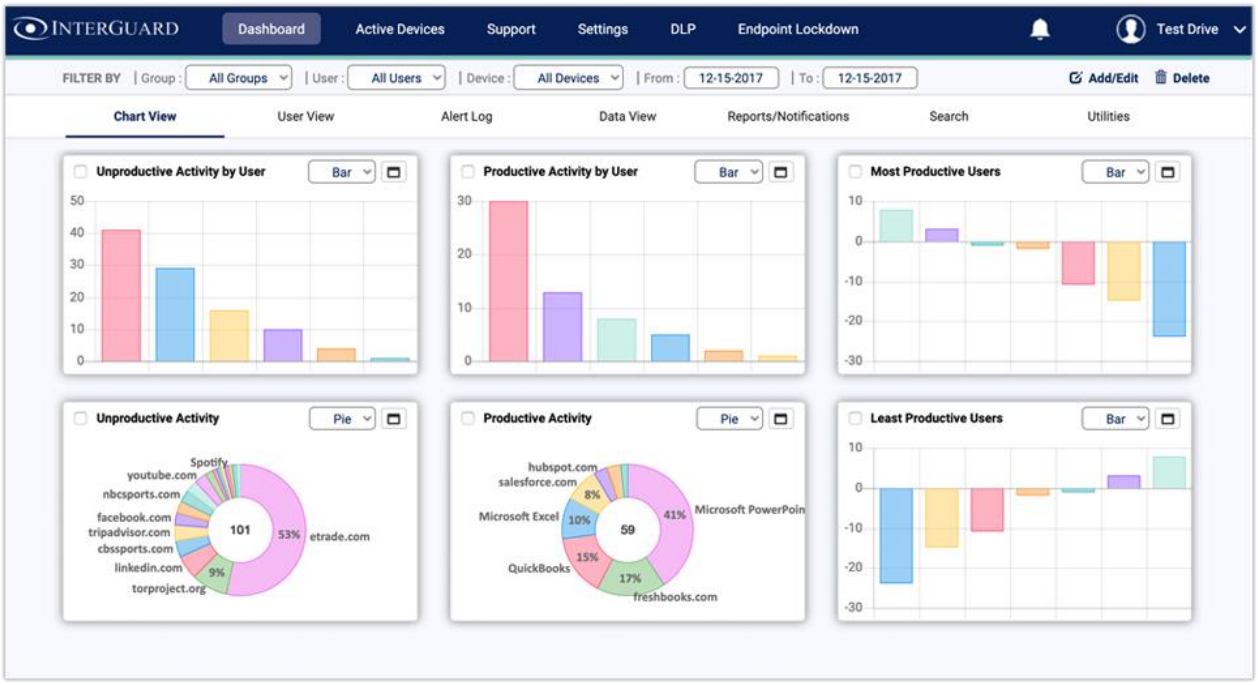


Рисунок 1.4 – Фрагмент роботи застосунку InterGuard

Із реалізованого у застосунку InterGuard функціоналу можемо бачити вже знайомі нам методи відстеження діяльності, вебсайтів, скріншоти та інше. Але виділимо ще декілька методів, що запроваджено у застосунку InterGuard. І перший з них блокування вебсайтів. Так саме блокування, а не просто відстеження, що дозволяє встановлювати обмеження на відвідування певних вебсайтів. Такий функціонал може, насамперед, забезпечити безпеку корпоративної мережі підвищити контроль і продуктивність користувачів. Другий важливий аспект – це контроль над електронною поштою. InterGuard забезпечує глибокий контроль над

електронною поштою співробітників, включаючи відстеження надсилання та отримання електронних листів, а також відстеження вмісту цих повідомлень. InterGuard є солідним вибором для компаній, які шукають всебічне рішення для моніторингу користувачів. Це потужний, але гнучкий інструмент, який дозволяє підлаштовувати налаштування моніторингу під конкретні потреби компанії.

Система *RescueTime* – це програмне забезпечення для моніторингу та аналізу використання часу користувачами ПК та мобільних пристроїв (див. рис. 1.5). Із особливих ключових можливостей RescueTime відзначаються персоналізовані цілі та сповіщення. Користувачі можуть встановити свої цілі щодо продуктивності та отримувати сповіщення про досягнення цілей або відхилення від них. Тим саме відстежуючи у фоновому режимі діяльність, система прораховує та аналізує планову продуктивність для користувача. Із особливостей також варто відзначити інтеграцію з іншими інструментами, такими як календар, інструментами сповіщень та керування завданнями.

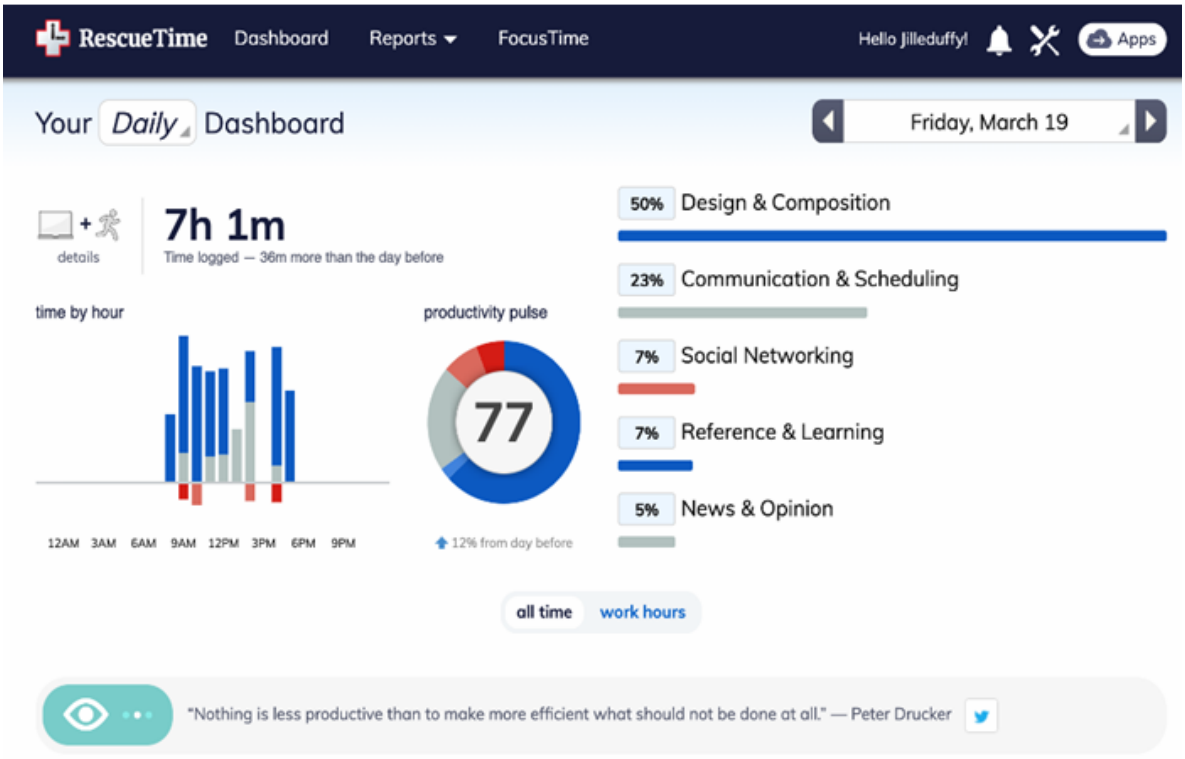


Рисунок 1.5 – Приклад UI застосунку RescueTime

Розглянуті системи моніторингу є висококласними інструментами для моніторингу активності користувачів, але вони мають відмінності в підходах та функціональності. Система RescueTime більше орієнтована на індивідуальну продуктивність, хоча також має функції для команд. Time Doctor, з іншого боку, більше сфокусований на моніторингу та оптимізації продуктивності команд, особливо для віддаленої роботи.

InterGuard надає більш комплексний моніторинг, включаючи функції, такі як скріншоти екрана, відстеження електронної пошти та блокування вебсайтів. Це робить його більш придатним для корпоративного контексту, де є потреба в більш строгому контролі. Teramind є одним із найбільш повних рішень, із широким спектром функцій для моніторингу користувачів, включаючи відеонагляд та аудіомоніторинг. Це робить Teramind підходящим для організацій, яким потрібно вести детальний нагляд за активністю користувачів з питань безпеки та виявлення внутрішніх загроз.

Деякі програми мають лише кілька базових опцій, таких як скріншоти, запис клавіатури та моніторинг активності в Інтернеті. Інші першокласні системи реєструють кожне клацання миші та кожен рух працівника за комп’ютером.

Серед спільних особливостей розглянутих систем можна виділити здатність відстежувати діяльність на комп’ютері, аналізувати продуктивність користувачів та генерувати звіти на основі зібраних даних. Вони також пропонують різні рівні персоналізації та налаштування, що дозволяє організаціям адаптувати інструменти під власні потреби.

Переважна більшість систем моніторингу, представлених на ринку програмного забезпечення, є комерційними продуктами. Тому доцільною є розробка системи моніторингу та аналізу діяльності користувачів. Здійснений аналіз є допоміг виявити основні характеристики, функції та можливості таких систем, що, в свою чергу, може допомогти у розробці нового продукту.

1.3 Постановка задачі

Розробка системи моніторингу та аналізу діяльності користувача ПК є актуальною задачею в умовах високих темпів інформатизації усіх сфер діяльності, оскільки дозволяє забезпечувати безпеку конфіденційної інформації та оптимально планувати й налагоджувати роботу індивідуальних користувачів і співробітників компаній.

Об’єкт роботи – процес діяльності користувача за комп’ютером.

Предмет роботи – програмні засоби та методи моніторингу й аналізу діяльності користувача ПК.

Мета роботи – підвищення ефективності моніторингу активності користувача за комп’ютером з метою оптимального планування діяльності шляхом розробки автоматизованої системи моніторингу та аналізу діяльності користувача ПК.

Для досягнення поставленої мети було поставлено такі **завдання**:

1) дослідити теоретичні засади відслідковування діяльності користувача за комп’ютером та провести аналіз існуючих програмних засобів забезпечення моніторингу;

2) обґрунтувати вибір технологій і засобів розробки системи моніторингу та аналізу діяльності користувача ПК;

3) здійснити проєктування, розробку та програмну реалізацію системи моніторингу й аналізу діяльності за комп’ютером індивідуальних користувачів та співробітників компаній.

Розроблювана система моніторингу та аналізу діяльності користувача за комп’ютером повинна задовольняти наступним вимогам:

1) функціональність – система повинна включати наступні ключові функції: прослуховувати клавіатуру, робити знімки екрану з певною періодичністю, відстежувати відкриті сторінки вебресурсів та вікон застосунків, генерувати звіти про продуктивність, забезпечувати дистанційний моніторинг та управління;

2) користувацька відповідність: система повинна бути простою в користуванні та налаштуванні, включаючи адміністраторів системи та кінцевих користувачів – зручність використання є ключовими для успіху системи;

3) доступність: система повинна бути надійною і доступною цілодобово протягом усіх днів тижня для забезпечення постійного моніторингу діяльності користувачів, що вимагає ретельного планування резервного копіювання та відновлення даних, а також надійної системи підтримки;

4) забезпечення безпеки: проєкт має враховувати важливість безпеки даних, включаючи захист від несанкціонованого доступу, шифрування, відстеження витоків інформації та інше;

5) масштабованість: система повинна мати можливість масштабування, щоб відповідати змінюваним потребам, включаючи збільшення кількості користувачів або розширення функціональності.

Ці вимоги є основою для подальшого проєктування та реалізації системи.

Висновки до розділу 1

В даному розділі бакалаврської кваліфікаційної роботи розглянуто теоретичні засади моніторингу та аналізу діяльності користувача за комп’ютером. Виявлено основні вимоги до таких систем та набір їх функціональних можливостей. Здійснений огляд існуючих систем моніторингу діяльності персоналу та індивідуальних користувачів ПК дав можливість сформувати вимоги до розроблюваної системи аналізу та моніторингу діяльності користувачів за комп’ютером.

2 ТЕХНОЛОГІЇ І ЗАСОБИ РОЗРОБКИ СИСТЕМИ МОНІТОРИНГУ ТА АНАЛІЗУ ДІЯЛЬНОСТІ КОРИСТУВАЧА ПК

2.1 Функції Windows API для відстеження користування застосунками та вебресурсами

Моніторинг системи включає в себе збір, аналіз та відображення інформації про діяльність та продуктивність користувачів, які працюють за комп’ютером. Одним із основних у роботі таких системи є можливість відстеження вебактивності та користування застосунками.

Windows API надає розробникам можливість використовувати функції та служби, які надає операційна система Windows для моніторингу активності застосунків. В контексті проекту, що реалізується, необхідними є дві функції Windows API: GetForegroundWindow та GetWindowThreadProcessId [6].

GetForegroundWindow – це функція, яка повертає дескриптор до активного вікна, тобто вікна, яке в даний момент має фокус введення. Це вікно, яке користувач в даний момент найбільш ймовірно використовує.

GetWindowThreadProcessId – це функція, яку надає Windows, для отримання ідентифікатора процесу, який створив конкретне вікно. Тобто, якщо є вікно, і необхідно знати, яка програма це вікно створила, можна використати цю функцію.

Під час реалізації, GetForegroundWindow використовують для отримання активного вікна, а GetWindowThreadProcessId для отримання ідентифікатора процесу, який створив це вікно. Це дозволяє дізнатися, яка програма в даний момент активна. Загалом описані функції дають можливість відстежувати, який застосунок в даний момент є активним на комп’ютері користувача [7].

Відстеження вебактивності є доволі клопітливим як в технічному, так і в етичному плані процесом, який може також порушувати питання щодо конфіденційності та безпеки. Але реалізація даного функціоналу є базовою вимогою для більшості систем моніторингу. На прикладі найпопулярнішого

браузера Google Chrome розглянемо можливу реалізацію даного функціоналу [8]. Процес збору даних та їх обробки відбувається за наступним сценарієм:

1) отримання логів: різні браузери зберігають логи в різних форматах і в різних місцях. Google Chrome зберігає логи перегляду в SQLite базі даних в профілі користувача. Реалізація на базі іншого браузера потребує знань місцезнаходження і формату цих файлів;

2) читання логів: як тільки файл логів знайдено, потрібно розібрати його. Знову ж таки, це буде залежати від формату, в якому браузер зберігає логи. Для SQLite бази даних, наприклад, використаємо стандартні інструменти та бібліотеки для читання та обробки цих даних;

3) обробка даних: після того, як дані було отримано , ми зможемо обробити їх і показати їх у застосунку. Знову ж таки, конкретний процес буде залежати від того, як ці дані будуть використовуватися, тож процес обробки може включати фільтрацію даних, відсівання неістотної інформації та перетворення даних у формат, який зручно використовувати.

Згідно даних методів алгоритм збору та обробки даних може бути відтворено у вигляді обробки простого SQLite файлу.

Кейлоггер – функціонал, що використовується для реалізації читання натисків клавіатури, насамперед є досить суперечливим рішенням яке в плані законодавства може бути незаконним чи порушувати приватність людей. Та в рамках роботи такий функціонал може бути описано та використано у застосунку, згідно з правами користувача.

Реалізація цього алгоритму є доволі простою – зчитуються клавіші, що натискаються та записують їх у вихідний файл. За допомогою мови C# можна відстежувати натискання клавіш за допомогою функцій KeyPress або KeyDown. Записаний текст заноситься у документ бази даних, як текстове поле та надалі можу бути зчитаний для відображення.

Відповідно, використовуючи цей функціонал будь–яке натискання клавиші у сторонніх застосунках буде записано до файлу.

Створення скріншотів екрану – ще один важливий аспект моніторингу діяльності користувача на комп’ютері. Застосування скріншотів дозволяє забезпечити зрозуміле і наглядне представлення дій, виконаних користувачем, та допомагає виявити некоректне або небезпечне використання системи.

Для реалізації скріншотів у середовищі C# існує клас System.Windows.Forms.Screen, який надає методи та властивості для взаємодії з екранами, підключеними до комп’ютера [9]. Використовуючи методи можна отримати розмір поточного екрану, а потім створити нове зображення цього розміру. Зображення зберігається в файловій системі або у базі даних з використанням поточного часу як частини імені файлу, що дозволяє нам легко визначити коли зображення було зроблене. Цей процес автоматизовано та періодично запускається з використанням таймера.

Та як було зазначено у попередньому розділі, збереження скріншотів вимагає певних ресурсів. Відповідно реалізація цього функціоналу потребує достатньо місця для зберігання.

2.2 Мова програмування C# та платформа .NET

Для розробки системи моніторингу активності користувача було вибрано мову програмування C# та технологію .NET. Розроблена компанією Microsoft, мова C# та платформа .NET є оптимальним вибором для розробки застосунку, від якого вимагають прямого доступу до пакетів Windows або логів системи. Гнучкість, висока продуктивність, та глибока інтеграція з ОС Windows – все це надійні інструменти для реалізації моніторинг системи [10].

Плюси використання цих засобів полягають ще й у можливості побудови на їх основі UI, що беззаперечно полегшує умови розробки та використання застосунку. Розберемо детальніше вищевказані та інші, необхідні для реалізації технології.

C# – це об'єктно–орієнтована мова програмування, розроблена Microsoft для

створення широкого спектра застосунків. Мова забезпечує усі необхідні інструменти для ефективного програмування: від розробки простих консольних додатків до створення складних багатопотокових програм. Вона має чіткий синтаксис, сильну типізацію, що дозволяє забезпечити високу надійність програмного коду. Багаторівнева підтримка фреймворків та бібліотек дозволяє розробнику не гаяти час розробляючи велосипед, а скористатися вже наданими рішеннями для реалізації необхідної задачі [11]. Впродовж роботи, звісно, буде продемонстровано гнучкість та широкий профіль цієї МП.

На прикладі рис. 2.1. показано різносторонність використання ресурсів мови C# для підтримки застосунків на різних платформах.

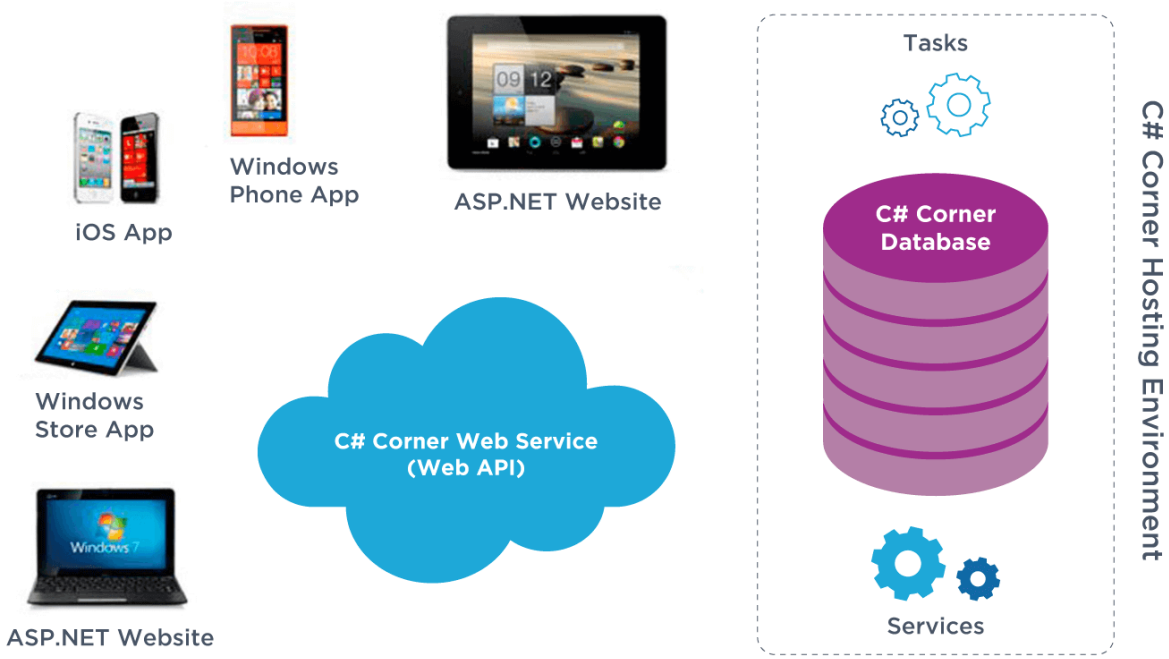


Рисунок 2.1 – Взаємодія архітектури C# із різними системами

Перечислимо загальні аспекти мови програмування C#, які зробили її однією з найпопулярніших мов програмування. C# є повністю об'єктно–орієнтованою мовою програмування. Це означає, що вона підтримує парадигми об'єктно–орієнтованого програмування, такі як інкапсуляція, наслідування та поліморфізм.

Як частина .NET, C# було створено спеціально для платформи .NET Microsoft, що дозволяє розробникам створювати різноманітні застосунки, включаючи настільні, веб, мобільні, гральні, та обчислювальні програми. C# тісно інтегровано з платформою .NET і є однією з основних мов програмування, які використовуються для розробки на цій платформі.

Сумісність з CLR (Common Language Runtime): C# є однією з багатьох мов, що можуть використовувати CLR .NET для виконання коду. CLR – це компонент платформи .NET, який забезпечує середовище виконання для коду. Він обробляє важливі аспекти виконання програми, такі як керування пам'яттю, управління потоками, безпеку та виключення. Коли ви пишете код на C#, CLR компілює його в машинний код при виконанні [11].

Сучасна, продумана мова: C# постійно оновлюється і вдосконалюється. Вона має багато сучасних особливостей, які спрощують розробку програмного забезпечення, такі як автоматична збірка сміття, LINQ (Language Integrated Query) для роботи з даними, асинхронне програмування за допомогою async/await, і багато іншого.

.NET – це платформа від Microsoft, яка забезпечує потужну та гнучку інфраструктуру для розробки, запуску та управління різноманітними типами програмних застосунків. Вона включає в себе багатий набір інструментів, сервісів та бібліотек, що, як ми пам’ятаємо з попереднього пункту, значно полегшують процес розробки та покращують продуктивність розробників (див. рис. 2.2) [12]. Розберемося в потужних можливостях платформи та яким чином її використання може бути корисним.

1. Ядрові компоненти .NET Framework складається з трьох основних компонентів: CLR (Common Language Runtime), FCL (Framework Class Library) і ASP.NET:

а) CLR надає середовище для виконання коду, написаного на .NET–сумісних мовах програмування (C#, Managed C++, Visual Basic .NET, F# та інших), управління пам’яттю, обробку винятків і безпеку;

б) FCL це бібліотека класів, які надають готові до використання функції для розробки додатків;

с) ASP.NET використовується для розробки вебдодатків і вебсервісів.

2. Багатомовність: .NET підтримує багато мов програмування, включаючи C#, VB.NET, F# і інші. Це дозволяє розробникам вибрати мову, яка найкраще підходить для їх задач.

3. Переносимість: із випуском .NET Core і .NET 5 та 6, платформа .NET стала переносимою, тобто можливою для використання не тільки на Windows, але і на MacOS і Linux.

4. Продуктивність: .NET надає високу продуктивність завдяки оптимізованому управлінню пам'яттю і можливостям компіляції Just–In–Time.

5. Безпека: .NET надає багато механізмів безпеки, таких як управління областями доступу, автентифікацію, криптографічні сервіси тощо.

6. Спільнота та підтримка: .NET має велику активну спільноту розробників і велику кількість відкритого коду. Microsoft підтримує і розвиває платформу .NET.

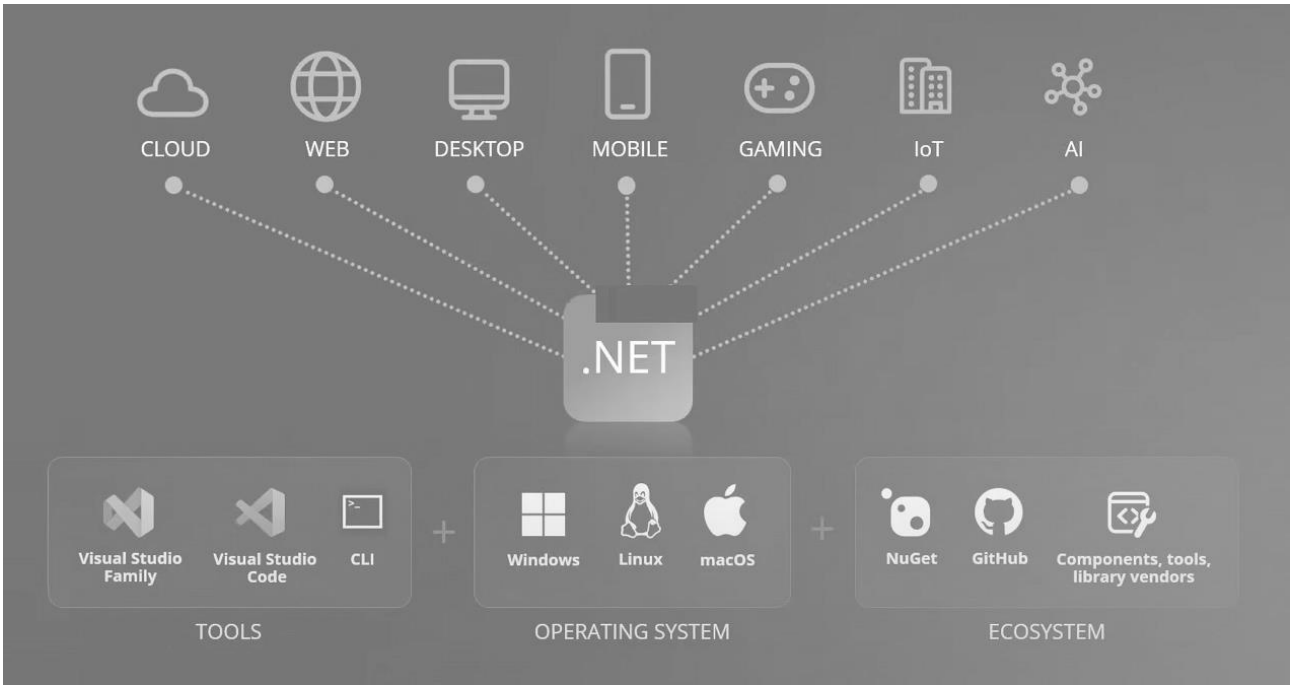


Рисунок 2.2 – Можливості платформи .NET

У рамках розробки системи моніторингу та аналізу користувача ПК, вибір даних технологій виправданий здатністю C# і .NET ефективно працювати з системними ресурсами, включаючи використання Windows API, логів системи, роботи з базами даних та самою Windows.

Внутрішній код і код C # пов'язані один з одним за допомогою часткових визначень класів, а потім компілятор .NET створює додаток.

2.3 Система Windows Presentation Foundation

Windows Presentation Foundation – WPF, є графічною системою для створення десктопних та мобільних застосунків на платформі .NET. Як і описані вище технології, WPF розроблена корпорацією Microsoft та дозволяє значно підвищити продуктивність застосунків, які інтенсивно працюють з графікою [13].

WPF підтримує розширений набір можливостей графічного дизайну, включаючи векторну графіку, використання градієнтів, використання тривимірної графіки та анімації. Ці можливості відкривають перед розробниками широкі перспективи для створення ефектних та зручних для користувача інтерфейсів.

Одна з основних особливостей WPF – використання XAML, мови розмітки для опису інтерфейсу. XAML дозволяє розробникам створювати інтерфейс в декларативному стилі, що спрощує процес розробки і підтримки коду.

XAML – мова розмітки, розроблена компанією Microsoft та використовується для визначення користувацького інтерфейсу в .NET Framework та .NET Core додатках (див. рис. 2.3). XAML в основному використовується в WPF і Xamarin.Forms для опису графічного інтерфейсу додатка [14]. Існують інші мови розмітки, такі як HTML, CSS або XML, кожна з яких має власні особливості, та вибір між якими залежить від багатьох факторів, в першу чергу включаючи потреби проекту, що реалізується.

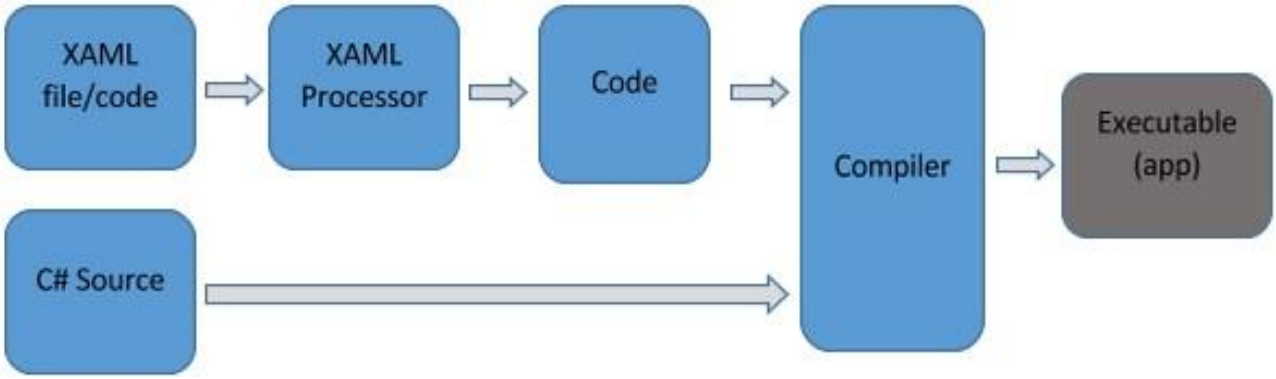


Рисунок 2.3 – Схема процесу роботи XAML

В свою чергу, мова розмітки XAML має свої особливості, серед яких:

1) відокремлення логіки від презентації: XAML дозволяє вам відокремити дизайн інтерфейсу від його логіки. Таке відокремлення зручно, оскільки дизайнери та розробники можуть працювати над UI відокремлено один від одного;

2) декларативна мова: XAML – це декларативна мова, тобто ми описуємо "що" ви хочете зробити, а не "як". Це відрізняє XAML від більшості процедурних мов програмування, таких як C#;

3) використання XML синтаксису. XAML використовує синтаксис XML, тому він досить знайомий багатьом розробникам і легко читається;

4) однією з ключових особливостей XAML є його потужна підтримка прив'язки даних. Це дозволяє зв’язати елементи інтерфейсу безпосередньо з даними, що спрощує роботу з динамічною інформацією;

5) XAML може бути використаний для створення інтерфейсу в додатках WPF. Це робить його універсальним інструментом для розробки UI на платформі .NET.

Файл XAML інтерпретується XAML процесором для конкретної платформи. Процесор XAML перетворює XAML у внутрішній код, який описує елемент інтерфейсу користувача.

Ще одна важлива особливість WPF — це підтримка прив’язки даних, яка спрощує процес відображення даних у інтерфейсі. Дата байндинг дозволяє

автоматично синхронізувати елементи інтерфейсу з даними моделі, без необхідності написання додаткового коду. Все це робить WPF відмінним вибором для створення сучасних додатків для Windows, які потребують багатого на візуальні ефекти інтерфейсу і високої продуктивності при роботі з графікою.

2.4 СУБД MongoDB

MongoDB – це відкрита база даних NoSQL, що використовує модель документів [15]. Вона була створена для сучасних вебдодатків, де потрібна швидка обробка великих обсягів даних і можливість горизонтального масштабування (див. рис. 2.4, рис. 2.5).

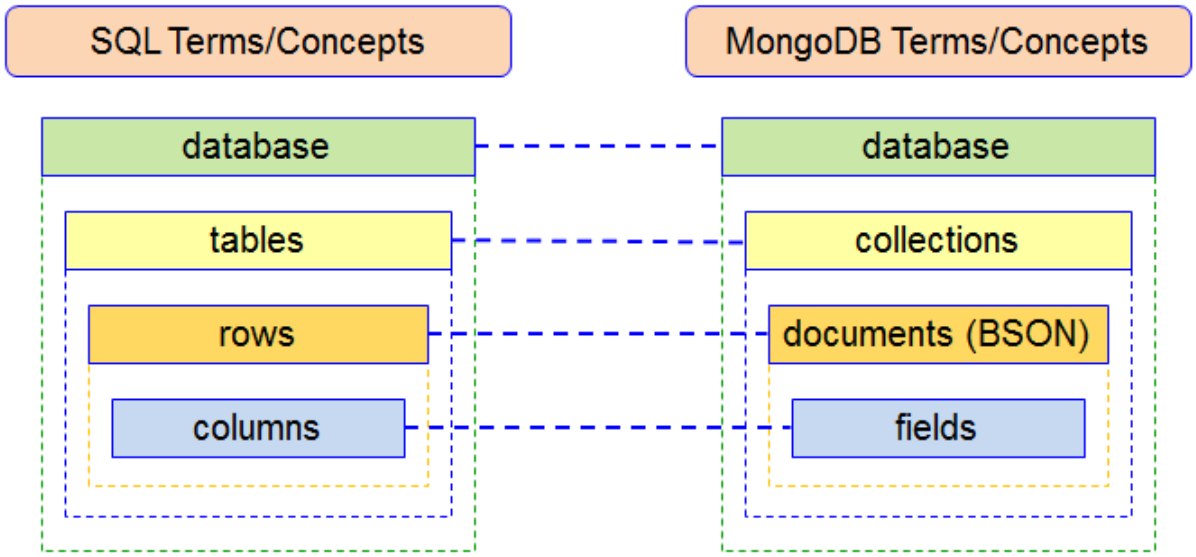


Рисунок 2.4 – Порівняння архітектури та структури збереження та запису даних у SQL та NoSQL базах даних

MongoDB використовує формат BSON для зберігання даних. Документи в MongoDB можуть мати різну структуру, що дозволяє зберігати складні структури даних, такі як масиви і вкладені документи, в одному запису. Це робить базу даних гнучкою і легкою для використання, особливо при роботі з даними, структура яких може змінюватися з часом [15].

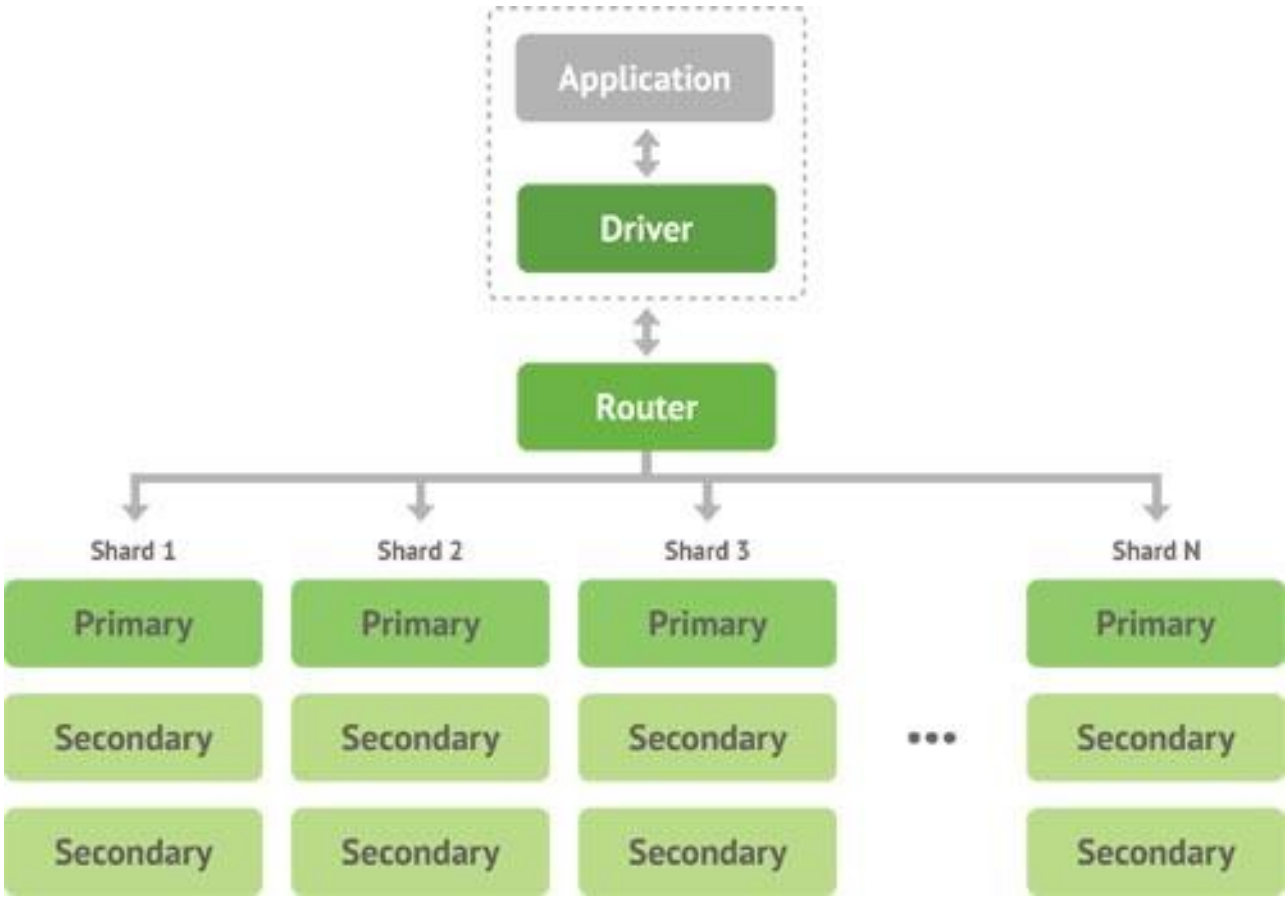


Рисунок 2.5 – Архітектура NoSQL бази даних MongoDB

MongoDB підтримує розподілені запити і може масштабуватися горизонтально шляхом додавання додаткових серверів до кластера. Відповідно MongoDB є відмінним вибором для великих проектів, де потрібно обробляти великі обсяги даних. Крім того, MongoDB має різні можливості індексації, у тому числі підтримку текстового пошуку та геопросторових індексів. Це дозволяє ефективно виконувати різноманітні запити до бази даних.

Порівнюючи СУБД, завжди виділяють два основних типи систем управління базами даних, які використовуються для зберігання та керування даними. Виділяючи структуру саме SQL моделей, кажуть, що вона має структуровану або табличну модель даних. В той час як MongoDB використовує неструктуровану або напівструктуровану модель. Що це означає для розробника який стоїть перед вибором використання однієї з цих систем?

Визначення, чи є дані структурованими, напівструктурованими або неструктурованими, відноситься до того, наскільки гнучкими або незмінними є схеми даних у базі даних. Структуровані дані відносяться до даних, які вписуються в чітко визначені моделі або схеми. Вони зазвичай зберігаються у реляційних базах даних (RDBMS), таких як MySQL або PostgreSQL, у формі таблиць. Такі дані часто використовуються в ситуаціях, де є велика потреба в консистентності і контролі даними.

Напівструктуровані або неструктуровані дані не вписуються в ці визначені схеми так само, як структуровані дані. Це означає, що у них є певна структура, але вони є значно більш гнучкими, що стосується формату і зміни даних. Прикладом і є описана вище MongoDB, яка зберігають дані у форматі JSON. Ці бази даних були створені для обробки великих обсягів нерегулярних або змінних даних.

Для розробника вибір між використанням структурованої та неструктурованої бази даних часто зводиться до вимог до даних та вимог до масштабування:

1) *вимоги до даних*: якщо дані відповідають чіткій схемі, яка не змінюється, можете використовувати структуровану базу даних. Якщо ми знаємо, що дані можуть змінюватися чи розширюватися, NoSQL база даних, як MongoDB, може бути більш гнучким вибором;

2) *вимоги до масштабування*: NoSQL бази даних, як правило, легше масштабувати горизонтально, а саме додаванням більшої кількості серверів до пулу, а реляційні бази даних часто масштабуються вертикально, тобто додаванням більшої кількості ресурсів до одного сервера.

У контексті проєкту, MongoDB може використовуватися для зберігання і обробки даних про активність користувача, зокрема інформації про використання окремих додатків, часу їх використання, скріншотів екрану тощо. Завдяки своїй високій продуктивності та можливості горизонтального масштабування, MongoDB є ідеально підходящим вибором для такого типу даних [16]. Вона також дозволить створювати складні запити до даних, що зберігаються в базі, а також

використовуватися для зберігання конфігурацій та налаштувань додатку для кожного з користувачів. MongoDB надає гнучкість для зберігання даних у форматі, який найкраще відповідає потребам.

Отже, в контексті проєкту, що реалізується, MongoDB дозволить ефективно зберігати, обробляти і аналізувати дані, необхідні для моніторингу активності користувача.

2.5 Шаблон проєктування Model–View–ViewModel

Розробляючи програмний застосунок, часто виникає питання по написанню алгоритму дій або існує повторювана архітектурна конструкцій під час проєктування програмного забезпечення. Паттерни проєктування, іншими словами шаблон – це приклад вирішення задачі, який можна використовувати у різних ситуаціях, що пропонує вирішення проблеми проектування в рамках деякого контексту, що часто виникає. Одним з таких шаблонів є шаблон Model–View–ViewModel (далі MVVM) (див. рис. 2.6).

MVVM – це шаблон архітектури додатків, який розділяє взаємодію користувача (UI логіку) та бізнес–логіку на три відокремлені компоненти: модель, представлення і ViewModel [17]. Це забезпечує чітке відокремлення відповідальностей між компонентами і полегшує тестування, підтримку і розвиток додатків [18]:

1) модель (Model): Модель представляє бізнес–логіку та дані додатку. Вона включає в себе бізнес–правила, валідацію даних, обчислювальну логіку, та інші аспекти, що стосуються збереження і обробки даних;

2) представлення (View): Представлення відповідає за відображення даних із моделі користувачеві, і за отримання відповіді від користувача. Воно не містить бізнес–логіки або стану додатку, його основна мета – представити дані і дозволити взаємодію;

3) ViewModel є мостом між моделлю і представленням. Вона зберігає стан представлення і обробляє команди, які отримуються від представлення.

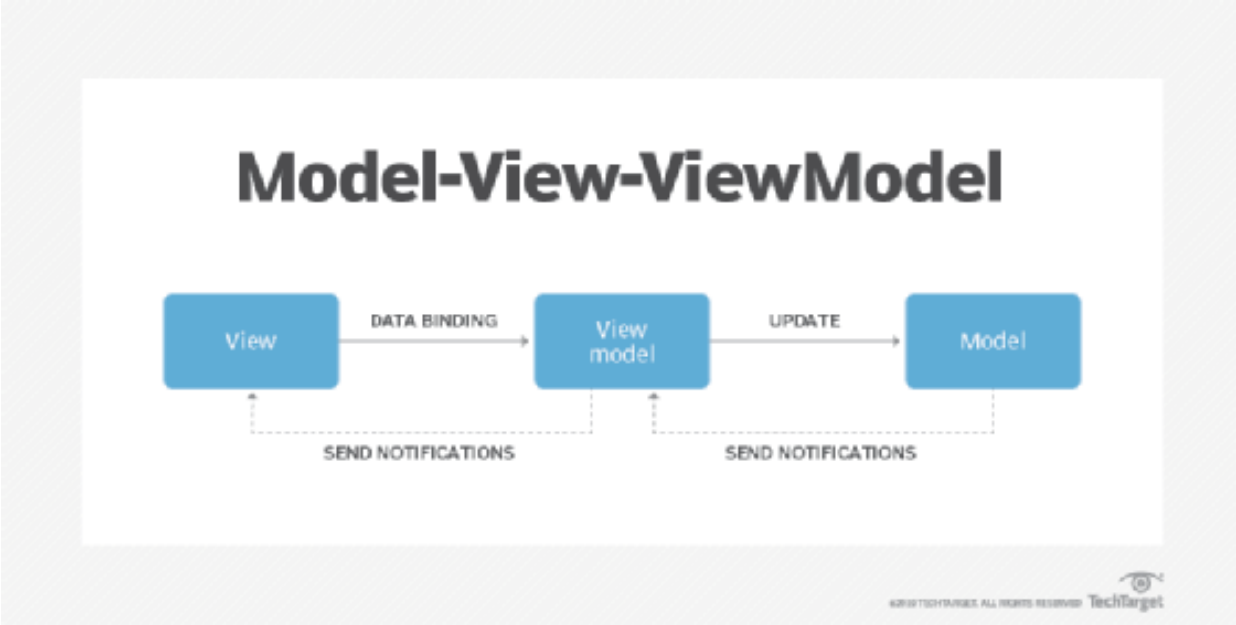


Рисунок 2.6 – Бізнес–логіка паттерну Model–View–ViewModel

MVVM було спеціально створено для підтримки двосторонньої прив’язки даних в XAML, що робить його ідеально підходящим для WPF, та інших XAML–базованих UI фреймворків [19]. Основні переваги MVVM включають:

1) відокремлення логіки – розділ бізнес–логіки і UI логіки полегшує розробку, тестування і підтримку додатку;

2) підтримка двосторонньої прив'язки даних – дозволяє моделі представлення використовувати повноцінну прив’язку даних до представлення, що зменшує потребу у коді взаємодії користувача;

3) підтримка автоматичного оновлення UI. Зміни в моделі представлення автоматично відображаються в UI за допомогою прив’язки даних;

4) тести – завдяки відокремленню представлення від логіки, можна легко тестувати бізнес–логіку без необхідності інтерфейсу користувача.

2.6 Середовище розробки Visual Studio 2022

Питання IDE є, насамперед, першочерговим від час прийняття рішення що до технологічних засад проекту. Розробка програмного забезпечення цілком і

повність залежить від вибраної мови та звісно сучасної IDE, що надає не тільки змогу писати код, але й повністю відповідає за написання правильного синтаксису, надає змогу посилатися на бібліотеки та фреймворки, сумісна з інструментами управління версіями для хостингу IT–проектів та їх спільної розробки.

Підійшовши до цього питання, та врахувавши усі попередні методи розробки, можна впевнено сказати, що для створення застосунку на базі .NET та C#, враховуючи використання усієї необхідної документації бібліотек, фреймворків та сторонніх систем – Visual Studio є впевненішим рішенням для реалізації системи моніторингу діяльності користувача.

Visual Studio – це інтегроване середовище розробки від компанії Microsoft, яке використовується для розробки програмного забезпечення для Windows, Android, iOS, вебсайтів, вебдодатків та вебслужб. Засноване на .NET Framework, Visual Studio підтримує розробку на мовах C#, Visual Basic .NET, C++, F#, JavaScript і Python, а також може бути розширено за допомогою плагінів для підтримки багатьох інших мов [20].

Visual Studio надає набір потужних інструментів та послуг, що спрощують процес розробки. Це включає редактор коду, дизайнер UI, сервер для налагодження, інструменти для управління версіями коду, інструменти для роботи з базами даних, інструменти для створення вебслужб, інструменти для розгортання додатків, і багато іншого.

Серед переваг Visual Studio – його здатність до розширення і налаштування. Розробники можуть користуватися вбудованими інструментами, а також додавати власні плагіни та розширення для покращення функціональності IDE. Це означає, що Visual Studio може бути налаштований для підтримки широкого спектру процесів розробки, від простих вебдодатків до складних корпоративних рішень.

Висновки до розділу 2

У цьому розділі було розглянуто та обґрунтовано вибір технологій та інструментальних засобів розробки системи моніторингу та аналізу діяльності користувача ПК, застосування яких дозволяє реалізувати необхідний набір функціональності та забезпечує ефективну і гнучку розробку.

Для розробки системи моніторингу та аналізу діяльності користувачів за комп'ютером обґрунтовано використання середовища розробки Visual Studio 2022, яке є ефективним та потужним інструментом для розробки застосунків, підтримує обрані для розробки мови програмування, має зручний інтерфейс. У якості мови програмування обрано C# і платформу .NET, які входять до складу стеку технологій Microsoft, відомого своєю надійністю і великою кількістю можливостей. Для створення графічного інтерфейсу користувача було використано систему побудови клієнтських застосунків Windows Presentation Foundation, який дозволяє створювати інтерфейс в декларативному стилі з використанням для опису інтерфейсу мови розмітки XAML, що спрощує процес розробки і підтримки коду.

Для роботи з базами даних використано документоорієнтовану СУБД MongoDB – представника NoSQL баз даних. Проєктування архітектури застосунку та розділення логіки і інтерфейсу здійснювалося з використанням шаблону MVVM: Model–View–ViewModel.

3 ПРОЄКТУВАННЯ СИСТЕМИ МОНІТОРИНГУ ДІЯЛЬНОСТІ КОРИСТУВАЧА

3.1 Опис вхідних даних та структури системи

Реалізація процесу відстеження діяльності має на увазі отримання інформації, що надає нам ПК, та роботу з отриманими даними. Під час збору таких даних необхідно вирішувати декілька питань що до їх наявності, місця збереження і звісно обсягу цих даних.

Почнемо з даних про активні програми та процеси. Дані про активні процеси відносяться до інформації, яка витікає з діючих програм, іншими словами процесів, на комп'ютері. Коли користувач запускає програму, операційна система створює "процес" для керування цією програмою. Кожен процес має ряд атрибутів, які можуть бути відстеженні і зберігатися для подальшого аналізу.

Операційна система Windows управляє і слідкує за всіма процесами, які виконуються в системі, та зберігає відповідну інформацію в системних ресурсах, які відомі як «управління завданнями». Однак ця інформація не зберігається в доступному для користувача файлі або базі даних, які можна просто відкрити і переглянути [21].

Замість цього, для доступу до цієї інформації потрібно використовувати спеціальні API, які надає Windows. Ці API включають вже відомі нам функції, такі як GetForegroundWindow і GetWindowThreadProcessId, які ми згадували у попередніх розділах. Функції дозволяють отримати додаткову інформацію про процеси. Таким чином, щоб отримати дані про активні процеси для вашої системи моніторингу, ви маєте написати код, який викликає ці API [22].

Щодо структури даних, дані про процеси зазвичай включають наступні елементи:

1) ім’я процесу – це назва виконуваного файлу (наприклад, notepad.exe або firefox.exe);

2) ID процесу – це унікальний ідентифікатор, який призначає операційна система кожному процесу;

3) час запуску – це момент часу, коли процес був запущений;

4) час завершення – це момент часу, коли процес був завершений. Зауважте, що для діючих процесів цей параметр може бути невідомим;

5) статус – це поточний статус процесу (наприклад, виконується, призупинено, завершено і так далі);

6) використання ресурсів – це може включати такі параметри, як використання ЦП, використання пам’яті, використання диска та інші.

Варто зазначити що дані такого типу не є ресурсозалежними і не потребують великої кількості місця у базі даних або у файлах. На рис. 3.1 зображено схему, яка дозволяє зрозуміти, як саме застосунок має змогу отримувати системні дані.

Ми викликаємо Windows API, який потім відправляє SYSCALL (системний запит) в ядро для збирання інформації або внесення необхідних змін.

Опис вхідних даних не завершується збором даних про активні вікна. Одним з переліку методів розробки системи моніторингу є відстеження вебактивності. Зауважимо, що на прикладі веббраузера Google Chrome було описано загальну методику що до даних, але ця інформація потребує деякого розкриття та детального опису процесів від початку збору цих даних до їх правильної обробки та використання [23].

Використовуючи браузер, користувач користується різними сторінками та вебдодатками, на основі відвідувань браузер формує історію пошуку (див. рис. 3.2). Усі процеси що відбуваються під час вебсерфінга потрапляють у визначене, розробником браузера, місцезнаходження, яким зазвичай є профілі користувача. Для Google Chrome та в ОС Windows цей шлях або стандартне місцезнаходження виглядає наступним чином:

C:\Users\<username>\AppData\Local\Google\Chrome\User Data\Default\History.

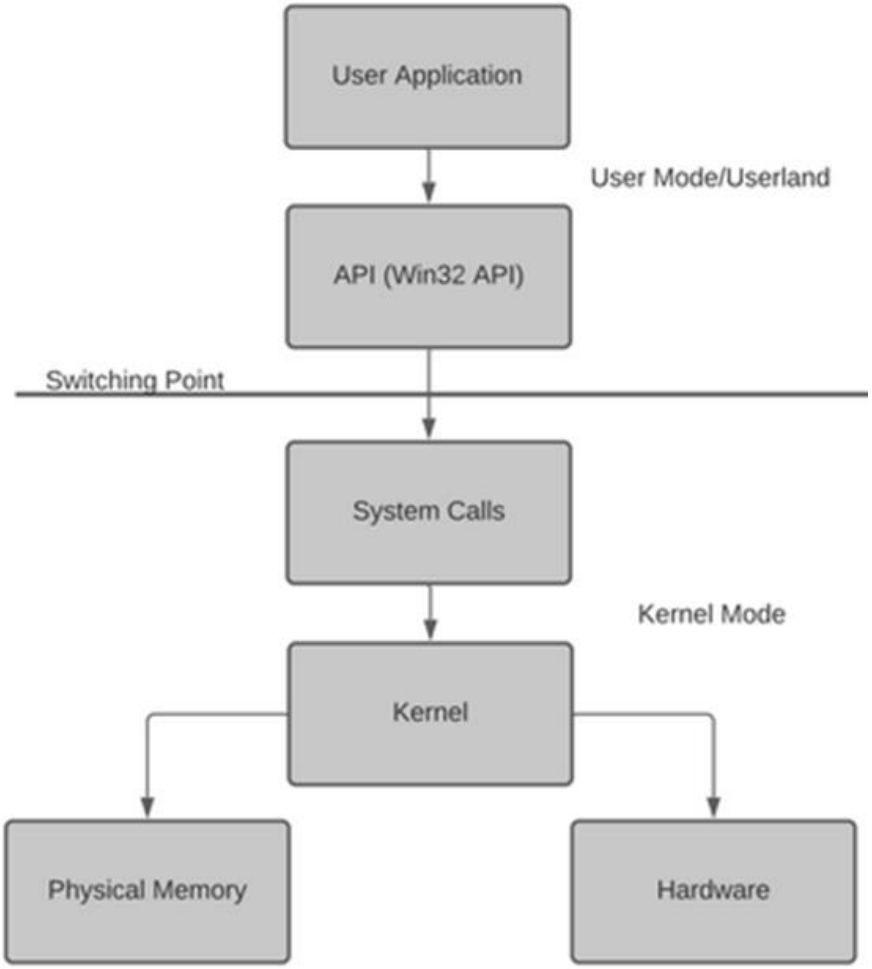


Рисунок 3.1 – Діаграма процесу запиту до системи через Windows API

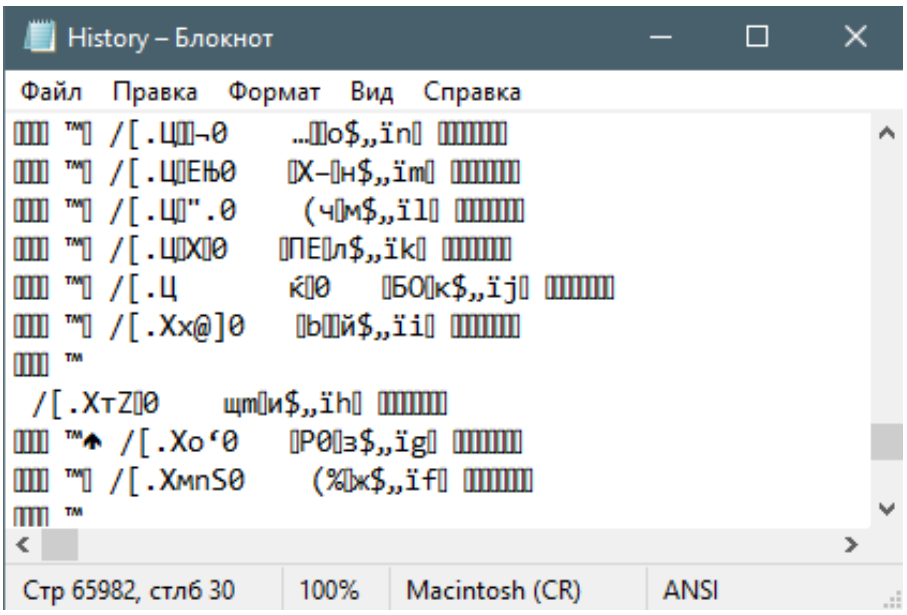


Рисунок 3.2 – Вигляд файлу History у форматі SQLite

Наступним кроком є читання та збору цих даних з вказаної директиви. Файл історії Google Chrome це база даних SQLite. Відповідно, щоб прочитати цей файл, нам знадобиться бібліотека або інструмент, що може працювати з SQLite. В C#, звісно, присутні фреймворки для роботи з усіма можливими БД, які розробник буде використовувати. У даному випадку, для конвертації даних з SQLite до виду, що може читати людина, використаємо бібліотеку System.Data.SQLite.

Розбір даних. В базі даних є таблиця *urls*, яка містить історію відвідування вебсайтів. Основні колонки цієї таблиці, які нам будуть цікаві, – це *url, title* та *last_visit_time*. Перші дві колонки є досить зрозумілими, але *last_visit_time* зберігається у форматі, який відлічується від '1601–01–01'. Тому знову ж таки з'являється задача конвертації цих даних до звичайного формату дати і часу.

Збір даних з таблиці *urls* здійснюється за допомогою SQL–запитів. Ми можемо використати SELECT–запит, щоб витягти потрібні дані. Наприклад, наступний SQL–запит витягне всі URL–и та часи їхнього останнього відвідування:

SELECT url, last_visit_time FROM urls.

Наступним кроком є збереження і аналіз даних (див. рис. 3.3).

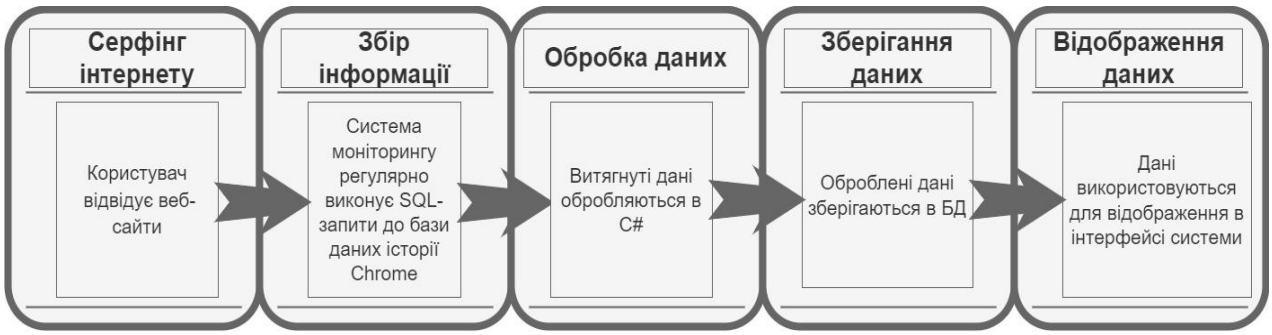


Рисунок 3.3 – Схема процесу отримання даних про вебактивність

Після того, як було опрацьовано файл *History* дані, ми можемо зберегти їх в MongoDB для подальшого аналізу і звітів. Звісно, також є можливість створення запитів до MongoDB для отримання необхідної інформації, наприклад, для отримання переліку вебсайтів, які користувач відвідав найбільше разів, тим самим створивши звітність що до відображення вебактивності користувача. Також

важливо зазначити, що не всі дії в браузері можуть бути відслідкувано цим способом, особливо якщо користувач використовує режим "інкогніто".

Кожен етап в цій послідовності є важливою частиною процесу моніторингу вебактивності, починаючи від користувацької активності і закінчуючи представленням зібраної і обробленої інформації.

3.2 Проєктування системи моніторингу

Проектування системи – процес доволі клопітливий та повинен визначати усі необхідні компоненти, ресурси, архітектури, модулі, інтерфейси та з рештою вхідні і вихідні дані для роботи системи з метою задовольнити визначені вимоги. Це також включає визначення стратегій та планів реалізації системи.

Під час проєктування системи беруться до уваги різні аспекти, включаючи архітектуру даних, програмне забезпечення та апаратне забезпечення системи, сценарії використання системи, інтерфейси користувача, а також проблеми безпеки та продуктивності [24]. Тож цей етап є найважливішим у циклі розробки системи, що дозволяє команді розробників або розробнику мати чітке бачення про те, як буде функціонувати система.

Почнемо проєктувати систему з визначення ролі користувачів у системі, а також визначення ролі адміністратора групи, якщо таких необхідний для керуванням та збору аналізу діяльності користувачів.

Користувач – user, що використовує систему моніторингу для визначення власної активності, перегляду часу витраченого на окремі програми та аналізу вебсайтів. Для своєї діяльності йому необхідно мати під рукою аналіз процесів у вигляді графіків та діаграм. Для свого рівня доступу він має належний функціонал, що до діяльності.

Керівник групи/Адміністратор – user, що відповідно має більш розширений доступ до функціоналу системи та користується нею для повного аналізу не тільки своєї діяльності, але й для моніторингу інших користувачів, що під’єднані до

групи. Такий користувач має доступ до списків використаних програм, вебсайтів, переліку скріншотів, або файлів кейлоггеру.

Для відокремлення різних рівнів користувачів, перш за все необхідно представити вимоги до системи реєстрації та авторизації, на якому user визначається зі своєю роллю. Цей етап також включає визначення того, які дані потрібні для реєстрації користувача, як буде відбуватися процес авторизації та які засоби відновлення доступу будуть доступні.

Для початку визначимося з переліком даних що до реєстрації. Для повного доступу до системи та функціонування і реалізації зв’язку між простим користувачем та адміністратором. Наступних даних під час реєстрації буде достатньо:

– логін відображає індивідуальне ім’я user у системі та авторизує користувача у систему за потреби;

– електронна адреса, знову ж таки необхідна для авторизації користувача та прив’язки даних до групи, а також для зв'язку з користувачем по за межами моніторинг системи;

– пароль звісно захищає користувача від небажаного входу в систему та є ключом що зберігається у БД у зашифрованому вигляді.

Даний перелік реєстраційних даних є цілком достатнім для виконання повного процесу реєстрації.

Під час процесу, як ми визначилися, буде необхідно відокремити звичайного користувача та адміністратора. Для цього лише буде необхідно поставити необхідну кнопку під час процесу реєстрації.

Процес авторизації повинен запустити вже зареєстрованого користувача у систему. З цією задачею можуть впоратись деякі описані у переліку про реєстрацію дані. Для прикладу це може бути поле або для введення електронної пошти або для логіну користувача. В розумних системах, зазвичай, такі поля об’єднують у одне, тим самим не ставлячи питання що до вибору способу авторизації. Це зручно не тільки для самого користувача але й для розробника UI.

Сам процес авторизації та реєстрації не є технічно складним у реалізації. Найголовнішою проблемою є запис та зчитування даних про користувача під час цього процесу. Тож вже на цьому етапі нам необхідно підключити БД для збереження даних та їх зчитування. На цьому етапі ми вирішуємо, як будуть зберігатися дані користувачів. Це включає в себе визначення схеми бази даних та створення необхідних таблиць та інших структур даних.

Отже, використовуючи MongoDB дані повинні автоматично записуватися у базу даних під час реєстрації та зчитуватись під час авторизації. Логіка даного процесу описана наступним чином:

– користувач зареєструвався та вписав дані у систему;

– в уже створеній базі даних та колекції *users* з’являється новий документ, кожен з яких представляє інформацію про окремого користувача. Кожен документ містить рядки інформації, такі як логін користувача, електронна адреса та зашифрований пароль. Отже, коли новий користувач реєструється, ми створюємо новий лист, записуємо на нього цю інформацію, а потім зберігаємо лист колекції *users.*

Для процесу авторизації реалізують наступні дії:

– коли користувач намагається увійти, ми перевіряємо колекцію користувачів, щоб знайти документ, який має таке ж ім'я користувача та електронну адресу;

– перевіряємо, чи зашифрований пароль на цьому листі відповідає тому, що ввів користувач. Якщо це так, ми дозволяємо користувачу увійти.

Це – основна ідея того, як працює система реєстрації та авторизації в контексті бази даних MongoDB.

Відновлення даних – необхідна функціональність для кожної системи. Для реалізації цього функціоналу нам буде необхідна пошта, яку користувач вже записав до БД застосунку. Цей функціонал повинен надавати можливість зміни паролю, за умови що користувач має прямий доступ до своєї пошти. Тож наступні кроки відновлення профілю матимуть такий вигляд:

– користувач вписує свій логін або електронну пошту;

– користувач виписує новий пароль та обов’язково підтверджує його;

– користувач вводить код що прийшов йому на пошту у спеціальне поле.

Таким чином, відновлення паролю також потребує зв’язку як з базою даних, так і з самим користувачем для відновлення діяльності його системи.

Підбиваючи підсумки про проєктування систем реєстрації, авторизації та відновлення даних у системі, створимо схему, що надає вище описаним словам логічне та інтерактивне представлення (див. рис. 3.4).

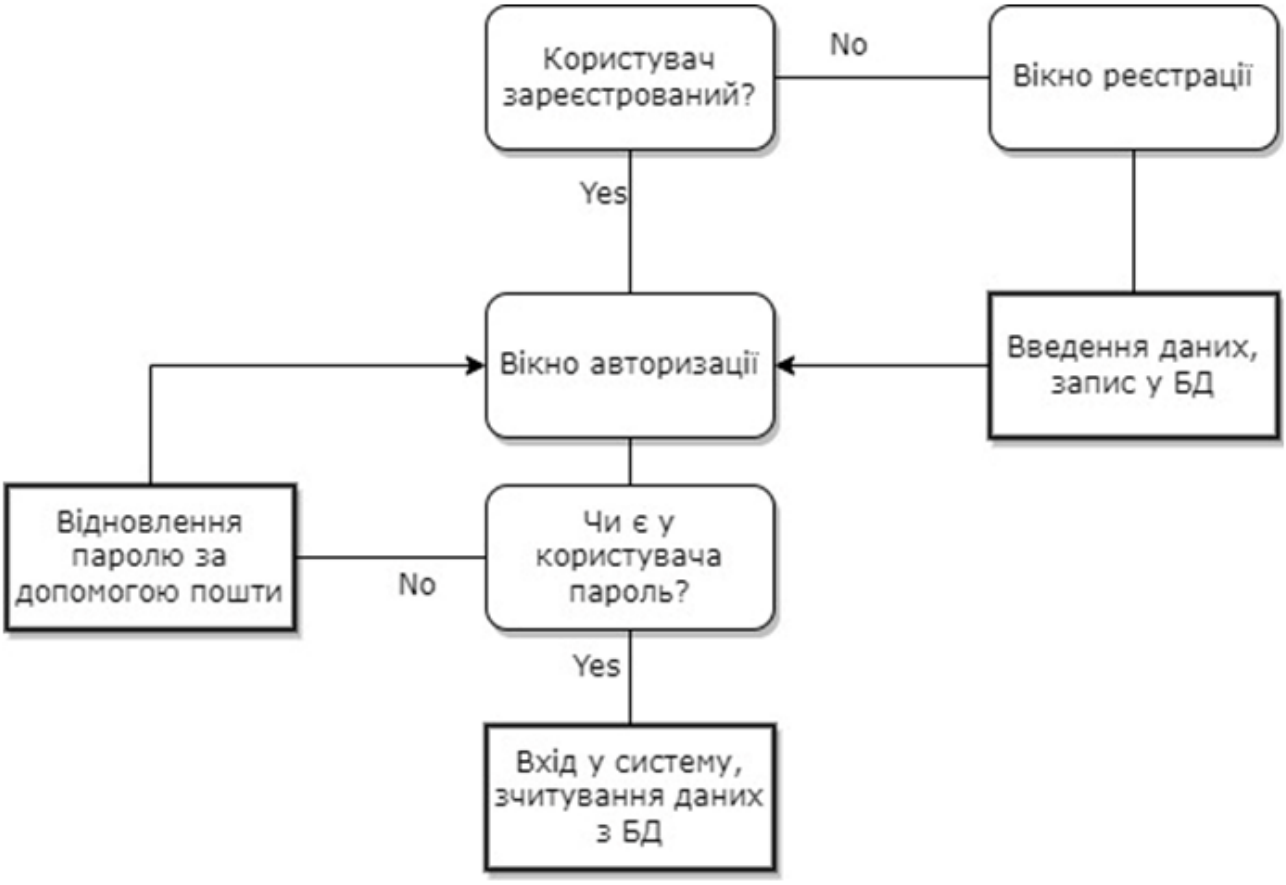


Рисунок 3.4 – Схема бізнес–логіки системи моніторингу діяльності

Опишемо детально процес отримання даних про діяльність. Описуючи збір даних про активні вікна та вебсайти, можна використовувати однин алгоритм.

На прикладі отримання та обробки даних вебактивності користувача розберемо процес моніторингу активності у вебзастосунках.

Впродовж роботи не раз було поверхнево описано процес отримання вебданих та їх обробки для звітування у графічній складовій системи, тож перейдемо одразу до структурної реалізації алгоритму обробки цих даних, запису та відображення у UI.

Після отримання документації щодо історії браузера користувача та їх зчитування за допомогою SQLite ми маємо на руках набір даних. Для зберігання цих даних будемо використовувати тимчасовий файл. Це зумовлено тим, що отримавши дані про пошукові запити, файл History буде продовжувати накопичувати дані про запити якщо людина використовує браузер у даний момент. Тож для відображення актуальних даних, та для запобігання виникненню помилок, файл History повинен обновлюватися.

Після екстракції даних, вони можуть потребувати подальшої обробки, щоб бути сумісними з базою даних, до якої вони будуть передані. Наприклад, можемо створити колекцію JSON–об'єктів, де кожний об’єкт представляє окремий запис вебактивності.

Після того, як дані готові до передачі, вони відправляються до бази даних завдяки бібліотеці MongoDB.NET Driver [25]. Використовуючи MongoDB, будемо створювати новий документ web_activity у колекції для кожної вебактивності.

Коли нові дані про вебактивність користувача стають доступними, вони додаються до відповідного документу web_activity в базі даних. Якщо вже існує документ для цього користувача, ми оновлюємо його новими даними додаючи та зберігаючи усі останні зміни активності.

Для отримання даних про вебактивність користувача з бази даних, використовуємо методи запитів в базі даних NoSQL, щоб витягнути відповідні документи з бази даних. Надалі ми можемо використовувати ці дані для аналізу активності користувача або відображення даних в графічному інтерфейсі.

Опишемо загальну архітектуру системи за допомогою ілюстрації бізнес–логіки (див. рис.3.5).

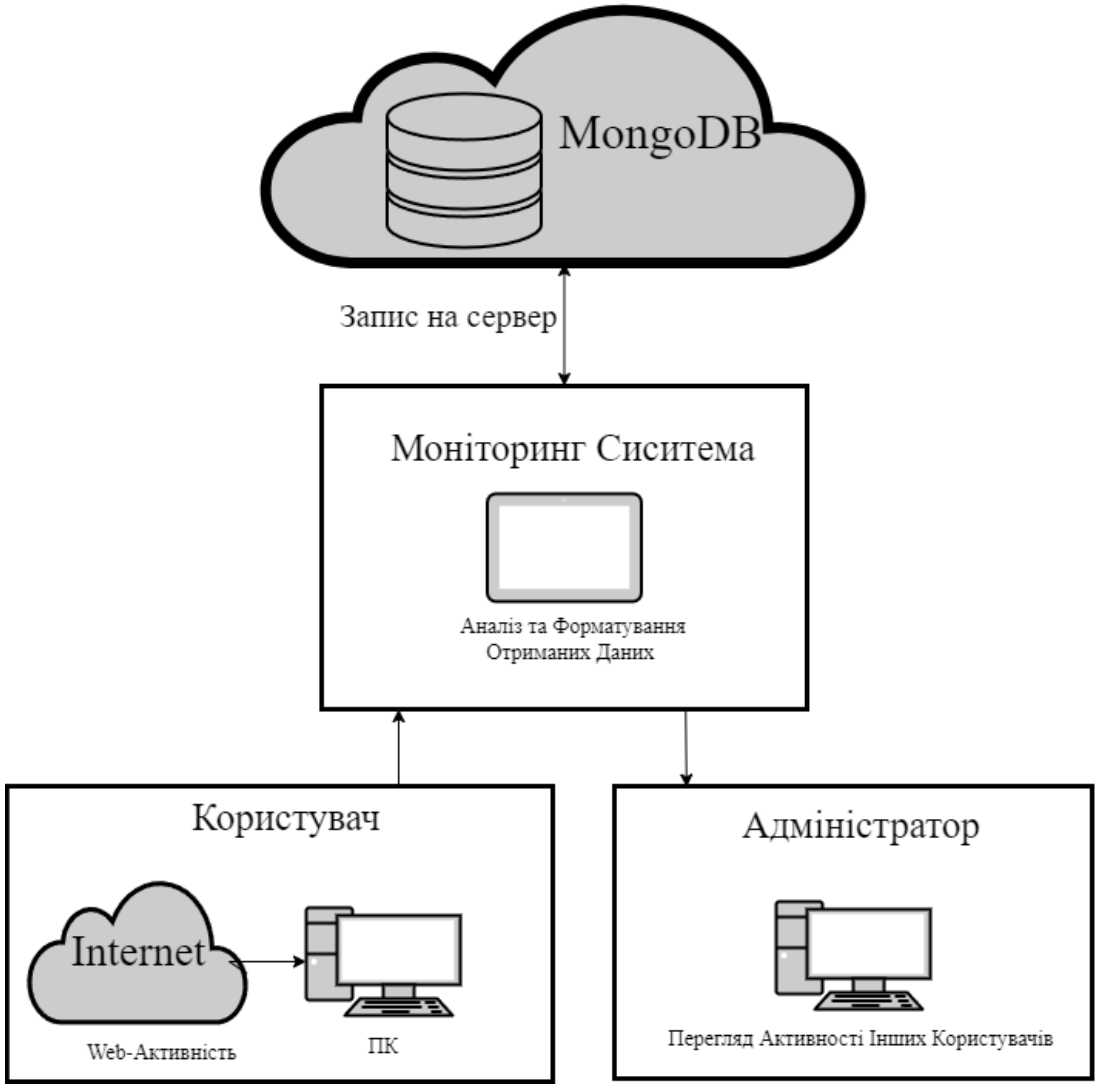


Рисунок 3.5 – Бізнес–логіка роботи системи моніторингу вебактивності

Висновки до розділу 3

У даному розділі було описано проєктування системи моніторингу діяльності користувача ПК. Побудова чіткої структури даних, яка враховує всі потрібні аспекти активності користувача, є фундаментом для розробки моніторинг систем. Тому було детально розглянуто структуру вхідних даних, яка є критичною для ефективного збору й аналізу інформації про активність користувача..

Вивчено та узагальнено основні характеристики та властивості вхідних даних, що формують основу для моніторингу, розглянуто структуру даних, які

підлягають аналізу, та визначено ключові параметри, необхідні для подальшої роботи над системою.

Таким чином, розділ відображає ключові моменти процесу проєктування системи моніторингу, включаючи розробку структури і проектування системи в цілому. Детальний аналіз вхідних даних та ретельне проєктування є підґрунтям для створення ефективної та надійної системи моніторингу та аналізу діяльності користувача у відповідності до поставлених задач.

4 РОЗРОБКА ТА ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ МОНІТОРИНГУ Й АНАЛІЗУ ДІЯЛЬНОСТІ КОРИСТУВАЧА ПК

4.1 Початковий етап розробки системи

Початковий етап розробки є останньою складовою створення системи, впровадження усіх необхідних бібліотек, фреймворків, вибір технологій, розробка моделей даних, алгоритмів, створення UI та безпосередньо тестування системи.

Відповівши на перших етапах роботи на основні питання що до визначення вимог до системи було вирішено наступні ключові положення:

1) що має робити система;
2) хто буде її використовувати;
3) які дані вона повинна збирати і аналізувати.

Це визначає багато важливих аспектів системи, включаючи її архітектуру, дизайн, та механізми роботи алгоритмів. Враховуючи ці вимоги та потреби, було розроблено систему моніторингу та аналізу активності: визначено її архітектуру, обрано технології, розроблено моделі даних.

Визначившись на попередніх етапах з усіма необхідними рішеннями, підійшли до критично важливого для успіху проєкту пункту, оскільки він визначає, як з точки зору користувача система буде виконувати свої ключові функції та, найголовніше, яким чином вона буде реалізована.

Ключовими інструментальними засобами та технологіями, задіяними при розробці системи є такі компоненти:

– C# та платформа .NET;
– UI на базі системи WPF;
– MongoDB як БД та MongoDB.Driver як фреймворк для роботи з БД у середовищі C#;
– Material Design як графічна бібліотека для WPF [26];
– MVVM як шаблон реалізації усієї логіки у UI.

Для початку створюється C# застосунок на базі WPF та підключаються усі необхідні бібліотеки і фреймворки. Після чого буде в наявності пуста панель під назвою *MainWindow.xaml* та декілька інших файлів. Наразі не потрібно займатися графічною складовою системи, вона буде дороблена у подальшому.

Після створення самого проєкту відбувається створення БД. Впродовж роботи було зазначено, що користувач та адміністратор зможуть мати доступ до записів віддалено. Тож цей аспект враховують, створюючи БД не локально, а використовуючи сервер, де й будуть зберігатися записи.

На вебресурсі – сервері MongoDB Cloud, доступному 24/7, створюється база даних, зображена на рис. 4.1.

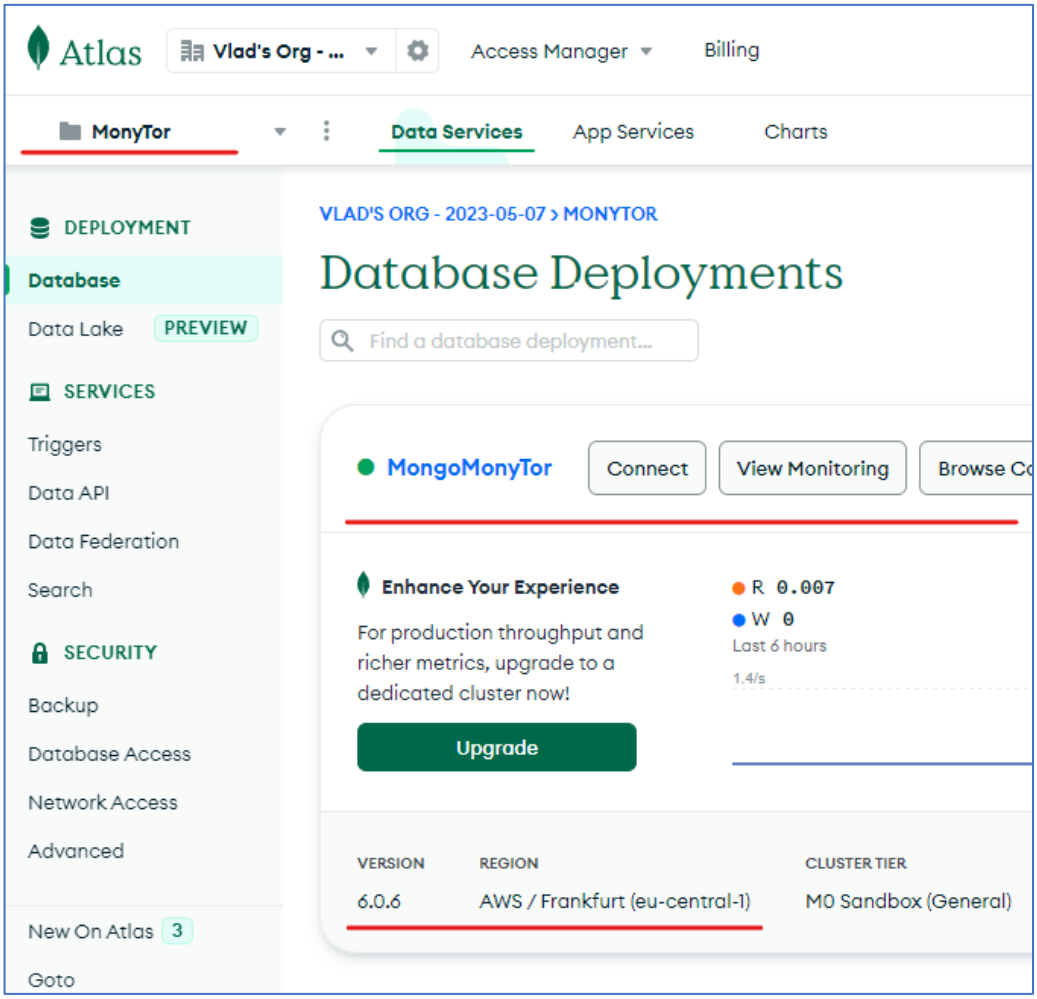


Рисунок 4.1 – Новостворена БД

На рис. 4.1 відображено (підкреслено червоним) сервер що містить базу, назву та інші вхідні дані про створені колекції. Для зручності MongoDB надає можливість завантажити десктоп застосунок для роботи з БД, що буде відображати усю необхідну інформацію про стан бази, документів, колекції, тощо. Завантажуємо додаток MongoDB Compass та створюємо нове підключення, використовуючи рядок підключення формату URI (Universal Resource Identifier). На рис. 4.2 зображено вже підключену базу з доданими колекціями та документами.

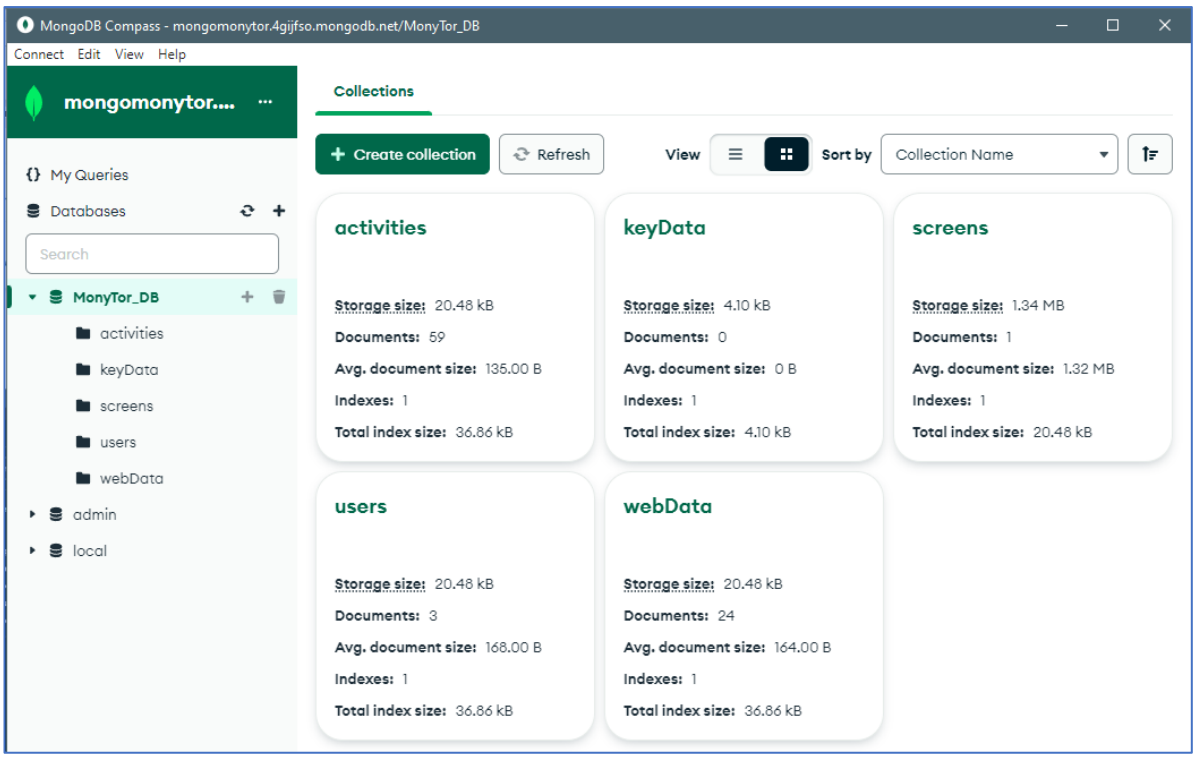


Рисунок 4.2 – Робота над БД у застосунку MongoDB Compass

Останнім кроком при підключенні БД є створення у застосунку класу *MongoDbContext* та інших додаткових класів і інтерфейсів (див. додаток А), які будуть проводити взаємодію між системою та базою даних, впроваджуючи свою логіку для створення, оновлення, видалення та пошуку даних у документах різних колекцій.

4.2 Моніторинг та аналіз вебдіяльності

Моніторинг та аналіз вебдіяльності – це важливий інструмент, який дозволяє краще розуміти поведінку користувачів у мережі. Впродовж роботи було розкрито методику збору, відстеження та аналізу даних про вебсайти, які відвідують користувачі.

На прикладі використання браузера Chrome можна бачити, як це працює. Хром автоматично веде історію перегляду, збираючи інформацію про всі відвідані вебсайти. Дані, що збираються, включають багато інформації про дії користувача з сайтом, а також дані про завантаження різноманітних файлів та інше.

Ці дані можуть бути використані для аналізу поведінки користувача. Аналізуючи історію перегляду, можна виявити патерни поведінки користувача, виявити часто відвідувані вебсайти, визначити періоди найбільшої активності тощо. Все це може бути корисним для надання персоналізованих рекомендацій, покращення продуктивності користувача, а також для інших цілей.

Почнемо з першого етапу, а саме зі збору даних. Історія дій користувача в браузері Google Chrome зберігається log–файлі. За допомогою стороннього вебзастосунку продемонстровано початковий вигляд даних у файлі *History* (див. рис. 4.3). Файл містить посилання, назву ресурсу та час візиту у форматі, відомому як «Webkit timestamp». *Webkit timestamp* – це число мікросекунд, яке минуло з півночі 1 січня 1601 року [27]. Вміст цього файлу необхідно проаналізувати та довести до вигляду, який можна прочитати, перед тим як вносити записи у БД.

Для читання файлу було створено клас *ChromeHistoryData* (додаток Б) та підключено необхідну бібліотеку System.Data.SQLite. Функція *GetChromeHistoryDataAsync()* даного класу повертає список об’єктів свого класу. Приймати список та оброблювати його має функціонал описаний у класі *Service.Start()*, що викликається кожен раз коли користувач оновлює початкову сторінку *MainView.xaml*. *Service.Start()* відповідає за повне оновлення даних під час моніторингу не тільки вебактивності, але й активних вікон.

Оброблені дані приймаються у *Service.Start()*, аналізуються та записуються їх до бази даних у функції *Service.AddOrUpdateWebActivity()* (додаток В).

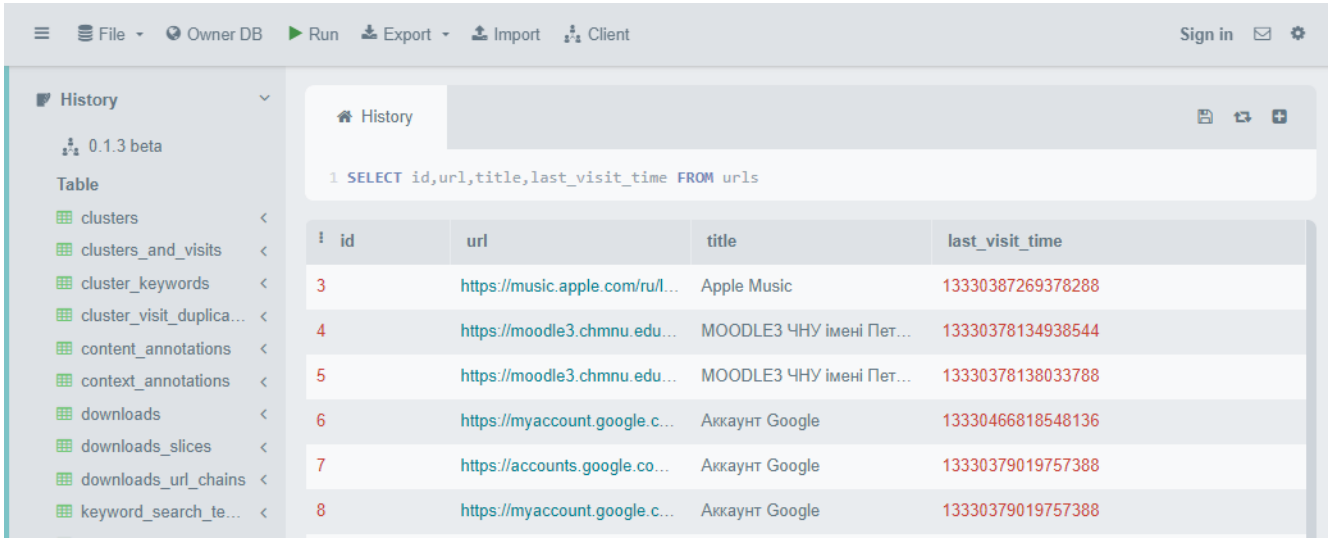


Рисунок 4.3 – Вміст SQLite файлу History

Процес аналізу та обробки даних у останній функції розглянемо детальніше. Функція відповідає за додавання нової активності в мережі (об’єкта Web) або оновлення існуючої активності для певного користувача:

1) спочатку функція викликає метод *GetUserWebData()*, який забирає дані з БД для отримання всіх вебактивностей даного користувача;

2) потім перевіряє, чи існує активність з таким же URL і назвою сторінки;

3) якщо активність з таким URL і назвою вже існує, вона оновлює дані цієї активності методом Update;

4) в іншому випадку вона перевіряє, чи існує активність або з таким URL, або з такою назвою сторінки. Якщо так, і при цьому URL і назва не відповідають одночасно, тоді вона оновлює час останнього відвідування для активності з відповідною назвою;

5) якщо ж активність з таким URL і назвою не було знайдено, то вона створює нову активність за допомогою методу Create.

Функція *Service.AddOrUpdateWebActivity()* реалізує логіку «додати або оновити», тому вона важлива для уникнення дублювання даних у базі даних. Завдяки їй можна тримати дані про активність користувача в мережі актуальними та організованими. Розберемо це на конкретних прикладах.

Аналіз вмісту розкритого SQLite файлу History (див. рис. 4.3) показав, що користувач, мабуть, три рази відкривав сторінку під назвою «Аккаунт Google», тобто заходив три рази поспіль на один і той же сайт. Та ця заява є хибною. Система Google для початку реєструє активність користувача що перейшов за посиланням, потім встановлює чи є користувач авторизованим, тож непомітно переадресовує [28]. Процес переадресації є непомітним, та історія все одно відображає переходи за кожним посиланням, і їх може бути безліч. Важливою задачею є встановлення факту, що користувач зайшов на сайт і користується ним повноцінно, використовуючи одне й те саме доменне ім’я.

На рис. 4.4 показано, що відвідуючи один сайт із виділеним доменом, можна користуватися різноманітним функціоналом. Та все ж таки аналіз пулу даних про вебактивність враховує цей пункт і записує до БД лише один домен сайту, оновлюючи лише час останнього візиту та безпосередньо кількість цих візитів. Аналіз оброблених в MongoDB даних дає можливість перевірити правильність роботи аналізу та запису (див. рис. 4.5).

Вирішивши питання що до дублювання даних та їх правильності запису може виникнути інше питання. Як саме можна відслідковувати дії користувача, коли він вийде з сайту, а потім знову увійде? На це питання є проста відповідь. Коли користувач перший раз заходить на сайт, дані, збережені у log–файлі, встановлюють за цим посиланням кількість візитів, а система в свою чергу зчитує дані з кінця до початку. Побачивши це посилання, титул сайту та змінену кількість візитів, система просто оновлює запис. Це дозволяє правильно та логічно коригувати дані.

Рисунок 4.4 – Користування різним функціоналом одного сайту

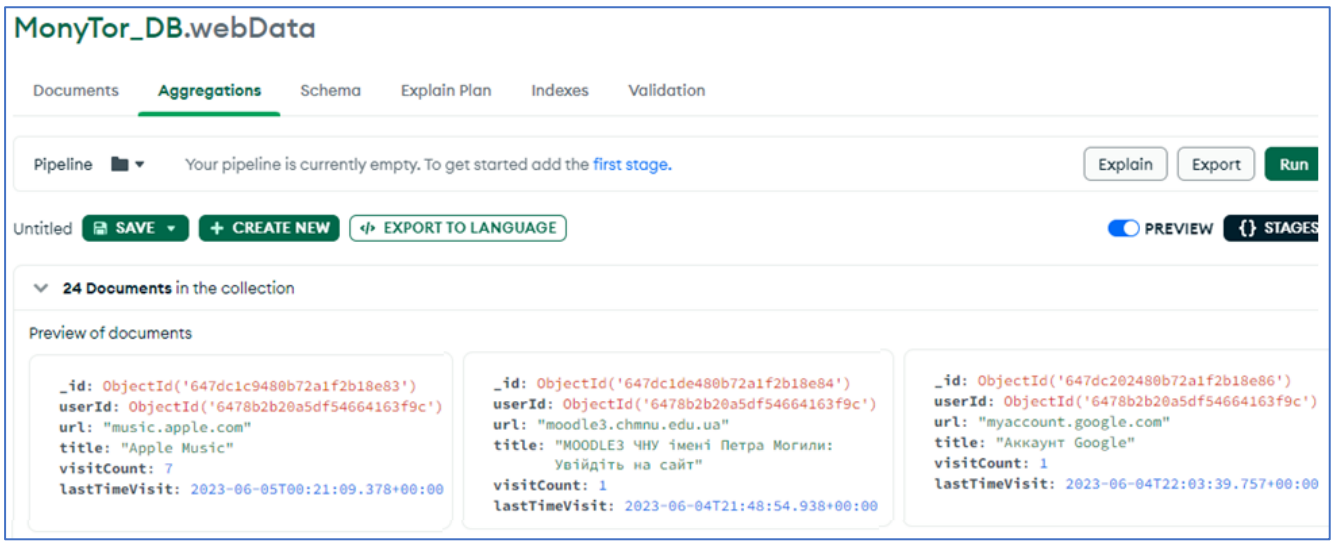


Рисунок 4.5 – Записані і збережені дані у БД MongoDB

Працюючи далі з даними про вебактивність, переходять до збору даних з БД, їх аналізу, та виведення на UI. Для початку було створено представлення, на якому відображаються дані, та згідно шаблону MVVM створюється ViewModel до цього

виду. Опишемо клас *WebAnalysisViewModel* та інші класи, розроблені для реалізації взаємодії та аналізу вхідних даних (додаток Г).

Клас *WebAnalysisViewModel* забезпечує взаємодію між видом та логікою, збирає дані з БД які безпосередньо впроваджуються у графічний інтерфейс. У свою чергу, клас *WebHoursActivityViewModel* проводить аналіз даних та виводить на екран аналіз активності користувача погодинно.

Фінальна версія сторінки з моніторингом вебактивності містить дві області. Зліва на сторінці відображено ресурси та посилання, за якими користувач здійснював перехід, дату, час та кількість зроблених візитів (див. рис. 4.6). Користувач та адміністратор мають можливість здійснювати перехід за активним посиланням.

Рисунок 4.6 – Моніторинг вебактивності користувача

Справа на сторінці з моніторингом вебактивності міститься діаграма, яка відображає загальну активність по годинах – період часу, коли користувач більш за все користується інтернетом (див. рис. 4.7). Відмічаємо коректну роботу посилань та аналізу вебактивності по годинах, які відображають у який період часу користувач був особливо активний під час роботи з вебресурсами.

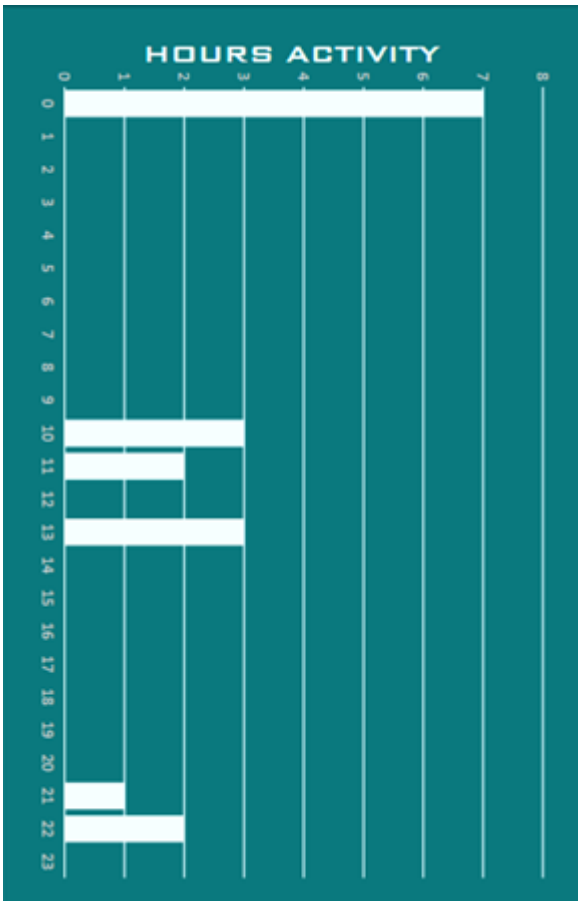


Рисунок 4.7 – Діаграма загальної вебактивності по годинах

4.3 Моніторинг та аналіз діяльності активних вікон

Моніторинг та аналіз діяльності активних вікон – це процес слідкування та аналізу дій, що виконуються користувачем під час роботи з десктоп застосунками, з метою отримання цінної інформації про робочу активність або використання ресурсів. Він може допомогти у визначенні шаблонів поведінки, ідентифікації недоліків у продуктивності, забезпеченні безпеки системи та багатьох інших задачах.

Описаний впродовж роботи функціонал включає в себе збір даних про діяльність користувача та системи, їх обробку та представлення у вигляді, який полегшує аналіз. Почнемо описувати процес реалізації з опису системної функції *GetForegroundWindow()*, яка вже згадувалася у попередніх параграфах. Даний метод – це частина Windows API, яка використовується для отримання

ідентифікатора вікна, яке в даний час знаходиться на передньому плані. Зазначимо, що *GetForegroundWindow()* повертає лише вікна верхнього рівня, вона не повертає дескриптори дочірніх вікон або вікон контролю. Важливо зауважити, що ця функція повертає дескриптор вікна, а не інформацію про застосунок, який використовує це вікно. Для отримання такої інформації можуть використовуватися інші функції API Windows, такі як *GetWindowThreadProcessId()*.

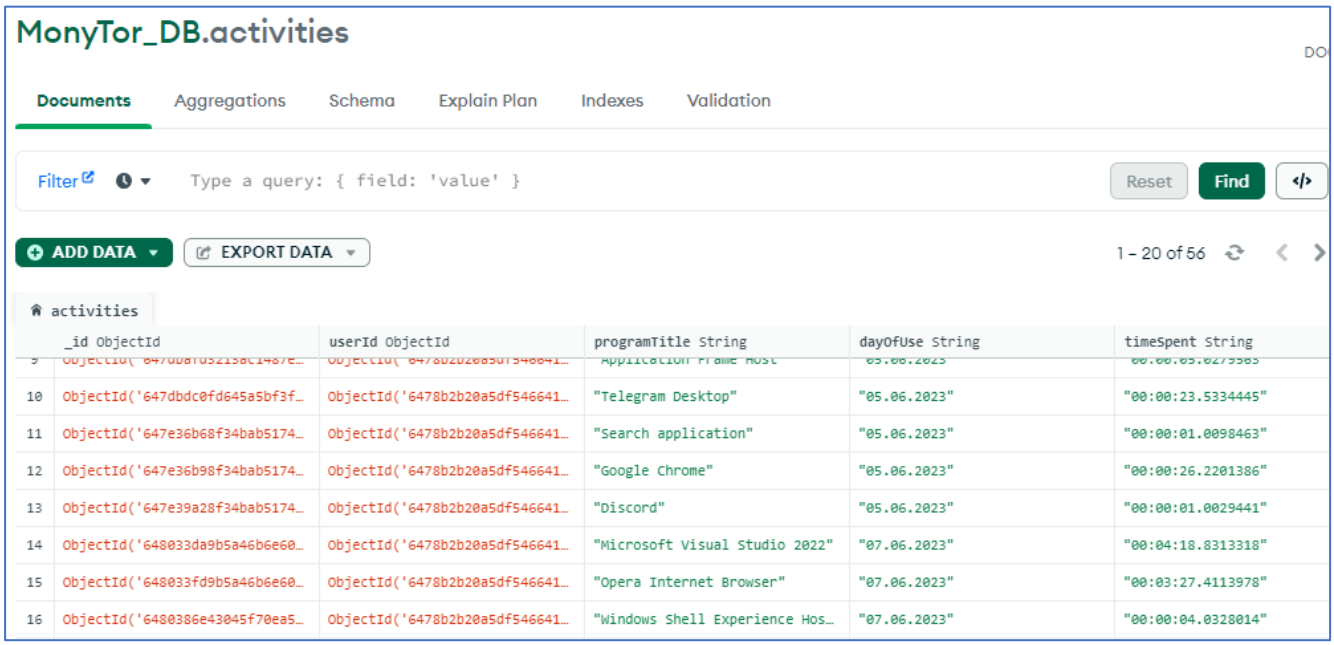
Було створено метод *GetActiveWindowTitle()* (див. додаток Д) у тому ж класі *System* та його методі *Start(),* у якому викликаються методи читання та запису під час відстеження вебактивності. За його допомогою отримується опис активного процесу, який відображається у вікні переднього плану. Спочатку отримується ідентифікатор процесу, який створив вікно переднього плану. Далі здійснюється перевірка, чи входить цей процес у список заблокованих системних процесів *_blockedSystemProcesses*. Якщо так, функція повертає *null*. Якщо процес не заблоковано, за допомогою *Process.GetProcessById()* отримуємо об'єкт Process, що відповідає даному ідентифікатору процесу.

Нарешті, опис файлу основного модуля процесу повертається як результат. Саме таким чином реалізовано отримання інформації про програму, що використовує активне вікно. Функція є корисною саме при моніторингу того, які додатки користувач використовує у реальному часі.

Далі БД заповнюється даними, отриманими від системи, із використанням аналогічної вже описаної вище функції *AddOrUpdateActivity()* (додаток Е) для запису та оновлення документів у базі даних. Також у методі *Start()* відбувається процес вимірювання часу, що дозволяє запустити таймер під час запуску вікна, а оновлення даних у БД відбувається за рахунок суми часу, який вже записано і того що зчитується у поточний момент часу.

У цілому описаний функціонал виконує зчитування та запис даних до MongoDB, використовуючи пряме з’єднання з системою Windows та атрибутом *DllImport("user32.dll"),* який вказує на файл, в якому містяться описані функції Windows API.

Протестувавши систему, було здійснено перевірку записаних даних про активні вікна у базі даних (див. рис. 4.8).



	_id ObjectId	userId ObjectId	programTitle String	dayOfUse String	timeSpent String
10	ObjectId('647dbdc0fd645a5bf3f…	ObjectId('6478b2b20a5df546641…	"Telegram Desktop"	"05.06.2023"	"00:00:23.5334445"
11	ObjectId('647e36b68f34bab5174…	ObjectId('6478b2b20a5df546641…	"Search application"	"05.06.2023"	"00:00:01.0098463"
12	ObjectId('647e36b98f34bab5174…	ObjectId('6478b2b20a5df546641…	"Google Chrome"	"05.06.2023"	"00:00:26.2201386"
13	ObjectId('647e39a28f34bab5174…	ObjectId('6478b2b20a5df546641…	"Discord"	"05.06.2023"	"00:00:01.0029441"
14	ObjectId('648033da9b5a46b6e60…	ObjectId('6478b2b20a5df546641…	"Microsoft Visual Studio 2022"	"07.06.2023"	"00:04:18.8313318"
15	ObjectId('648033fd9b5a46b6e60…	ObjectId('6478b2b20a5df546641…	"Opera Internet Browser"	"07.06.2023"	"00:03:27.4113978"
16	ObjectId('6480386e43045f70ea5…	ObjectId('6478b2b20a5df546641…	"Windows Shell Experience Hos…	"07.06.2023"	"00:00:04.0328014"

Рисунок 4.8 – Записи у БД про активні вікна

Подальша робота з цими даними потребує UI для реалізації видимих діаграм, списків, відображення даних, тощо. Аналогічно до реалізації представлення вебактивності, створюється сторінка, яка буде містити повну інформацію про аналітичну складову моніторингу. До даного xaml–файлу було написано клас, що реалізує шаблон MVVM – *DesctopAnalysisViewModel* (додаток Ж). Клас реалізує поля об’єктів інших класів, що мають алгоритми аналізу отриманих з БД даних. Клас *DesctopRatingViewModel()* збирає дані з БД, які безпосередньо впроваджуються у графічний інтерфейс.

Опишемо декілька прикладів реалізації аналізу активних вікон на прикладі класів *WeekActivityViewModel* та *DiagramWeekActivityViewModel* (додаток З).

Клас *WeekActivityViewModel* використовується для збору та аналізу інформації про усі активні вікна та період їх роботи протягом останнього тижня, інші класи даного типу оброблюють інформацію для днів тижня та місяців.

Розглянемо дані, на основі яких проведено розрахунки у класі *WeekActivityViewModel*. Перш за все, проводиться розрахунок кількості активностей за тиждень. Це робиться за допомогою обчислення кількості елементів в колекції завантаженій з БД, дати яких відповідають вказаному тижню. Умова перевіряє, чи дата кожної активності співпадає з поточним тижнем.

Наступним обчисленням є розрахунок сумарного часу, проведеного у роботі з активними вікнами. Це робиться за допомогою сумування часу для всіх вікон, дата яких відповідає вказаному тижню. Дані, що були зібрані у даному та інших подібних класах, відображаються на панелі користувача окремо.

Перейдемо до опису реалізації діаграм на UI панелі, та їх розробки. За задумкою діаграми повинні аналізувати колекцію завантажених з БД даних для визначення часу проведеного у активних вікнах протягом днів тижня, або місяців.

Клас *DiagramWeekActivityViewModel* реалізує наступну логіку, за якою визначається та розраховується загальний час активності для кожного дня поточного тижня. Це виконується шляхом створення словника, де ключі відповідають дням, а значеннями є сумарний час активності для кожного дня. Забираючи по одному запису з колекції ми сумуємо час проведений у вікні, поки записи не закінчиться умовою, що дата роботи додатку не входить у дату поточного тижня. Аналогічно цій системі обчислення працює обчислення сумарного часу за місяцями.

У цілому дані формули та обчислення використовуються для створення колекції даних, яка використовується для відображення даних про активність користувача за поточний тиждень у графічному відображенні за допомогою діаграм.

Таким чином, було визначено, як працюють та використовуються дані, проаналізовані та оброблені спеціальними класами. Було зазначено, що після об'єкти класів збираються у *DesctopAnalysisViewModel* та здійснюється їх виведення на представлення. Фінальний вигляд сторінки зображено на рис. 4.9.

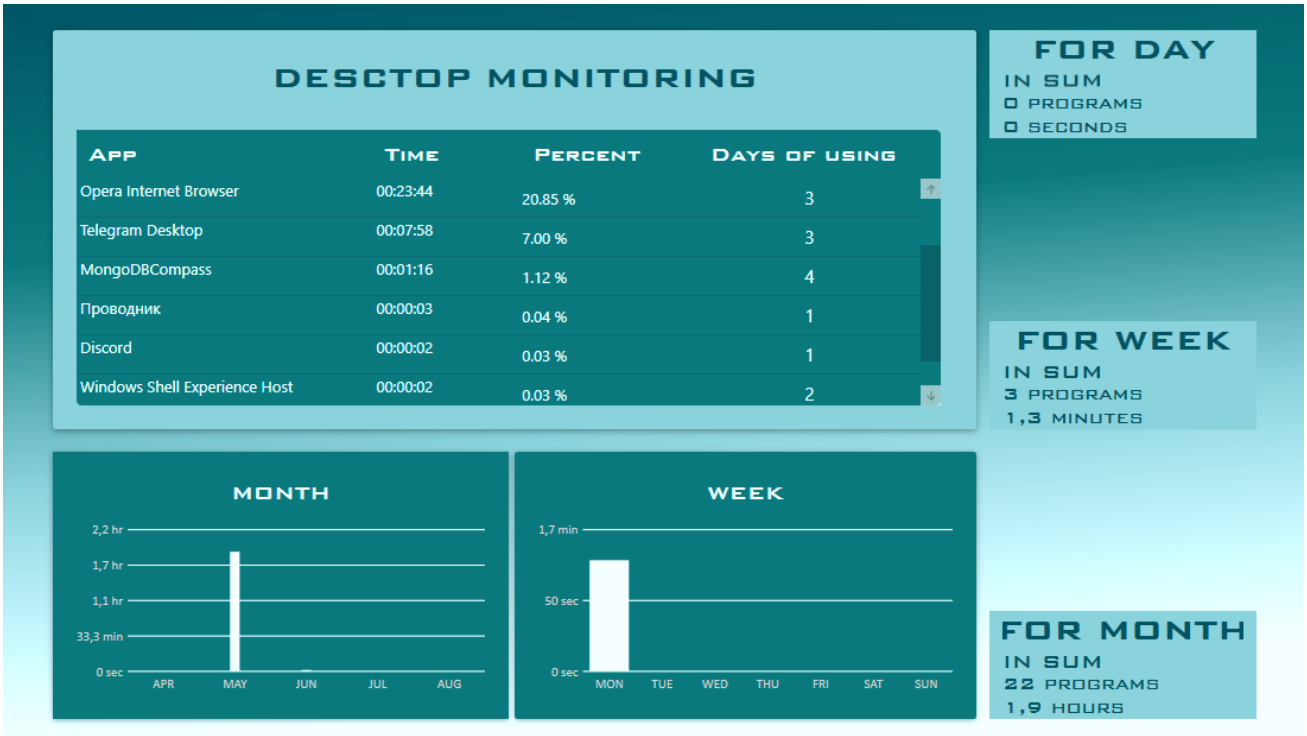


Рисунок 4.9 – Представлення Desktop Monitoring

4.4 Панель адміністратора

Для більшості вебресурсів, організацій, інтернет–магазинів, комерційних платформ, вебзастосунків та інших цифрових сервісів, панель адміністратора стає невід'ємною частиною їх екосистеми. Вона служить стратегічним центром управління, який надає користувачам – зазвичай адміністраторам або менеджерам можливість контролювати, налаштовувати та переглядати важливі аспекти своєї системи.

Адміністративна панель, також відома як панель управління – це інтерфейс, де дозволяє отримати доступ до різних параметрів та налаштувань ресурсу. Вона дозволяє адміністраторам не лише керувати контентом сайту, але й виконувати багато технічних задач, таких як управління базами даних, відстеження статистики та аналітики, налаштування параметрів безпеки та конфігурації системи, управління користувачами та їх ролями, і багато іншого.

Розглянемо створення адміністративної панелі, що використовується для керування та оптимізації ресурсів та сервісів, а також опишемо різні інструменти, можливості та налаштування, які надає така панель.

Отже, у попередніх розділах при описі структури програми, ми згадували про адміністрування як про основну вимогу до структурної організації. При реєстрації адміністратора він встановлює параметр, що і дозволяє йому не тільки мати додаткову галочку при записі в БД, але й міняє навігаційну систему, даючи доступ до окремої адміністративної панелі.

Дана панель може представляти собою вже відомі графіки, списки чи діаграми. Та заповнюватися вони будуть в окремо визначеному порядку взаємодії зі звичайним користувачем. Також панель може містити додатковий функціонал, як наприклад встановлення методу, що робить скріншоти під час моніторингу окремого користувача, або інших, різноманітних програмних засобів стеження.

На рис. 4.10 зображено UI панель для адміністратора та опишемо, який додатковий функціонал встановлено.

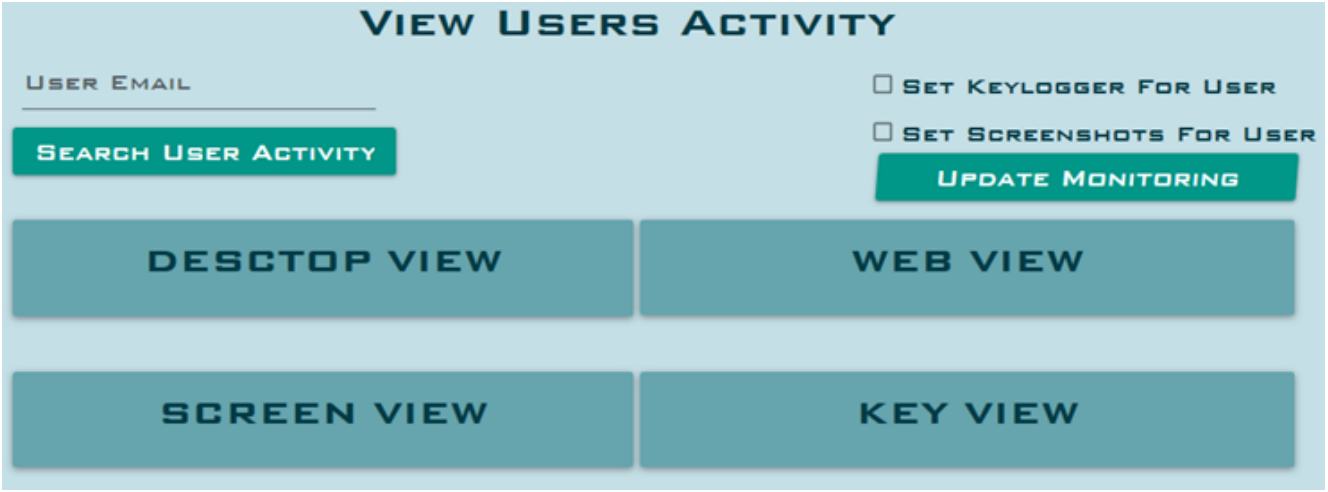


Рисунок 4.10 – Графічне представлення панелі адміністратора

Перш за все бачимо, що панель адміністратора має активні поля для пошуку користувача. У поле необхідно ввести електронну пошту та натиснути пошук. Клас *AdminViewModel* оброблює запит на конкретного користувача, виводить

інформацію про нього та надає можливість перегляду як панелей, що вже описані, так і панелей додаткового функціоналу.

Чинний функціонал дозволяє віддалено слідкувати за змістом документів БД, куди записується інформація про діяльність користувача. Для його реалізації у клас *AdminViewModel* (додаток И) додається обробка подій та зчитуються дані з MongoDB.

Панелі, що містять інформацію про веб та десктоп активності, розгортаються та згортаються, показуючи інформацію про обраного адміністратором користувача. На прикладі демонстрації вебактивності продемонструємо, як адміністратор може переглядати активність користувача (див. рис 4.11).

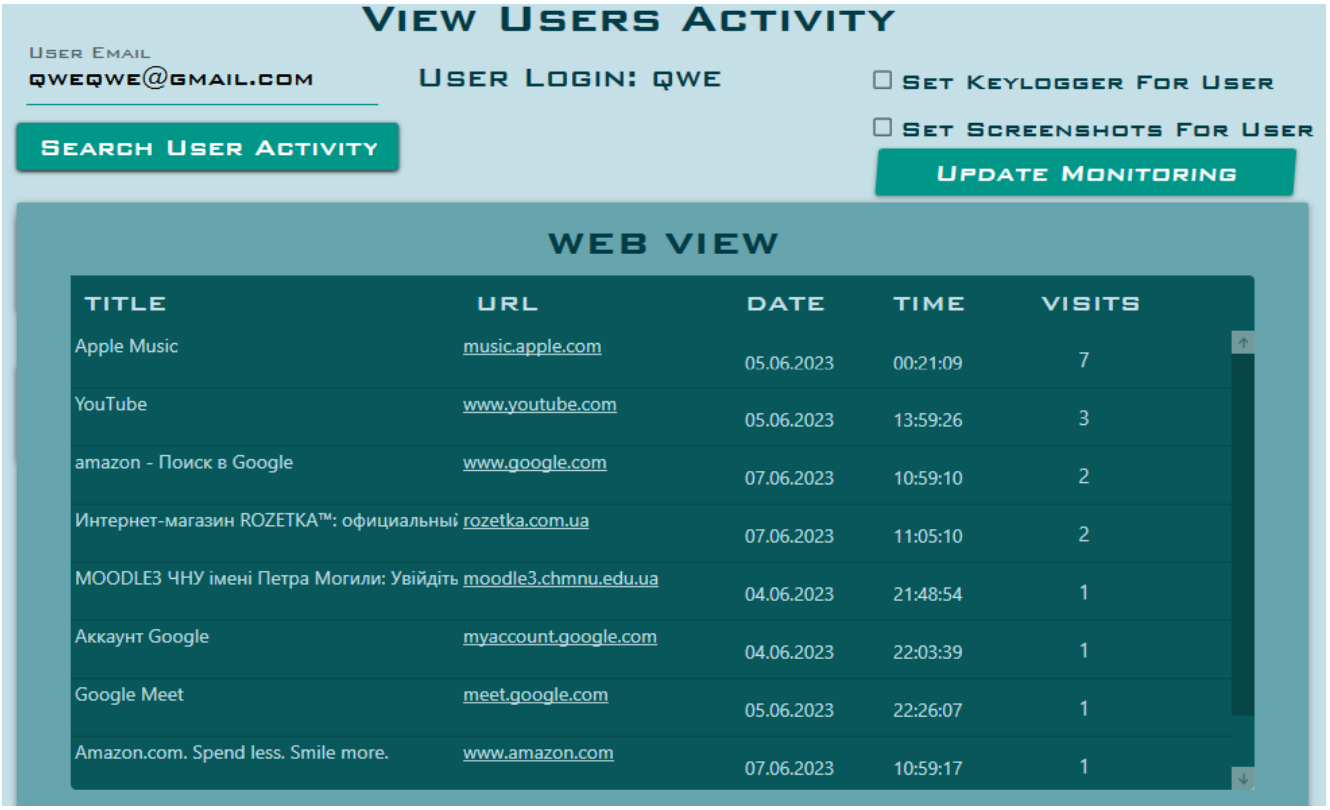


Рисунок 4.11 – Перегляд вебактивності обраного користувача

При більш розширеному функціоналі може знадобитися не тільки перегляд основних списків програм, але й додатковий функціонал, що відображає аналіз та поведінку користувача, описаний у пунктах 4.2 та 4.3 даного розділу. Додати його

до вже створеної панелі задач є простою задачею, що не потребує написання додаткового C# та xaml-коду. Але при демонстрації можливостей адміністративної діяльності та моніторингу, основних панелей активності може бути достатньо списків вебресурсів та активних вікон. Відповідно, можемо опустити реалізацію діаграмних панелей, та умовно відзначити їх корисність при аналізі користувацької діяльності адміністратором.

4.5 Панель скріншотів та кейлоггеру

Відповідно до розкритого у попередньому підрозділі представлення адміністративної панелі, відмічено присутність інших двох панелей під назвою «Screen View» та «Key View». За активацію цих панелей відповідає адміністратор. На рис. 4.10 є додаткові прапорці та кнопка, яка дозволяє адміністратору встановлювати функціонал для обраного користувача.

Що це означає для користувача? Якщо адміністратор встановив прапорець у обраній комірці, під час роботи з системою застосунок автоматично буде оброблювати булеві дані з БД, які і вказують чи проводити відносно користувача додаткові дії. Встановлення прапорця запустить код у класі *Service*, коли користувач знову оновить систему. За замовчуванням та щоб не навантажувати систему було налаштовано періодичність, з якою система буде робити скріншоти екрану – один скріншот на пів-години часу.

Тож функція *Service.Start()* запускає на виконання код класу *ScreenMonitor* (додаток І), який створює об'єкт *Bitmap* із розмірами робочого столу. Для збереження у БД MongoDB необхідно перетворити це зображення на масив байтів. Надалі створений об'єкт класу *Screen* зберігається у базі даних у вигляді байтів (див. рис. 4.12).

Рисунок 4.12 – Запис скріншоту у вигляді байтів

Функціонал кейлоггеру схожим чином запускається за допомогою *Service.Start()*. Сам процес читання відбувається у класі *Keylog* та *KeyboardHook* (додаток К). Клас обробляє подію, яка спрацьовує при натисканні клавіши, та записує дані. Записані впродовж процесу клавіши зберігаються до БД та можуть бути зчитані у панелі Key View.

Розглянемо створений у БД запис з клавіатури користувача (див. рис. 4.13).

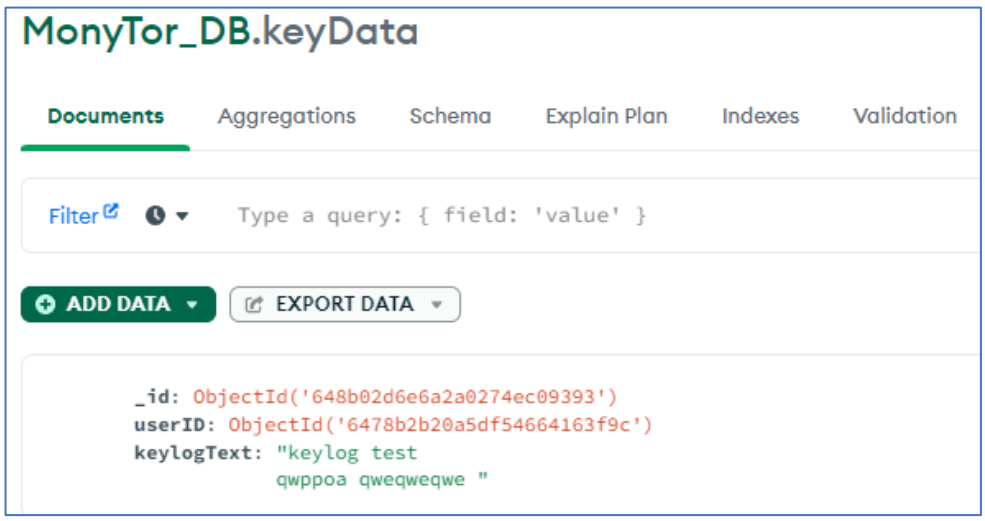


Рисунок 4.13 – Запис з клавіатури у БД з індексом користувача

На прикладі панелі для перегляду скріншотів під час активності користувача відображено результат роботи системи зчитування рисунку, що працює в зворотному порядку – з байтів у зображення (див. рис 4.14).

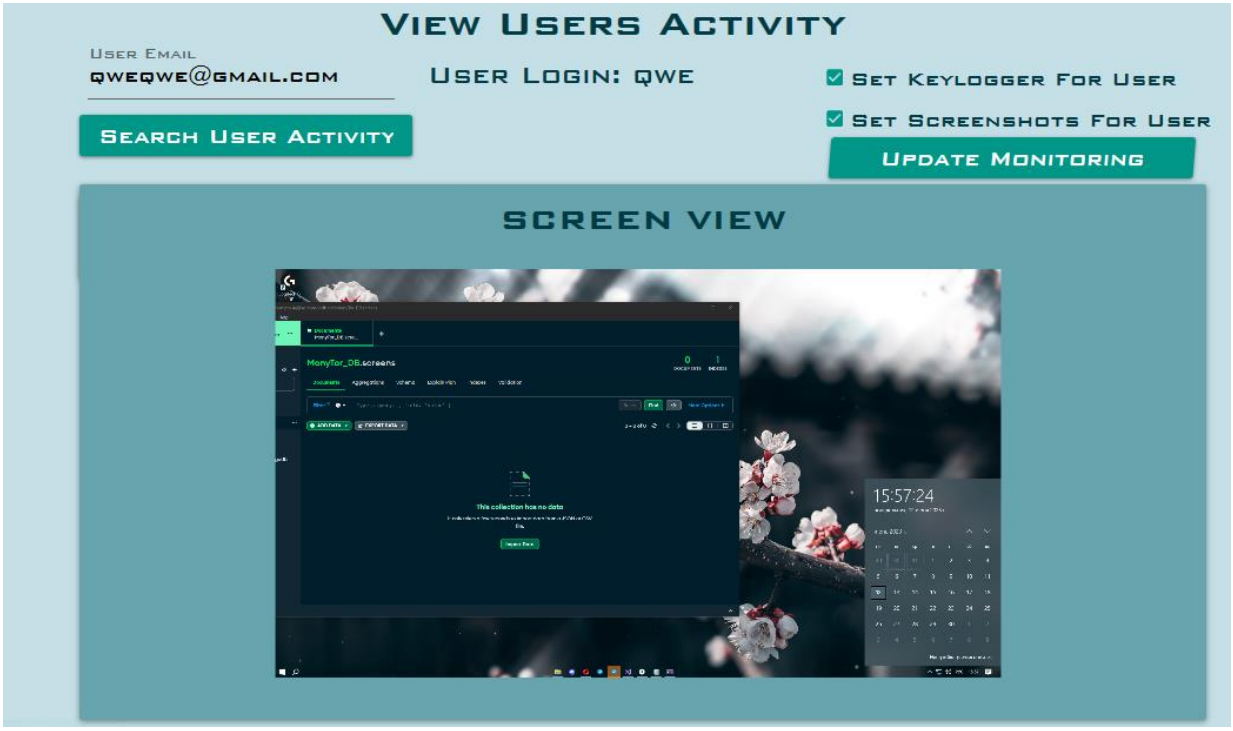


Рисунок 4.14 – Панель виведення скріншоту для адміністратора

Висновки до розділу 4

У даному розділі було описано розробку і програмну реалізацію системи моніторингу та аналізу діяльності користувача за комп'ютером. Розроблена система дозволяє прослуховувати клавіатуру, робити знімки екрану з певною періодичністю, відстежувати відкриті сторінки вебресурсів та вікон застосунків, генерувати звіти про продуктивність, забезпечувати дистанційний моніторинг та управління, формувати звіти з результатами проведеного моніторингу.

Однією з головних функцій системи є моніторинг та аналіз вебдіяльності користувача. Було реалізовано збір і аналіз історії перегляду вебсторінок, збереження інформації про відвідані URL-адреси, назви сторінок, час останнього

відвідування тощо. Це дозволяє встановити активність та інтереси користувача в онлайн-середовищі. Також була реалізована функція моніторингу та аналізу діяльності активних вікон. Система відслідковує активне вікно, зчитує його назву та час перебування в активному стані. Це дозволяє визначити, які програми і вікна використовує користувач найбільше часу і як вони взаємодіють між собою.

Для забезпечення зручного керування та адміністрування системи було розроблено панель адміністратора, яка надає можливість перегляду та аналізу зібраних даних, управління користувачами і доступом до функціональності системи. Додатковий функціонал системи може включати налаштування параметрів моніторингу, створення звітів, інтеграцію з іншими системами тощо.

На основі аналізу отриманої інформації є можливість ефективно планувати й налагоджувати роботу індивідуальних користувачів та співробітників компаній.

ВИСНОВКИ

У сучасних умовах інформатизації суспільства та поширенням віддаленого методу роботи, важливим є розвиток систем моніторингу діяльності користувача ПК, оскільки це забезпечує збільшення продуктивності роботи, та задовольняє потреби в аналізі користувацької активності. Проведена робота дозволяє зробити наступні висновки.

У зв’язку з широким поширенням комп’ютерних технологій у всіх сферах життєдіяльності зростає потреба в системах, які дозволяють контролювати та аналізувати діяльність користувачів ПК. Ефективність роботи як окремого користувача так і співробітника компанії можна поліпшити, аналізуючи їх діяльність за комп’ютером та надаючи зведену інформацію стосовно дій та виконуваної роботи. Було виявлено, що на сьогодні існує багато різних підходів і технологій для моніторингу користувацької активності. Це здебільш корпоративні комерційні розробки, або системи моніторингу на мобільних платформах. Тому є потреба у створенні спеціалізованої системи, орієнтованої на моніторинг та аналіз діяльності користувача ПК у ОС Windows.

Для розробки системи моніторингу та аналізу діяльності користувачів за комп’ютером обґрунтовано використання середовища розробки Visual Studio 2022, яке є ефективним та потужним інструментом для розробки застосунків, підтримує обрані для розробки мови програмування, має зручний інтерфейс. У якості мови програмування обрано C# і платформу .NET, які входять до складу стеку технологій Microsoft, відомого своєю надійністю і великою кількістю можливостей. Для створення графічного інтерфейсу користувача було використано систему побудови клієнтських застосунків Windows Presentation Foundation, який дозволяє створювати інтерфейс в декларативному стилі з використанням для опису інтерфейсу мови розмітки XAML, що спрощує процес розробки і підтримки коду.

Для роботи з базами даних використано документоорієнтовану СУБД MongoDB – представника NoSQL баз даних. Проєктування архітектури застосунку

та розділення логіки і інтерфейсу здійснювалося з використанням шаблону MVVM: Model–View–ViewModel.

Було здійснено проєктування, розробку та програмну реалізацію системи моніторингу та аналізу діяльності користувача ПК. Розроблена система дозволяє прослуховувати клавіатуру, робити знімки екрану з певною періодичністю, відстежувати відкриті сторінки вебресурсів та вікон застосунків, генерувати звіти про продуктивність, забезпечувати дистанційний моніторинг та управління, формувати звіти з результатами проведеного моніторингу.

Однією з головних функцій системи є моніторинг та аналіз вебдіяльності користувача. Було реалізовано збір і аналіз історії перегляду вебсторінок, збереження інформації про відвідані URL-адреси, назви сторінок, час останнього відвідування тощо. Це дозволяє встановити активність та інтереси користувача в онлайн-середовищі. Також була реалізована функція моніторингу та аналізу діяльності активних вікон. Система відслідковує активне вікно, зчитує його назву та час перебування в активному стані. Це дозволяє визначити, які програми і вікна використовує користувач найбільше часу і як вони взаємодіють між собою.

Для забезпечення зручного керування та адміністрування системи було розроблено панель адміністратора, яка надає можливість перегляду та аналізу зібраних даних, управління користувачами і доступом до функціональності системи. Додатковий функціонал системи може включати налаштування параметрів моніторингу, створення звітів, інтеграцію з іншими системами тощо.

На основі аналізу отриманої інформації є можливість ефективно планувати й налагоджувати роботу індивідуальних користувачів та співробітників компаній.

Поставлені завдання виконано, однак функціонал створеної системи моніторингу та аналізу діяльності користувача ПК може бути розширено шляхом доданням нових методів для автоматичного виявлення аномальної поведінки, розробки моделей прогнозу та рекомендації, інтеграції з іншими системами, такими як календар, інструментами сповіщень та керування завданнями.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Work From Home During the COVID-19 Outbreak / Galanti T., Guidetti G. and other. *Journal of Occupational and Environmental Medicine*. 2021. Vol. 63(7). pp. 426-432. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8247534 (дата звернення 04.05.2023).

2. Найкраще програмне забезпечення для моніторингу співробітників та все, що вам потрібно знати про моніторинг співробітників: вебсайт. URL: https://clevercontrol.com/uk/best-employee-monitoring-software/#title2 (дата звернення 07.05.2023).

3. Віддалений контроль співробітників. Методи моніторингу працівників: вебсайт: URL: https://westelecom.ua/blog/udalennyj-kontrol-sotrudnikov (дата звернення 07.05.2023).

4. Про інформацію: Закон України від 02.10.1992 № 2657-XII. URL: https://zakon.rada.gov.ua/laws/show/2657-12#Text (дата звернення 08.05.2023).

5. Top 10 Employee Monitoring Software: вебсайт. URL: https://www.insightful.io/blog/top-10-employee-monitoring-software (дата звернення 09.05.2023).

6. Windows API index: вебсайт. URL: https://learn.microsoft.com/en-us/windows/win32/apiindex/windows-api-list (дата звернення 15.05.2023).

7. Programming reference for the Win32 API: вебсайт. URL: https://learn.microsoft.com/en-us/windows/win32/api/ (дата звернення 15.05.2023).

8. Abdallah W.K., Asem A.S., Senousy M.B. User Intent Discovery using Analysis of Browsing History. *International Journal of Advanced Computer Science and Applications(IJACSA)*. 2016. Vol. 7(10). URL: http://dx.doi.org/10.14569/IJACSA.2016.071015 (дата звернення 16.04.2023).

9. System.Windows.Forms Namespace: вебсайт. URL: https://learn.microsoft.com/en/dotnet/api/system.windows.forms?view=windowsdesktop-7.0 (дата звернення 17.05.2023).

10. Mark J. Price C# 10 and .NET 6 – Modern Cross-Platform Development, Sixth Edition. Packt Publishing, 2022. 826 p.

11. Khang A. Professional WPF and C# Programming. Independently Published, 2019. 405 p.

12. .NET documentation: вебсайт. URL: https://learn.microsoft.com/en-us/dotnet/ (дата звернення 19.05.2023).

13. Windows Presentation Foundation documentation : вебсайт. URL: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/?view=netdesktop-7.0 (дата звернення 19.05.2023).

14. XAML overview (WPF .NET) : вебсайт. URL: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/xaml/?view=netdesktop-7.0 (дата звернення 19.05.2023).

15. MongoDB Documentation: вебсайт. URL: https://www.mongodb.com/docs (дата звернення 20.05.2023).

16. Bradshaw S., Brazil E., Chodorow K. MongoDB: The Definitive Guide: Powerful and Scalable Data Storage. 3rd Edition. 2020. Publisher(s): O'Reilly Media, Inc. 511 p.

17. Weil A. Learn WPF MVVM - XAML, C# and the MVVM pattern. 2016. 174 p.

18. The Model-View-ViewModel Pattern: вебсайт. URL: https://learn.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm (дата звернення 22.05.2023).

19. Anderson C. The Model-View-ViewModel (MVVM) Design Pattern. In: Pro Business Applications with Silverlight 5. Apress, Berkeley, CA. 2012. pp. 461-469/ URL: https://doi.org/10.1007/978-1-4302-3501-9_13 (дата звернення 22.05.2023).

20. Preez O.J. Visual Studio 2022 In-Depth: Explore the Fantastic Features of Visual Studio 2022 - 2nd Edition Paperback. 2022. BPB Publication. 260 p.

21. A useful set of Windows API functions: вебсайт. URL: https://stevedonovan.github.io/winapi/api.html (дата звернення 24.05.2023).

22. Кіндій В.А., Парамуд Я.С. Алгоритмічно-програмні засоби моніторингу часу роботи працівника за комп'ютером. *Комп'ютерні системи та мережі*. 2022. Вип. 4, №1. С. 59-66.

23. Як відстежувати використання браузера Chrome на комп'ютерах з Windows: вебсайт. URL: https://support.google.com/chrome/a/answer/7652902?hl=ru (дата звернення 26.05.2023).

24. Карпенко Н.В. Проектування інтерфейсу користувача: навчально-методичний посібник. Дніпро : ЛІРА, 2018. 80 с.

25. MongoDB C# Driver: вебсайт. URL: https://www.mongodb.com/docs/drivers/csharp/current/ (дата звернення 27.05.2023).

26. Material Design In XAML Documentation: вебсайт. URL: http://materialdesigninxaml.net (дата звернення 27.05.2023).

27. File Times: вебсайт. URL: https://learn.microsoft.com/en-us/windows/win32/sysinfo/file-times (дата звернення 28.05.2023).

28. Переадресація та Google Пошук: вебсайт. URL: https://developers.google.com/search/docs/crawling-indexing/301-redirects?hl=ru (дата звернення 28.05.2023).

ДОДАТОК А

Реалізація підключення до БД

// Лістинг коду класу MongoDbContext

```
public MongoDbContext(string connectionString, string mongoDatabase)
        {
            try
            {
                IMongoClient client = new MongoClient(connectionString);
                _database = client.GetDatabase(mongoDatabase);
            }
            catch (Exception ex)
            {
                Console.WriteLine("DataBase error connection: "+
ex.Message);
            }
}
```

// Лістинг коду інтерфейсу IDataService

```
public interface IDataService<T>
    {
        Task<T> Get(ObjectId id);
        Task<T> Create(T entity);
        Task<T> Update(T entity);
        Task<bool> Delete(ObjectId id);
     }
```

Лістинг коду інтерфейсу IWebDataService:

```
public interface IWebDataService : IDataService<Web>
    {
        Task<ICollection<Web>> GetUserWebData(ObjectId userId);
    }
```

// Лістинг коду класу WebDataService

```
public class WebDataService : IWebDataService
    {
        private readonly MongoDbContext _mongoDbContext;

        public WebDataService(MongoDbContext mongoDbContext)
        {
            _mongoDbContext = mongoDbContext;
        }
        public async Task<Web> Get(ObjectId id)
        {
            return await _mongoDbContext.WebData.Find(u => u.Id ==
id).FirstAsync();
        }

        public async Task<ICollection<Web>> GetUserWebData(ObjectId userId)
        {
            return await _mongoDbContext.WebData.Find(a => a.UserId ==
userId).ToListAsync();
        }

        public async Task<Web> Create(Web entity)
        {
            await _mongoDbContext.WebData.InsertOneAsync(entity);
            return entity;
        }

        public async Task<Web> Update(Web entity)
        {
            await _mongoDbContext.WebData.ReplaceOneAsync(a => a.Id ==
entity.Id, entity);
            return entity;
        }
        public async Task<bool> Delete(ObjectId id)
        {
            await _mongoDbContext.WebData.DeleteOneAsync(a => a.Id == id);
            return true;        }  }
```

ДОДАТОК Б

Зчитування даних про вебактивність

// Лістинг коду класу ChromeHistoryData

```
public class ChromeHistoryData
{
    public string Url { get; set; }
    public string Title { get; set; }
    public int VisitCount { get; set; }
    public DateTime? LastVisitTime { get; set; }

    public async Task<List<ChromeHistoryData>> GetChromeHistoryDataAsync()
    {
        var historyDataList = new List<ChromeHistoryData>();

        string userProfilePath =
Environment.GetEnvironmentVariable("USERPROFILE");
    string historyPath = @"\AppData\Local\Google\Chrome\User
Data\Default\History";
string fullHistoryPath = userProfilePath + historyPath;        string
copyHistoryFile = @"E:\Диплом\MonyTor\HistoryCopy";

        Directory.CreateDirectory(Path.GetDirectoryName(copyHistoryFile));

        File.Copy(originalHistoryFile, copyHistoryFile, true);

        using (var connection = new SQLiteConnection($"Data
Source={copyHistoryFile};Version=3;"))
        {
            await connection.OpenAsync();

            string sql = "SELECT url, title, visit_count, last_visit_time
FROM urls";
            using var command = new SQLiteCommand(sql, connection);

            using var reader = await command.ExecuteReaderAsync();
            try
            {
                while (await reader.ReadAsync())
                {
                    var data = new ChromeHistoryData
                    {
                        Url = new Uri(reader.GetString(0)).Host,
                        Title = reader.GetString(1),
                        VisitCount = reader.GetInt32(2),
```

```
                        LastVisitTime =
ConvertUnixToDateTime(reader.GetInt64(3))
                    };
                    historyDataList.Add(data);
                }
            }
            catch (Exception)
            {
                await Console.Out.WriteLineAsync("File reading Exp");
            }

        }
        return historyDataList;
    }
    private DateTime? ConvertUnixToDateTime(long unixTimeMicroseconds)
    {

        if (unixTimeMicroseconds == 0)
            return null;

        return new DateTime(1601, 1, 1, 0, 0, 0,
DateTimeKind.Utc).Add(TimeSpan.FromMilliseconds(unixTimeMicroseconds /
1000d)).AddHours(3);
    }
}
```

ДОДАТОК В

Аналіз даних про вебактивність та запис до БД

// Лістинг коду функції Service.AddOrUpdateWebActivity():

```
public async Task AddOrUpdateWebActivity(Web webActivity)
        {
            var allUserWebActivities = await
_webDataService.GetUserWebData(webActivity.UserId);

            var existingWebActivityByUrl =
allUserWebActivities.FirstOrDefault(a =>
a.WebUrl.Equals(webActivity.WebUrl));
            var existingWebActivityByTitle =
allUserWebActivities.FirstOrDefault(a =>
a.WebTitle.Equals(webActivity.WebTitle));

            if (existingWebActivityByUrl != null &&
existingWebActivityByTitle != null &&
existingWebActivityByUrl.WebTitle.Equals(webActivity.WebTitle))
            {
                await _webDataService.Update(webActivity);        }
            else            {
                if ((existingWebActivityByUrl != null &&
!existingWebActivityByUrl.WebTitle.Equals(webActivity.WebTitle)) ||
                    (existingWebActivityByTitle != null &&
!existingWebActivityByTitle.WebUrl.Equals(webActivity.WebUrl)))
                {
                    if (existingWebActivityByTitle != null)
                    {
                        existingWebActivityByTitle.LastTimeVisit =
webActivity.LastTimeVisit;
                        await
_webDataService.Update(existingWebActivityByTitle);
                    }                }
                else                {
               await _webDataService.Create(webActivity); }}}
```

ДОДАТОК Г

Взаємодія представлення та моделі вебактивності

// Лістинг коду класу WebAnalysisViewModel:

```
public class WebAnalysisViewModel : ViewModelBase
    {
        private readonly WebRatingViewModel _timeOfUseRatingViewModel;
        public ObservableCollection<WebRatingItemViewModel> WebRatingItems
{ get; private set; }
        public WebHoursActivityViewModel WebHoursActivityViewModel { get;}

        public WebAnalysisViewModel(IAccountStore accountStore)
        {
            ICollection<Web> activities =
accountStore.CurrentAccount.WebActivities;
            _timeOfUseRatingViewModel = new WebRatingViewModel(activities);
            WebRatingItems = _timeOfUseRatingViewModel.WebActivity;
            WebHoursActivityViewModel = new
WebHoursActivityViewModel(activities);  }}
```

// Лістинг коду класу WebRatingViewModel:

```
public class WebRatingViewModel : ViewModelBase
    {

        public ObservableCollection<WebRatingItemViewModel> WebActivity {
get; }
        public WebRatingViewModel(ICollection<Web> webActivities)
        {
            WebActivity = GetProgramViewModels(webActivities);
        }
          private ObservableCollection<WebRatingItemViewModel>
GetProgramViewModels(ICollection<Web> webActivities)
        {
            ObservableCollection<WebRatingItemViewModel> resultCollection =
new ObservableCollection<WebRatingItemViewModel>();

            if (webActivities.Count() != 0)
                foreach (string webTitle in webActivities.Select(a =>
a.WebTitle).Distinct())
                {
```

```
                    var webActivity = webActivities.FirstOrDefault(a =>
a.WebTitle == webTitle);

                    DateTime? lastVisitDate = null;
                    string lastVisitTimeString = null;

                    if (webActivity.LastTimeVisit.HasValue)
                    {
                        lastVisitDate =
webActivity.LastTimeVisit.Value.Date;
                        lastVisitTimeString =
webActivity.LastTimeVisit.Value.TimeOfDay.ToString("hh\\:mm\\:ss");
}

                    resultCollection.Add(
                        new WebRatingItemViewModel
                        {
                            WebTitle = webTitle.Length > 50 ?
webActivity.WebTitle.Substring(0, 50) + "..." : webActivity.WebTitle,
                            WebUrl = webActivity.WebUrl,
                            VisitCount = webActivity.VisitCount,
                            LastVisitDate = lastVisitDate?.ToString("d"),
                            LastVisitTime = lastVisitTimeString,
                        }
 );              }
                resultCollection.Sort((p1, p2) =>
p2.VisitCount.CompareTo(p1.VisitCount));
            return resultCollection;
        }
 }
```

// Лістинг коду класу WebHoursActivityViewModel:

```
public class WebHoursActivityViewModel : ViewModelBase
        {
        public SeriesCollection HoursActivity { get; }
        public Func<double, string> Formatter { get; set; } = value =>
value.ToString();

        public WebHoursActivityViewModel(ICollection<Web> activities)
        {            HoursActivity =
GetHoursActivitySeriesCollection(activities);
        }
        private SeriesCollection
GetHoursActivitySeriesCollection(ICollection<Web> activities)
            {
            return new SeriesCollection {
```

```
            new ColumnSeries
            {
                Values = new
ChartValues<double>(GetHoursActivity(activities)),
                MaxColumnWidth = 24,
                LabelPoint = value => value.Instance.ToString(),
                Title = "Web-Site Visit Count"
            }            };        }
        private ICollection<double> GetHoursActivity(ICollection<Web>
activities)
        {
            Dictionary<int, double> resultTimes = new Dictionary<int,
double>();
            for (int i = 0; i < 24; i++)
            {
                resultTimes[i] = 0;            }
            foreach (var webActivity in activities)
            {
                if (webActivity.LastTimeVisit.HasValue)
                {
                var hour = webActivity.LastTimeVisit.Value.Hour;
                    resultTimes[hour] += webActivity.VisitCount;
                }            }
            return resultTimes.Values;        }     }
```

ДОДАТОК Д

Відстеження активних вікон

// Лістинг коду функції Service.GetActiveWindowTitle():

```
private string GetActiveWindowTitle()
        {
            GetWindowThreadProcessId(GetForegroundWindow(), out int
windowId);

            if (Array.Exists(_blockedSystemProcesses, element => element ==
windowId))
            {
                return null;
            }
            Process currentProcess = Process.GetProcessById(windowId);

            return
currentProcess.MainModule?.FileVersionInfo.FileDescription;
        }
```

ДОДАТОК Е

Запис та оновлення даних про активні вікна

// Лістинг коду функції Service.AddOrUpdateActivity ():

```
private async Task AddOrUpdateActivity(Activity activity)
        {
            Activity activityContext =
_loggedAccount.Activities.FirstOrDefault(a =>
                a.ProgramTitle.Equals(activity.ProgramTitle) &&
a.DayOfUse.Equals(activity.DayOfUse));
            if (activityContext == null)
            {
                _loggedAccount.Activities.Add(activity);
                await _activityService.Create(activity);
            }..
            else
            {
                activityContext.TimeSpent += activity.TimeSpent;
                await _activityService.Update(activityContext);
            }
        }
```

ДОДАТОК Ж

Взаємодія представлення та моделі десктоп активності

Лістинг коду класу DesctopAnalysisViewModel:

```
public class DesctopAnalysisViewModel : ViewModelBase
    {
        private readonly DesctopRatingViewModel _timeOfUseRatingViewModel;
   public DayActivityViewModel DayActivityViewModel { get; }
   public WeekActivityViewModel WeekActivityViewModel { get; }
   public MonthActivityViewModel MonthActivityViewModel { get; }
   public ObservableCollection<TimeOfUseRatingItemViewModel>
TimeOfUsePrograms => _timeOfUseRatingViewModel.Programs;
   public ObservableCollection<DaysOfUseRatingItemViewModel>
DaysOfUsePrograms => _daysOfUseRatingViewModel.Programs;
   public DiagramWeekActivityViewModel DiagramWeekActivityViewModel { get;
}
   public DiagramMonthActivityViewModel DiagramMonthActivityViewModel {
get; }
   public DesctopAnalysisViewModel(IAccountStore accountStore)
        {
            ICollection<Activity> activities =
accountStore.CurrentAccount.Activities;
            _timeOfUseRatingViewModel = new
DesctopRatingViewModel(activities);
            DayActivityViewModel = new DayActivityViewModel(activities);
            WeekActivityViewModel = new WeekActivityViewModel(activities);
            MonthActivityViewModel = new
MonthActivityViewModel(activities);
            DiagramWeekActivityViewModel = new
DiagramWeekActivityViewModel(activities);
            DiagramMonthActivityViewModel = new
DiagramMonthActivityViewModel(activities);        }    }
```

ДОДАТОК З

Аналіз даних про активні вікна

Лістинг коду класу WeekActivityViewModel:

```
public class WeekActivityViewModel : ViewModelBase
    {
        public string WeekActivity { get; }
        public string WeekTimeActivity { get; }
        public WeekActivityViewModel(ICollection<Activity> activities)
        {
            int currentWeekActivitiesCount =
GetActivityCountForWeek(DateTime.Now, activities);
            int lastWeekActivitiesCount =
GetActivityCountForWeek(DateTime.Now.AddDays(-7), activities);
            TimeSpan currentWeekTimeSpent =
GetTimeSpentForWeek(DateTime.Now, activities);

            double activityChangePercentage =
CalculatePercentageChange(lastWeekActivitiesCount,
currentWeekActivitiesCount);

            WeekActivity = $"{currentWeekActivitiesCount} programs";
            WeekTimeActivity = currentWeekTimeSpent.SecondsToFullString();
        }

        private static int GetActivityCountForWeek(DateTime date,
ICollection<Activity> activities)
        {
            int weekOfYear = ISOWeek.GetWeekOfYear(date);
            return activities
                .Count(a =>
ISOWeek.GetWeekOfYear(DateTime.Parse(a.DayOfUse)) == weekOfYear);
        }
        private static TimeSpan GetTimeSpentForWeek(DateTime date,
ICollection<Activity> activities)
        {
            int weekOfYear = ISOWeek.GetWeekOfYear(date);
            return new TimeSpan(
                activities
                    .Where(a =>
ISOWeek.GetWeekOfYear(DateTime.Parse(a.DayOfUse)) == weekOfYear)
                    .Sum(a => a.TimeSpent.Ticks));
        }
        private static double CalculatePercentageChange(int lastWeekCount,
int thisWeekCount)
```

```
        {
            // To prevent division by zero.
            if (lastWeekCount == 0 && thisWeekCount == 0)
                return 100;
            if (lastWeekCount == 0)
                return 0;
            return ((double)thisWeekCount / lastWeekCount) * 100;        }
}
```

Лістинг коду класу DiagramWeekActivityViewModel:

```
public class DiagramWeekActivityViewModel : ViewModelBase
    {
        public SeriesCollection CurrentWeekActivity { get; }
        public Func<double, string> Formatter { get; set; } = value =>
value.SecondsToAbbreviatedString();
        public DiagramWeekActivityViewModel(ICollection<Activity>
activities)
        {            if (activities == null)
            {
                throw new ArgumentNullException(nameof(activities));
            }
            CurrentWeekActivity =
GetActivitySeriesForCurrentWeek(activities);        }
        private SeriesCollection
GetActivitySeriesForCurrentWeek(ICollection<Activity> activities)
        {            return new SeriesCollection        {
            new ColumnSeries
            {
                Values = new
ChartValues<double>(GetActivityDurationForEachDayOfWeek(activities)),
                LabelPoint = value =>
((double)value.Instance).SecondsToFullString(),
                Title = String.Empty
            }        };        }
        private ICollection<double>
GetActivityDurationForEachDayOfWeek(ICollection<Activity> activities)
        {
            DateTime startOfCurrentWeek = DateTime.Now.GetFirstDayOfWeek();
            DateTime endOfCurrentWeek = startOfCurrentWeek.AddDays(6);
            var activityDurationForEachDayOfWeek =
InitializeActivityDurationForWeek();
            foreach (Activity activity in activities)
            {
                DateTime activityDate = DateTime.Parse(activity.DayOfUse);
                if (activityDate.IsDateInRange(startOfCurrentWeek,
endOfCurrentWeek))
                {                    int index =
((int)activityDate.DayOfWeek == 0) ? 6 : (int)activityDate.DayOfWeek - 1;
```

```
                    activityDurationForEachDayOfWeek[index] +=
TimeSpan.FromTicks(activity.TimeSpent.Ticks).TotalSeconds;
                }           }           return
activityDurationForEachDayOfWeek.Values;        }
        private Dictionary<int, double> InitializeActivityDurationForWeek()
        {
            var activityDurationForEachDayOfWeek = new Dictionary<int,
double>();
            for (int i = 0; i < 7; i++)
            {                activityDurationForEachDayOfWeek.Add(i, 0.0);
}
            return activityDurationForEachDayOfWeek;       }   }
```

ДОДАТОК И

Взаємодія представлення та моделі адміністратора

Лістинг коду класу AdminViewModel:

```
public class AdminViewModel : ViewModelBase, INotifyPropertyChanged
    {
        private WebRatingViewModel _timeOfUsePrograms;
        private DaysOfUseRatingViewModel _daysOfUsePrograms;
        public ObservableCollection<TimeOfUseRatingItemViewModel>
TimeOfUsePrograms => _timeOfUsePrograms?.Programs;
        public ObservableCollection<DaysOfUseRatingItemViewModel>
DaysOfUsePrograms => _daysOfUsePrograms?.Programs;
        public ObservableCollection<WebRatingItemViewModel> WebRatingItems
{ get; private set; }
        public event PropertyChangedEventHandler PropertyChanged;

        private readonly IUserSearchService _userSearchService;
        private readonly IActivityService _activityService;
        private readonly IWebDataService _webDataService;
        private readonly IAccountStore _accountStore;
        private readonly IUserService _userService;
        public ICommand GetUserActivityCommand { get; }
        public ICommand SetUserSpyCommand { get; }

        private bool _isAdmin;
        public string _findEmailUser;
        public string _loginValue;
        private bool _isKeylogger;
        private bool _isScreen;
        private Image _screenshot;
        public string _keylogger;
        public string LoginOfUser
        {
            get { return _loginValue; }
            set
            {      _loginValue = value;
                OnPropertyChanged();           }       }
        public string FindEmailUser
        {           get { return _findEmailUser; }
            set           {
                _findEmailUser = value;
                OnPropertyChanged();           }
     }
        public bool IsAdmin => _isAdmin;
        public bool IsKeylogger       {
```

```
            get => _isKeylogger;
            set            {
                _isKeylogger = value;
                OnPropertyChanged(nameof(IsKeylogger));    }
 }
        public bool IsScreen
        {            get => _isScreen;
            set            {
                _isScreen = value;
                OnPropertyChanged(nameof(IsScreen));
}      }
               public string Keylogger        {
            get => _keylogger;
            set            {
                _keylogger = value;
                OnPropertyChanged(nameof(Keylogger));
     }
        }        public Image Screenshot        {
            get => _screenshot;            set
 {
                _screenshot = value;
                OnPropertyChanged(nameof(Screenshot));
            }        }
        User User { get; set; }
        public AdminViewModel(IAccountStore accountStore,
IUserSearchService userService, IActivityService activityService,
IWebDataService webDataService)
        {        _isAdmin =
accountStore.CurrentAccount.AccountHolder.IsAdmin;
            _userSearchService = userService;
            _activityService = activityService;
            _webDataService = webDataService;
            _accountStore = accountStore;
            GetUserActivityCommand = new AsyncCommand(GetUserActivity);
            SetUserSpyCommand = new AsyncCommand(GetUserUpdate);
        }

        private async Task GetUserActivity()
        {
            User = await _userSearchService.GetUserByEmail(FindEmailUser);
            if (User != null)
            {
                _loginValue = "User Login: " + User.Login;
                var res = new Account
                {
                    AccountHolder = User,
                    Activities = await
_activityService.GetUserActivities(User.Id),
```

```
                    WebActivities = await
_webDataService.GetUserWebData(User.Id),
                };

                BsonDocument screenshotDocument =
_userSearchService.GetUserScreen(new BsonDocument()).FirstOrDefault();
                if (screenshotDocument != null)
                {
                    BsonBinaryData screenshotBinary =
screenshotDocument["screenshot"].AsBsonBinaryData;
                    byte[] screenshotData = screenshotBinary.Bytes;
                    using (MemoryStream memoryStream = new
MemoryStream(screenshotData))
                    {
                        _screenshot = Image.FromStream(memoryStream);
                    }
                }
                var keylogger = await
_userSearchService.GetUserKey(User.Id);
                if (keylogger != null)
                {
                    _keylogger = keylogger.Text;
                }
                ICollection<Activity> activities = res.Activities;
                ICollection<Web> webActivities = res.WebActivities;
                _timeOfUsePrograms = new WebRatingViewModel(activities);
                _daysOfUsePrograms = new
DaysOfUseRatingViewModel(activities);
                WebRatingItems = new
WebRatingViewModel(webActivities).WebActivity;
                OnPropertyChanged(nameof(TimeOfUsePrograms));
                OnPropertyChanged(nameof(DaysOfUsePrograms));
                OnPropertyChanged(nameof(WebRatingItems));
                OnPropertyChanged(nameof(LoginOfUser));
                OnPropertyChanged(nameof(Keylogger));
            }        }
        private async Task GetUserUpdate()
        {
            User = await _userSearchService.GetUserByEmail(FindEmailUser);
            if (User != null)
            {
                User.IsScreen = _isScreen;
                User.IsKeylogger = _isKeylogger;
                await _userSearchService.Update(User);            }       }

        protected new virtual void OnPropertyChanged([CallerMemberName]
string propertyName = null)        {
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));        }    }
```

ДОДАТОК І

Процесс реалізації скріншотів

Лістинг коду класу ScreenMonitor:

```
public class ScreenMonitor
{
    [DllImport("user32.dll")]
    public static extern IntPtr GetDesktopWindow();
    [DllImport("user32.dll")]
    public static extern IntPtr GetWindowDC(IntPtr hWnd);
    [DllImport("user32.dll")]
    public static extern IntPtr ReleaseDC(IntPtr hWnd, IntPtr hDC);
    [DllImport("gdi32.dll")]
    public static extern uint BitBlt(IntPtr hdcDest, int nXDest, int
nYDest, int nWidth, int nHeight, IntPtr hdcSrc, int nXSrc, int nYSrc, uint
dwRop);
    [DllImport("user32.dll", SetLastError = true)]
    static extern bool GetWindowRect(IntPtr hWnd, out Rectangle lpRect);
    IUserSearchService userSearchService;
    public async Task CaptureScreenshot(Account loggedAccount)
    {
        IntPtr handle = GetDesktopWindow();
        IntPtr hdcSrc = GetWindowDC(handle);
        Rectangle windowRect;
        GetWindowRect(handle, out windowRect);
        var screenshot = new Bitmap(windowRect.Width, windowRect.Height,
PixelFormat.Format32bppArgb);
        using (Graphics gfxScreenshot = Graphics.FromImage(screenshot))
        {
            IntPtr hdcDest = gfxScreenshot.GetHdc();
            BitBlt(hdcDest, 0, 0, windowRect.Width, windowRect.Height,
hdcSrc, 0, 0, (uint)CopyPixelOperation.SourceCopy |
(uint)CopyPixelOperation.CaptureBlt);
            gfxScreenshot.ReleaseHdc(hdcDest);
        }
        ReleaseDC(handle, hdcSrc);

        using (MemoryStream memoryStream = new MemoryStream())
        {
            screenshot.Save(memoryStream, ImageFormat.Png);
            byte[] screenshotData = memoryStream.ToArray();
            Screen screen = new Screen(loggedAccount.AccountHolder.Id,
screenshotData);
            await userSearchService.Create(screen);
        }
        screenshot.Dispose();    }
```

ДОДАТОК К

Процес реалізації кейлоггеру

// Лістинг коду класу Keylog

```
public class Keylog
    {
        public string LoggedString { get; set; }
        public void StartLogging()
        {
            KeyboardHook.KeyDown += KeyDownHandler;
            Application.Run();
        }
        private void KeyDownHandler(Keys key)
        {
            string keyName = key.ToString();
            SaveKeyPress(keyName, timestamp);
        }
        private void SaveKeyPress(string key, string timestamp)
        {
            LoggedString += key;
        }
    }
```

// Лістинг коду класу KeyboardHook

```
public static class KeyboardHook : IMessageFilter
    {
        public static event Action<Keys> KeyDown;
        static KeyboardHook()
        {
            Application.AddMessageFilter(PreFilterMessage());
        }
        public bool PreFilterMessage(ref Message m)           {
                const int WM_KEYDOWN = 0x0100;
                const int WM_SYSKEYDOWN = 0x0104;
                if (m.Msg == WM_KEYDOWN || m.Msg == WM_SYSKEYDOWN)
                {
      KeyDown?.Invoke((Keys)m.WParam.ToInt32());              }
                return false;           }     }
```