

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Додаток контролю активності та організації роботи
користувача»**

Виконав студент групи ІПЗ-22-1
ХОМНЮК Віктор Олександрович

Керівник роботи:
САВІЦЬКИЙ Роман Святославович

Рецензент:
ЛОКТИКОВА Тамара Миколаївна

Житомир – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна Вакалюк
«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ХОМНЮК Віктор Олександрович**

Керівник роботи: **САВІЦЬКИЙ Роман Святославович**

Тема роботи: **«Додаток контролю активності та організації роботи користувача»,**

затверджена Наказом закладу вищої освіти від «27» лютого 2026 р., №92/с

Вихідні дані для роботи: **процес моніторингу й обмеження деструктивної активності користувача для підвищення продуктивності на базі Win32 API, паттерну MVVM (WPF/.NET) та СУБД MongoDB для автоматизації тайм-менеджменту, логування інцидентів та аналітики сесій фокусування.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Савіцький Р.С.	01.03.2026р.	01.03.2026р
2	Савіцький Р.С.	01.03.2026р	01.03.2026р
3	Савіцький Р.С.	01.03.2026р	01.03.2026р

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи містить 57 сторінок, 25 рисунків, 13 таблиць, 30 джерел літератури, 2 додатки.

Метою роботи є розробка настільного додатку для операційної системи Windows, що забезпечує контроль продуктивності користувача шляхом блокування відволікаючих процесів та застосування науково обґрунтованих методів часового менеджменту.

У першому розділі проведено аналіз предметної області, досліджено існуючі аналоги, сформульовано задачу та обґрунтовано вибір технологій.

У другому розділі спроектовано архітектуру додатку, описано алгоритми блокування процесів через Win32 API, структуру бази даних та ключові модулі застосунку.

У третьому розділі описано порядок встановлення системи, представлено графічний інтерфейс та результати тестування.

В результаті розроблено програмний продукт, що дозволяє оптимізувати робочий час за допомогою наукових методик фокусування та автоматично блокувати відволікаючі процеси.

КЛЮЧОВІ СЛОВА: БЛОКУВАННЯ ПРОЦЕСІВ, MONGODB, РЕЖИМИ ФОКУСУВАННЯ, ТАЙМ-МЕНЕДЖМЕНТ, WPF.

					ІПЗ.КР.Б-121-26-ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Хомнюк В.О.			Додаток контролю активності та організації роботи користувача	Літ.	Арк.	Аркушів
Керівник		Савіцький Р.С.					3	
Реценз.		Локтікова Т.Л.				Житомирська політехніка, група ІПЗ-22-1		
Н. контр.								
Затверд.								

ABSTRACT

The explanatory note to the thesis consists of 57 pages, 25 figures, 13 tables, 30 references, and 2 appendices.

The aim of the work is to develop a desktop application for the Windows operating system that monitors user productivity by blocking distracting processes and applying scientifically grounded time management methods.

The first chapter analyzes the subject area, examines existing analogues, formulates the problem, and justifies the choice of technologies.

The second chapter designs the application architecture, describes algorithms for blocking processes via the Win32 API, the database structure, and the application's key modules.

The third chapter describes the system installation procedure, presents the graphical user interface, and reports on the test results.

As a result, a software product has been developed that allows users to optimize their working time using scientific focusing techniques and automatically block distracting processes.

KEYWORDS: FOCUS MODES, MONGODB, PROCESS BLOCKING, TIME MANAGEMENT, WPF.

		Хомніук В.О.			ІІЗ.КР.Б - 121 - 26 - ІІЗ	Арк.
		Савіцький Р.С.				
						4

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП	7
РОЗДІЛ 1 АНАЛІЗ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ	9
1.1 Постановка задачі	9
1.2 Аналіз аналогів програмного продукту	11
1.3 Вибір архітектури системи управління спільнотами та подіями	15
1.4 Обґрунтування вибору засобів реалізації та вимоги до апаратного забезпечення	18
Висновки до першого розділу	21
РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ ДЛЯ УПРАВЛІННЯ СПІЛЬНОТАМИ ТА ПОДІЯМИ	23
2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи	23
2.2. Розробка бази даних системи	31
2.3. Проєктування та реалізація алгоритмів роботи системи	35
2.4. Реалізація функціоналу системи організації спільнот та подій	38
Висновки до другого розділу	46
РОЗДІЛ 3. ІНСТРУКЦІЯ КОРИСТУВАЧА ТА РОЗГОРТАННЯ СИСТЕМИ. РЕЗУЛЬТАТИ ТЕСТУВАННЯ	47
3.1. Порядок встановлення та налаштування параметрів платформи	47
3.2. Структура інтерфейсу системи для управління спільнотами та подіями	50
Висновки до третього розділу	59
ВИСНОВКИ	61
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	62
ДОДАТКИ	65

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API (Application Programming Interface) – прикладний інтерфейс для взаємодії між програмами

ASR (Architecturally Significant Requirements) – архітектурно значущі вимоги до системи

JSON (Binary JSON) – бінарний формат збереження даних у СУБД MongoDB

C# – об'єктно-орієнтована мова програмування для платформи .NET

GUI (Graphical User Interface) – графічний інтерфейс користувача

JSON (JavaScript Object Notation) – текстовий формат обміну даними

MongoDB – документоорієнтована NoSQL база даних

MVVM (Model-View-ViewModel) – архітектурний патерн розділення інтерфейсу та логіки

NoSQL (Not Only SQL) – нереляційні системи керування базами даних

OAuth (Open Authorization) – протокол безпечної авторизації через сторонні сервіси

ViewModel – компонент MVVM, що пов'язує дані (Model) з інтерфейсом (View)

WPF (Windows Presentation Foundation) – графічна підсистема .NET для десктопних додатків

XAML (Extensible Application Markup Language) – мова розмітки інтерфейсу у WPF

		Хомніук В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				
						6

ВСТУП

Актуальність теми в умовах сучасного інформаційного суспільства проблема збереження концентрації уваги під час роботи за комп'ютером набуває дедалі більшої актуальності. Постійні сповіщення, месенджери, соціальні мережі та розважальний контент відволікають користувачів і суттєво знижують продуктивність праці. За даними досліджень, середній офісний працівник перемикається між завданнями до 50 разів на день, а відновлення концентрації після кожного відволікання займає від 15 до 23 хвилин.

Незважаючи на наявність різноманітних застосунків для управління часом, більшість із них зосереджені лише на таймерах та нагадуваннях, не вирішуючи ключову проблему запобігання доступу до відволікаючих ресурсів та програм безпосередньо на рівні операційної системи. Цей розрив між потребою у примусовому обмеженні та наявними інструментами обумовлює необхідність розробки комплексного рішення.

Метою кваліфікаційної роботи є проектування та розробка настільного додатку для ОС Windows, що поєднує активне блокування небажаних системних процесів із науково обґрунтованими техніками управління часом, а також ведення статистики ефективності роботи користувача.

Для досягнення мети визначено такі **завдання**:

1. Провести аналіз предметної області та огляд існуючих рішень.
2. Дослідити методи фокусування, такі як Pomodoro, FlowTime, 52/17, ультрадіанний ритм.
3. Спроекувати архітектуру системи та структуру бази даних.
4. Реалізувати функціонал блокування процесів на рівні ОС Windows.
5. Розробити профіль користувача зі статистикою використання методів.
6. Провести тестування розробленого програмного забезпечення.

Об'єктом дослідження є процес організації продуктивної роботи користувача персонального комп'ютера. Предметом дослідження методи та

		Хомнюк В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				
						7

засоби програмної реалізації систем контролю активності на платформі Windows із застосуванням C# WPF та MongoDB.

Предметом дослідження є процеси, методи та програмні технології автоматизації тайм-менеджменту, що поєднують низькорівневе блокування системних дистракцій за допомогою Win32 API, адаптивні режими фокусування та хмарний аналіз метрик продуктивності користувача.

У процесі виконання роботи застосовано методи об'єктно-орієнтованого проєктування, моделювання структур локальних баз даних, розробки настільних застосунків та проєктування графічних інтерфейсів користувача.

Практичне значення. Розроблений застосунок контролю активності та організації роботи користувача може бути впроваджений у повсякденну практику студентів, фахівців у галузі програмної інженерії та будь-яких користувачів персональних комп'ютерів під управлінням Windows, які прагнуть підвищити особисту продуктивність. Застосунок дозволяє структурувати робочий час за науково обґрунтованими методиками, автоматично усувати цифрові відволікання на рівні операційної системи та відстежувати динаміку продуктивності через статистику завершених сесій. Реалізований функціонал блокування процесів та гнучкі режими фокусування роблять застосунок готовим інструментом для щоденного використання без необхідності додаткового налаштування.

		Хомніук В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				8

РОЗДІЛ 1 АНАЛІЗ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

1.1 Постановка задачі

Метою роботи є розробка настільного застосунку для операційної системи Windows, що забезпечує контроль продуктивності користувача шляхом примусового блокування відволікаючих системних процесів та реалізації науково обґрунтованих методів часового менеджменту із використанням технології WPF та локальної бази даних MongoDB.

Проблема зниження продуктивності через цифрові відволікання набула особливої гостроти з поширенням дистанційної форми зайнятості та навчання. Постійний доступ до месенджерів, браузерів і розважального контенту безпосередньо під час роботи за комп'ютером формує стійкі патерни переривання концентрації, що суттєво знижує якість і обсяг виконуваної роботи. Після кожного незапланованого переривання користувачу потрібно в середньому від 15 до 23 хвилин для повного відновлення рівня концентрації, що при багаторазовому повторенні протягом робочого дня призводить до значних втрат продуктивної праці. Особливо гостро ця проблема відчувається у середовищі програмістів, студентів та фахівців розумової праці, для яких стан глибокої концентрації є необхідною умовою якісного виконання завдань. Зважаючи на неможливість повного відмовлення від цифрових інструментів у сучасних умовах роботи, найбільш ефективним рішенням є програмне обмеження доступу до відволікаючих ресурсів — не через рекомендації чи нагадування, а через автоматизоване примусове блокування на рівні операційної системи.

Для вирішення поставленої задачі визначено такі функціональні складові системи.

Функціонал методу Pomodoro має забезпечувати реалізацію класичної схеми чергування 25-хвилинних робочих інтервалів із 5-хвилинними паузами, з автоматичним переходом до подовженої 15–30-хвилинної перерви після кожного четвертого циклу. Таймер повинен відображати поточну фазу та час

		Хомнюк В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				9

що залишився, а лічильник завершених циклів — оновлюватись автоматично після кожної успішної сесії.

Функціонал методу FlowTime має реалізовувати відлік робочого часу у прямому напрямку без фіксованих лімітів завершення. Система повинна динамічно розраховувати рекомендовану тривалість перерви залежно від накопиченого часу фокусування та відображати відповідну підказку користувачу після завершення робочої фази.

Модуль блокування процесів надає користувачу можливість через спеціалізоване вікно налаштувань формувати персональний список заборонених застосунків та ігор. Окреме вікно управління відображає перелік активних обмежень і забезпечує примусове завершення небажаних процесів на рівні операційної системи.

Функціонал методу 52/17 має забезпечувати автоматичне чергування 52-хвилинних робочих блоків із 17-хвилинними паузами відновлення з відображенням поточної фази та часу що залишився до її завершення.

Функціонал ультрадіанного ритму має реалізовувати 90-хвилинні робочі цикли з 20-хвилинними паузами відновлення. Особливістю цього режиму є автоматичне розширення списку блокування — система без додаткових налаштувань має примусово завершувати процеси браузерів та Telegram протягом усього робочого блоку.

Функціонал блокування процесів має надавати можливість формувати персональний список заборонених застосунків через спеціалізоване вікно налаштувань із визначенням дня тижня та часового діапазону дозволеного використання. Фоновий сервіс моніторингу повинен циклічно перевіряти активні процеси операційної системи та примусово завершувати заборонені через Win32 API.

Функціонал профілю та статистики має автоматично фіксувати кількість успішно завершених сесій для кожного з чотирьох методів фокусування та зберігати ці дані у локальній базі даних MongoDB. Сторінка профілю повинна

		Хомніук В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				10

відображати зведену статистику у зручному форматі для аналізу динаміки продуктивності.

1.2 Аналіз аналогів програмного продукту

Cold Turkey Blocker [3] один з найпопулярніших і найжорсткіших десктопних блокерів відволікаючих факторів у світі (рис. 1.1). Додаток відомий своєю безкомпромісністю. Після запуску таймера фокусування користувач не може обійти обмеження, закрити процес через диспетчер завдань або видалити програму до завершення сесії. Завдяки такій архітектурі Cold Turkey допомагає сотням тисяч студентів, розробників та віддалених працівників повністю ізолювати свій робочий простір.

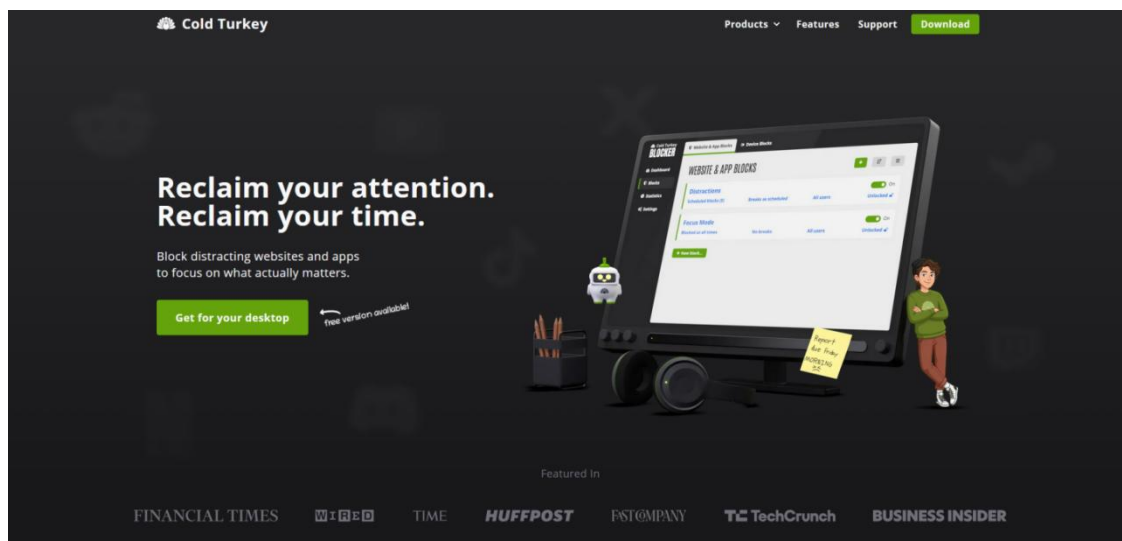


Рис. 1.1. Головна сторінка Cold Turkey Blocker

На офіційному сайті Cold Turkey Blocker є можливість завантажити десктопну версію додатку (рис. 1.2) для керування списками обмежень. У головному робочому просторі програми реалізовано інструментарій для створення та активації блокувань (Website & App Blocks), де користувач може гнучко налаштовувати групи заборонених ресурсів, як-от списки загальних дистракцій або суворий режим фокусування. Також у структурі інтерфейсу передбачено навігаційну панель для швидкого переходу між розділами аналітики, поточних розкладів обмежень та системних налаштувань профілю.

		Хомніук В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				11

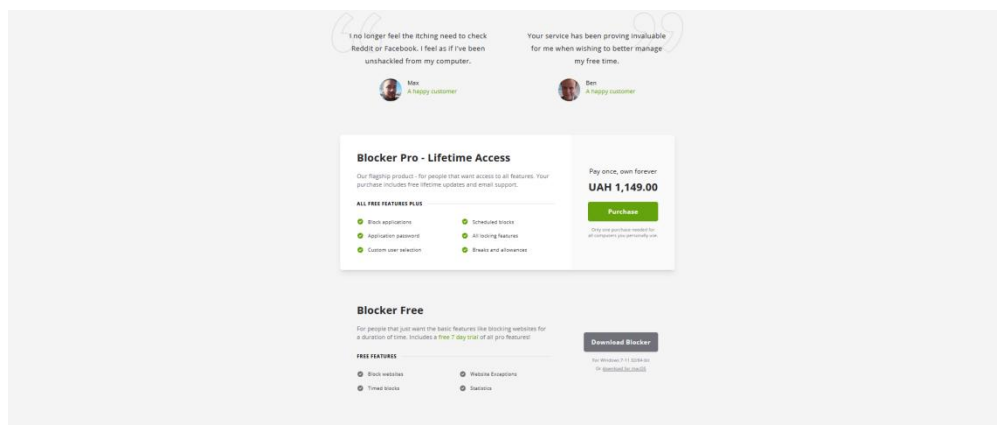


Рис 1.2. Завантаження додатку

У розділі можливостей (рис. 1.3) Cold Turkey Blocker представлено перелік персональних цілей та сценаріїв використання інструменту для боротьби з цифровими дистракціями. Система підтримує гнучке блокування вебресурсів і десктопного програмного забезпечення, що дозволяє користувачам лімітувати час на перегляд контенту, дозволяти доступ лише до конкретних каналів або повністю обмежувати окремі категорії сайтів. Функціонал програми спрямований і на формування корисних звичок у повсякденному житті – контроль часу в комп'ютерних іграх, запобігання безконтрольному гортанню стрічки новин, підвищення продуктивності на роботі та виділення фіксованого часу для навчання, хобі чи своєчасного сну.

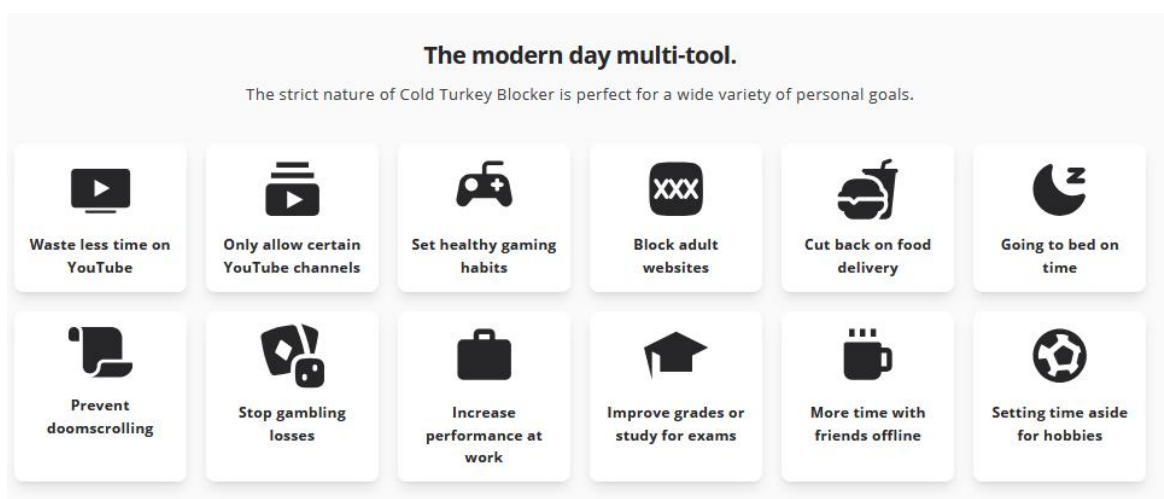


Рис. 1.3. Каталог можливостей

На ринку присутній ще один відомий аналог Freedom (рис. 1.4), кросплатформний сервіс для блокування цифрових дистракцій та підвищення концентрації [3]. Програма дозволяє користувачам створювати індивідуальні

		Хомніук В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				12

сесії фокусування з повним обмеженням доступу до вибраних мобільних додатків, десктопного програмного забезпечення та вебресурсів. Головною особливістю сервісу виступає функція синхронізації, за якої запуск режиму блокування на одному пристрої автоматично активує обмеження на всіх інших підключених гаджетах (ПК, смартфонах та планшетах), що допомагає користувачеві уникати перемикання уваги на альтернативні екрани.

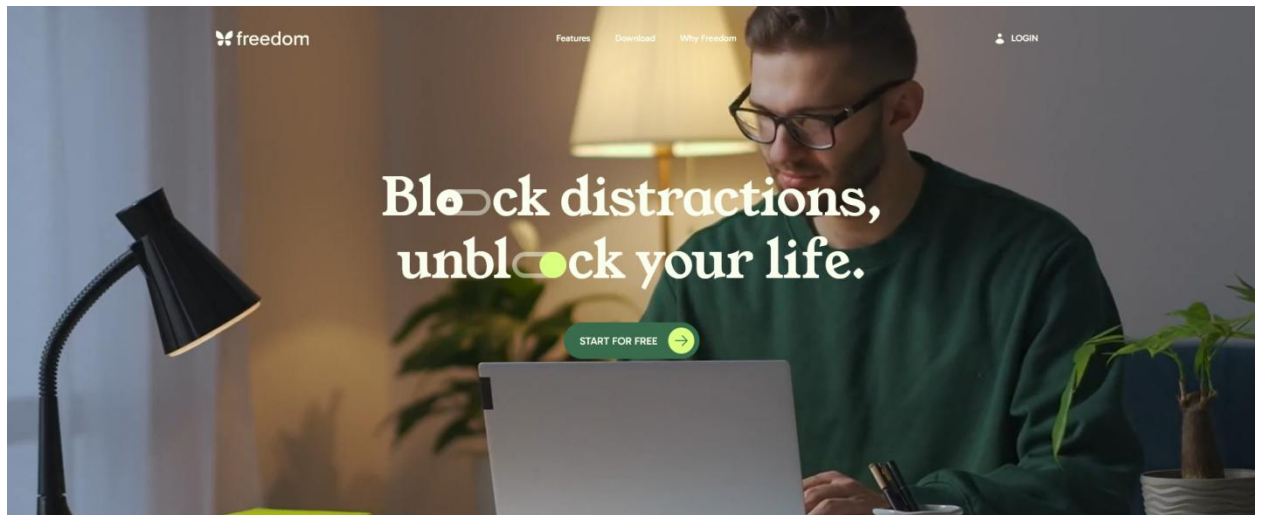


Рис. 1.4. Головне меню Freedom

На офіційному сайті Freedom [6] доступна безкоштовна реєстрація та завантаження кросплатформних клієнтів для управління сесіями фокусування (рис. 1.5). Через головний інтерфейс вебресурсу користувачі отримують безпосередній доступ до інструментарію створення чорних списків додатка з гнучким обмеженням роботи відволікаючого програмного забезпечення та мобільних програм. У структурі сайту передбачено верхню навігаційну панель для швидкого переходу до розділів з детальним описом функціоналу сервісу, посилань на завантаження версій для різних операційних систем, інформаційних матеріалів про методики концентрації та форми авторизації в особистому кабінеті користувача.

		Хомніук В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				13

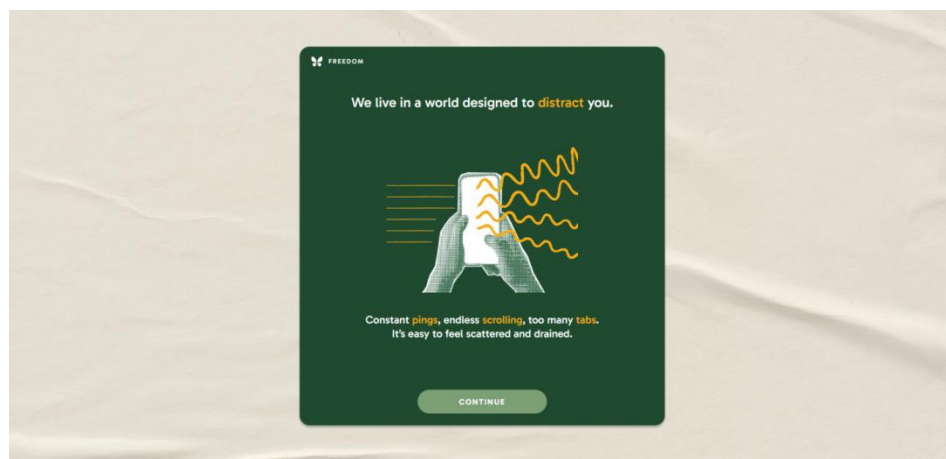


Рис 1.5. Завантаження додатку

У внутрішньому інформаційному блоці сервісу Freedom детально обґрунтовано необхідність використання інструментів для блокування дистракцій та утримання стану потоку під час роботи (рис. 1.6). Платформа надає користувачам можливість інтерактивного ознайомлення з механізмами своєї роботи через функціональні кнопки демонстрації та швидкого старту. Нижній розділ сторінки фокусується на проблематиці втрати концентрації через дрібні цифрові подразники, де реалізовано опис негативного впливу постійних сповіщень, звукових сигналів та регулярних перевірок соціальних мереж на загальну продуктивність і часовий бюджет користувача.

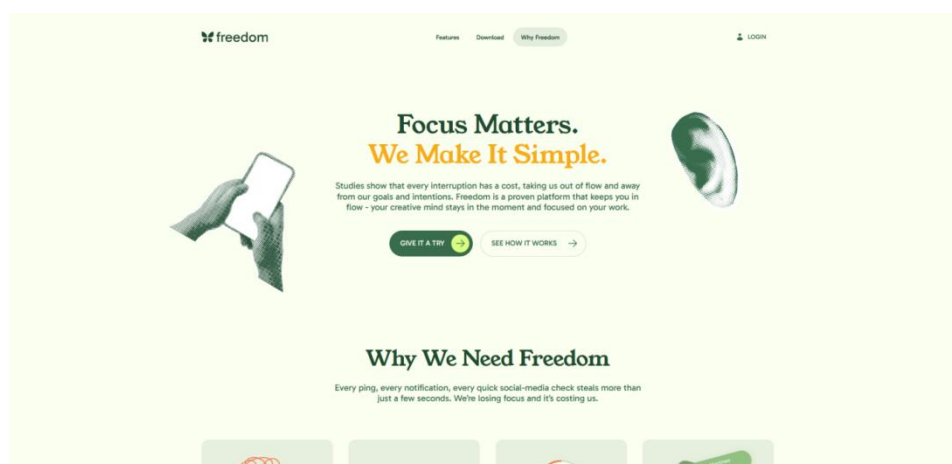


Рис 1.6. Інформація про Freedom

У таблиці представлено порівняльний аналіз функціональних можливостей популярних блокувальників продуктивності та власної розробки автора за ключовими критеріями ефективності (табл. 1.1).

		Хомніук В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				14

Порівняльна таблиця аналогів

Критерій порівняння	Cold Turkey Blocker	Freedom	Моя розробк
Фіксація та трекінг часу	+	+	+
Жорстке закриття процесів (Win32 API)	+	-	+
Автоматичний адаптивний відпочинок (Flowtime)	-	-	+
Інтеграція методик (Pomodoro, 52/17, Ультрадiанні)	-	-	+
Автоматичне розширення списку блокування	-	-	+
Збереження історії та статистики в базі даних	-	+	+(MongoDB)
Локальне десктопне керування без хмари та VPN	+	-	

1.3 Вибір архітектури системи управління спільнотами та подіями

Проектування програмного комплексу для моніторингу користувацької активності та блокування дистракцій вимагає вирішення важливої архітектурної дилеми, забезпечення високої швидкодії локальних модулів контролю при збереженні гнучкості обробки аналітичних даних. Архітектура програмного забезпечення у даному контексті розглядається не просто як набір структурних компонентів, а як комплекс стратегічних рішень щодо розподілу системних ресурсів, які безпосередньо впливають на стабільність операційної системи користувача та трудовитрати на подальший супровід кодової бази.

Для формування оптимального інженерного дизайну системи FocusMind застосовано методологію аналізу архітектурно значущих вимог ASR. Специфіка розроблюваного додатку сформулила три ключові групи вимог.

Функціональні обмеження – необхідність поєднання в єдиному робочому просторі низькорівневого інтерфейсу Win32 API [11] для закриття процесів, локальних таймерів фокусування, текстового модуля нотаток та мережевого чату.

		Хомнюк В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				15

Атрибути якості – критично низьке навантаження на центральний процесор (частота опитування дерева процесів Windows становить 500 мс), ізоляція помилок та абсолютна конфіденційність користувацьких профілів.

Наведена структурна схема ілюструє трирівневу клієнт-серверну архітектуру системи FocusMind, що реалізує бізнес-концепцію масштабованості через чітке розмежування клієнтських пристроїв, модуля мережевого підключення та хмарної бази даних MongoDB Atlas (рис. 1.7).



Рис 1.7. Клієнт-сервера архітектура

Монолітна архітектура [5] – традиційна уніфікована модель побудови програмного забезпечення з об'єднанням всіх компонентів в одному виконуваному файлі (рис. 1.8). У контексті розробки системи контролю продуктивності монолітність означає повну скомпонованість функціоналу (модулів блокування процесів, таймерів, чату та аналітики) в єдиній локальній збірці, що робить кодову базу масивною та взаємозалежною. Така архітектура працює як система з єдиним потоком виконання та спільною пам'яттю, що інкапсулює всю бізнес-логіку взаємодії з Win32 API та базою даних. У разі виникнення потреби внести локальні зміни чи додати новий режим фокусування розробнику необхідно повністю перебудовувати (перекомпілювати) та розгортати всю систему заново. Водночас монолітна модель залишається доволі зручною на ранній стадії життєвого циклу проєкту для швидкого створення функціонального MVP-прототипу додатка.



Рис. 1.8. Монолітна архітектура

Отже, тришарова архітектура найкраще підходить для настільних застосунків з обмеженою бізнес-логікою та чітко визначеним колом завдань. Її доцільно обирати тоді, коли проєкт не містить складних бізнес-правил із великою кількістю взаємозалежностей, має бути розроблений у стислі терміни та не передбачає змін технологічного стеку чи великої кількості зовнішніх інтеграцій (рис. 1.9).



Рис. 1.9. Тришарова архітектура

Зважаючи на перелічені фактори, для додатку контролю активності було обрано тришарову архітектуру як оптимальне рішення, що забезпечує чіткий розподіл відповідальності між шарами, простоту підтримки коду та достатню гнучкість для реалізації всього необхідного функціоналу без зайвого ускладнення структури проєкту.

1.4 Обґрунтування вибору засобів реалізації та вимоги до апаратного забезпечення

Вибір засобів реалізації є важливим етапом при розробці програмного забезпечення. Від цього залежатиме якість розробленого додатку, його продуктивність і швидкість реалізації запланованого функціоналу. Неправильно обраний технологічний стек може призвести до суттєвих архітектурних проблем на пізніх етапах розробки, тому аналіз доступних інструментів та обґрунтування кінцевого вибору є невід'ємною частиною проєктування.

Оскільки система є настільним додатком для операційної системи Windows, усі технічні рішення обирались з урахуванням глибокої інтеграції з платформою та забезпечення стабільної роботи в локальному середовищі без залежності від мережі. Ключовими критеріями відбору технологій були: доступ до системного API операційної системи, зрілість інструментарію розробки, продуктивність виконання та можливість локального зберігання даних без хмарних залежностей.

Проєктування логіки додатків і системних компонентів реалізовано на базі платформи C# [10]. Дана мова програмування вирізняється високим рівнем типізації та оптимізацією під екосистему Windows, що гарантує максимальну продуктивність під час виконання низькорівневих операцій. Завдяки рідній підтримці парадигми ООП, інструментарій C# дозволяє створювати гнучкі ієрархії об'єктів та ізолювати критично важливі ділянки коду. Це забезпечує надійну основу для побудови модульних десктопних рішень із тривалим життєвим циклом та спрощує подальше розширення архітектури програми.

		Хомніук В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				18

Для побудови графічного інтерфейсу користувача обрано технологію WPF [21]. яка є підсистемою створення настільних інтерфейсів, що входить до складу платформи .NET, і використовує декларативну мову розмітки XAML для опису структури та зовнішнього вигляду інтерфейсу [10]. На відміну від своїх попередників WinForms та MFC WPF побудована на основі векторної графіки з апаратним прискоренням через DirectX, що забезпечує чітке відображення інтерфейсу на екранах із будь-якою роздільною здатністю та щільністю пікселів.

Технологія WPF природно підтримує архітектурний паттерн MVVM [13], який забезпечує чітке розмежування між логікою застосунку та його представленням. Завдяки механізму прив'язки даних зміни у стані ViewModel автоматично відображаються в інтерфейсі без необхідності ручного оновлення елементів керування. Серед конкретних переваг WPF для даного проєкту виділяються такі: потужна система прив'язки даних забезпечує відображення стану таймера та статистики у реальному часі; підтримка стилів та ресурсних словників дозволяє централізовано визначати єдине оформлення для всіх вікон застосунку; векторна графіка використовується для побудови кругових індикаторів прогресу режимів фокусування; широкі можливості кастомізації шаблонів елементів керування дозволяють реалізувати унікальний дизайн кнопок, полів введення та списків без залежності від зовнішніх бібліотек.

Для зберігання даних користувача обрано базу даних MongoDB [27]. Предметна область застосунку включає гнучкі структури даних: профіль користувача з вкладеною статистикою методів фокусування, списки заблокованих процесів із часовими діапазонами та журнал сесій усі ці структури природно представляються у вигляді BSON-документів [22] без необхідності визначення жорсткої реляційної схеми.

Середовищем розробки обрано Microsoft Visual Studio [20], яке забезпечує повноцінну підтримку C# та WPF [21]. Серед ключових інструментів середовища, що використовувались під час розробки: вбудований дизайнер XAML із режимом розділеного перегляду коду та візуального представлення,

		Хомнюк В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				19

відладчик із підтримкою точок зупину та інспекцією стану об'єктів у реальному часі, профілювальник продуктивності для виявлення вузьких місць у роботі фонових потоків, а також вбудована інтеграція з системою контролю версій Git для управління кодовою базою. Менеджер пакетів NuGet використовувався для підключення залежностей, MongoDB.Driver для роботи з базою даних та Google.Apis.PeopleService для інтеграції з Google OAuth [8].

Додаток не потребує підключення до мережі Інтернет у стандартному режимі роботи. Єдиним винятком є процедура першого входу через Google OAuth, яка потребує активного інтернет-з'єднання для проходження авторизації та отримання даних профілю Права адміністратора операційної системи є обов'язковими для коректної роботи функції блокування системних процесів, оскільки процес блокування для сторонніх процесів вимагає відповідних привілеїв безпеки для Windows.

Для реалізації авторизації через Google акаунт у застосунку використовується бібліотека Google.Apis.Auth та Google.Apis.PeopleService.v1. Google OAuth 2.0 є стандартним протоколом делегованої авторизації, що дозволяє застосунку отримати обмежений доступ до даних Google-акаунту користувача без зберігання його пароля. Процес авторизації відбувається наступним чином застосунок відкриває системний браузер із запитом на авторизацію, користувач підтверджує доступ на сторінці Google, після чого Google перенаправляє відповідь на локальний HTTP-сервер застосунку з кодом авторизації. Отриманий код обмінюється на токен доступу, який використовується для подальших запитів до Google People API.

Google People API [9] надає доступ до профільних даних авторизованого користувача. Запит із параметром personFields "names, emailAddresses" повертає лише необхідні поля ім'я та email без зайвих даних. Отримані дані використовуються для автоматичного заповнення профілю при першому вході ім'я користувача записується у поле Login та RegisterNickName колекції Profile, а email у поле Email обох колекцій Profile та User. Це усуває необхідність

		Хомнюк В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				20

ручного введення даних при першому запуску для користувачів, що обирають авторизацію через Google.

При виборі технологічного стеку було розглянуто ряд альтернативних рішень та обґрунтовано причини їх відхилення. WinForms як альтернатива WPF [21] була відхилена через відсутність підтримки векторної графіки, обмежені можливості кастомізації елементів керування та відсутність нативної підтримки паттерну MVVM, що ускладнило б організацію коду при зростанні складності застосунку. MAUI як сучасна кросплатформна альтернатива була відхилена через надмірну складність для суто Windows-застосунку та менш зрілий інструментарій розробки порівняно з WPF.

Entity Framework Core як ORM для роботи з реляційною базою даних також розглядався, однак складність налаштування міграцій та необхідність підтримки схеми для відносно простої структури даних застосунку зробили цей варіант надлишковим. MongoDB.Driver [24] із нативною підтримкою документної моделі забезпечує більш пряме та просте API для задач даного проєкту без додаткових абстракцій.

Висновки до першого розділу

Під час виконання першого розділу було вирішено ряд аналітичних задач, необхідних для подальшої розробки застосунку. Розгляд проблематики цифрових відволікань підтвердив актуальність теми сучасний користувач персонального комп'ютера стикається з систематичними перериваннями концентрації, що негативно впливає на якість і обсяг виконуваної роботи. Це обумовило необхідність створення інструменту, який діє не на рівні рекомендацій, а на рівні примусового обмеження доступу до відволікаючих програм.

Вивчення існуючих рішень на ринку показало, що більшість із них вирішують лише частину проблеми або відстежують активність без блокування, або обмежують доступ до сайтів без впливу на настільні застосунки. Жоден із розглянутих продуктів не пропонує локального рішення, що об'єднує кілька технік фокусування з керуванням процесами ОС в одному інтерфейсі. Саме цей

		Хомнюк В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				21

гар став відправною точкою для формування функціональних вимог до власної розробки.

Детальне вивчення методів Pomodoro, FlowTime, 52/17 та ультрадіяного ритму дозволило зрозуміти природу кожної техніки і закласти відповідну логіку в структуру режимів фокусування. Особливістю ультрадіяного режиму визначено підвищений рівень обмежень, що відображає більш жорсткі вимоги цього методу до середовища роботи.

Порівняльний аналіз архітектурних підходів довів доцільність застосування тришарової архітектури для проєкту з чітко окресленим функціоналом і локальним середовищем виконання. Обґрунтовано стек технологій C# з доступом до системного API, WPF для побудови інтерфейсу та MongoDB для локального зберігання даних. Сформовано перелік вимог до апаратного і програмного забезпечення.

Отримані результати формують повноцінну аналітичну базу та визначають чіткий вектор для проєктування і реалізації системи у наступних розділах.

		Хомнюк В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				
						22

РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ ДЛЯ УПРАВЛІННЯ СПІЛЬНОТАМИ ТА ПОДІЯМИ

2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи

Проведено аналіз вимог, за результатами якого виділено такі можливості: реєстрація користувача, авторизація користувача, доступ до додавання та перегляду нотаток з можливістю їх редагування та видалення, додавання домашніх завдань з дедлайнами, а також набір інструментів із системами «Pomodoro», «Flowtime», «Метод 24/7», «Ультрадiанний» та «Ігровий режим».

У застосунку реалізовано модуль контролю процесів операційної системи, який дозволяє користувачеві примусово обмежувати доступ до програм, що відволікають від роботи. Повторний запуск та використання заблокованого програмного забезпечення стають можливими лише після завершення сесії фокусування або у визначений програмою час.

У системі визначено одного актора (табл. 2.1) Оскільки застосунок є персональним інструментом для підвищення продуктивності, розмежування ролей не передбачено - кожен користувач має повний доступ до всього функціоналу без обмежень. Користувач може формувати та редагувати список заблокованих процесів, обирати та запускати режими фокусування, переглядати і редагувати нотатки, керувати профілем та переглядати накопичену статистику завершених сесій. Таким чином єдиний актор системи поєднує в собі як роль звичайного користувача, так і роль адміністратора власних налаштувань.

Таблиця 2.1

Актори

Актор	Опис актора
Користувач	Має можливість повністю керувати програмою: додавати видаляти записи, користуватись наданими інструментами та повністю змінювати програму.

У прецеденті Вхід/Реєстрація (табл. 2.2) автентифікація у застосунку реалізована виключно через інтеграцію з сервісом Google OAuth, що повністю скасовує потребу у класичній реєстрації за логіном та паролем. При першому

		Хомніук В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				23

запуску застосунків перевіряє наявність збережених даних акаунту у MongoDB - за їх відсутності користувач проходить одноразову авторизацію через Google, після чого система автоматично отримує ім'я та email через Google People API і створює профіль у локальній базі даних. При наступних запусках профіль завантажується автоматично без повторної авторизації. Такий підхід забезпечує миттєвий і безпечний доступ до застосунку, усуваючи необхідність запам'ятовування паролів та ризики, пов'язані із зберіганням облікових даних локально.

Таблиця 2.2

Опис прецеденту “Вхід/Реєстрація”

Назва варіанту	Вхід/Реєстрація
Основні актори	Користувач
Короткий опис	Автентифікація в системі реалізована виключно через інтеграцію з сервісом Google. Це скасовує потребу в класичній реєстрації за паролем, забезпечуючи користувачам миттєвий і безпечний доступ до додатка.

У прецеденті Додавання нотаток (табл. 2.3) користувач має можливість створювати, редагувати та видаляти текстові нотатки безпосередньо у застосунку без необхідності переключатися на сторонні програми. Функціонал нотаток призначений для фіксації важливої інформації під час або між сесіями фокусування — записів лекцій, завдань, ідей чи нагадувань. Кожна нотатка містить заголовок та текстовий вміст і зберігається у колекції Notes локальної бази даних MongoDB із прив'язкою до акаунту поточного користувача. Пошук по нотатках здійснюється у реальному часі за заголовком та вмістом, що забезпечує швидкий доступ до потрібного запису навіть за великої кількості збережених нотаток. Усі зміни зберігаються локально та залишаються доступними між перезапусками застосунку без необхідності підключення до мережі Інтернет.

Таблиця 2.3

Описа преценденту “Додавання нотаток”

Назва варіанту	Додавання нотаток
Основні актори	Користувач
Короткий опис	Користувачі програми можуть додавати нотатки щоб занотувати якусь подію, лекцію або завдання.

У преценденті Відкривати систему Pomodoro (табл. 2.4) користувач має можливість запускати режим фокусування безпосередньо з головного меню застосунку. Система фокусування об'єднує чотири науково обґрунтовані методики - Pomodoro, FlowTime, метод 52/17 та ультрадіанний ритм — кожна з яких має власне вікно з таймером, індикатором прогресу та кнопками керування. Після вибору методу та натискання кнопки запуску система автоматично активує фоновий сервіс моніторингу процесів, який протягом усієї робочої фази блокує застосунки із персонального списку користувача. Елементи навігації головного меню блокуються на час активної сесії, що унеможливлює випадкове переключення між розділами під час роботи. Після завершення кожного робочого циклу система автоматично сповіщає користувача звуковим сигналом, зберігає результат сесії у колекції Profile бази даних MongoDB та активує таймер відпочинку із рекомендованою тривалістю паузи для обраного методу.

Таблиця 2.4

Описа преценденту “Відкривати систему Pomodoro”

Назва варіанту	Відкривати систему фокусування
Основні актори	Користувач
Короткий опис	Користувач програми може відкривати систему фокусування, використовувати методи концентрації уваги та керувати блокуванням систем і застосунків для підвищення продуктивності та ефективної організації роботи.

		Хомнюк В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				
						25

У прецеденті Відкривати систему Метод 52/17 (табл. 2.5) користувач запускає режим фокусування що базується на чергуванні 52-хвилинних робочих інтервалів із 17-хвилинними паузами відпочинку. Після запуску система активує таймер зворотного відліку та фоновий сервіс блокування процесів із персонального списку користувача. Після завершення кожного циклу результат автоматично зберігається у профілі MongoDB та активується таймер відпочинку.

Таблиця 2.5

Описа преценденту “Відкривати систему Метод 24/7”

Назва варіанту	Відкривати систему Метод 24/7
Основні актори	Користувач
Короткий опис	Користувач програми може відкривати систему Метод 24/7, використовувати методи фокусування та керувати блокуванням систем і застосунків для підтримки продуктивної роботи.

У прецеденті Відкривати систему Ультрадiанний (табл 2.6) користувач запускає найінтенсивніший режим фокусування застосунку, що базується на природних 90-хвилинних циклах активності мозку з 20-хвилинними паузами відновлення. Особливістю цього режиму є автоматичне розширення списку блокування — система без додаткових налаштувань примусово завершує процеси браузерів та Telegram протягом усього робочого циклу, забезпечуючи максимальний рівень ізоляції від відволікань для виконання складних інтелектуальних завдань.

Таблиця 2.6

Описа преценденту “Відкривати систему Ультрадiанний”

Назва варіанту	Відкривати систему Ультрадiанний
Основні актори	Користувач
Короткий опис	Користувач програми може відкривати систему Ультрадiанний, використовувати методи концентрації уваги та керувати блокуванням.

У прецеденті Відкривати систему Ігровий режим (табл 2.7) користувач отримує доступ до спеціалізованого модуля керування блокуванням ігор. У цьому режимі система відображає список застосунків та ігор що підлягають обмеженню, надає можливість налаштовувати розклад їх використання та активує фоновий моніторинг для примусового завершення заборонених процесів поза дозволеним часовим діапазоном.

Таблиця 2.7

Опис прецеденту “Відкривати систему ігровий режим”

Назва варіанту	Відкривати систему Ігровий режим
Основні актори	Користувач
Короткий опис	Користувач програми може відкривати систему Ігровий режим, активувати необхідні функції та керувати блокуванням систем і застосунків для комфортного використання програми під час гри.

У прецеденті Блокування систем та застосунків (табл 2.8) користувач вручну вводить назву виконуваного файлу у форматі .exe, обирає день тижня та встановлює часовий діапазон у межах якого програма є дозволеною. Поза визначеним розкладом фоновий сервіс моніторингу автоматично виявляє запущений процес та примусово завершує його через process.Kill(), не вимагаючи жодних додаткових дій з боку користувача.

Таблиця 2.8

Опис прецеденту «Блокування систем та застосунків»

Назва варіанту	Блокування систем та застосунків
Основні актори	Користувач
Короткий опис	Користувач може вводити назву виконуваного файлу у форматі .exe, обирати день тижня та часовий діапазон, у межах якого програма є дозволеною. Поза визначеним розкладом фоновий сервіс автоматично завершує її процес.

У прецеденті Блокування гри зі Steam бібліотеки (табл 2.9) користувач відкриває вікно інтеграції зі Steam, де автоматично підвантажуються список ігор встановлених на комп'ютері. Після вибору гри система самостійно визначає її виконуваний файл без необхідності ручного введення. Користувач задає розклад дозволеного запуску за днями тижня та часовим діапазоном, після чого фоновий сервіс примусово завершує процес гри у будь-який час поза встановленим розкладом.

Таблиця 2.9

Опис прецеденту «Блокування гри зі Steam бібліотеки»

Назва варіанту	Блокування гри зі Steam бібліотеки
Основні актори	Користувач
Короткий опис	У вікні інтеграції зі Steam відображаються встановлені на ПК ігри користувача. Після вибору гри система автоматично визначає її виконуваний файл. Користувач може задати розклад (дні та години) дозволеного запуску, поза яким фоновий сервіс примусово завершуватиме процес гри.

Додаток контролю активності є настільним застосунком для Windows, тому взаємодія з ним відбувається безпосередньо через графічний інтерфейс без необхідності підключення до мережі. Проаналізувавши предметну область, було виділено одного актора - Користувач, оскільки застосунок є персональним інструментом і не передбачає розмежування ролей чи багатокористувацького режиму.

Взаємодія користувача із системою охоплює чотири основні напрями. Перший напрям – управління профілем з переглядом накопиченої статистики використання методів фокусування та налаштуванням персональних параметрів. Другий напрям – робота з нотатками із записом, переглядом та редагуванням нотаток безпосередньо під час сесії. Третій напрям – використання режимів фокусування із запуском сесії як у базовому режимі, так і з активованим блокуванням процесів для розширення таймера функцією примусового обмеження доступу до відволікаючих застосунків. Четвертий напрям – окреме

		Хомніук В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				28

блокування процесів через головне меню без прив'язки до конкретного режиму фокусування.

Далі представлено наочне відображення цього у діаграмі варіантів використання яка предоставлена нижче (рис. 2.1).

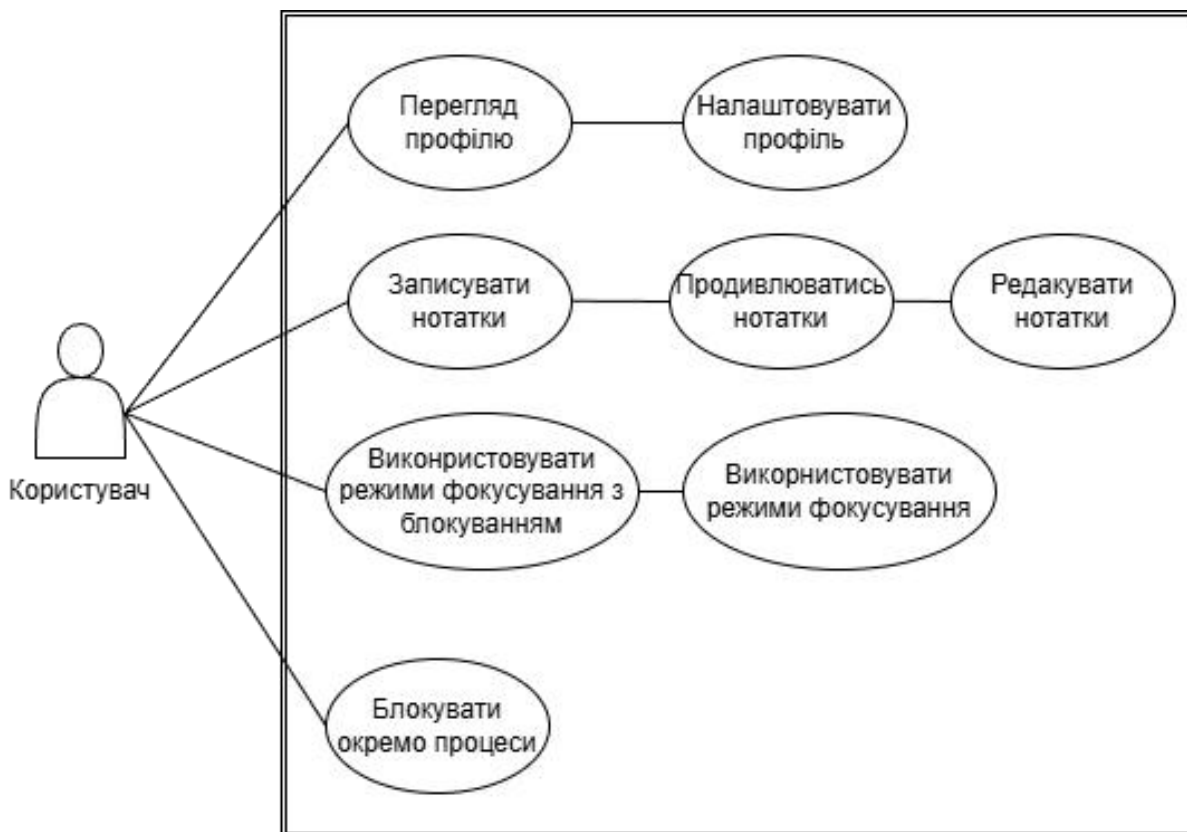


Рис. 2.1. Діаграма варіантів

При розробці настільних застосунків на платформі WPF [21] існують усталені підходи до організації коду, які визначають спосіб взаємодії інтерфейсу з логікою програми.

У застосунку реалізовано підхід Code-Behind з розміщенням логіки обробки подій та управління станом безпосередньо у файлах вікон. Кожне вікно (UltraFocus, Pomodoro, FlowtimeWindow та Productivity) інкапсулює власні поля стану – прапорці режимів, таймери, лічильники циклів – та методи реакції на дії користувача. Такий підхід природний для WPF і забезпечує простоту реалізації за умови чітко визначеного та обмеженого функціоналу кожного вікна (рис. 2.2).

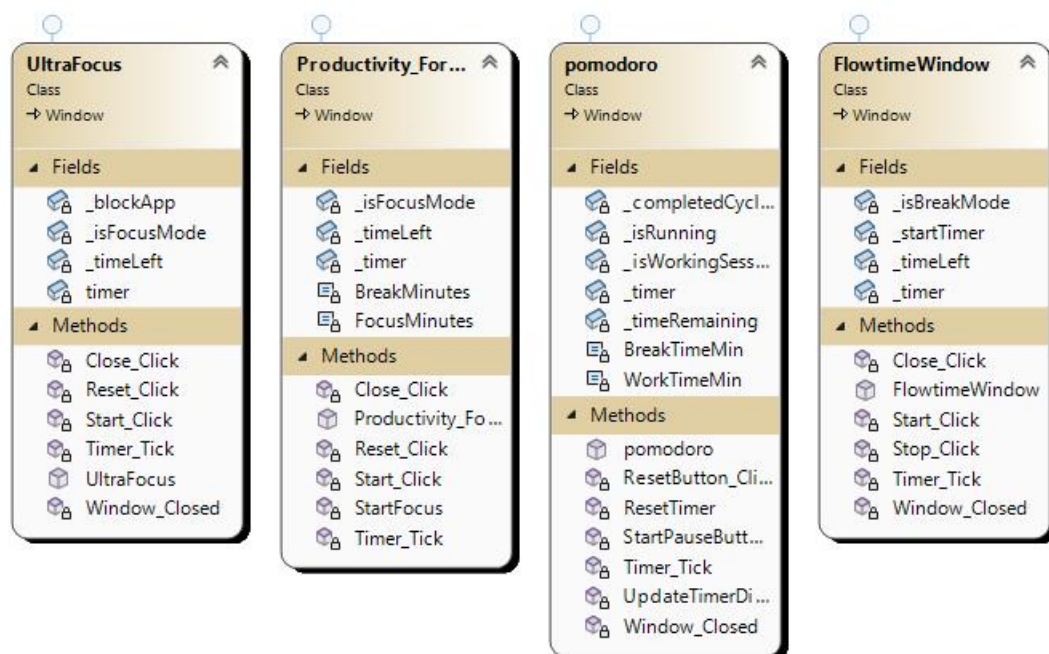


Рис. 2.2. Підхід Code-Behind

Для реалізації контролю активності застосунків потребував механізму постійного спостереження за станом операційної системи в реальному часі. З цією метою у застосунку реалізовано патерн Monitor для безперервного нагляду за запущеними процесами. Клас BlockApp інкапсулює повний цикл моніторингу. Метод StartMonitoring ініціює фоновий процес спостереження, MonitorGameS циклічно перевіряє поточний стан системи через IsAppRunning , а StopMonitoring коректно завершує спостереження після закінчення сесії. Клас Unblock доповнює цю логіку, надаючи користувачу інтерфейс для оперативного керування списком програм, що перебувають під контролем (рис. 2.3).

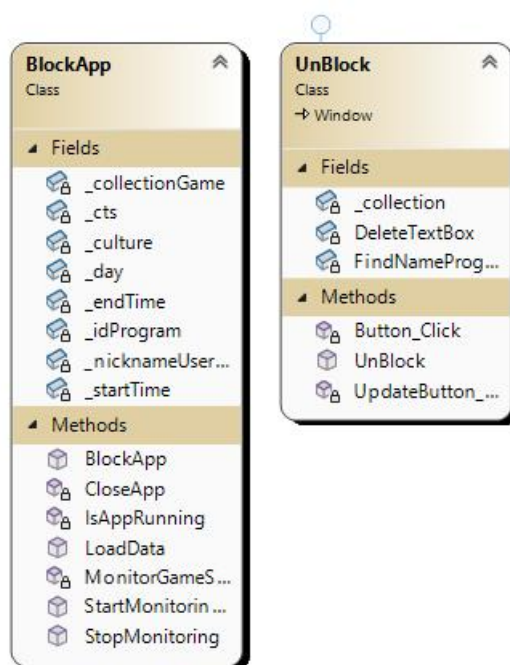


Рис. 2.3. Патерн Monitor

2.2. Розробка бази даних системи

Для надійного збереження та ефективного управління даними у проєкті обрано систему управління базами даних MongoDB [27]. Вибір цієї NoSQL-СУБД зумовлений високою продуктивністю обробки великих обсягів даних і гнучкістю моделі документів з оперативною адаптацією структури даних під функціональні зміни застосунку. На відміну від реляційних баз даних, MongoDB не вимагає визначення жорсткої схеми таблиць, що дозволяє розширювати структуру колекцій без міграцій та простоїв системи. Доступ до бази даних здійснюється через офіційний драйвер MongoDB.Driver [20], який надає типізований асинхронний API для виконання усіх операцій зі збереження та отримання даних без блокування UI-потoku.

Колекція BlockUserGame (табл 2.10) зберігає налаштування блокування програм для кожного користувача. Кожен документ містить унікальний ідентифікатор, назву виконуваного файлу та відображувану назву програми, прив'язку до акаунту користувача, день тижня та часовий діапазон дозволеного використання через поля StartTime і EndTime, а також час фактичного спрацювання блокування. На основі цих даних фоновий сервіс моніторингу визначає коли і який процес має бути примусово завершений. Документна

модель MongoDB дозволяє гнучко розширювати структуру колекції без необхідності міграцій схеми, що є суттєвою перевагою порівняно з реляційними базами даних при додаванні нових параметрів блокування у майбутніх версіях.

Таблиця 2.10.

Структура таблиці “BlockUserGame”

Поле	Опис	Тип	Призначення
id	[BsonId]	Objectid	Унікальний ідентифікатор
Id_Game	Програма	string	Зберігає саму програму
Name_Game	Назва гри	string	Зберігає назву гри
User	Користувач	string	Зберігає ім'я користувача
Day	День коли не можна використовувати програму	string	Зберігає день коли можна використовувати програму
StartTime	Початок блокування	string	Зберігає початок часу коли можна використовувати програму
EndTime	Кінець блокування	string	Зберігає кінець часу коли можна використовувати програму
StartTimeBlockPrograms	Час коли була заблокована програма	string	Зберігає час блокування програми

Колекція Profile (табл 2.11) зберігає повний профіль користувача застосунку. Кожен документ містить унікальний ідентифікатор, логін та email отримані через Google People API під час авторизації, реєстраційний нікнейм та текстовий опис акаунту. Ключовою частиною документа є чотири числові лічильники — UltraFocus, Pomodoro, Flowtime та Productivity_Formula — які автоматично інкрементуються після кожної успішно завершеної сесії відповідного методу фокусування та відображаються на сторінці профілю користувача. Такий підхід дозволяє у реальному часі відстежувати динаміку

використання кожного методу без необхідності виконання складних агрегаційних запитів до бази даних, оскільки всі лічильники зберігаються безпосередньо у єдиному документі профілю та оновлюються атомарно через операцію Inc драйвера MongoDB.Driver.

Таблиця 2.11

Структура таблиці “Profile”

Поле	Опис	Тип	Призначення
id	[BsonId]	Objectid	Унікальний ідентифікатор в базі даних
Login	Логін користувача	string	Використовується для авторизації в системі
RegisterNickName	Реєстраційний нікнейм	string	Зберігає ім'я користувача, вказане під час реєстрації
Email	Електронна пошта	string	Містить коротку інформацію або статус акаунта
UserDescription	Опис користувача	string	Відображає кількість використань режиму UltraFocus
UltraFocus	Лічильник UltraFocus	Int32	Відображає кількість використань режиму Pomodoro
Pomodoro	Лічильник Pomodoro	Int32	Відображає кількість використань режиму Pomodoro
Flowtime	Лічильник Flowtime	Int32	Відображає кількість використань режиму Flowtime
Productivity_Formula	Лічильник Productivity	Int32	Відображає кількість використань формули продуктивності

Колекція User (табл 2.12) є базовою колекцією ідентифікації користувача у системі. Документ містить мінімально необхідний набір полів: унікальний ідентифікатор, логін та email отримані під час авторизації через Google OAuth. Саме поле Email є ключовим зв'язуючим елементом між колекціями — сервіс моніторингу BlockApp використовує його для фільтрації записів у колекції BlockUserGame, а колекція Profile прив'язує статистику фокусування до конкретного акаунту через це ж поле. Мінімалістична структура колекції

обумовлена архітектурним рішенням зберігати детальні дані профілю окремо у колекції Profile, що забезпечує чіткий розподіл відповідальності між колекціями та спрощує операції пошуку користувача при авторизації.

Таблиця 2.12

Структура таблиці “User”

Поле	Опис	Тип	Призначення
id	[BsonId]	Objectid	Унікальний ідентифікатор в базі даних
Login	Логін користувача	string	Використовується для авторизації та входу в систему
Email	Електронна пошта користувача	string	Використовується для зв'язку та відновлення доступу до акаунта

База даних застосунку побудована на основі документно-орієнтованої NoSQL-СУБД MongoDB, для взаємодії з якою використовується офіційний драйвер MongoDB.Driver, що забезпечує типізований доступ до колекцій через C#-класи та автоматично серіалізує об'єкти у формат BSON при збереженні та десеріалізує при читанні.

Між колекціями встановлено логічні зв'язки через поле Email/Login користувача. Таким чином система не дозволить відобразити статистику або налаштування блокування для неіснуючого акаунту. При отриманні даних профілю драйвер автоматично мапить BSON-документ на відповідний C#-клас через атрибут для ідентифікатора та відповідність назв полів.

Операція створення нового користувача виконується послідовно спочатку створюється запис у колекції User із логіном та email, після чого у колекції Profile створюється відповідний документ із нульовими лічильниками методів фокусування. Якщо на будь-якому з етапів виникає помилка, другий запис не створюється, що унеможливорює існування профілю без відповідного акаунту.

База даних застосунку складається з трьох колекцій BlockUserGame зберігає налаштування блокування програм, Profile зберігає профіль користувача та статистику методів фокусування, User зберігає дані акаунту.

		Хомнюк В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				
						34

2.3. Проектування та реалізація алгоритмів роботи системи

Для наочного відображення логіки роботи застосунку розроблено блок-схему алгоритму виконання сесії фокусування. Схема охоплює повний життєвий цикл застосунку від моменту запуску до завершення сесії та переходу у стан очікування з відображенням взаємодії між паралельними потоками виконання [18], механізмом блокування процесів та підсистемою збереження даних [25].

Точкою входу виступає запуск застосунку FocusMind з подальшою ініціалізацією головної форми. На цьому етапі система одночасно запускає два незалежні потоки: UI-потік відповідає за побудову та відображення головного вікна застосунку, тоді як DB-потік паралельно встановлює підключення до локального екземпляра MongoDB. Паралельне виконання цих потоків суттєво скорочує час запуску застосунку, оскільки підключення до бази даних відбувається одночасно з ініціалізацією інтерфейсу, не блокуючи відображення головного вікна та не змушуючи користувача очікувати завершення мережевих операцій перед появою інтерфейсу. Такий підхід забезпечує миттєву реакцію застосунку на запуск незалежно від швидкості встановлення підключення до локальної бази даних, що є особливо важливим при першому старті системи після завантаження операційної системи, коли служба MongoDB може ще не встигнути повністю ініціалізуватися.

Після завершення ініціалізації обох потоків система виконує перевірку наявності збережених даних акаунту у колекції User бази даних MongoDB. Ця перевірка визначає подальший маршрут виконання алгоритму. Відсутність записів у колекції означає перший запуск застосунку на пристрої. У такому випадку користувачу пропонується форма введення імені та нікнейму, після заповнення якої система автоматично створює два документи – запис у колекції User із логіном та email, а також відповідний документ у колекції Profile із нульовими значеннями лічильників для кожного з чотирьох методів фокусування. За наявності збережених даних профіль завантажується

		Хомніук В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				
						35

автоматично без додаткових дій з боку користувача, і він одразу потрапляє до головного меню застосунку.

Перша гілка реалізує фоновий сервіс моніторингу процесів. Після отримання сигналу про запуск сесії сервіс звертається до колекції BlockUserGame у MongoDB та завантажує актуальний список заблокованих програм для поточного користувача. Далі запускається нескінченний цикл опитування системних процесів через Win32 API з інтервалом 500 мс. На кожній ітерації метод Process.GetProcesses() [17] повертає масив усіх активних процесів операційної системи. Для кожного процесу виконується порівняння його імені з заборонених програм через Path.GetFileNameWithoutExtension() без урахування регістру символів. У разі виявлення збігу процес примусово завершується через process.Kill() [17], а відповідна подія фіксується у журналі колекції logs із зазначенням часу, імені процесу та ідентифікатора сесії. Цикл моніторингу виконується безперервно до моменту деактивації таймера сесії, після чого він коректно завершується через механізм CancellationToken [16], звільняючи системні ресурси.

У режимі ультрадіяного ритму [28] сервіс моніторингу отримує розширений список блокування. Окрім програм, визначених користувачем вручну, до переліку автоматично додаються імена процесів браузерів (chrome, msedge, firefox, opera) та месенджера Telegram. Це забезпечує максимальний рівень ізоляції від відволікань протягом 90-хвилинного робочого циклу без необхідності додаткового налаштування з боку користувача.

Друга гілка обробляє взаємодію користувача з інтерфейсом у головному потоці UI. Після натискання кнопки «Старт» усі навігаційні елементи меню блокуються, що унеможливлює перехід до інших розділів застосунку під час активної сесії. Одночасно запускається зворотній відлік хронометра - DispatcherTimer з інтервалом одна секунда оновлює відображення таймера у форматі hh:mm:ss. Для кожного методу фокусування логіка перемикавання фаз реалізована окремо: Pomodoro чергує 25-хвилинні робочі інтервали з 5-хвилинними перервами, метод 52/17 використовує цикли відповідної

		Хомнюк В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				36

тривалості, ультрадіанний режим будує структуру з 90-хвилинного робочого блоку та 20-хвилинної відновлювальної перерви. FlowTime [30] є винятком - його таймер рахує час у прямому напрямку, відображаючи тривалість поточної сесії без фіксованого ліміту завершення.

Третя гілка є асинхронним потоком бази даних, який паралельно отримує сигнали від репозиторію та забезпечує синхронізацію стану між інтерфейсом і збереженими даними. Цей потік відповідає за оперативне оновлення лічильників та запис проміжних подій без блокування UI-потoku, що є критично важливим для забезпечення плавної роботи інтерфейсу під час активної сесії фокусування. Усі операції запису до MongoDB виконуються асинхронно через методи InsertOneAsync та UpdateOneAsync офіційного драйвера MongoDB.Driver [20], що унеможливує ситуацію коли операція збереження даних затримує оновлення таймера або реакцію інтерфейсу на дії користувача. Завдяки механізму async/await платформи .NET [15] потік бази даних не займає системні ресурси під час очікування відповіді від MongoDB, а звільняє потік виконання для обробки інших завдань. Це особливо важливо у момент завершення сесії, коли одночасно виконується кілька операцій: подається звуковий сигнал, оновлюється лічильник у колекції Profile, активується таймер відпочинку та знімаються обмеження на запуск програм — усі ці дії мають відбуватися миттєво і злагоджено без жодних затримок у відображенні інтерфейсу.

Коли таймер досягає нульової позначки, всі три паралельні гілки синхронізуються. Система подає звуковий сигнал, що сповіщає користувача про завершення робочої фази. Після цього у колекції Profile MongoDB оновлюється лічильник завершених сесій відповідного методу фокусування через інкремент відповідного поля Pomodoro, Flowtime, UltraFocus або Productivity_Formula. Одночасно активується таймер відпочинку з тривалістю, що відповідає поточному методу, знімаються всі обмеження на запуск заблокованих програм та розблоковуються елементи навігації головного меню. Після завершення таймера відпочинку алгоритм повертається у стан очікування,

		Хомніук В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				
						37

готовий до запуску наступної сесії фокусування без необхідності перезапуску застосунку.

2.4. Реалізація функціоналу системи організації спільнот та подій

Розглянемо основний функціонал додатку блокування. Метод LoadData() завантажує з колекції MongoDB BlockUserGame налаштування блокування ідентифікатор програми, email користувача, дозволений часовий діапазон та день тижня. Ці дані використовуються надалі для перевірки умов блокування.

```
public void LoadData()
{
    var gameBlock = _collectionGame.Find(new BsonDocument()).ToList();

    foreach (var item in gameBlock)
    {
        _idProgram = item.Contains("Id_Game") ? item["Id_Game"].ToString() :
string.Empty;
        _nicknameUserEmail = item.Contains("User") ? item["User"].ToString() :
string.Empty;
        _startTime = item.Contains("StartTime") ?
TimeSpan.Parse(item["StartTime"].ToString()) : TimeSpan.Zero;
        _endTime = item.Contains("EndTime") ?
TimeSpan.Parse(item["EndTime"].ToString()) : TimeSpan.Zero;
        _day = item.Contains("Day") ? item["Day"].ToString() : string.Empty;
    }
}
```

Блокування процесів у режимі ультрадіяного ритму. Код формує масив із назвами заборонених застосунків - Steam та Telegram - після чого перебирає всі активні системні процеси та порівнює їх імена з переліком без урахування регістру символів через StringComparison.OrdinalIgnoreCase. При виявленні збігу процес примусово завершується через process.Kill().

Такий підхід обумовлений специфікою ультрадіяного режиму, який орієнтований на глибоку зосереджену роботу протягом тривалого 90-хвилинного циклу. Steam та Telegram визначено як найбільш поширені джерела імпульсивного відволікання серед цільової аудиторії застосунку - розробників та студентів. Steam провокує переключення на ігри, а Telegram генерує безперервний потік сповіщень, що руйнують стан глибокої концентрації. На

		Хомніук В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				
						38

відміну від режиму BD, де користувач самостійно визначає список заблокованих програм через базу даних, у режимі ультрадіяного ритму список блокування є фіксованим та застосовується автоматично без будь-яких додаткових налаштувань. Це рішення прийнято свідомо користувач, який обирає ультрадіяний режим, апріорі погоджується на максимальний рівень обмежень заради досягнення стану повного фокусування. Цикл перевірки виконується у фоновому потоці з інтервалом 5000 мс і залишається активним протягом усього 90-хвилинного робочого блоку, автоматично деактивууючись під час фази відпочинку.

```
string[] blockedApps = { "steam", "Telegram" };
foreach (string blockedApp in blockedApps)
{
    if (process.ProcessName.Equals(blockedApp,
StringComparison.OrdinalIgnoreCase))
    {
        process.Kill();
    }
};
```

Подібний процес відбувається і в режимі BD. Код отримує ім'я виконуваного файлу з `_idProgram` через `Path.GetFileNameWithoutExtension()`, що дозволяє коректно порівнювати назву процесу без урахування розширення файлу. Після цього отримане ім'я порівнюється з іменем кожного активного системного процесу без урахування регістру символів через `StringComparison.OrdinalIgnoreCase`. При виявленні збігу процес примусово завершується через `process.Kill()`.

Відмінність від режиму NoBD полягає у тому, що тут блокується конкретна програма, визначена користувачем через базу даних, а не фіксований перелік застосунків. Це забезпечує гнучкість налаштування кожен користувач може самостійно визначити який застосунок має блокуватися у його випадку, що робить систему персоналізованою та адаптованою під індивідуальні потреби. Такий підхід дозволяє розширювати список заблокованих програм без змін у дії застосунку, оскільки всі налаштування зберігаються та зчитуються безпосередньо з MongoDB.

		Хомніук В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				
						39

```

if (!string.IsNullOrEmpty(_nicknameUserEmail) && currentUserEmail ==
_nicknameUserEmail)
{
    bool isCorrectDay = dayOfWeekUkrainian.Equals(_day,
StringComparison.OrdinalIgnoreCase);
    bool isAllowedTime = currentTime >= _startTime && currentTime < _endTime;

    if (!(isCorrectDay && isAllowedTime))
    {
        if (!string.IsNullOrEmpty(_idProgram) && IsAppRunning(_idProgram))
        {
            CloseApp(numberFocus, mode);
        }
    }
}

```

Цей фрагмент коду виконує рекурсивний обхід заданого переліку директорій для пошуку виконуваного файлу гри за її назвою. На кожній ітерації циклу програма зчитує всі файли з розширенням .exe у поточній папці та її піддиректоріях. Для кожного знайденого файлу виокремлюється його ім'я без розширення, після чого перевіряється наявність часткового збігу із шуканою назвою гри. Порівняння рядків здійснюється без урахування регістру символів, що дозволяє уникати помилок через різне написання великих і малих літер. У разі успішного збігу та за умови, що ім'я файлу не є порожнім, робота фрагмента негайно переривається із поверненням повного шляху до знайденого виконуваного файлу.

```

foreach (var dir in dirs)
{
    var exeFiles = Directory.GetFiles(dir, "*.exe", SearchOption.AllDirectories);
    foreach (var exe in exeFiles)
    {
        string exeName = Path.GetFileNameWithoutExtension(exe);
        if (exeName != null &&
            exeName.Contains(gameName, StringComparison.OrdinalIgnoreCase))
        {
            return exe;
        }
    }
}

```

Фрагмент коду реалізує пошук виконуваного файлу гри в директорії Steam. Спочатку перевіряється наявність кореневої папки, після чого у її піддиректоріях запускається рекурсивне зчитування всіх файлів із розширенням .exe. Для кожного знайденого об'єкта виокремлюється ім'я та перевіряється його збіг із назвою гри без урахування регістру символів. У разі успішного виявлення фрагмент повертає повний шлях до файлу, який використовується для автоматичного заповнення поля назви процесу у формі блокування.

```
foreach (string blockedApp in blockedApps)
{
    if (process.ProcessName.Equals(blockedApp,
StringComparison.OrdinalIgnoreCase))
    {
        process.Kill();
    }
}
```

Код отримує список ігор користувача через Steam Web API. Формується URL-запит до офіційного ендпоінту IPlayerService/GetOwnedGames із підстановкою API-ключа та Steam ID користувача. Через HttpClient виконується асинхронний GET-запит, відповідь зчитується як рядок та парситься у JSON-об'єкт через JObject.Parse [25]. З отриманого об'єкта витягується масив ігор за шляхом response.games - якщо масив порожній або відсутній, метод завершується достроково. За наявності ігор колекція gameNames очищається та заповнюється іменами ігор для кожного елементу масиву витягується поле name, перевіряється на порожнечу через IsNullOrWhiteSpace та додається до колекції. Після заповнення колекція прив'язується до елемента ListSteamGame як джерело даних для відображення списку ігор у інтерфейсі користувача.

```
private async Task LoadGamesAsync()
{
    string url =
$"https://api.steampowered.com/IPlayerService/GetOwnedGames/v1/?key={apiKey}
&steamid={steamId}&include_appinfo=1";

    using HttpClient client = new();
    var response = await client.GetAsync(url);
```

		Хомнюк В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				41

```

string json = await response.Content.ReadAsStringAsync();
JsonObject data = JObject.Parse(json);

var games = data["response"]?["games"];

if (games == null) return;

gameNames.Clear();

foreach (var game in games)
{
    string name = game["name"]?.ToString();
    if (!string.IsNullOrEmpty(name))
        gameNames.Add(name);
}

ListSteamGame.ItemsSource = gameNames;
}

```

Логіка перемикання фаз у режимі Pomodoro. Код перевіряє поточний стан сесії через прапорець `_isWorkingSession`. Якщо активна робоча фаза збільшується лічильник завершених циклів, викликається метод `UpdatePomodoro()` для збереження статистики у базі даних, оновлюється текст інтерфейсу з кількістю виконаних циклів, після чого система переводиться у фазу відпочинку встановлюється відповідний статус та завантажується тривалість перерви з `BreakTimeMin`. В іншому випадку, коли завершується фаза відпочинку, система повертається до робочого режиму встановлюється спеціальний статус.

```

if (_isWorkingSession)
{
    _completedCycles++;
    AddNumberFocus.UpdatePomodoro();
    CycleText.Text = $"Завершено циклів: {_completedCycles}";
    _isWorkingSession = false;
    StatusText.Text = "Час відпочити";
    _timeRemaining = TimeSpan.FromMinutes(BreakTimeMin);
}
else
{
    _isWorkingSession = true;
}

```

		Хомнюк В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				42

```

StatusText.Text = "Час фокусуватися!";
_timeRemaining = TimeSpan.FromMinutes(WorkTimeMin);
}

```

Наведено метод `Timer_Tick`, який викликається щосекунди та керує логікою відліку часу у режимі `Productivity Formula`. Поки значення `_timeLeft` більше нуля таймер зменшується на одну секунду і оновлює відображення у форматі `hh:mm:ss`, після чого метод завершує виконання.

Коли таймер досягає нуля, перевіряється поточна фаза через прапорець `_isFocusMode`. Якщо активна фаза фокусування викликається `UpdateProductivity_Formula()` для збереження статистики, система переводиться у фазу перерви із завантаженням тривалості з `BreakMinutes` та відображенням статусу «ПЕРЕПВА». В іншому випадку система повертається до робочої фази встановлюється тривалість із `FocusMinutes` та статус «WAVE FLOW». Незалежно від фази, після перемикання оновлюється відображення таймера на екрані.

```

if (_timeLeft > TimeSpan.Zero)
{
    _timeLeft -= TimeSpan.FromSeconds(1);
    TimerText.Text = _timeLeft.ToString(@"hh\:mm\:ss");
    return;
}
if (_isFocusMode)
{
    AddNumberFocus.UpdateProductivity_Formula();
    _isFocusMode = false;
    _timeLeft = TimeSpan.FromMinutes(BreakMinutes);
    StatusText.Text = "ПЕРЕПВА";
}
else
{
    _isFocusMode = true;
    _timeLeft = TimeSpan.FromMinutes(FocusMinutes);
    StatusText.Text = "WAVE FLOW";
}
TimerText.Text = _timeLeft.ToString(@"hh\:mm\:ss");

```

Наведено метод `Timer_Tick` для режиму `UltraFocus`. Метод викликається щосекунди через `DispatcherTimer` і є центральним елементом управління

		Хомнюк В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				
						43

циклами ультрадіанного ритму. Поки `_timeLeft` більше нуля - таймер щосекунди зменшується на одну секунду через `TimeSpan.FromSeconds(1)` з оновленням відображення у форматі `hh:mm:ss`.

Після досягнення нуля перевіряється поточна фаза через прапорець `_isFocusMode`. Якщо активна робоча фаза - викликається метод `UpdateUltraFocus()`, який звертається до колекції `Profile` у `MongoDB` та інкрементує поле `UltraFocus` для поточного користувача, фіксуючи факт успішного завершення робочого циклу. Після збереження статистики таймер переводиться на 2-хвилинну перерву шляхом присвоєння `_timeLeft = TimeSpan.FromMinutes(2)`, прапорець `_isFocusMode` скидається у `false` та користувачу відображається діалогове вікно `WpfMessageBox` із повідомленням «Час фокусування завершено!». Відображення повідомлення виконується в UI-поточці через `Dispatcher.Invoke`, що забезпечує коректну роботу без помилок міжпоточкового доступу.

В іншому випадку, коли завершується фаза відпочинку, система автоматично готує наступний робочий цикл таймер повертається до 30-хвилинного робочого інтервалу через `TimeSpan.FromMinutes(30)`, `_isFocusMode` встановлюється у `true` та знову відображається повідомлення про завершення фази відпочинку. Такий підхід забезпечує безперервне автоматичне чергування робочих і відновлювальних циклів без необхідності ручного перезапуску таймера після кожної фази.

Незалежно від фази, після кожного перемикання таймер оновлює відображення часу на екрані через `TimerText.Text` `_timeLeft.ToString(@"hh\:mm\:ss)`. Це дозволяє користувачу у реальному часі бачити скільки часу залишилось до завершення поточної фази - як робочої, так і відновлювальної. Архітектурне рішення розмістити всю логіку перемикання фаз безпосередньо у методі `Timer_Tick` відповідає підходу `Code-Behind`, характерному для WPF-застосунків, і забезпечує простоту реалізації за умови чітко визначеної та незмінної структури циклів ультрадіанного ритму.

`if (_timeLeft > TimeSpan.Zero)`

		Хомніук В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				44

```

{
    _timeLeft -= TimeSpan.FromSeconds(1);
    TimerText.Text = _timeLeft.ToString(@"hh\:mm\:ss");
    return;
}

if (_isFocusMode)
{
    AddNumberFocus.UpdateUltraFocus();
    _timeLeft = TimeSpan.FromMinutes(2);
    _isFocusMode = false;
    WpfMessageBox.Show("Час фокусування завершено!", "Ultra Focus",
        MessageBoxButton.OK, MessageBoxImage.Information);
}
else
{
    _timeLeft = TimeSpan.FromMinutes(30);
    _isFocusMode = true;
    WpfMessageBox.Show("Час фокусування завершено!", "Ultra Focus",
        MessageBoxButton.OK, MessageBoxImage.Information);
}
TimerText.Text = _timeLeft.ToString(@"hh\:mm\:ss");

```

Наведено метод `Timer_Tick` для режиму `FlowTime`. Метод розгалужується залежно від поточної фази через прапорець `_isBreakMode`.

Якщо активна фаза відпочинку (`_isBreakMode = false`) - таймер рахує час у прямому напрямку обчислюється різниця між поточним часом і моментом старту через `DateTime.Now - _startTimer`, після чого результат відображається у форматі `hh:mm:ss`. Це реалізує характерну особливість `FlowTime` - відлік тривалості робочої сесії вгору без фіксованого ліміту.

Якщо активна фаза перерви - таймер працює у зворотному напрямку щосекунди зменшується на одну секунду. Коли відлік досягає нуля, таймер зупиняється через `_timer.Stop()`, прапорець `_isBreakMode` скидається у `false`, статус оновлюється на «WAVE FLOW» та відображення таймера скидається до `00:00:00`, сигналізуючи про готовність до нової робочої сесії.

Висновки до другого розділу

У ході виконання другого розділу було вирішено комплекс задач, пов'язаних із проєктуванням та реалізацією ключових модулів застосунку контролю активності та організації роботи користувача.

Визначено та описано варіанти використання системи, сформовано діаграму варіантів використання з детальним описом взаємодії єдиного актора із функціональними блоками застосунку. Виокремлено та обґрунтовано застосування паттернів проєктування - Code-Behind для організації логіки вікон та Monitor для реалізації фонового сервісу спостереження за процесами.

Спроектовано структуру бази даних MongoDB: визначено колекції для зберігання профілю користувача, налаштувань блокування та журналу сесій. Описано логіку першого запуску застосунку з перевіркою наявності збережених даних акаунту та автоматичним створенням нового профілю через Google People API.

Розроблено та описано алгоритм роботи застосунку у вигляді блок-схеми, що охоплює паралельне виконання потоків моніторингу та інтерфейсу, логіку перемикання фаз роботи і відпочинку, а також механізм синхронізації через точки FORK та JOIN.

Реалізовано та детально описано основні модулі системи сервіс блокування процесів BlockApp із підтримкою режимів BD та NoBD, логіку таймерів для кожного з чотирьох методів фокусування - Pomodoro, FlowTime, метод 52/17 та механізмами перемикання фаз і збереження статистики у MongoDB. Наведено лістинги ключових методів із поясненням їх функціонального призначення.

		Хомніук В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				
						46

РОЗДІЛ 3. ІНСТРУКЦІЯ КОРИСТУВАЧА ТА РОЗГОРТАННЯ СИСТЕМИ. РЕЗУЛЬТАТИ ТЕСТУВАННЯ

3.1. Порядок встановлення та налаштування параметрів платформи

Для коректної роботи застосунку контролю активності та організації роботи користувача необхідно виконати ряд підготовчих кроків, пов'язаних із встановленням залежностей та налаштуванням середовища. Правильне розгортання системи є запорукою стабільної роботи всіх функціональних модулів — блокування процесів, таймерів фокусування, збереження нотаток та статистики продуктивності. Процес встановлення не потребує спеціальних технічних знань і складається з послідовної інсталяції кількох компонентів, після чого застосунок готовий до використання без додаткового налаштування. Усі необхідні залежності є безкоштовними та доступні для завантаження з офіційних джерел, а сам застосунок не потребує підключення до мережі Інтернет у стандартному режимі роботи.

Перед запуском застосунку на комп'ютері має бути встановлено такі компоненти. Кожен із них є обов'язковим для забезпечення повноцінної роботи всіх функціональних модулів системи - від підключення до бази даних до коректного відображення графічного інтерфейсу та виконання операцій блокування процесів на рівні операційної системи. Відсутність будь-якого з перерахованих компонентів призведе до некоректної роботи застосунку або унеможливить його запуск. Тому перед першим використанням рекомендується послідовно встановити всі залежності у порядку, описаному нижче, та перевірити їх коректну роботу.

.NET 6 Runtime або вище середовище виконання для запуску застосунків на платформі .NET [10, 21]. Завантажується з офіційного сайту Microsoft [9]. На сторінці завантаження необхідно обрати розділ .NET 6.0 та завантажити пакет .NET Desktop Runtime, оскільки він містить компоненти WPF [21], необхідні для запуску настільного застосунку. Версія «ASP.NET Core Runtime» не підходить, оскільки не включає WPF-компоненти. Після завантаження

		Хомнюк В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				
						47

запустити інсталятор від імені адміністратора та дотримуватись інструкцій майстра встановлення. Перевірити успішність встановлення можна через командний рядок команда `dotnet --version` має повернути версію 6.0 або вище.

MongoDB Community Server [25, 27] локальний екземпляр бази даних, що є обов'язковим компонентом для роботи застосунку, оскільки всі дані користувача зберігаються виключно локально без використання хмарних сервісів. На сторінці завантаження необхідно обрати платформу «Windows», формат пакету «MSI» та актуальну стабільну версію із позначкою «Current Release». Формат MSI є пріоритетним порівняно з ZIP-архівом, оскільки містить майстер встановлення з автоматичним налаштуванням служби Windows.

Після входу в головне меню користувач може сформувати персональний список програм для блокування. Для цього необхідно перейти до відповідного розділу меню з відображенням переліку запущених у системі процесів. Відмічені процеси зберігаються у колекції `BlockUserGame` та автоматично завантажуються за кожного наступного запуску сесії фокусування.

Google Chrome або інший сучасний браузер необхідний для проходження авторизації через Google OAuth [8] під час першого запуску застосунку. Під час авторизації застосунок відкриває системний браузер зі сторінкою підтвердження доступу Google, де користувач надає дозвіл на читання базових даних профілю. Після завершення авторизації браузер більше не потрібен, усі подальші операції виконуються локально без мережевої взаємодії. За відсутності планів на авторизацію через Google та введення імені вручну наявність браузера не є обов'язковою.

Активне підключення до мережі Інтернет необхідне виключно для першого входу через Google OAuth [8]. Під час первинної авторизації застосунок відкриває системний браузер зі сторінкою підтвердження доступу Google, після чого отримує ім'я та email користувача через Google People API для автоматичного заповнення профілю. Після успішної авторизації токен зберігається локально у файловій системі через механізм `FileDataStore`, що забезпечує автоматичне оновлення сесії при наступних запусках без повторного

		Хомнюк В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				48

проходження процедури входу. Починаючи з другого запуску застосунок повністю не потребує мережевого з'єднання - усі основні функції, зокрема блокування процесів [15, 17], режими фокусування [1, 2], збереження нотаток та перегляд статистики продуктивності, працюють повністю офлайн із локальною базою даних MongoDB [27].

Google Chrome або інший сучасний браузер необхідний для проходження авторизації через Google OAuth при першому запуску застосунку. Під час авторизації застосунок автоматично відкриває системний браузер за замовчуванням зі сторінкою підтвердження доступу Google, де користувач обирає акаунт та надає дозвіл на читання базових даних профілю - імені та електронної пошти. Після підтвердження браузер можна закрити, оскільки застосунок отримує необхідні дані у фоновому режимі через локальний HTTP-сервер та зберігає токен авторизації на диску. Усі подальші операції виконуються локально без будь-якої мережевої взаємодії. Якщо користувач не планує використовувати авторизацію через Google і вводитиме ім'я та нікнейм вручну при першому запуску - наявність браузера не є обов'язковою умовою для роботи застосунку.

Активне підключення до мережі Інтернет необхідне виключно для першого входу через Google OAuth. Під час первинної авторизації застосунок відкриває системний браузер зі сторінкою підтвердження доступу Google, де користувач надає дозвіл на читання базових даних профілю — імені та email. Після успішного завершення авторизації отриманий токен зберігається локально у файловій системі через механізм FileDataStore бібліотеки Google.Apis, що забезпечує автоматичне оновлення токена при наступних запусках без повторного проходження повного циклу авторизації. Починаючи з другого запуску застосунок не потребує мережевого з'єднання — усі основні функції, зокрема блокування процесів, режими фокусування, збереження нотаток та перегляд статистики, працюють повністю офлайн із локальною базою даних MongoDB.

		Хомніук В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				
						49

3.2. Структура інтерфейсу системи для управління спільнотами та подіями

Під час проєктування інтерфейсу головним критерієм була простота: користувач відкриває застосунок і одразу розуміє що робити далі. Усі модулі - блокування процесів, режими фокусування, нотатки та профіль - винесено на головний екран без зайвих вкладених меню. Темна кольорова схема обрана не як стилістичне рішення, а як практична необхідність - вона не відволікає і не втомлює очі під час тривалої роботи.

При першому запуску відображається вікно авторизації, яке містить кнопку входу через Google OAuth для автоматичного заповнення профілю, а також поле для введення імені вручну як альтернативний варіант (рис. 3.1).

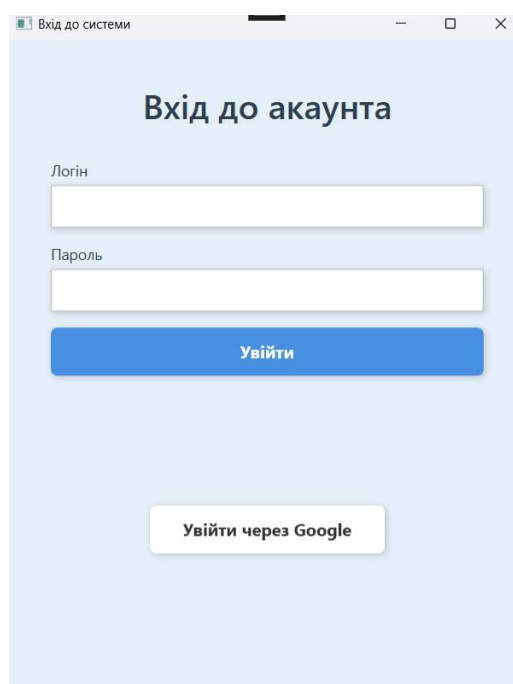


Рис.3.1. Реєстрація користувача

Першим етапом взаємодії користувача з програмним комплексом продуктивності та тайм-менеджменту є проходження процедури автентифікації. Для забезпечення безпеки персональних даних, збереження індивідуальних налаштувань фокусування, нотаток, доступ до головного функціонала системи надається виключно зареєстрованим користувачам через спеціалізоване вікно входу.

		Хомнюк В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				
						50

Графічний інтерфейс цього вікна розроблено на базі платформи WPF (Windows Presentation Foundation) із впровадженням сучасних стандартів проєктування UI/UX. Також користувач має змогу зареєструватись через свій гугл акаунт (рис. 3.2).

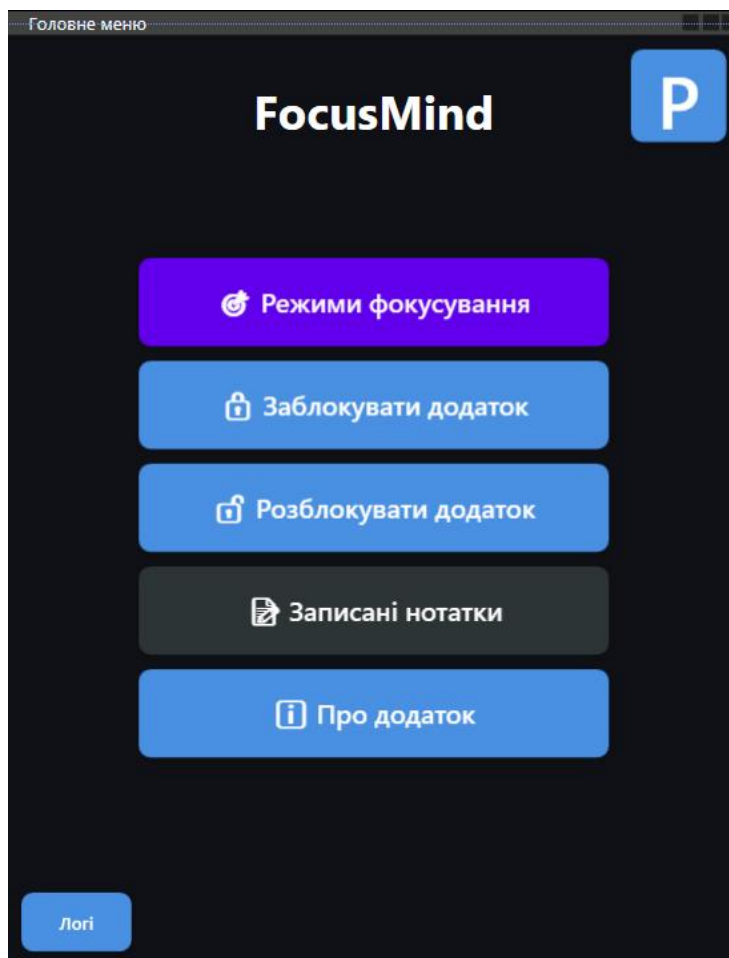


Рис. 3.2. Головне меню

При активації відповідного керуючого елемента в головному меню системи, додаток ініціює відкриття екранної форми Профіль користувача. Цей інтерфейсний модуль призначений для агрегації персональних даних користувача, відображення поточної інформації про його статус, а також наочного представлення аналітичної статистики щодо використання різних технік фокусування часу.

Вікно профілю зберігає загальну стилістику додатка та слугує основним компонентом для відображення персональних даних авторизованого користувача. У цьому вікні виводиться збережене в базі даних ім'я користувача, а також його текстовий опис, який містить додаткову інформацію про

		Хомніук В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				51

обліковий запис або поточний статус користувача в системі. Зазначені дані динамічно підвантажуються з моделі даних і відображаються у відповідних текстових полях інтерфейсу, що забезпечує швидкий доступ до актуальної інформації про поточний профіль користувача (рис. 3.3).

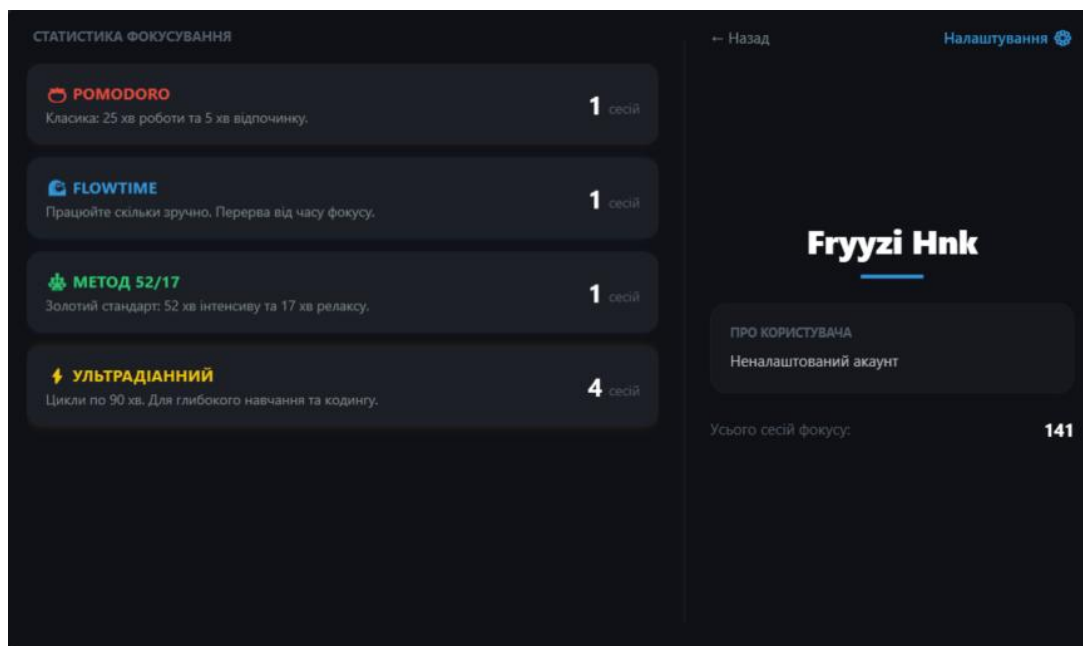


Рис. 3.3. Профіль користувача.

При виборі опції Налаштування у вікні персонального профілю, додаток відкриває екранну форму Налаштування профілю. Цей інтерфейсний модуль надає користувачеві інструментарій для модифікації персональних ідентифікаторів, редагування публічної інформації про поточний вид діяльності, а також для попереднього перегляду оновленої картки акаунта перед її збереженням у базі даних MongoDB.

Вікно налаштувань спроектовано із збереженням загальної темної стилістики додатку (Dark Mode) та має чіткий розподіл на дві функціональні зони: ліва панель призначена для безпосереднього введення й конфігурації даних, тоді як права виступає інтерактивною зоною моніторингу та збереження (рис. 3.4).

		Хомнюк В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				52

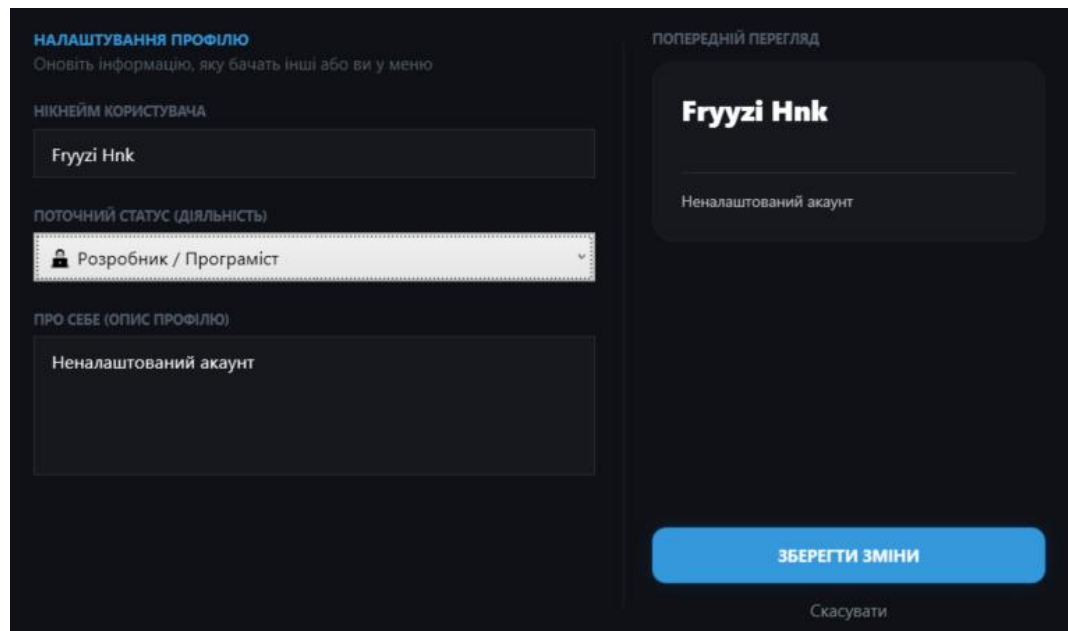


Рис. 3.4. Налаштування профілю

Наведено екранну форму вибору режиму роботи застосунку. Інтерфейс виконано у темній кольоровій схемі та містить чотири картки з доступними методами фокусування.

Картка Pomodoro описує класичну техніку чергування 25 хвилин роботи та 5 хвилин відпочинку з рекомендацією для рутинних завдань. Запуск здійснюється кнопкою Запустити (25/5).

Картка Flowtime пропонує адаптивний режим без фіксованого ліміту, де тривалість перерви розраховується автоматично від часу фокусування. Перехід до режиму відбувається через кнопку «Увійти в потік».

Картка Метод 52/17 описує золотий стандарт продуктивності - 52 хвилини інтенсивної роботи та 17 хвилин повного релаксу. Активується кнопкою «Почати цикл».

Картка Ультрадіанний орієнтована на глибоке навчання та складне програмування з робочими циклами по 90 хвилин (рис. 3.5).

		Хомнюк В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				
						53

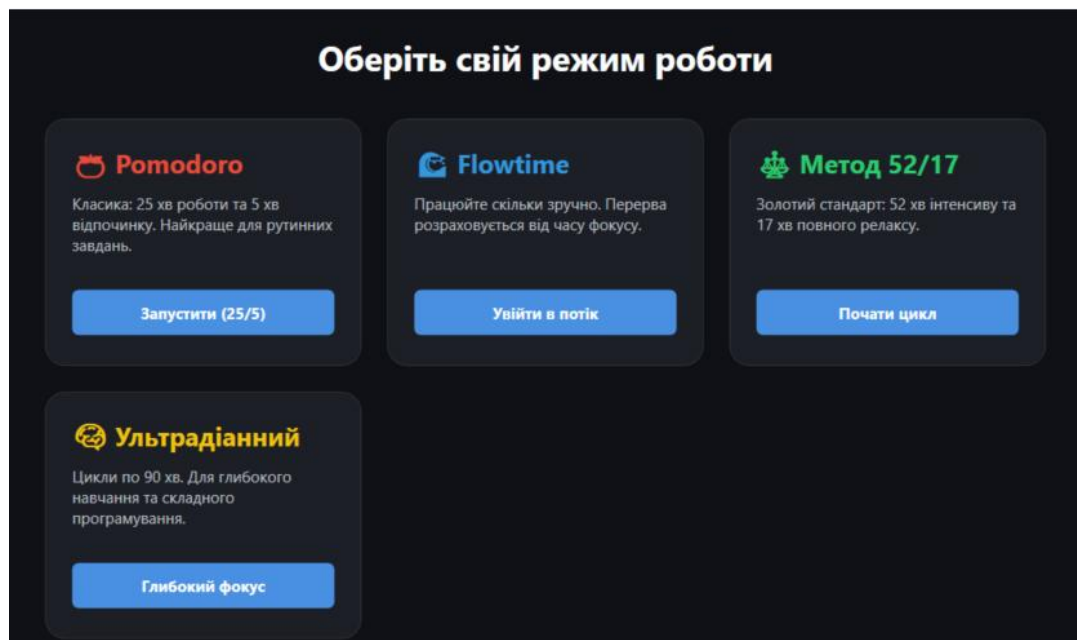


Рис.3.5. Режими роботи

Режим Pomodoro [1]. У верхній частині відображається статус поточної фази - Час фокусуватися!, виділений червоним кольором для привернення уваги. Центральну частину займає великий цифровий таймер із початковим значенням 25 хвилин. Нижче розташовано дві кнопки керування зелена кнопка «Старт» для запуску зворотного відліку та сіра кнопка Скидання для повернення таймера до початкового значення. У нижній частині вікна відображається лічильник завершених циклів, який на момент старту має значення 0 та збільшується після кожного успішно завершеного робочого інтервалу (рис. 3.6).

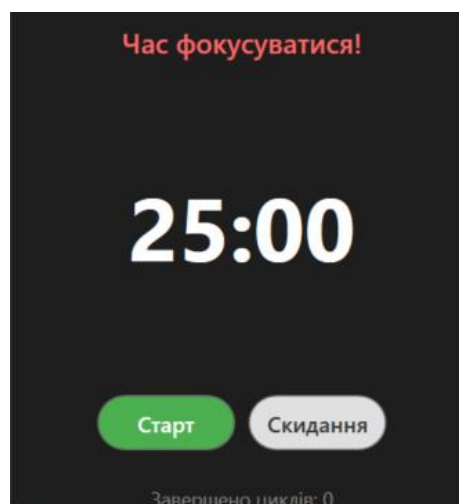


Рис. 3.6. Pomodoro таймер

		Хомнюк В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				
						54

Режим FlowTime [2, 30]. У верхній частині відображається назва режиму WAVE FLOW. Центральну частину займає кругова індикаторна шкала із цифровим таймером у форматі hh:mm:ss та підписом ЧАС У ПОТОЦІ що відраховує тривалість робочої сесії у прямому напрямку. Керування здійснюється двома круглими кнопками - синьою для запуску та червоною для зупинки сесії.

У нижній частині вікна розміщено довідкову таблицю рекомендованих перерв залежно від тривалості робочої сесії до 25 хвилин - перерва 5 хвилин, від 25 до 50 хвилин - перерва 8 хвилин, від 50 до 90 хвилин - перерва від 10 до 15 хвилин, понад 90 хвилин - перерва 20 хвилин. Ця підказка допомагає користувачу самостійно визначити оптимальну тривалість відпочинку після завершення робочого інтервалу (рис. 3.7).

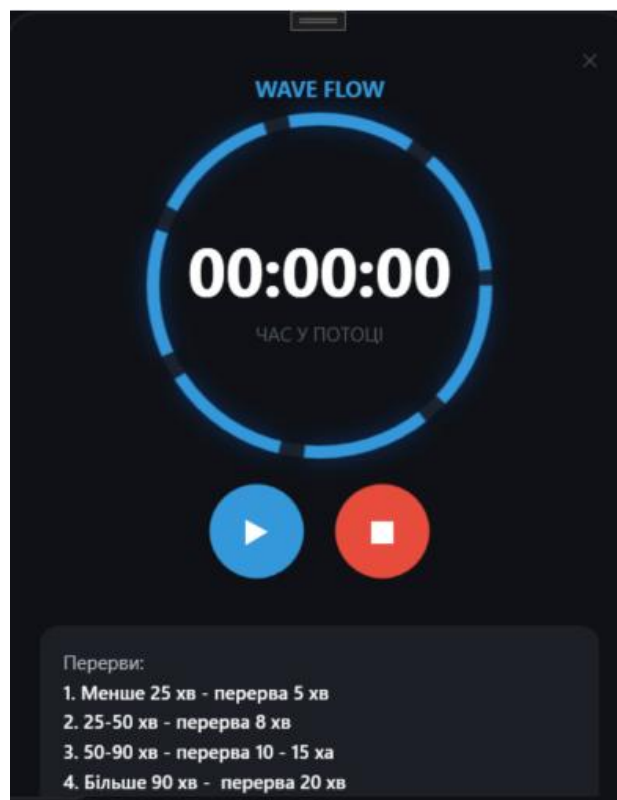


Рис. 3.7. Wave Flow таймер

Режим 54/17 [4]. У верхній частині відображається назва режиму. Центральну частину займає кругова індикаторна шкала фіолетового кольору із цифровим таймером у форматі hh:mm:ss із початковим значенням 5 хвилин та

підписом WAVE FLOW. Керування здійснюється двома круглими кнопками - фіолетовою для запуску та сірою для скидання таймера.

У нижній частині вікна розміщено інформаційну панель ПОТОЧНИЙ ЦИКЛ, яка відображає параметри активного циклу тривалість робочого інтервалу - 54 хвилини та тривалість перерви - 17 хвилин, виділену фіолетовим кольором для візуального розмежування між фазами. Це дозволяє користувачу у будь-який момент бачити структуру поточного циклу без необхідності звертатися до налаштувань (рис. 3.8).

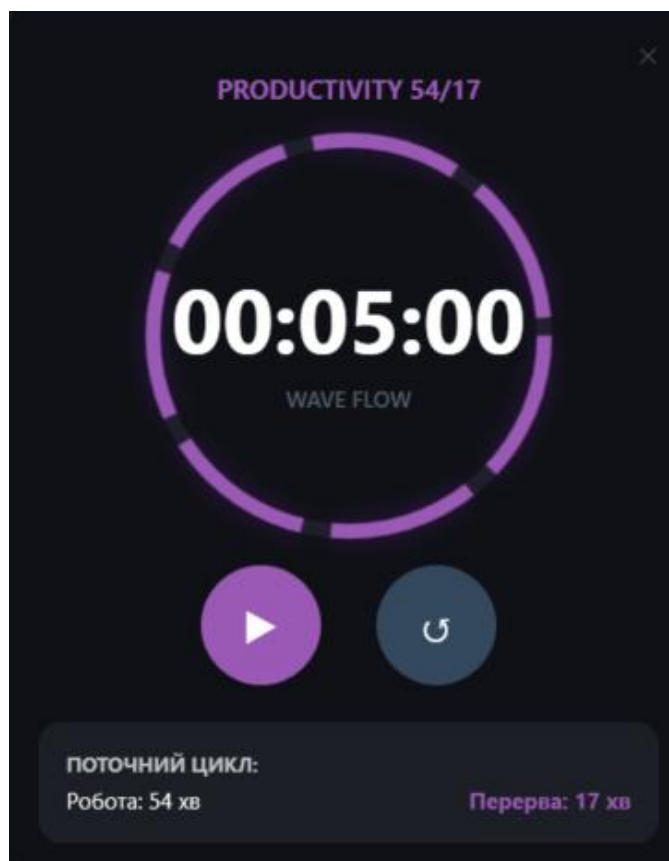


Рис. 3.8. Режим 54/17

Режиму Ultra Focus [28]. У верхній частині відображається назва режиму ULTRA FOCUS MODE червоним кольором. Центральну частину займає кругова індикаторна шкала з червоним світінням та цифровим таймером із початковим значенням 30 хвилин і підписом DEEP WORK, що підкреслює призначення режиму для глибокої зосередженої роботи. Керування здійснюється двома круглими кнопками - червоною для запуску та сірою для зупинки сесії.

У нижній частині вікна розміщено статусний рядок із червоною індикаторною точкою та написом SYSTEMS LOCKED - FOCUS REQUIRED, який сигналізує користувачу про те, що блокування процесів активоване і система перебуває у режимі повного контролю. Загальне візуальне оформлення у червоних тонах із ефектом світіння підсилює відчуття підвищеної концентрації та невідворотності обмежень (рис. 3.9).

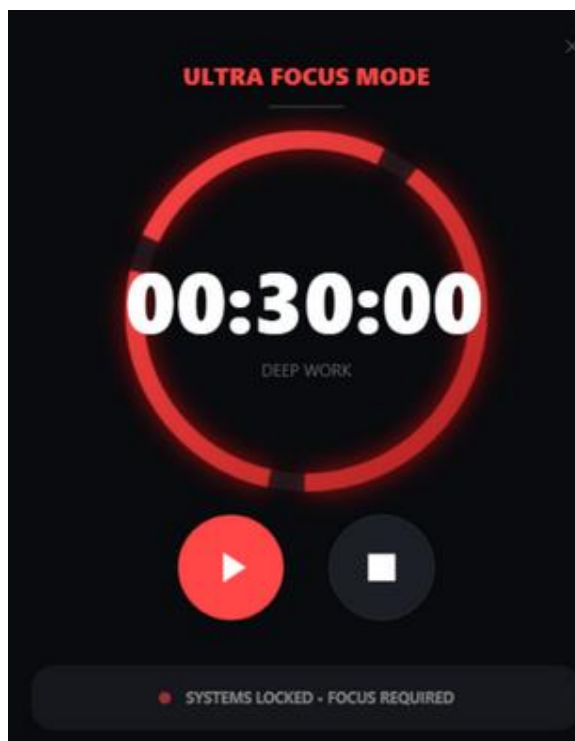


Рис. 3.9. Ultra Focus Mode таймер

Нижче наведений екран форми вікна блокування гри. Форма містить три поля для налаштування параметрів блокування поле для введення імені виконуваного файлу програми, що підлягає блокуванню, поле :назва гри для зручної ідентифікації застосунку користувачем, та випадаючий список день для вибору дня тижня, в який діятиме обмеження. Прапорець встановити час дозволяє додатково задати часовий діапазон блокування, розширюючи гнучкість налаштування.

У нижній частині вікна розміщено кнопку steam для швидкого автоматичного заповнення назви процесу без ручного введення. Навігація здійснюється через кнопки «Назад» для повернення до попереднього екрану та застосувати для збереження налаштувань у базі даних MongoDB (рис. 3.10).

		Хомніук В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				
						57

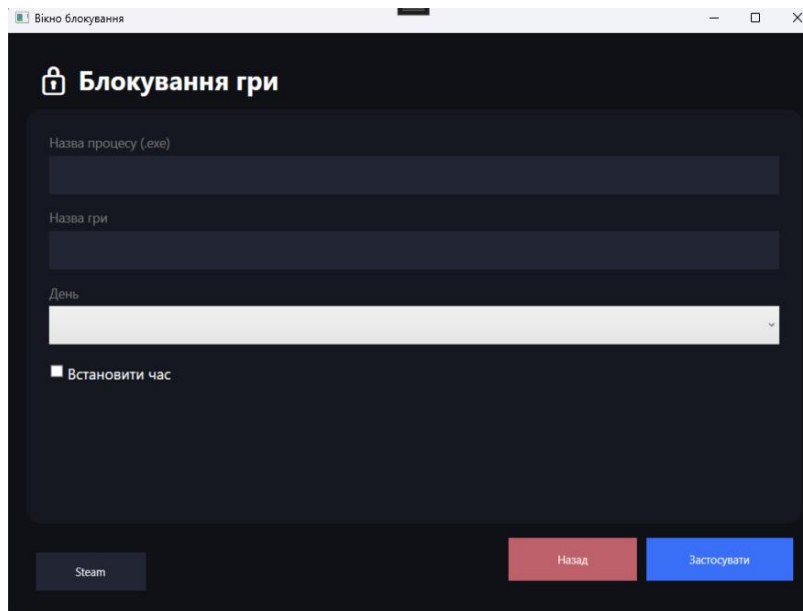


Рис. 3.10. Блокувальник

У даному вікні реалізовано функціонал швидкого блокування ігор із бібліотеки Steam [26]. Натискання кнопки Steam автоматично звертається до локально встановленого клієнта Steam та отримує перелік ігор користувача, що усуває необхідність вручну вводити назву виконуваного файлу. Після вибору гри зі списку поля назва процесу (.exe) та назва гри заповнюються автоматично.

Додатково користувач може налаштувати розклад блокування -о брати конкретний день тижня через випадаючий список та встановити часовий діапазон через прапорець «Встановити час». Це дозволяє, наприклад, заблокувати запуск ігор у робочі дні в години, відведені для навчання чи роботи, залишаючи доступ у вільний час (рис. 3.11).

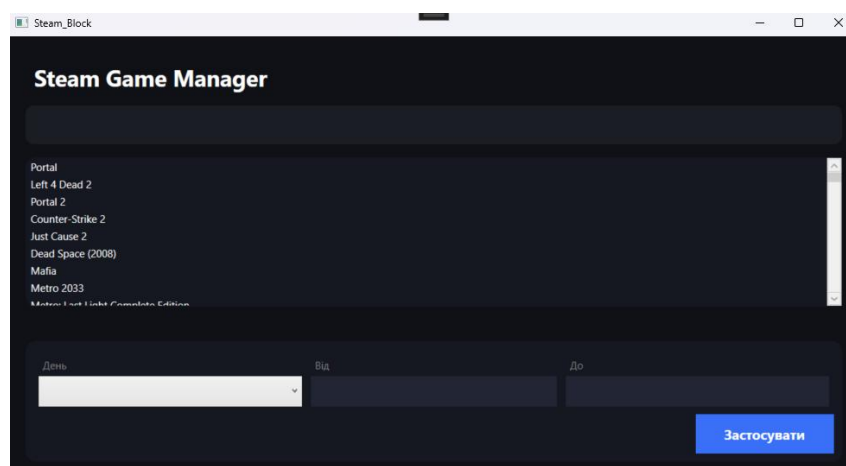


Рис. 3.11. Блокувальник steam

		Хомніук В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				58

Для забезпечення гнучкості конфігурування захисного середовища та надання користувачеві можливості оперативного редагування поточної матриці обмежень, в архітектуру FocusMind інтегровано спеціалізований керуючий модуль розблокування програм (рис. 3.12). Ця екранна форма ініціюється при активації відповідного пункту «Розблокувати додаток» у головному навігаційному хабі програми.

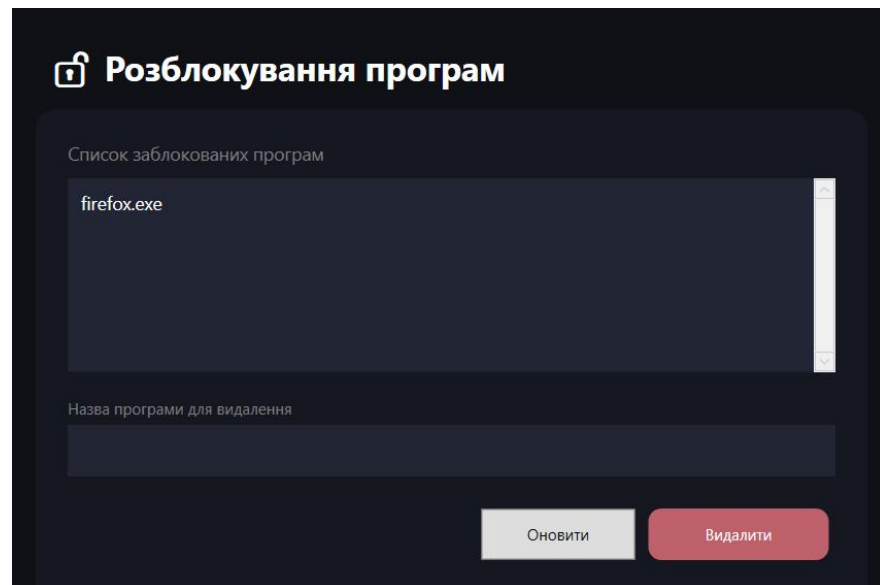


Рис. 3.12. Розблокування програми

Висновки до третього розділу

У третьому розділі розглянуто практичний етап реалізації та тестування застосунку контролю активності та організації роботи користувача. Описано порядок встановлення і налаштування системи, визначено мінімальні вимоги до апаратного та програмного забезпечення, а також детально розглянуто процес першого запуску з автоматичною перевіркою наявності збережених даних акаунту.

Наведено екранні форми всіх реалізованих модулів застосунку з детальним описом їх функціонального призначення та елементів керування. Розглянуто інтерфейс вибору режиму роботи, що об'єднує чотири методи фокусування - Pomodoro, FlowTime, метод 52/17 та ультрадіанний ритм - кожен з яких має власне візуальне оформлення та структуру керування. Описано вікно блокування ігор із підтримкою автоматичного отримання переліку ігор із

		Хомніук В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				
						59

бібліотеки Steam та гнучким налаштуванням розкладу блокування за днем тижня і часовим діапазоном.

Наведено та проаналізовано лістинги ключових фрагментів коду сервісу моніторингу процесів BlockApp, логіки таймерів для кожного режиму фокусування, механізму авторизації через Google People API та алгоритму створення профілю користувача у MongoDB. Реалізований функціонал повністю відповідає вимогам, сформульованим у першому розділі.

		Хомніук В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				60

ВИСНОВКИ

У процесі виконання бакалаврської кваліфікаційної роботи вирішено комплекс теоретичних та практичних задач, результатом яких стало створення повнофункціонального настільного застосунку контролю активності та організації роботи користувача для операційної системи Windows.

У першому розділі проведено детальний аналіз предметної області, досліджено проблематику цифрових відволікань та їх вплив на продуктивність користувачів персональних комп'ютерів. Виконано порівняльний аналіз існуючих програмних рішень, за результатами якого виявлено ключовий недолік наявних продуктів - жодне з них не поєднує кілька науково обґрунтованих методик фокусування з активним блокуванням процесів на рівні операційної системи у єдиному локальному рішенні. Обґрунтовано вибір технологічного стеку та архітектурного підходу.

У другому розділі спроектовано архітектуру застосунку на основі тришарової моделі, розроблено структуру локальної бази даних MongoDB із трьома колекціями, побудовано діаграму варіантів використання та блок-схему алгоритму роботи системи. Реалізовано та детально описано ключові модулі сервіс блокування процесів BlockApp із підтримкою режимів BD та NoBD, логіку таймерів чотирьох методів фокусування, інтеграцію зі Steam Web API та механізм авторизації через Google OAuth.

У третьому розділі описано порядок встановлення та налаштування системи, наведено екранні форми всіх реалізованих модулів із детальним описом їх функціонального призначення та підтверджено коректність роботи застосунку.

Практична цінність розробленого застосунку полягає у наданні користувачам інструменту примусового обмеження доступу до відволікаючих програм через Win32 API. Поєднання трьох методик керування часом, гнучкого налаштування розкладу та локального зберігання статистики робить цей застосунок ефективним рішенням для підвищення продуктивності.

		Хомніук В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				
						61

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Cirillo F. The Pomodoro Technique : official website. URL: <https://www.pomodorotechnique.com/>
2. Csikszentmihalyi M. Flow: The Psychology of Optimal Experience. Research Overview. URL: <https://www.psychologytoday.com/us/basics/flow>
3. Cold Turkey Blocker - The toughest website blocker on the internet : official website. URL: <https://getcoldturkey.com/>
4. DeskTime. The 52/17 Rule: How the Most Productive People Work. URL: <https://deskttime.com/blog/52-17-updated/>
5. Fowler M. MonolithFirst. Guide to Software Architecture Patterns. URL: <https://martinfowler.com/bliki/MonolithFirst.html>
6. Freedom – Block Distractions on iOS, Android, Mac, and Windows : official website. URL: <https://freedom.to/>
7. Git - Distributed is the New Centralized: official version control website. URL: <https://git-scm.com/>
8. Google. OAuth 2.0 for Desktop Apps. Google Identity Platform. URL: <https://developers.google.com/identity>
9. Google. People API Overview. Google Developers. URL: <https://developers.google.com/people>
10. Microsoft. C# documentation. URL: <https://learn.microsoft.com/uk-ua/dotnet/csharp/tour-of-csharp/>
11. Microsoft. CreateToolhelp32Snapshot function (tlhelp32.h). Windows API. URL: <https://learn.microsoft.com/en-us/windows/win32/api/tlhelp32/nf-tlhelp32-createtoolhelp32snapshot>
12. Microsoft. App architecture guide for Windows applications. Microsoft Learn. URL: <https://learn.microsoft.com/en-us/windows/apps/>
13. Microsoft. Model-View-ViewModel (MVVM) Pattern Overview. Microsoft Learn. URL: <https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm>

		Хомніук В.О.			ІІЗ.КР.Б - 121 - 26 - ІІЗ	Арк.
		Савіцький Р.С.				62

14. Microsoft. Native Interoperability (P/Invoke) in .NET. Microsoft Learn.
URL: <https://learn.microsoft.com/en-us/dotnet/standard/native-interop/pinvoke>
15. Microsoft. Process and Thread Functions. Microsoft Learn. URL:
<https://learn.microsoft.com/en-us/windows/win32/procthread/process-and-thread-functions>
16. Microsoft. CancellationToken Struct. .NET API Documentation. URL:
<https://learn.microsoft.com/en-us/dotnet/api/system.threading.cancellationtoken>
17. Microsoft. System.Diagnostics.Process. URL:
<https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics.process>
18. Microsoft. Task Parallel Library (TPL). Microsoft Learn. URL:
<https://learn.microsoft.com/en-us/dotnet/standard/parallel-programming/task-parallel-library-tpl>
19. Microsoft. Threading in C#. Microsoft Learn. URL:
<https://learn.microsoft.com/en-us/dotnet/standard/threading/>
20. Microsoft. Visual Studio documentation. URL:
<https://docs.microsoft.com/en-us/visualstudio/>
21. Microsoft. Windows Presentation Foundation documentation. URL:
<https://docs.microsoft.com/en-us/dotnet/desktop/wpf/>
22. MongoDB, Inc. BSON Types. URL:
<https://www.mongodb.com/docs/manual/reference/bson-types/>
23. MongoDB, Inc. Data Modeling Introduction. MongoDB Documentation.
URL: <https://www.mongodb.com/docs/manual/core/data-modeling-introduction/>
24. MongoDB, Inc. MongoDB C# Driver documentation. URL:
<https://www.mongodb.com/docs/drivers/csharp/>
25. MongoDB, Inc. MongoDB Manual. Introduction to MongoDB. URL:
<https://www.mongodb.com/docs/manual/introduction/>
26. Steam. Steamworks Web API Reference. Valve Developer Community.
URL: https://partner.steamgames.com/doc/webapi_overview
27. MongoDB, Inc. MongoDB Manual. URL:
<https://www.mongodb.com/docs/manual/>

		Хомніук В.О.			ІІЗ.КР.Б - 121 - 26 - ІІЗ	Арк.
		Савіцький Р.С.				63

28. ScienceDirect. Thermoregulation and the ultradian basic rest-activity cycle. Handbook of Clinical Neurology, 156, 321–335. URL: <https://www.sciencedirect.com/science/chapter/handbook/abs/pii/B9780444639127000229>
29. Single Responsibility Principle (SRP) in C#. .NET Design Patterns Guide. URL: <https://www.mono-project.com/docs/advanced/pinvoke/>
30. Timing App. The Flowtime Technique: A Guide to the Flexible Pomodoro Alternative. URL: <https://timingapp.com/blog/flowtime-technique/>

		Хомніук В.О.			ІПЗ.КР.Б - 121 - 26 - ІЗ	Арк.
		Савіцький Р.С.				
						64

ДОДАТКИ

		Хомніук В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				
						65

Фрагменти коду

```

using DnsClient;
using BLockGame.Model;
using Google.Apis.Auth.OAuth2;
using Google.Apis.Auth.OAuth2.Flows;
using Google.Apis.Auth.OAuth2.Requests;
using Google.Apis.PeopleService.v1;
using Google.Apis.PeopleService.v1.Data;
using Google.Apis.Services;
using Microsoft.VisualBasic.ApplicationServices;
using MongoDB.Bson;
using MongoDB.Driver;
using System;
using System.Threading;
using System.Windows;
using System.Windows.Input;
using System.Xml.Linq;
using static BLockGame.Model_database;
using static System.Windows.Forms.VisualStyles.VisualStyleElement.ListView;
using WinFormsMessageBox = System.Windows.Forms.MessageBox;
using WpfMessageBox = System.Windows.MessageBox;
namespace BLockGame
{
    public partial class MainWindow : Window
    {
        private static IMongoCollection<BsonDocument> _collection;
        private static IMongoCollection<ProfileModel> _collectionProfile;
        private Main_Manu main_Manu;
        public MainWindow()
        {
            InitializeComponent();
            var client = new MongoClient("mongodb://localhost:27017/");
            var database = client.GetDatabase("BLockGame");
            _collection = database.GetCollection<BsonDocument>("User");
            _collectionProfile = database.GetCollection<ProfileModel>("Profile");
        }
        private void LoginButton_Click(object sender, RoutedEventArgs e)
        {
            var enterLogin = MyLoginBox.Text.Trim();
            var enterPassword = MyPasswordBox.Password;
            if (string.IsNullOrEmpty(enterLogin) || string.IsNullOrEmpty(enterPassword))
            {
                WpfMessageBox.Show("Будь ласка, заповніть усі поля!");
                return;
            }
            var filter = Builders<BsonDocument>.Filter.Eq("Login", enterLogin);
            var user = _collection.Find(filter).FirstOrDefault();
            if (user != null)
            {
                string dbPassword = user["Password"].ToString();
                if (enterPassword == dbPassword)
                {
                    Base_User.IsAutotization = true;
                    Base_User.User = enterLogin;
                    main_Manu = new Main_Manu();
                    main_Manu.Show();
                    this.Close();
                }
            }
            else
            {
                WpfMessageBox.Show("Невірний пароль! Повторіть спробу.");
            }
        }
    }
}

```

```

    }
    else
    {
        WpfMessageBox.Show("Користувача з таким логіном не знайдено!");
    }
}

private void Register_Click(object sender,
System.Windows.Navigation.RequestNavigateEventArgs e)
{
    Register registerWindow = new Register();
    registerWindow.Show();
}

private async void Button_Click_1(object sender, RoutedEventArgs e)
{
    try
    {
        var secret = new Secret();
        var clientSecrets = new ClientSecrets
        {
            ClientId = secret.ClientId,
            ClientSecret = secret.ClientSecret
        };
        var flow = new GoogleAuthorizationCodeFlow(new
GoogleAuthorizationCodeFlow.Initializer
        {
            ClientSecrets = clientSecrets,
            Scopes = new[] { "https://www.googleapis.com/auth/userinfo.email",
"https://www.googleapis.com/auth/userinfo.profile" }
        });
        var codeReceiver = new LocalServerCodeReceiver();
        var credential = await new AuthorizationCodeInstalledApp(flow, codeReceiver)
.AuthorizeAsync("user", Cancellation.Token.None);
        var peopleService = new PeopleServiceService(new BaseClientService.Initializer
        {
            HttpClientInitializer = credential,
            ApplicationName = "GoogleAuthWpfApp"
        });
        var request = peopleService.People.Get("people/me");
        request.PersonFields = "names,emailAddresses";
        Person profile = await request.ExecuteAsync();
        string name = profile.Names?[0]?.DisplayName ?? "User";
        string email = profile.EmailAddresses?[0]?.Value ?? string.Empty;
        if (string.IsNullOrEmpty(email))
        {
            WpfMessageBox.Show("Не вдалося отримати електронну пошту від Google.");
            return;
        }
        var bsonEmailFilter = Builders<BsonDocument>.Filter.Eq("Email", email);
        var profileEmailFilter = Builders<ProfileModel>.Filter.Eq(p => p.Email, email);
        var existingUser = _collection.Find(bsonEmailFilter).FirstOrDefault();
        var existingProfile = _collectionProfile.Find(profileEmailFilter).FirstOrDefault();
        main_Manu = new Main_Manu();
        if (existingUser != null)
        {
            Base_User.IsAutotization = true;
            Base_User.User = existingUser.Contains("Login") ? existingUser["Login"].ToString() :
name;
            main_Manu.Show();
            this.Close();
        }
        else
        {
            var newUser = new BsonDocument
            {
                { "Login", name },
            }
        }
    }
}

```

		Хомнюк В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				67

```

        { "Email", email }
    };
    var newProfile = new ProfileModel
    {
        Login = name,
        RegisterNickName = name,
        Email = email,
        UserDescription = "Неналаштований акаунт",
        UltraFocus = 0,
        Pomodoro = 0,
        Flowtime = 0,
        Productivity_Formula = 0
    };
    _collectionProfile.InsertOne(newProfile);
    _collection.InsertOne(newUser);
    Base_User.IsAutotization = true;
    Base_User.User = name;
    main_Manu.Show();
    this.Close();
    }
    }
    catch (Exception ex)
    {
        WpfMessageBox.Show($"Помилка авторизації: {ex.Message}", "Помилка");
    }
    }
}

using System;
using System.Threading;
using System.Windows;
using System.Text.RegularExpressions;
using System.Net.Http;
using System.Net;
using WpfMessageBox = System.Windows.MessageBox;
using Google.Apis.Auth.OAuth2;
using Google.Apis.Auth.OAuth2.Flows;
using Google.Apis.Auth.OAuth2.Requests;
using Google.Apis.PeopleService.v1;
using Google.Apis.PeopleService.v1.Data;
using Google.Apis.Services;
using MongoDB.Bson;
using MongoDB.Driver;
namespace BlockGame
{
    public partial class Register : Window
    {
        private static IMongoCollection<BsonDocument> _collection;
        public Register()
        {
            InitializeComponent();
            var client = new MongoClient("mongodb://localhost:27017/");
            var database = client.GetDatabase("BBlockGame");
            _collection = database.GetCollection<BsonDocument>("User");
        }
        private void Button_Click(object sender, RoutedEventArgs e)
        {
            var Gmail = GmailTextBox.Text;
            var Login = LoginTextBox.Text;
            var Password = PasswordTextBox.Password;
            if (string.IsNullOrEmpty(Gmail))
            {
                WpfMessageBox.Show("Будь ласка введіть пошту!");
            }
            if (string.IsNullOrEmpty(Login))
            {

```

```

WpfMessageBox.Show("Введіть будь ласка свій логін");
}
if (string.IsNullOrEmpty>Password))
{
WpfMessageBox.Show("Введіть будь ласка свій пароль");
}
else
{
try
{
if (IsValidEmail(Gmail))
{
var document = new BsonDocument
{
{"Gmail", Gmail},
{"Login", Login},
{"Password", Password}
};
_collection.InsertOne(document);
MainWindow mainWindow = new MainWindow();
mainWindow.Show();
this.Close();
}
else
{
WpfMessageBox.Show("Введена пошта не коректна! Введіть будь ласка коректну пошту
(Приклад test@example.com)");
}
}
catch
{
WpfMessageBox.Show("Помилка не вдалось створити аккаунт! Повторіть спробу пізніше");
}
}
static bool IsValidEmail(string email)
{
string pattern = @"^[^@\s]+@[^@\s]+\.[^@\s]+$";
Regex regex = new Regex(pattern);
return regex.IsMatch(email);
}
private async void Button_Click_1(object sender, RoutedEventArgs e)
{
try
{
var secret = new Secret();
var clientSecrets = new ClientSecrets
{
ClientId = secret.ClientId,
ClientSecret = secret.ClientSecret
};
var flow = new GoogleAuthorizationCodeFlow(new
GoogleAuthorizationCodeFlow.Initializer
{
ClientSecrets = clientSecrets,
Scopes = new[] { "https://www.googleapis.com/auth/userinfo.email",
"https://www.googleapis.com/auth/userinfo.profile" }
});
var codeReceiver = new LocalServerCodeReceiver();
var credential = await new AuthorizationCodeInstalledApp(flow, codeReceiver)
.AuthorizeAsync("user", Cancellation.Token.None);
WpfMessageBox.Show($"Access Token: {credential.Token.AccessToken}", "Успішна
авторизація");
var peopleService = new PeopleServiceService(new BaseClientService.Initializer
{

```

```

HttpClientInitializer = credential,
ApplicationName = "GoogleAuthWpfApp"
});
var request = peopleService.People.Get("people/me");
request.PersonFields = "names,emailAddresses";
Person profile = await request.ExecuteAsync();
string name = profile.Names?[0]?.DisplayName ?? "Ім'я не знайдено";
string email = profile.EmailAddresses?[0]?.Value ?? "Електрону пошту не знайдено";

WpfMessageBox.Show($"Імя {name}\nEmail {email}", "Information in User");
}
catch (Exception ex)
{
WpfMessageBox.Show($"Помилка авторизації: {ex.Message}", "Помилка");
}
}
}
}

using BlockGame.Class;
using System;
using System.Windows;
using System.Windows.Threading;
using WpfMessageBox = System.Windows.MessageBox;
namespace BlockGame.FocusFolder.UltraFocus
{
public partial class UltraFocus : Window
{
private readonly DispatcherTimer timer;
private TimeSpan _timeLeft;
private bool _isFocusMode;
private readonly BlockApp _blockApp;
public UltraFocus()
{
InitializeComponent();
_blockApp = new BlockApp();
_blockApp.StartMonitoring(Base_User.User, 3, "NoBD");
timer = new DispatcherTimer
{
Interval = TimeSpan.FromSeconds(1)
};
timer.Tick += Timer_Tick;
_timeLeft = TimeSpan.FromMinutes(30);
TimerText.Text = _timeLeft.ToString(@"hh\:mm\:ss");
_isFocusMode = true;
}
private void Timer_Tick(object? sender, EventArgs e)
{
if (_timeLeft > TimeSpan.Zero)
{
_timeLeft -= TimeSpan.FromSeconds(1);
TimerText.Text = _timeLeft.ToString(@"hh\:mm\:ss");
return;
}
if (_isFocusMode)
{
AddNumberFocus.UpdateUltraFocus();
_timeLeft = TimeSpan.FromMinutes(2);
_isFocusMode = false;
WpfMessageBox.Show("Час фокусування завершено!", "Ultra Focus", MessageBoxButton.OK,
MessageBoxImage.Information);
}
else
{
_timeLeft = TimeSpan.FromMinutes(30);
}
}
}
}

```

		Хомніук В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				70

```

_isFocusMode = true;
WpfMessageBox.Show("Час фокусування завершено!", "Ultra Focus", MessageBoxButton.OK,
MessageBoxImage.Information);
}
TimerText.Text = _timeLeft.ToString(@"hh\:mm\:ss");
}
private void Start_Click(object sender, RoutedEventArgs e)
{
timer.Start();
}
private void Reset_Click(object sender, RoutedEventArgs e)
{
timer.Stop();
_timeLeft = TimeSpan.FromMinutes(30);
_isFocusMode = true;
TimerText.Text = _timeLeft.ToString(@"hh\:mm\:ss");
}
private void Close_Click(object sender, RoutedEventArgs e)
{
Close();
}
private void Window_Closed(object sender, EventArgs e)
{
timer?.Stop();
}
}
}

using BlockGame.Class;
using System;
using System.Windows;
using System.Windows.Threading;
namespace BlockGame.FocusFolder.Productivity_Formula
{
public partial class Productivity_Formula : Window
{
private readonly DispatcherTimer _timer;
private TimeSpan _timeLeft;
private bool _isFocusMode = true;
private const int FocusMinutes = 5;
private const int BreakMinutes = 17;
public Productivity_Formula()
{
InitializeComponent();
_timer = new DispatcherTimer
{
Interval = TimeSpan.FromSeconds(1)
};
_timer.Tick += Timer_Tick;
StartFocus();
}
private void Timer_Tick(object sender, EventArgs e)
{
if (_timeLeft > TimeSpan.Zero)
{
_timeLeft -= TimeSpan.FromSeconds(1);
TimerText.Text = _timeLeft.ToString(@"hh\:mm\:ss");
return;
}
if (!_isFocusMode)
{
AddNumberFocus.UpdateProductivity_Formula();
_isFocusMode = false;
_timeLeft = TimeSpan.FromMinutes(BreakMinutes);
StatusText.Text = "ПЕРЕРВА";
}
}
}

```

```

    }
    else
    {
        _isFocusMode = true;
        _timeLeft = TimeSpan.FromMinutes(FocusMinutes);
        StatusText.Text = "WAVE FLOW";
    }
    TimerText.Text = _timeLeft.ToString(@"hh\:mm\:ss");
}
private void Start_Click(object sender, RoutedEventArgs e)
{
    _timer.Start();
}
private void Reset_Click(object sender, RoutedEventArgs e)
{
    _timer.Stop();
    StartFocus();
}
private void StartFocus()
{
    _isFocusMode = true;
    _timeLeft = TimeSpan.FromMinutes(FocusMinutes);
    StatusText.Text = "WAVE FLOW";
    TimerText.Text = _timeLeft.ToString(@"hh\:mm\:ss");
}
private void Close_Click(object sender, RoutedEventArgs e)
{
    _timer.Stop();
    Close();
}
}
}

```

```

using BLockGame.Class;
using System;
using System.Windows;
using System.Windows.Media;
using System.Windows.Threading;

```

```

namespace BLockGame.FocusFolder.Pomodoro
{
    public partial class pomodoro : Window
    {
        private readonly DispatcherTimer _timer;
        private TimeSpan _timeRemaining;
        private bool _isRunning;
        private bool _isWorkingSession = true;
        private int _completedCycles;
        private const int WorkTimeMin = 25;
        private const int BreakTimeMin = 5;
        public pomodoro()
        {
            InitializeComponent();
            _timer = new DispatcherTimer
            {
                Interval = TimeSpan.FromSeconds(1)
            };
            _timer.Tick += Timer_Tick;
            ResetTimer();
        }
        private void Timer_Tick(object? sender, EventArgs e)
        {
            if (_timeRemaining > TimeSpan.Zero)
            {
                _timeRemaining -= TimeSpan.FromSeconds(1);
                UpdateTimerDisplay();
            }
        }
    }
}

```

```

return;
}
_timer.Stop();
_isRunning = false;
StartPauseButton.Content = "Старт";
if (_isWorkingSession)
{
    _completedCycles++;
    AddNumberFocus.UpdatePomodoro();
    CycleText.Text = $"Завершено циклів: {_completedCycles}";
    _isWorkingSession = false;
    StatusText.Text = "Час відпочити";
    _timeRemaining = TimeSpan.FromMinutes(BreakTimeMin);
}
else
{
    _isWorkingSession = true;
    StatusText.Text = "Час фокусуватися!";
    _timeRemaining = TimeSpan.FromMinutes(WorkTimeMin);
}
UpdateTimerDisplay();
System.Windows.MessageBox.Show(
    _isWorkingSession
    ? "Перерва завершена. Починаємо працювати!"
    : "Робочий цикл завершено. Час відпочити!",
    "Pomodoro",
    MessageBoxButton.OK,
    MessageBoxImage.Information);
}
private void StartPauseButton_Click(object sender, RoutedEventArgs e)
{
    if (_isRunning)
    {
        _timer.Stop();
        StartPauseButton.Content = "Старт";
    }
    else
    {
        _timer.Start();
        StartPauseButton.Content = "Пауза";
    }
    _isRunning = !_isRunning;
}
private void ResetButton_Click(object sender, RoutedEventArgs e)
{
    _timer.Stop();
    _isRunning = false;
    _isWorkingSession = true;
    StartPauseButton.Content = "Старт";
    ResetTimer();
}
private void ResetTimer()
{
    _timeRemaining = TimeSpan.FromMinutes(WorkTimeMin);
    StatusText.Text = "Час фокусуватися!";
    UpdateTimerDisplay();
}
private void UpdateTimerDisplay()
{
    TimerText.Text = _timeRemaining.ToString(@"mm\:ss");
}
private void Window_Closed(object sender, EventArgs e)
{
    _timer.Stop();
}

```

```

    }
}

using BlockGame.Class;
using System;
using System.Windows;
using System.Windows.Threading;

namespace BlockGame.FocusFolder.Flowtime
{
    public partial class FlowtimeWindow : Window
    {
        private readonly DispatcherTimer _timer;
        private DateTime _startTimer;
        private TimeSpan _timeLeft;
        private bool _isBreakMode = false;
        public FlowtimeWindow()
        {
            InitializeComponent();
            _timer = new DispatcherTimer
            {
                Interval = TimeSpan.FromSeconds(1)
            };
            _timer.Tick += Timer_Tick;
            TimerText.Text = "00:00:00";
        }
        private void Timer_Tick(object? sender, EventArgs e)
        {
            if (!_isBreakMode)
            {
                TimeSpan elapsed = DateTime.Now - _startTimer;
                TimerText.Text = elapsed.ToString(@"hh\:mm\:ss");
            }
            else
            {
                if (_timeLeft > TimeSpan.Zero)
                {
                    _timeLeft -= TimeSpan.FromSeconds(1);
                    TimerText.Text = _timeLeft.ToString(@"hh\:mm\:ss");
                }
                else
                {
                    _timer.Stop();
                    _isBreakMode = false;
                    TextTime.Text = "WAVE FLOW";
                    TimerText.Text = "00:00:00";
                }
            }
        }
        private void Start_Click(object sender, RoutedEventArgs e)
        {
            _isBreakMode = false;
            TextTime.Text = "Фокус";
            _startTimer = DateTime.Now;
            if (!_timer.IsEnabled)
                _timer.Start();
        }
        private void Stop_Click(object sender, RoutedEventArgs e)
        {
            TimeSpan elapsed = DateTime.Now - _startTimer;
            AddNumberFocus.UpdateFlowtimeWindow();
            _isBreakMode = true;
            TextTime.Text = "Перерва";
            if (elapsed <= TimeSpan.FromMinutes(25))
            {
                _timeLeft = TimeSpan.FromMinutes(5);
            }
        }
    }
}

```

```

    }
    else if (elapsed <= TimeSpan.FromMinutes(50))
    {
        _timeLeft = TimeSpan.FromMinutes(8);
    }
    else if (elapsed <= TimeSpan.FromMinutes(90))
    {
        _timeLeft = TimeSpan.FromMinutes(10);
    }
    else
    {
        _timeLeft = TimeSpan.FromMinutes(20);
    }
    TimerText.Text = _timeLeft.ToString(@"hh\:mm\:ss");
}
private void Close_Click(object sender, RoutedEventArgs e)
{
    _timer?.Stop();
    Close();
}
private void Window_Closed(object sender, EventArgs e)
{
    _timer?.Stop();
}
}

using MongoDB.Bson;
using MongoDB.Driver;
using System;
using System.Diagnostics;
using System.Globalization;
using System.IO;
using System.Threading;
using System.Threading.Tasks;
using System.Windows;
namespace BlockGame.Class
{
    public class BlockApp
    {
        private readonly IMongoCollection<BsonDocument> _collectionGame;
        private readonly CultureInfo _culture = new CultureInfo("uk-UA");
        private string _idProgram;
        private string _nicknameUserEmail;
        private TimeSpan _startTime;
        private TimeSpan _endTime;
        private string _day;
        private CancellationTokenSource _cts;
        public BlockApp()
        {
            var client = new MongoClient("mongodb://localhost:27017/");
            var database = client.GetDatabase("BlockGame");
            _collectionGame = database.GetCollection<BsonDocument>("BlockUserGame");
        }
        public void LoadData()
        {
            var gameBlock = _collectionGame.Find(new BsonDocument()).ToList();
            foreach (var item in gameBlock)
            {
                _idProgram = item.Contains("Id_Game") ? item["Id_Game"].ToString() : string.Empty;
                _nicknameUserEmail = item.Contains("User") ? item["User"].ToString() : string.Empty;
                _startTime = item.Contains("StartTime") ?
                TimeSpan.Parse(item["StartTime"].ToString()) : TimeSpan.Zero;
                _endTime = item.Contains("EndTime") ? TimeSpan.Parse(item["EndTime"].ToString()) :
                TimeSpan.Zero;
                _day = item.Contains("Day") ? item["Day"].ToString() : string.Empty;
            }
        }
    }
}

```

```

}
}
public void StartMonitoring(string currentUserEmail)
{
    StartMonitoring(currentUserEmail, 0, "BD");
}
public void StartMonitoring(string currentUserEmail, int numberFocus, string mode)
{
    StopMonitoring();
    _cts = new CancellationTokenSource();
    LoadData();
    CancellationToken token = _cts.Token;
    Task.Run(() => MonitorGameStatus(currentUserEmail, numberFocus, mode, token), token);
}
public void StopMonitoring()
{
    if (_cts != null)
    {
        _cts.Cancel();
        _cts.Dispose();
        _cts = null;
    }
}
private void MonitorGameStatus(string currentUserEmail, int numberFocus, string mode,
CancellationToken token)
{
    while (!token.IsCancellationRequested)
    {
        DateTime now = DateTime.Now;
        TimeSpan currentTime = now.TimeOfDay;
        string dayOfWeekUkrainian = now.ToString("dddd", _culture);
        if (mode == "BD")
        {
            if (!string.IsNullOrEmpty(_nicknameUserEmail) && currentUserEmail ==
_nicknameUserEmail)
            {
                bool isCorrectDay = dayOfWeekUkrainian.Equals(_day,
StringComparison.OrdinalIgnoreCase);
                bool isAllowedTime = currentTime >= _startTime && currentTime < _endTime;
                if (!(isCorrectDay && isAllowedTime))
                {
                    if (!string.IsNullOrEmpty(_idProgram) && IsAppRunning(_idProgram))
                    {
                        CloseApp(numberFocus, mode);
                    }
                }
            }
            else if (mode == "NoBD")
            {
                CloseApp(numberFocus, mode);
            }
            try
            {
                Task.Delay(5000, token).Wait(token);
            }
            catch (OperationCanceledException)
            {
                break;
            }
        }
        private bool IsAppRunning(string exeName)
        {
            foreach (var process in Process.GetProcesses())

```

```

    {
    try
    {
        if (process.ProcessName.Equals(Path.GetFileNameWithoutExtension(exeName),
StringComparison.OrdinalIgnoreCase))
        {
            return true;
        }
    }
    catch { }
    }
    return false;
}
private void CloseApp(int numberFocus, string mode)
{
    Process[] processes = Process.GetProcesses();
    foreach (var process in processes)
    {
        try
        {
            switch (mode)
            {
                case "BD":
                    if (process.ProcessName.Equals(Path.GetFileNameWithoutExtension(_idProgram),
StringComparison.OrdinalIgnoreCase))
                    {
                        process.Kill();
                    }
                    break;
                case "NoBD":
                    switch (numberFocus)
                    {
                        case 3:
                            string[] blockedApps = { "steam", "Telegram" };
                            foreach (string blockedApp in blockedApps)
                            {
                                if (process.ProcessName.Equals(blockedApp, StringComparison.OrdinalIgnoreCase))
                                {
                                    process.Kill();
                                }
                            }
                            break;
                        }
                    }
                    break;
            }
        }
        catch { }
    }
}

using MongoDB.Bson;
using MongoDB.Driver;

namespace BlockGame.Class
{
    public class ViewProfile
    {
        public static string FindNameProgramDataBase { get; set; }
        public static string FindUserDescription { get; set; }
        private static readonly IMongoCollection<BsonDocument> _collectionProfile;
        public static string Nickname;
        static ViewProfile()
        {
            var client = new MongoClient("mongodb://localhost:27017/");

```

		Хомнюк В.О.			ІПЗ.КР.Б - 121 - 26 - ІЗ	Арк.
		Савіцький Р.С.				77

```

var database = client.GetDatabase("BLockGame");
_collectionProfile = database.GetCollection<BsonDocument>("Profile");
}
public static void ViewProfileData()
{
if (string.IsNullOrEmpty(Base_User.User))
{
FindNameProgramDataBase = "Гість";
FindUserDescription = "Опис відсутній.";
return;
}
var filter = Builders<BsonDocument>.Filter.Eq("RegisterNickName", Base_User.User);
var userProfile = _collectionProfile.Find(filter).FirstOrDefault();
if (userProfile != null)
{
FindNameProgramDataBase = userProfile.Contains("Login") ?
userProfile["Login"].ToString() : Base_User.User;
FindUserDescription = userProfile.Contains("UserDescription") ?
userProfile["UserDescription"].ToString() : "Опис відсутній.";
}
else
{
FindNameProgramDataBase = Base_User.User;
FindUserDescription = "Неналаштований акаунт";
}
}
public static void UpdateProfileData(string newNickname, string newDescription)
{
var filter = Builders<BsonDocument>.Filter.Eq("Login", FindNameProgramDataBase);
var update = Builders<BsonDocument>.Update
.Set("Login", newNickname)
.Set("UserDescription", newDescription);
_collectionProfile.UpdateOne(filter, update);
FindNameProgramDataBase = newNickname;
FindUserDescription = newDescription;
}
}
}

```

		Хомніук В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				
						78

```

using BLockGame.FocusFolder.UltraFocus;
using MongoDB.Bson;
using MongoDB.Driver;

namespace BLockGame.Class
{
    public class AddNumberFocus
    {
        public static string FindUltraFocus { get; set; }
        public static string Pomodoro { get; set; }
        public static string FlowtimeWindow { get; set; }
        public static string Productivity_Formula { get; set; }
        private static readonly IMongoCollection<BsonDocument> _collectionProfile;
        static AddNumberFocus()
        {
            var client = new MongoClient("mongodb://localhost:27017/");
            var database = client.GetDatabase("BLockGame");
            _collectionProfile = database.GetCollection<BsonDocument>("Profile");
        }
        public static void UpdateUltraFocus()
        {
            var filter = Builders<BsonDocument>.Filter.Eq("Login", Base_User.User);
            var update = Builders<BsonDocument>.Update.Inc("UltraFocus", 1);
            _collectionProfile.UpdateOne(filter, update);
        }
        public static void UpdatePomodoro()
        {
            var filter = Builders<BsonDocument>.Filter.Eq("Login", Base_User.User);
            var update = Builders<BsonDocument>.Update.Inc("Pomodoro", 1);
            _collectionProfile.UpdateOne(filter, update);
        }
        public static void UpdateFlowtimeWindow()
        {
            var filter = Builders<BsonDocument>.Filter.Eq("Login", Base_User.User);
            var update = Builders<BsonDocument>.Update.Inc("Flowtime", 1);
            _collectionProfile.UpdateOne(filter, update);
        }
        public static void UpdateProductivity_Formula()
        {
            var filter = Builders<BsonDocument>.Filter.Eq("Login", Base_User.User);
            var update = Builders<BsonDocument>.Update.Inc("Productivity_Formula", 1);
            _collectionProfile.UpdateOne(filter, update);
        }
        public static void GetFocus()
        {
            var filter = Builders<BsonDocument>.Filter.Eq("Login", Base_User.User);
            var userProfile = _collectionProfile.Find(filter).FirstOrDefault();
            if (userProfile != null)
            {
                FindUltraFocus = userProfile.Contains("UltraFocus") ?
                userProfile["UltraFocus"].ToString() : "0";
                Pomodoro = userProfile.Contains("Pomodoro") ? userProfile["Pomodoro"].ToString() :
                "0";
                FlowtimeWindow = userProfile.Contains("Flowtime") ?
                userProfile["Flowtime"].ToString() : "0";
                Productivity_Formula = userProfile.Contains("Productivity_Formula") ?
                userProfile["Productivity_Formula"].ToString() : "0";
            }
            else
            {
                FindUltraFocus = "0";
                Pomodoro = "0";
                FlowtimeWindow = "0";
                Productivity_Formula = "0";
            }
        }
    }
}

```

```

    }
    }
}

using BLockGame.Model;
using Microsoft.VisualBasic.Logging;
using MongoDB.Bson;
using MongoDB.Driver;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Forms;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace BLockGame.Admin
{
    public partial class Logi : Window
    {
        private static IMongoCollection<BsonDocument> _collection;
        public Logi()
        {
            var client = new MongoClient("mongodb://localhost:27017/");
            var database = client.GetDatabase("BLockGame");
            _collection = database.GetCollection<BsonDocument>("BlockUserGame");
            InitializeComponent();
            LoadLogs();
        }
        public void LoadLogs()
        {
            var logs = new List<LogsModel>();
            var logsData = _collection.Find(new BsonDocument()).ToList();
            foreach (var item in logsData)
            {
                var User = item["User"].ToString();
                var ProgramName = item["Id_Game"].ToString();
                var Day = item["Day"].ToString();
                var StartTime = item["StartTime"].ToString();
                var EartTime = item["EndTime"].ToString();
                var StartTimeBlockProgram = item["StartTimeBlockPrograms"].ToString();
                logs.Add(new LogsModel
                {
                    Name = User,
                    ProgramName = ProgramName,
                    Day = Day,
                    StartTime = StartTime,
                    EndTime = EartTime,
                    StartTimeBlockPrograms = StartTimeBlockProgram
                });
            }
            MyGrid.ItemsSource = logs;
        }
    }
}

using System.Net.Http;
using System.Windows;
using Newtonsoft.Json.Linq;
using System.Windows.Controls;

```

		Хомнюк В.О.			ІПЗ.КР.Б - 121 - 26 - ІЗ	Арк.
		Савіцький Р.С.				80

```

using System.Windows.Input;
using WpfMessageBox = System.Windows.MessageBox;
using System.IO;
using MongoDB.Driver;
using MongoDB.Bson;

namespace BlockGame
{
    public partial class Team_Block : Window
    {
        private string[] Day = { "Понеділок", "Вівторок", "Середа", "Четверг", "П'ятниця",
"Суббота", "Неділя" };
        private static IMongoCollection<BsonDocument> _collection;
        string apiKey = "1744EA18CBBFDCF855948A249C398F30";
        string steamId = "76561198961054657";
        private string GameDirectory = @"D:\Game Programs\Steam\steamapps\";
        private static string ItemComboBox;
        private static string StartText;
        private static string EndText;
        private static string FileName;
        private static string foldername;
        private List<string> gameNames = new();
        public Team_Block()
        {
            InitializeComponent();
            var client = new MongoClient("mongodb://localhost:27017/");
            var database = client.GetDatabase("BlockGame");
            _collection = database.GetCollection<BsonDocument>("BlockUserGame");
            foreach (var day in Day)
            DayComboBox.Items.Add(day);
            _ = LoadGamesAsync();
        }
        private async Task LoadGamesAsync()
        {
            string url =
$"https://api.steampowered.com/IPlayerService/GetOwnedGames/v1/?key={apiKey}&steamid={
steamId}&include_appinfo=1";
            using HttpClient client = new();
            var response = await client.GetAsync(url);
            string json = await response.Content.ReadAsStringAsync();
            JObject data = JObject.Parse(json);
            var games = data["response"]?["games"];
            if (games == null) return;
            gameNames.Clear();
            foreach (var game in games)
            {
                string name = game["name"]?.ToString();
                if (!string.IsNullOrEmpty(name))
                gameNames.Add(name);
            }
            ListSteamGame.ItemsSource = gameNames;
        }
        private async Task<string> FindGameExecutableAsync(string gameName)
        {
            return await Task.Run(() =>
            {
                try
                {
                    if (!Directory.Exists(GameDirectory))
                    return null;
                    var dirs = Directory.GetDirectories(GameDirectory);
                    foreach (var dir in dirs)
                    {
                        var exeFiles = Directory.GetFiles(dir, "*.exe", SearchOption.AllDirectories);
                        foreach (var exe in exeFiles)

```

```

{
string exeName = Path.GetFileNameWithoutExtension(exe);
if (exeName != null &&
exeName.Contains(gameName, StringComparison.OrdinalIgnoreCase))
{
return exe;
}
}
}
}
catch
{
return null;
}
return null;
});
}
private void SearchGame_TextChanged(object sender, TextChangedEventArgs e)
{
string text = SearchGame.Text?.ToLower() ?? "";
List<SteamGame>.ItemsSource = gameNames
.Where(g => g.ToLower().Contains(text))
.ToList();
}
private void Button_Click(object sender, RoutedEventArgs e)
{
if (DayComboBox.SelectedItem == null)
{
WpfMessageBox.Show("Виберіть день!");
return;
}
if (string.IsNullOrEmpty(MinDay.Text) ||
string.IsNullOrEmpty(MaxDay.Text))
{
WpfMessageBox.Show("Введіть час!");
return;
}
if (string.IsNullOrEmpty(FileName) ||
string.IsNullOrEmpty(foldername) ||
string.IsNullOrEmpty(Base_User.User))
{
WpfMessageBox.Show("Не вибрана гра або користувач!");
return;
}
ItemComboBox = DayComboBox.SelectedItem.ToString();
StartText = MinDay.Text;
EndText = MaxDay.Text;
var doc = new BsonDocument
{
{ "Id_Game", FileName },
{ "Name_Game", foldername },
{ "User", Base_User.User },
{ "Day", ItemComboBox },
{ "StartTime", StartText },
{ "EndTime", EndText }
};
try
{
_collection.InsertOne(doc);
WpfMessageBox.Show("Збережено!");
}
catch (Exception ex)
{
WpfMessageBox.Show(ex.Message);
}
}

```

		Хомніук В.О.			ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
		Савіцький Р.С.				
						82

```

    }
    private async void ListSteamGame_MouseDoubleClick(object sender, MouseButtonEventArgs
e)
    {
        if (ListSteamGame.SelectedItem is not string selectedGame)
            return;
        selectedGame = selectedGame.Trim();
        string foundPath = await FindGameExecutableAsync(selectedGame);
        if (!string.IsNullOrEmpty(foundPath))
        {
            foldername = foundPath;
            FileName = Path.GetFileName(foundPath);
            System.Windows.Clipboard.SetText(FileName);
            WpfMessageBox.Show($"Знайдено: {FileName}");
        }
        else
        {
            WpfMessageBox.Show("Файл гри не знайдено");
        }
    }
    private void ListSteamGame_SelectionChanged(object sender, SelectionChangedEventArgs
e)
    {
    }
}

using System;
using System.Windows;
using System.Threading;
using WinFormsApp = System.Windows.Forms.Application;
using WpfApp = System.Windows.Application;
using WpfMessageBox = System.Windows.MessageBox;
using MongoDB.Bson;
using MongoDB.Driver;
using static System.Windows.Forms.VisualStyles.VisualStyleElement.StartPanel;

namespace BlockGame
{
    public partial class Window_Block_Game : Window
    {
        private string[] Day = { "Понеділок", "Вівторок", "Середа", "Четверг", "П'ятниця",
"Суббота", "Неділя" };
        private static IMongoCollection<BsonDocument> _collection;
        private static string ItemComboBox;
        private static string StartText;
        private static string EndText;
        private static string StartTimeBlockPrograms;
        string GetEmailDataBase;
        string GetUserName;
        public Window_Block_Game()
        {
            InitializeComponent();
            var client = new MongoClient("mongodb://localhost:27017/");
            var database = client.GetDatabase("BBlockGame");
            _collection = database.GetCollection<BsonDocument>("BlockUserGame");
            foreach (var day in Day)
            {
                DayComboBox.Items.Add(day);
            }
        }
        private async void Button_Click(object sender, RoutedEventArgs e)
        {
            if (!string.IsNullOrEmpty(Id_Game.Text))
            {
                if (!string.IsNullOrEmpty(NameBlockGameTextBox.Text))

```

		Хомнюк В.О.			ІПЗ.КР.Б - 121 - 26 - ІЗ	Арк.
		Савіцький Р.С.				83

```

{
string IdGame = Id_Game.Text;
string NameBlockGame = NameBlockGameTextBox.Text;
if (!string.IsNullOrEmpty(DayComboBox.SelectedItem?.ToString()))
{
ItemComboBox = DayComboBox.SelectedItem.ToString();
if (!string.IsNullOrEmpty(StartTimeTextBox.Text))
{
if (!string.IsNullOrEmpty(EndTimeTextBox.Text))
{
StartText = StartTimeTextBox.Text;
EndText = EndTimeTextBox.Text;
StartTimeBlockPrograms = DateTime.Now.ToString("HH:mm:ss");
var Block = new BsonDocument
{
{"Id_Game", IdGame},
{"Name_Game", NameBlockGame},
{"User", Base_User.User},
{"Day", ItemComboBox},
{"StartTime", StartText},
{"EndTime", EndText},
{"StartTimeBlockPrograms", StartTimeBlockPrograms}
};
_collection.InsertOne(Block);
}
else
{
WpfMessageBox.Show("Введіть час до якого хочете грати");
}
}
else
{
WpfMessageBox.Show("Введіть час з якого хочете почати грати");
}
}
else
{
WpfMessageBox.Show("Виберіть день в який хочете грати");
}
}
else
{
WpfMessageBox.Show("Введіть назву програми яку хочете заблокувати");
}
}
else
{
WpfMessageBox.Show("Введіть назву програми .exe");
}
}
private void ExitButton_Click(object sender, RoutedEventArgs e)
{
Main_Manu main_Manu = new Main_Manu();
main_Manu.Show();
this.Close();
}

private void TimeCheckBox_Checked(object sender, RoutedEventArgs e)
{
EndTimeTextBox.Visibility = Visibility.Visible;
StartTimeTextBox.Visibility = Visibility.Visible;
StartLable.Visibility = Visibility.Visible;
EndLable.Visibility = Visibility.Visible;
}

```

```

private void TimeCheckBox_Unchecked_1(object sender, RoutedEventArgs e)
{
    EndTimeTextBox.Visibility = Visibility.Hidden;
    StartTimeTextBox.Visibility = Visibility.Hidden;
    StartLable.Visibility = Visibility.Hidden;
    EndLable.Visibility = Visibility.Hidden;
}

private void Button_Click_1(object sender, RoutedEventArgs e)
{
    Team_Block team_Block = new Team_Block();
    team_Block.Show();
    this.Close();
}
}

```

		Хомніук В.О.			ІПЗ.КР.Б - 121 - 26 - ІЗ	Арк.
		Савіцький Р.С.				
						85

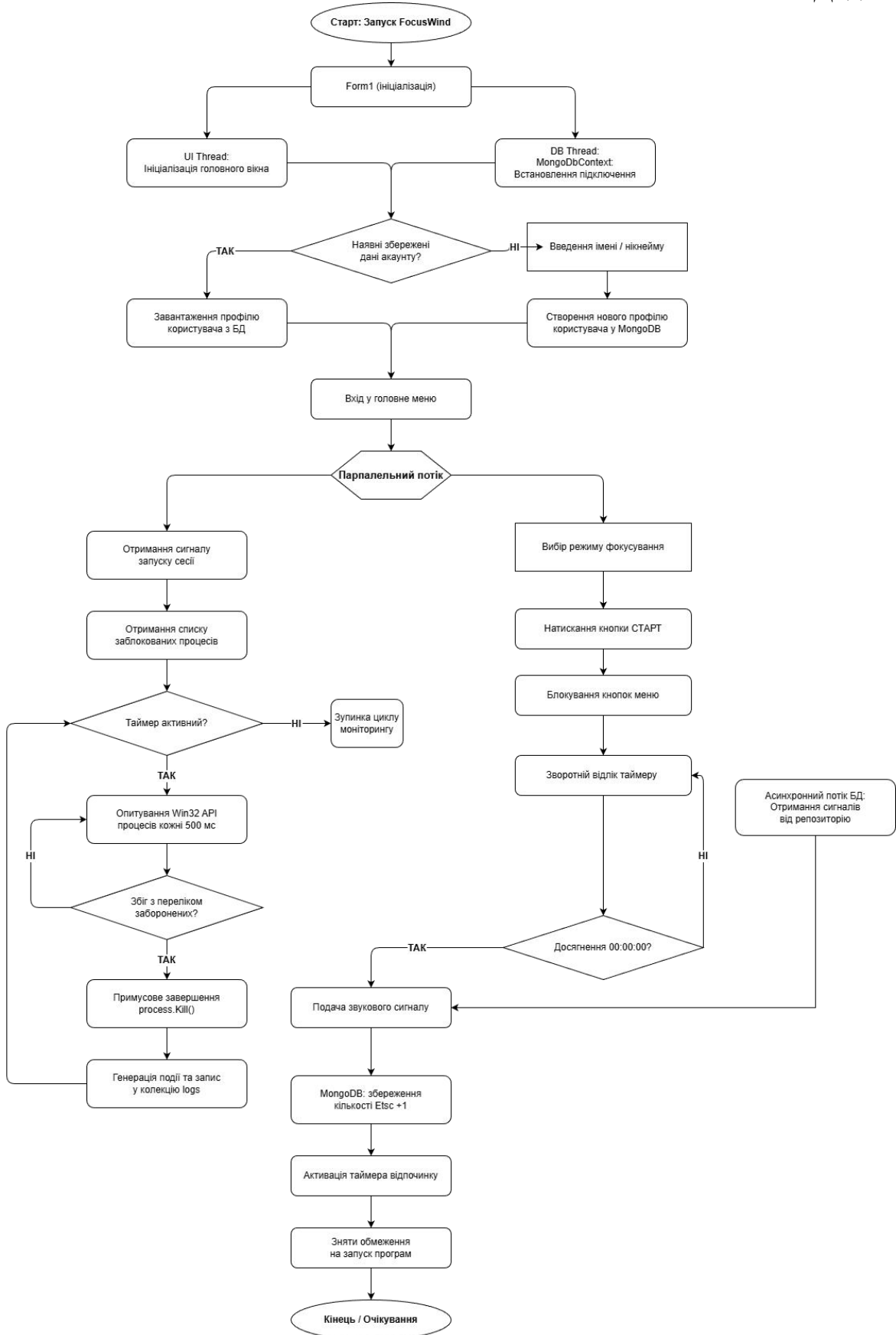


Рис В.1. Виконання сесій фокусування