Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБЛЕННЯ ЗАСОБІВ ДЛЯ КОНТРОЛЮ І КЕРУВАННЯ ДІЄЮ КОРИСТУВАЧА У ПРОГРАМНИХ ЗАСТОСУНКАХ
(тема)

Виконав:
здобувач 4 року навчання,
групи ІТІНФ-21-1
Гаєвий А. О.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник проф. Гороховатський В. О.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри інформатики _____ Кобилін О. А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту

Кафедра Інформатики

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Гаєвому Антону Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка засобів для контролю і керування дією користувача у програмних застосунках

затверджена наказом університету від 19 травня 2025 року № 381Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 25 травня 2025 р.
3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, платформа .Net, мова програмування C#, бібліотека LiveChartsCore.SkiaSharpView.WPF, середовище розробки Visual Studio, ORM-бібліотека Entity Framework Core, мова розмітки XAML, система керування базами даних SQLite, графічне середовище WPF.

4. Перелік питань, що потрібно опрацювати в роботі
1. Огляд існуючих програм для моніторингу продуктивності та цифрової саморегуляції.
2. Аналіз методів гейміфікації та формування корисних звичок користувача.
3. Проєктування архітектури застосунку та структури бази даних.
4. Реалізація WPF-застосунку на платформі .NET 8.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Актуальність проблеми цифрової залежності, постановка задачі, лістинги реалізації застосунку, тестові зображення.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	07.04.2025	
2	Аналіз завдання, підбір літератури	08.04.25-10.04.25	
3	Аналіз літератури з досліджуваної проблеми	11.04.25-14.04.25	
4	Аналіз технічних засобів	15.04.25-20.04.25	
5	Розробка методів	21.04.25-27.04.25	
6	Програмна реалізація	28.04.25-11.05.25	
7	Оформлення пояснювальної записки	12.05.25-20.05.25	
8	Перевірка на нормоконтроль	21.05.25-01.06.25	
9	Перевірка на плагіат	21.05.25-01.06.25	
10	Рецензування	21.05.25-01.06.25	
11	Підготовка презентації та доповіді	21.05.25-18.06.25	
12	Занесення роботи в електронний архів	02.06.25-18.06.25	
	Попередній захист кваліфікаційної роботи	02.06.25-18.06.25	

Дата видачі завдання 7 квітня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. Гороховатський В. О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 63 с., 11 табл., 22 рис., 36 джерел.

КОНТРОЛЬ ЧАСУ, ГЕЙМІФІКАЦІЯ, ОБМЕЖЕННЯ ЗАСТОСУНКІВ, ПРОДУКТИВНІСТЬ, САМОКОНТРОЛЬ, СТАТИСТИКА ВЗАЄМОДІЇ.

Об’єктом роботи є процес контролю щодо використання користувачем програмних інструментів у межах повсякденної діяльності.

Метою роботи є розробка програмного забезпечення, що дозволяє здійснювати облік, обмеження та мотиваційне управління часом використання інструментальних засобів із використанням гейміфікованого підходу.

Використано методи проєктування програмного забезпечення, аналітичного моделювання та гейміфікації. Проведено дослідження методів контролю часу використання програм, розроблено систему завдань із нарахуванням досвіду, реалізовано механізми обмеження доступу до застосунків, а також систему мотиваційних сповіщень. Здійснено аналіз підходів до вимірювання активного часу взаємодії з програмами. Розроблено архітектуру застосунку на основі патерну MVVM з використанням WPF та SQLite.

У результаті роботи здійснена програмна реалізація системи для підвищення особистої продуктивності та цифрового добробуту.

TIME CONTROL, GAMIFICATION, APPLICATION RESTRICTIONS, PRODUCTIVITY, SELF-CONTROL, INTERACTION STATISTICS.

The object of the work is the process of controlling the user's use of software tools in daily activities.

The aim is to develop software for tracking, limiting, and motivationally managing time spent using tools with a gamified approach.

Software design, analytical modeling, and gamification methods were applied. Usage time monitoring methods were studied, a task system with experience points was developed, as well as restriction mechanisms and motivational notifications. Approaches to measuring active interaction time were analyzed. The application architecture was built using the MVVM pattern with WPF and SQLite.

As a result, a software system was implemented to improve personal productivity and digital well-being.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

C# – C Sharp (мова програмування)

.NET – Microsoft .NET Platform (платформа Microsoft .NET)

WPF – Windows Presentation Foundation (Фонд презентацій Windows)

WPF – графічна підсистема Microsoft .NET для створення настільних застосунків з підтримкою сучасного інтерфейсу користувача, стилів, шаблонів, анімації та двостороннього зв’язування даних

XAML – Extensible Application Markup Language (розширювана мова розмітки застосунків)

MVVM – Model-View-ViewModel (модель-представлення-модель представлення)

MVVM – архітектурний шаблон проектування, який відокремлює інтерфейс користувача від бізнес-логіки шляхом використання проміжної логіки представлення

SQL – Structured Query Language (структурована мова запитів)

SQLite – Lightweight Structured Query Language Database (легка реляційна база даних)

JSON – JavaScript Object Notation (нотація об’єктів JavaScript)

API – Application Programming Interface (програмний інтерфейс застосунку)

IDE – Integrated Development Environment (інтегроване середовище розробки)

СУБД – система управління базами даних

DI – Dependency Injection (впровадження залежностей)

UI – User Interface (користувацький інтерфейс)

ORM – Object-Relational Mapping (об’єктно-реляційне відображення)

ВСТУП

У сучасному світі, де цифрові технології стали невід’ємною частиною повсякденного життя, зростає потреба у ефективному і свідомому використанні часу та ресурсів. Все більше людей стикаються з труднощами у підтриманні концентрації, самодисципліни та ефективного розподілу часу між продуктивною діяльністю й відпочинком. Постійна доступність до інтернету, соціальних мереж, відео та ігор сприяє формуванню цифрової залежності, що особливо відображається на молоді [1].

Разом із новими можливостями, які надають сучасні технології, з’являється виклик – ефективно керувати особистим часом. Це створює попит на інструментальний засіб, який не лише фіксує активність, а й мотивує до саморозвитку через систему завдань, досвіду та навичок.

Актуальність теми зумовлена зростаючою потребою в інструментах цифрового добробуту. У період, коли дедалі більше людей працюють дистанційно або навчаються онлайн, контроль над власним часом стає необхідністю. Програмне забезпечення, що не лише контролює, а й мотивує, відкриває нові підходи до самоменеджменту та персонального розвитку [2].

Отже, розробка застосунку для контролю часу використання інструментальних засобів у поєднанні з мотиваційними механізмами є важливим кроком до підвищення особистої ефективності та боротьби з цифровою залежністю.

1 ПРОГРАМНИЙ КОНТРОЛЬ ЧАСУ ВИКОРИСТАННЯ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ

1.1 Проблематика цифрової залежності та актуальність контролю часу

У сучасному цифровому середовищі все частіше спостерігається явище надмірного використання електронних пристроїв та інтернет-сервісів, що супроводжується втратою контролю над часом та зниженням особистої продуктивності. Однією з ключових причин цього явища є так зване дофамінове мислення – прагнення до постійного отримання легкого задоволення, яке викликає короткочасний дофаміновий виплеск, нейромедіатора, пов’язаного з мотивацією і винагородою.

Цифровий контент, зокрема короткі відео, соціальні мережі та ігри, формують швидкі цикли винагороди, що створюють відчуття моментального задоволення. На противагу цьому, складні завдання, навчання або тривала робота вимагають зусиль і не приносять миттєвого результату, що часто викликає уникнення дискомфорту – ще один важливий психологічний фактор, який сприяє прокрастинації.

Прокрастинація – це систематичне відкладання важливих справ, навіть за наявності усвідомлення їхньої значущості. Вона не є простою лінню, а радше психологічним захисним механізмом, спрямованим на уникнення внутрішнього стресу, пов’язаного зі складними або неприємними завданнями. У цифровому середовищі прокрастинація часто набуває форми постійного перемикання між вкладками браузера, перегляду розважального контенту або безцільного «скролінгу» в соціальних мережах [3].

Наслідками прокрастинації є хронічна неефективність, зниження самооцінки, накопичення незавершених справ і почуття провини, що лише посилює залежність від цифрових розваг як форми втечі від реальності.

Одна з ключових причин формування цифрової залежності – бажання уникнути дискомфорту. Складні або рутинні завдання вимагають

концентрації, волі та терпіння, а сучасні цифрові платформи пропонують миттєве задоволення без зусиль. Це створює небезпечну поведінкову модель – користувач частіше обирає дії, які приносять негайну насолоду, навіть якщо вони не мають цінності в довгостроковій перспективі.

Усе це пов’язано з дофаміновою системою винагороди: чим швидше користувач отримує «винагороду», тим більше він схильний повертатися до цього джерела знову. Саме тому користування TikTok, YouTube Shorts чи нескінченним скролінгом стрічки може набувати ознак поведінкової залежності.

У сучасному інформаційному середовищі людина постійно перебуває під впливом великої кількості зовнішніх подразників: сповіщення, реклама, оновлення в соціальних мережах, новинні заголовки тощо. Це призводить до інформаційного перевантаження – стану, коли мозок не встигає якісно обробляти вхідну інформацію, що негативно впливає на концентрацію та здатність до глибокої концентрації та роботи [4].

У результаті втрачається здатність довго зосереджуватися на одній задачі, зростає імпульсивність і знижується ефективність мислення. Людина все частіше перемикається між завданнями, не завершуючи жодного з них, що створює ілюзію активності при фактичній неефективності.

У контексті цифрової залежності це явище стає особливо небезпечним, адже саме високочастотне перемикання уваги закріплює поведінкову модель швидкого споживання інформації. Для подолання цього ефекту важливо створювати умови для глибокої, усвідомленої роботи, а також використовувати програмні інструменти, що сприяють фокусуванню уваги та відсіканню зайвих стимулів та подразників.

1.2 Огляд існуючих програмних рішень для контролю часу

Одним з ефективних інструментів у подоланні прокрастинації та цифрової залежності є метод двох хвилин, також відомий як метод імпульсу. Його суть полягає у тому, щоб зменшити психологічний бар'єр перед початком завдання – користувач не зобов'язується повністю виконати справу, а лише почати її протягом перших двох хвилин [5].

Це дозволяє обійти внутрішній опір, пов'язаний зі страхом складності або дискомфорту. На практиці навіть невеликий початковий крок (наприклад, відкрити потрібну програму, почати читати документ або написати наглядне завдання) активує стан фокусування, після чого людині набагато легше продовжити роботу. У більшості випадків ініціація дії призводить до її природного продовження.

У межах даної кваліфікаційної роботи метод двох хвилин буде реалізовано безпосередньо в програмному забезпеченні. Користувач отримуватиме повідомлення із закликом «почни з двох хвилин», що дозволить запустити процес продуктивності без примусу. Також передбачається мотиваційна система за «початок активності», яка формуватиме позитивне підкріплення і сприятиме розвитку звички до самостійного старту завдань.

Таким чином, метод імпульсу виступає не лише як психологічна техніка, а й як конкретний функціональний модуль програмного продукту, орієнтований на м'який, але стабільний розвиток дисципліни та фокусування уваги.

У сучасному цифровому середовищі користувачі мають доступ до широкого спектру програм, які допомагають управляти часом, аналізувати власну продуктивність, а також обмежувати доступ до розважальних або відволікаючих програм. Такі інструменти особливо актуальні в умовах зростання віддаленої роботи, навчання та зниження загального рівня концентрації серед користувачів.

Сучасні засоби можна умовно поділити на такі категорії [5]:

– трекери часу (RescueTime, ManicTime, Toggl);
– блокувальники сайтів та додатків (Cold Turkey, Freedom, FocusMe);
– ігрові та мотиваційні додатки (Forest, Habitica, FocusPlant);
– комплексні системи управління часом і цілями (TMetric, Clockify).

RescueTime – це одна з найвідоміших програм для відстеження активності користувача на комп'ютері або мобільному пристрої. Вона працює у фоновому режимі й автоматично реєструє, які програми та вебсайти були використані, скільки часу на них витрачено, і наскільки вони були продуктивними згідно з обраним профілем (робота, навчання, творчість тощо) [6].

Однією з головних переваг RescueTime є повна автоматизація процесу збору даних про активність користувача. Програма не потребує ручного введення інформації: вона самостійно фіксує, які програми та сайти були відкриті, і скільки часу на них витрачено. Це дозволяє мінімізувати зусилля користувача та уникнути викривлень у статистиці, що виникають при власноручному звітуванні.

Другою сильною стороною є деталізована аналітика: користувач отримує не лише загальну кількість витраченого часу, а й розбивку за категоріями (робота, розваги, навчання тощо), графіки ефективності та індивідуальні рейтинги продуктивності. Це допомагає не просто контролювати час, а аналізувати власні цифрові звички і виявляти слабкі місця в організації часу.

Основним недоліком RescueTime є відсутність активного впливу на поведінку користувача – програма не може блокувати доступ до сайтів або програм, які є «небажані». Вона виконує функцію «дзеркала», але не інструменту зміни поведінки. Тобто користувач може побачити, що витратив 3 години на YouTube, але нічого не заважає йому робити це знову.

Ще одним обмеженням є відсутність вбудованої системи мотивації: програма не пропонує нагород, завдань чи елементів гейміфікації. Це знижує

її ефективність для користувачів, які потребують зовнішніх стимулів або бажають отримувати від процесу відчуття прогресу. Крім того, деякі функції, зокрема розширена аналітика, доступні лише у платній версії, що може бути перепоною для подальшого використання (рис. 1.1).

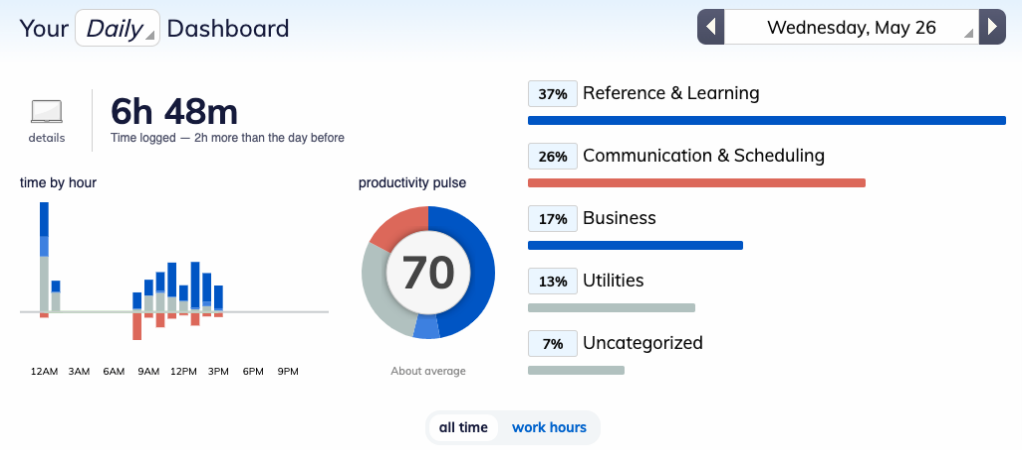

Рисунок 1.1 – Інтерфейс головної сторінки RescueTime

Cold Turkey – це програма жорсткого блокування сайтів і програм. Її головна особливість – незворотність. Після встановлення блокування користувач не зможе зняти його до завершення вказаного часу.

Cold Turkey відрізняється суворим підходом до блокування відволікаючих ресурсів, що робить його ефективним інструментом для тих, хто хоче жорстко обмежити себе. Програма дозволяє повністю заблокувати як окремі сайти чи додатки, так і цілі категорії або навіть весь інтернет на певний час. Унікальність полягає в тому, що після активації блокування його неможливо скасувати, навіть через перезавантаження системи. Це створює відчуття серйозності рішень користувача та сприяє формуванню самодисципліни.

Ще одна перевага – гнучкість налаштувань. Користувач може створювати власні списки небажаних сайтів, встановлювати графіки блокування, використовувати таймери фокусування або режими, призначені для певних видів роботи (наприклад, «режим письменника»). Це дозволяє адаптувати програму під індивідуальні потреби.

Такий жорсткий підхід, з одного боку, ефективний, але з іншого – може викликати внутрішній опір і зниження мотивації. Коли користувач не має вибору і змушений дотримуватись правил, встановлених програмою, це може викликати психологічний дискомфорт. У деяких випадках такий контроль може навіть призводити до спроб обійти систему або повністю відмовитись від її використання, що найчастіше спостерігається у залежних користувачів.

Також важливо відзначити, що Cold Turkey не має аналітичної частини: програма не показує, скільки часу було збережено, які звички змінились, які сайти блокувались частіше за інші. Відсутність зворотного зв'язку робить процес зміни поведінки менш усвідомленим. Крім того, програма не містить жодних елементів гейміфікації або підтримки цілей, що обмежує її довготривалу ефективність (рис. 1.2).

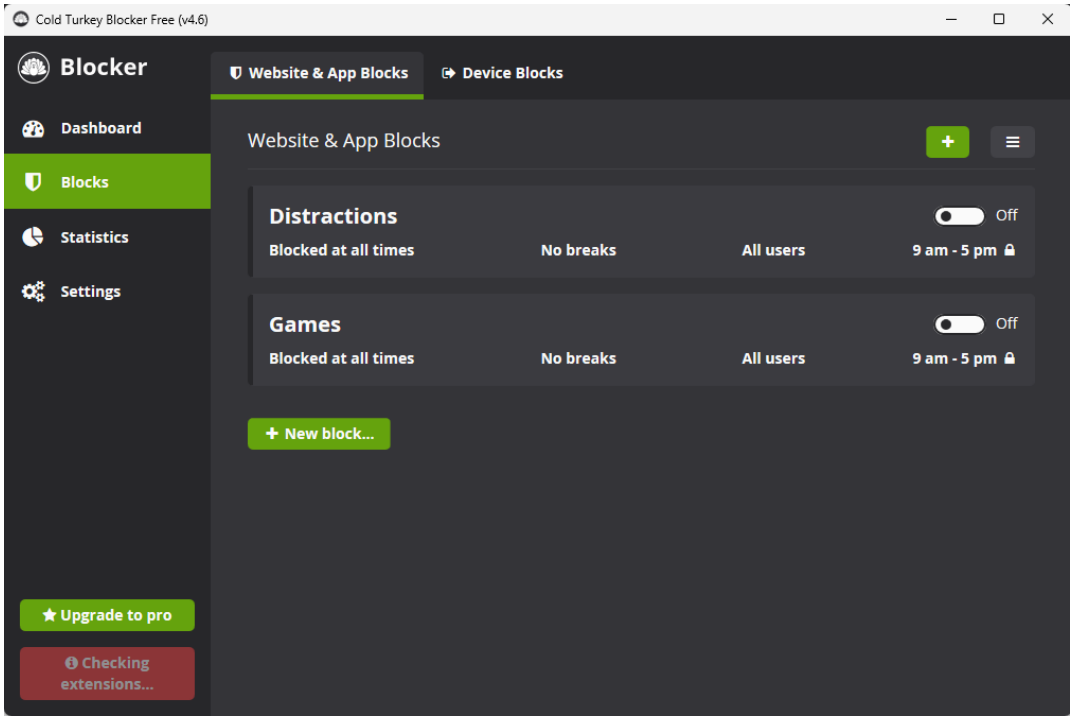

Рисунок 1.2 – Інтерфейс головної сторінки Cold Turkey

FocusMe – це гібридний застосунок, який поєднує функції блокувальника, планувальника та мотиваційного асистента. Він дає можливість створювати гнучкі правила обмеження, задавати продуктивні цілі [7].

FocusMe є одним із найгнучкіших і найбагатофункціональніших інструментів у сфері контролю часу. Його ключовою перевагою є унікальна система правил і шаблонів, яка дозволяє налаштувати програму практично під будь-який стиль роботи чи навчання. Користувач може створювати розклади, дозволяти короткі перерви, комбінувати блокування з повідомленнями-мотиваціями, створювати винятки та сценарії роботи. Така глибока кастомізація робить програму універсальною для різних типів користувачів – від фрілансерів до студентів.

Разом з тим, така багатофункціональність стає і її головним недоліком. Інтерфейс програми складний і перевантажений: початківцю може бути важко зрозуміти, як правильно налаштувати правила, які опції активувати, як використовувати аналітику. Це часто відлякує користувачів, які шукають простіше рішення.

Ще однією слабкою стороною є обмежений візуальний стиль і відсутність ігрових елементів. Програма виглядає більше як професійний інструмент для досвідчених користувачів, і може не викликати емоційного залучення. Відсутність системи нагород, балів, рівнів або внутрішньої мотивації може негативно впливати на бажання користувача повертатися до програми щодня (рис. 1.3).

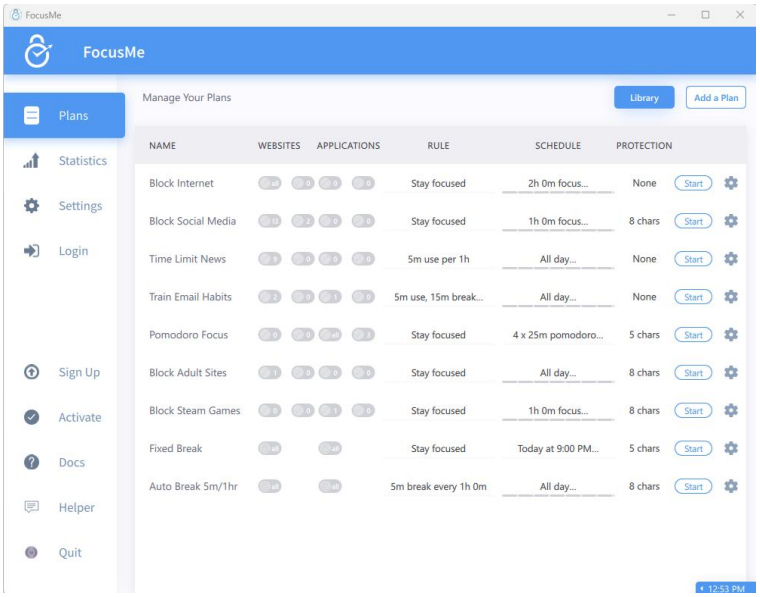

Рисунок 1.3 – Інтерфейс головної сторінки FocusMe

1.3 Інструменти розробки програмного забезпечення

Для реалізації програмного забезпечення, призначеного для контролю часу використання інструментальних засобів, необхідно використовувати надійні та сучасні засоби розробки. У контексті настільних додатків особливої актуальності набувають ті інструменти, які забезпечують ефективну реалізацію логіки, зручне створення інтерфейсу, просте зберігання даних і можливість масштабування в майбутньому.

C# – це високорівнева об’єктно-орієнтована мова програмування, розроблена корпорацією Microsoft. Вона є основною мовою для створення застосунків на платформі .NET, зокрема настільних програм під Windows. Серед переваг C# – зрозумілий синтаксис, підтримка сучасних парадигм програмування, велика кількість бібліотек і фреймворків, а також потужна інтеграція з середовищем Visual Studio. Мова активно використовується в корпоративному секторі, у розробці ігор, вебдодатків і настільних рішень.

.NET – це програмна платформа від Microsoft, яка надає інструменти та бібліотеки для створення додатків будь-якої складності. У проєкті використовується фреймворк фонд презентацій Windows (Windows Presentation Foundation, WPF) – потужний інструмент для створення настільних програм з графічним інтерфейсом.

WPF дозволяє будувати інтерфейси з використанням розширюваної мови розмітки застосунків (Extensible Application Markup Language, XAML), що забезпечує гнучке оформлення вигляду програми, підтримку стилів, шаблонів, анімацій та зв’язування даних. Особливістю WPF є можливість реалізувати архітектуру модель-представлення-модель представлення (Model-View-ViewModel, MVVM), яка розділяє логіку, подання та модель даних, що полегшує підтримку й розширення проєкту.

Для збереження даних у програмі, зокрема інформації про застосунки, завдання, часові обмеження, історію досвіду тощо, використовується легка реляційна база даних (Lightweight Structured Query Language Database,

SQLite). Це вбудована реляційна база даних, яка не потребує окремого сервера й зберігається у вигляді одного локального файлу, що робить її ідеальним рішенням для настільних застосунків.

На відміну від простого зберігання даних у форматі нотацій об’єктів JavaScript (JavaScript Object Notation, JSON), використання SQLite дозволяє надійно структурувати інформацію у вигляді таблиць, встановлювати зв’язки між об’єктами, легко виконувати складні запити, а також забезпечувати більшу безпеку та цілісність даних [8–10]. Завдяки реляційній структурі стає можливим зберігати історію змін, підраховувати статистику, фільтрувати завдання за датою, категорією чи статусом виконання тощо.

Використання SQLite дозволяє проєкту бути масштабованим, розширюваним і надійним – що є важливим фактором для програм, які працюють із чутливою інформацією користувача та повинні зберігати її в безпечному вигляді навіть при тривалому використанні.

1.4 Постановка задачі

Таким чином, проблема надмірного використання цифрових інструментів та виникнення залежності від миттєвого задоволення є актуальною у сучасному суспільстві. Тому ставиться завдання розробки програмного забезпечення для контролю часу використання інструментальних засобів, яке не лише обмежує час доступу до непродуктивних програм, а й мотивує користувача до саморозвитку через гейміфікований інтерфейс, систему досвіду, завдань і навичок.

Об’єктом є процес контролю щодо використання користувачем програмних інструментів у межах повсякденної діяльності.

Метою роботи є розробка програмного забезпечення, що дозволяє здійснювати облік, обмеження та мотиваційне управління часом

використання інструментальних засобів із використанням гейміфікованого підходу.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– здійснити аналіз існуючих програм контролю часу та методів мотивації користувачів;

– обґрунтувати вибір інструментів та архітектури програмного забезпечення;

– розробити структуру бази даних для зберігання інформації про навички, цілі, досвід, часові обмеження тощо;

– реалізувати програмну систему з графічним інтерфейсом на базі технології WPF;

– передбачити модуль гейміфікації: систему досвіду, рівнів, завдань і категорій навичок;

– реалізувати механізм контролю часу використання сторонніх програм та систему обмежень;

– протестувати роботу системи на прикладі типових сценаріїв користування.

2 ПРОЄКТУВАННЯ ФУНКЦІОНАЛЬНОЇ ЧАСТИНИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Аналіз вимог до програмної системи

У сучасному цифровому середовищі дедалі більше людей стикаються з проблемами, пов’язаними з організацією особистого часу, контролем за використанням комп’ютерних ресурсів та зниженням продуктивності. Постійна доступність розважального контенту, соціальних мереж, відеоплатформ і комп’ютерних ігор призводить до зростання рівня цифрової залежності та прокрастинації, що особливо помітно серед молоді та фахівців, які працюють за комп’ютером [11, 12].

З огляду на це, цільовою аудиторією розроблюваного програмного забезпечення є користувачі, що прагнуть досягти більшої особистої ефективності, підвищити самодисципліну та впровадити у своє життя сталі продуктивні звички. До таких користувачів передусім належать студенти, які навчаються в онлайн або гібридному форматі та потребують додаткових засобів самоконтролю для уникнення відволікань під час навчального процесу. Також до цільової аудиторії належать фрилансери та фахівці в галузі інформаційних технологій, які працюють у вільному графіку й самостійно організовують свій робочий час, що часто ускладнює підтримання фокусу та продуктивності.

Окрему групу користувачів становлять особи, які усвідомлюють наявність залежності від цифрових джерел задоволення – таких як YouTube, TikTok, стрімінгові платформи або відеоігри – і прагнуть поступово змінити свої звички на користь більш структурованого та цілеспрямованого способу життя. Водночас, усе більше людей активно займаються саморозвитком і шукають зручні інструменти для постановки цілей, моніторингу власного прогресу, а також підтримки особистої мотивації [13–17].

У цьому контексті потреби користувачів охоплюють декілька важливих аспектів. Серед них – необхідність у точному контролі за витраченим часом, розподілом діяльностей за категоріями та аналізом цифрових звичок. Не менш важливим є наявність мотиваційних механізмів, що сприяють початку і завершенню корисних справ, а також формуванню позитивного підкріплення через систему рівнів, досвіду та візуалізацію особистого прогресу [18–23]. Додатково користувачі очікують від програмного продукту можливості створення та ведення атомарних завдань, прив’язаних до конкретних навичок або життєвих цілей.

Функціональні вимоги визначають, які конкретні дії повинна виконувати система для досягнення поставлених цілей. З огляду на цільову аудиторію та поставленими перед користувачами завданнями, функціональність програмного забезпечення має бути спрямована на забезпечення комплексного підходу до контролю часу, стимулювання продуктивної поведінки та підтримки процесу самодисципліни.

Насамперед, програмний застосунок має реалізовувати механізм автоматичного обліку часу використання програм, що дозволяє фіксувати активність користувача на рівні вікон, які перебувають у фокусі. Це забезпечує збір даних без втручання з боку користувача та створює достовірну аналітику стосовно витраченого часу на різні застосунки чи типи програм – продуктивні, суміжні чи непродуктивні.

Іншою ключовою вимогою є можливість встановлення обмежень на використання певних програм, зокрема розважального характеру. Система повинна дозволяти налаштовувати ліміти за часом або умовами запуску, а також надавати користувачеві попередження та повністю блокувати доступ у випадку перевищення встановлених меж. Такий підхід сприяє зменшенню необдуманого використання часу та формуванню відповідального ставлення до цифрових звичок.

Важливою складовою функціональності є система завдань, яка дозволяє створювати персоналізовані завдання, об’єднувати їх за категоріями

навичок, фіксувати результати та отримувати винагороду у вигляді досвіду або підвищення рівня. Завдяки цьому користувач має змогу відслідковувати свій прогрес, структурувати повсякденні дії та закріплювати нові звички через повторювану практику.

Не менш значущою є реалізація гейміфікаційного шару: система рівнів, досвіду, візуалізації розвитку навичок. Такі механізми слугують інструментом внутрішньої мотивації та позитивного підкріплення. Користувач відчуває досягнення навіть у рамках невеликих кроків, що сприяє емоційному залученню до процесу та переходу від швидкого задоволення до планомірного та якісного.

Крім цього, програмне забезпечення повинно включати інструменти м’якої мотивації, зокрема реалізацію методу «двох хвилин». Цей метод полягає в тому, що користувач отримує підказку почати будь-яку заплановану справу хоча б на дві хвилини – що значно підвищує ймовірність продовження дії після подолання психологічного бар’єру старту.

Додатково система має надавати можливість перегляду статистики, історії активності, зведених графіків за певний період, що дозволяє аналізувати динаміку поведінки, виявляти слабкі місця та підтримувати саморефлексію у користувача.

Нефункціональні вимоги визначають загальні властивості програмного забезпечення, які не стосуються конкретної функціональності, але мають суттєвий вплив на якість, зручність і стабільність роботи системи. У контексті розроблюваного застосунку, ці вимоги є критичними для забезпечення довготривалого та ефективного використання програмного забезпечення користувачем.

Насамперед, одним із пріоритетів є зручність користування. Інтерфейс має бути інтуїтивно зрозумілим, доступним навіть для користувачів без спеціальної технічної підготовки. Особливу увагу слід приділити візуальній структурі, кольоровому оформленню, розміщенню елементів керування та загальній логіці навігації. Враховуючи щоденне використання програми,

надзвичайно важливо, щоб взаємодія з нею була комфортною та не викликала відчуття перевантаження та непорозумінь.

Другим важливим аспектом є продуктивність і швидкодія. Застосунок повинен працювати у фоновому режимі, не створюючи помітного навантаження на систему, особливо в частині обліку активності програм. Всі операції, зокрема зчитування активного вікна, перевірка обмежень або оновлення прогресу, повинні виконуватись швидко й ефективно, не заважаючи основній роботі користувача [24–28].

Надійність та стабільність – ще одна ключова вимога. Застосунок має безперебійно функціонувати протягом тривалого часу, не викликати збоїв або втрати даних у разі непередбачуваних ситуацій (наприклад, раптового завершення роботи системи, вимкнення живлення). Особливо це стосується обліку часу.

З цим пов’язана й вимога збереження даних у безпечному та автономному форматі. Вибір бази даних SQLite як засобу зберігання обумовлений потребою в локальному, захищеному, структурованому зберіганні інформації. Дані мають бути ізольовані від зовнішнього впливу, легко резервуватись, і в разі необхідності – відновлюватись без втрати цілісності.

Розширюваність та легкість супроводу системи також є важливими характеристиками. Архітектура застосунку має бути побудована так, щоб у майбутньому можна було легко додавати нові функції: додаткові модулі, інструменти візуалізації, інші моделі взаємодії з користувачем [29]. Для цього реалізовується розділення відповідальностей, дотримання шаблону MVVM та використання незалежних сервісів.

Окрему увагу слід приділити безпеці – хоча застосунок працює локально, він оперує персональними даними користувача: цілями, звичками, історією активності. Дані повинні бути недоступними для сторонніх додатків, а при бажанні користувача – видалятись повністю або експортуватись у зручному форматі.

2.2 Формування основи для реалізації функцій системи контролю

Однією з ключових функцій програмного забезпечення є фіксація активності користувача за комп’ютером, а саме – визначення, які програми використовуються протягом робочого часу пристрою. Цей процес є основою для подальшого аналізу продуктивності, визначення розподілу часу між різними видами діяльності, а також для ініціювання механізмів обмеження та мотивації.

Методика реалізується через періодичне визначення активного вікна, тобто того, яке перебуває у фокусі користувача в даний момент часу. У середовищі операційної системи Windows ця задача вирішується за допомогою спеціальних функцій програмного інтерфейсу застосунку (Application Programming Interface, API), зокрема «GetForegroundWindow()» та «GetWindowText()» [30], які дозволяють отримати дескриптор активного вікна і назву процесу, що його породив. Завдяки цим інструментам можливо точно і без потреби у втручанні користувача фіксувати, яка програма перебуває у центрі уваги в поточний момент.

Опитування активного вікна відбувається через регулярні інтервали часу (наприклад, кожну секунду) та при зміні активного вікна. Результати фіксуються в локальній базі даних у вигляді записів, що включають назву програми, часову позначку початку активності та її тривалість. Такий підхід забезпечує баланс між точністю та продуктивністю, оскільки не створює суттєвого навантаження на систему.

Важливою особливістю є те, що система не лише збирає дані, але й категоризує їх за типом активності. Наприклад, браузер Google Chrome може бути як інструментом для навчання, так і джерелом розваг – тому користувачу надається можливість самостійно класифікувати застосунки або конкретні назви вікон (наприклад, «Google Chrome» як суміжне, «Dota 2» як непродуктивне, а «Visual Studio» як продуктивне). Це дозволяє сформувати більш гнучку і точну модель обліку часу.

На основі зібраної інформації система надалі виконує інші функції: формує зведення, активує обмеження, відображає графіки продуктивності. У подальшому розвитку застосунку зібрані дані також можуть бути використані для виявлення пікових періодів концентрації, розпізнавання непродуктивних шаблонів поведінки та формування рекомендацій.

Фіксація активних програм є відносно простою, але концептуально важливою складовою системи, що забезпечує об’єктивну основу для прийняття рішень, побудови звітності та реалізації мотиваційних механізмів у рамках застосунку.

У контексті розробки програмного забезпечення для самоконтролю особливої актуальності набуває питання не лише обмеження доступу до відволікаючих програм, а й підтримки внутрішньої мотивації користувача. Одним із найефективніших підходів до вирішення цього завдання є впровадження елементів гейміфікації, що дозволяє перетворити рутинні або складні дії на емоційно привабливу взаємодію.

Гейміфікація – це процес застосування ігрових механік у неігровому контексті. У межах запропонованої системи вона реалізується через систему досвіду, рівнів, навичок та нагород, яка дає змогу користувачеві відчувати прогрес, навіть виконуючи незначні або звичні завдання. На практиці це виражається у наступному: за кожне успішно виконане завдання користувач отримує певну кількість досвіду, що накопичується у межах вибраної навички, наприклад: «Програмування», «Малювання», «Англійська мова».

По мірі накопичення досвіду користувач підвищує рівень навички, що візуально підкріплюється відповідними індикаторами: шкалами прогресу, рівнями. Це створює ефект досягнення, навіть у разі незначного просування, та формує позитивне підкріплення для повторення корисної дії [31]. Такий підхід відповідає психологічним механізмам звикання: мотивація виникає не лише від результату, а й від самого процесу розвитку.

Окрім цього, структура навичок дозволяє персоналізувати мотивацію, адже кожен користувач самостійно визначає, що саме він хоче прокачати. Це

створює емоційний зв’язок з обраним напрямком розвитку, що значно підвищує шанс повернення до розвинення навички в майбутньому. Наприклад, користувач, який прокачує «Навичку C#», має більшу ймовірність повернутися до навчання, аби досягти наступного рівня.

На відміну від жорстких обмежень, гейміфікація не викликає опору або фрустрації. Вона базується на внутрішній мотивації, бажанні досягати, порівнювати, ставити власні цілі. Система також дозволяє візуалізувати загальний прогрес, відображати добові статистики, що додає прозорості та відчуття зростання.

Таким чином, гейміфікаційна модель не лише стимулює продуктивність, а й формує довгострокову мотивацію, яка базується не на примусі, а на залученні. Завдяки цьому користувач залишається зацікавленим, повертається до взаємодії із системою й поступово вибудовує нові звички на основі добровільного рішення та власної ініціативи.

Однією з ключових складових внутрішньої логіки застосунку є система завдань, яка виступає в ролі посередника між повсякденною діяльністю користувача та його цілеспрямованим розвитком. Саме через виконання завдань реалізується поступове накопичення досвіду, прокачування навичок та побудова продуктивних моделей поведінки.

Завдання у системі подаються у вигляді конкретних дій, які користувач створює самостійно (наприклад: «Вивчити 20 слів англійською», «Прочитати 10 сторінок», «Пописати код 30 хвилин»). Кожне завдання пов’язане з певною категорією навичок – абстрактною або прикладною сферою діяльності, яку користувач прагне розвивати, наприклад: «Програмування», «Англійська мова», «Малювання», тощо. Це дозволяє структурувати цілісну картину розвитку, де кожна дія не є ізольованою, а інтегрована в загальну систему саморозвитку.

Виконання завдань винагороджується досвідом, який накопичується у відповідній навичці. Коли обсяг досвіду досягає заданого порогу, навичка підвищує рівень. Така система рівнів виконує роль мотиваційного

каталізатора, оскільки користувач бачить конкретний прогрес у вигляді шкал, цифр. Це формує ефект досягнення, що сприяє регулярному поверненню до розвитку та роботи із застосунком зокрема.

Особливістю системи завдань є її адаптивність: користувач сам обирає інтенсивність, складність і кількість щоденних цілей. Програма не нав’язує фіксованих сценаріїв, а виступає у ролі помічника, який гнучко підлаштовується під індивідуальний стиль життя. Такий підхід сприяє саморегуляції, коли відповідальність за досягнення цілей переноситься на самого користувача, але підтримується інструментами візуалізації й заохочення.

Важливо підкреслити, що система завдань і навичок не ізольована, а глибоко інтегрована з іншими модулями застосунку – гейміфікацією, статистикою, обмеженнями. Наприклад, після завершення завдання, яке потребувало зосередженості, система може автоматично порівняти витрачений час із фактичним використанням програм, що дозволяє проводити своєрідний перехресний контроль і підвищує усвідомленість користувача.

Серед багатьох психологічних підходів до подолання прокрастинації та стимулювання до дії метод «двох хвилин» є одним із найпростішим, але водночас ефективним інструментом. Його суть полягає в тому, що будь-яке завдання, яке можна виконати за дві хвилини або почати протягом цього часу, слід зробити негайно. Цей підхід базується на принципі зменшення початкового бар’єру до виконання справи, що особливо важливо в умовах цифрового перевантаження та дефіциту сили волі.

У межах розроблюваного застосунку метод «двох хвилин» реалізується як частина інтерфейсної взаємодії, що активується в момент, коли користувач відкладає завдання або уникає переходу до продуктивної активності. Наприклад, у разі надто тривалого перебування в розважальних програмах система може запропонувати просту альтернативу: «Спробуйте попрацювати 2 хвилини над запланованим завданням». Така підказка не є

примусовою, але створює психологічний імпульс, достатній для переходу до дії.

Психологічна ефективність методу пов’язана з тим, що користувач не відчуває тиску великої мети або складності завдання – лише дві хвилини зосередженої дії. На практиці це часто призводить до того, що після початку виконання користувач не зупиняється, а продовжує працювати значно довше. Таким чином, система ініціює позитивний цикл залучення, який запускається не зовнішньою силою, а м’яким нагадуванням.

Крім того, цей метод є частиною загальної мотиваційної логіки програми, яка не обмежується контролем і заборонами, а створює умови для психологічного комфорту і м’якого втягнення у діяльність. Його можна комбінувати з завданнями, досвідом, навичками – наприклад, запускати завдання в «режимі 2-х хвилин», після чого продовжувати у звичайному режимі з нарахуванням досвіду.

Технічно реалізація цього підходу не вимагає складних алгоритмів: достатньо відслідковувати бездіяльність або час перебування в застосунках не пов’язаних з завданнями, після чого відображати модальне вікно з пропозицією. Проте ефект цього методу полягає не в складності реалізації, а в його емоційному та когнітивному впливі на користувача, що підтверджується численними дослідженнями в галузі самоменеджменту [32].

Застосування методу «двох хвилин» у структурі програми дозволяє зробити взаємодію не лише ефективною, а й людяною: вона підтримує користувача, а не тисне на нього, допомагає почати замість карати за бездіяльність. Такий підхід особливо важливий для довготривалого користування системою та формування сталих продуктивних звичок.

Одним із ключових бар'єрів на шляху до самодисципліни та цілеспрямованої поведінки є явище уникнення дискомфорту. Цей механізм має глибокі психологічні корені й часто проявляється у формі відкладання складних або неприємних завдань, навіть за умови усвідомлення їхньої важливості. У контексті цифрового середовища така поведінка посилюється

миттєвими джерелами задоволення – відео, іграми, стрічками соцмереж, – які забезпечують негайне полегшення й відволікають від необхідної, але енергозатратної діяльності.

Психологічна модель уникнення дискомфорту базується на принципі короткострокової вигоди: наш мозок схильний уникати будь-якого стресу, навіть якщо в довгостроковій перспективі це шкодить досягненню цілей. Саме тому користувачі часто уникають відкривати «складні» додатки (наприклад, інтегроване середовище розробки (Integrated Development Environment, IDE), навчальні матеріали), натомість запускають YouTube, TikTok або ігри, що не потребують зусиль. Згодом це призводить до формування деструктивних звичок, зниження продуктивності та почуття провини.

У межах розроблюваного застосунку ця модель враховується як базовий психологічний механізм, на подолання якого спрямовані функції програми. Поєднання гейміфікації, методу двох хвилин, системи завдань та м’яких нагадувань формує альтернативну поведінкову модель, де замість уникнення користувач отримує безпечний і підтримувальний простір для дії. Наприклад, коли система фіксує тривале перебування в непродуктивній програмі, вона не просто обмежує доступ, а пропонує заміну – невелике завдання, яке легко почати.

Важливо також, що програма надає візуальний зворотний зв’язок, що дозволяє користувачеві бачити ефект виконаних дій: досвід, прогрес, підвищення рівня навичок. Це знижує рівень тривоги перед початком роботи й допомагає формувати позитивні асоціації з дискомфортом, що з часом трансформується в звичку долати перешкоди замість уникати їх.

Окрім цього, сама структура взаємодії з програмою не нав’язує тиску. Вона побудована на принципі добровільного включення, де користувач сам визначає ритм, обирає пріоритети та контролює рівень складності. Завдяки цьому замість опору виникає внутрішнє залучення, яке є більш стійким до психологічного виснаження.

Урахування моделі уникнення дискомфорту в логіці застосунку дозволяє не лише підвищити ефективність користувача, а й створити підґрунтя для довготривалої трансформації його поведінки, що є одною з цілей застосунку.

2.3 Особливості моделювання та атрибути даних

При проєктуванні бази даних для настільного застосунку з контролю часу використання інструментальних засобів основною задачею є забезпечення структурованого, надійного та розширюваного зберігання інформації про активність користувача, виконані завдання, розвиток навичок, а також налаштовані обмеження. Архітектура бази даних має дозволяти швидкий доступ до статистичних даних, зберігання історичних записів та відображення прогресу користувача.

Зважаючи на специфіку проєкту – тобто збереження зв’язаної інформації про завдання, досвід, обмеження, активність тощо – оптимальним вибором є реляційна база даних з чітко визначеною схемою, яка гарантує цілісність та несуперечність даних. Для цього було обрано SQLite – легковагову вбудовану систему управління базами даних (СУБД) [33], яка ідеально підходить для настільних застосунків, не вимагає налаштування сервера та зберігає всю інформацію у вигляді одного локального файлу.

Попри популярність NoSQL-рішень у задачах із великими обсягами неструктурованої інформації, саме реляційна модель краще відповідає вимогам строгої типізації сутностей, наявності зв’язків між об’єктами та забезпечення транзакційної цілісності. SQLite також підтримує транзакції, зовнішні ключі, фільтрацію та агрегацію, що є критично важливими для побудови логіки продуктивності [13–15], аналізу звичок користувача й реалізації історії змін.

На основі попереднього аналізу функціональних вимог, а також описаних у попередньому розділі сутностей, структура бази даних застосунку передбачає моделювання таких основних компонентів:

– дані про навички та їх категорії;
– завдання та історію їх виконання;
– обмеження на використання програм;
– записи про активність користувача;
– логіку накопичення досвіду.

Однією з найважливіших частин реалізації інформаційної системи є розробка концептуальної моделі предметної області, що охоплює всі ключові сутності та взаємозв’язки між ними. У контексті розробленого застосунку сутності відображають основні напрямки діяльності користувача: розвиток навичок, виконання завдань, накопичення досвіду, фіксацію взаємодії із іншими застосунками та регулювання цифрової поведінки через систему обмежень.

На основі логіки системи та вимог до функціональності було виділено низку базових таблиць, реалізованих у реляційній структурі бази даних. Усі сутності взаємопов’язані через зовнішні ключі, що забезпечує цілісність, уникнення дублювання інформації та спрощення обробки пов’язаних даних у бізнес-логіці програми [16–19].

У центрі системи знаходиться концепція набуття та вдосконалення навичок, що здійснюється шляхом виконання користувачем відповідних завдань. Для представлення навичок та їх класифікації використовується сутність «Категорії навичок» (табл. 2.1), яка дозволяє групувати навички за тематичними напрямами (наприклад, програмування, мистецтво, навчання тощо). Таким чином система транслює фактичний розвиток користувача із реального життя в цифрове середовище, забезпечуючи наочне відображення динаміки його прогресу за допомогою кількісних показників. Це дозволяє користувачу одразу бачити конкретні результати своєї діяльності та ефективніше оцінювати власні досягнення.

Таблиця 2.1 – Атрибути таблиці SkillCategory

Назва	**Тип**	**Опис**
id	INTEGER	Унікальний ідентифікатор категорії
name	TEXT	Назва категорії навичок
color	TEXT	Колір категорії
sort_order	INTEGER	Порядковий номер для сортування в інтерфейсі користувача

Кожна категорія містить набір навичок, що зберігаються у таблиці «Навички» (табл. 2.2). Навичка є основною одиницею розвитку та прив’язки завдань. Вона має назву, поточний рівень (який зростає зі збільшенням досвіду) та числовий показник накопиченого досвіду.

Таблиця 2.2 – Атрибути таблиці Skill

Назва	**Тип**	**Опис**
id	INTEGER	Унікальний ідентифікатор навички
name	TEXT	Назва навички
category_id	INTEGER	Зовнішній ключ на таблицю категорій
level	INTEGER	Поточний рівень
xp	INTEGER	Загальна кількість досвіду

Для розвитку навичок користувач створює завдання (табл. 2.3) – конкретні активності з навчальною або практичною ціллю. Кожне завдання має атрибути, що дозволяють відстежити його статус, складність, очікувану тривалість і пов’язану навичку. Такий підхід дозволяє системі автоматично розраховувати вплив виконання завдання на прогрес навички. Крім цього, зв’язок із програмами, у яких виконується завдання, дає змогу об’єктивно оцінити активний час і враховувати лише реальну взаємодію з інструментом. Це сприяє формуванню відповідального підходу до самонавчання та дає користувачу зворотний зв’язок про ефективність витраченого часу.

Таблиця 2.3 – Атрибути таблиці TaskItem

Назва	Тип	Опис
id	INTEGER	Унікальний ідентифікатор завдання
title	TEXT	Назва завдання
description	TEXT	Текстовий опис
skill_id	INTEGER	Зовнішній ключ на навичку
estimated_minutes	INTEGER	Очікувана тривалість виконання
difficulty	INTEGER	Складність за шкалою від 1 до 10
date_created	TEXT	Дата створення
completed	BOOLEAN	Статус виконання

Результатом успішного виконання завдання є нарахування досвіду, який фіксується у таблиці історії досвіду (табл. 2.4). Це дозволяє зберігати історичні дані про розвиток користувача, відстежувати динаміку зростання та використовувати ці дані для побудови графіків і звітності.

Таблиця 2.4 – Атрибути таблиці ExperienceHistory

Назва	Тип	Опис
id	INTEGER	Унікальний ідентифікатор запису
task_id	INTEGER	Зовнішній ключ на виконане завдання
xp_earned	INTEGER	Нарахований досвід
recorded_at	TEXT	Часова мітка завершення завдання

Одна з унікальних функцій системи – це автоматичне розрахування досвіду на основі активної взаємодії користувача із заданими програмами. Користувач може пов’язати конкретне завдання з однією або кількома програмами, в яких очікується його виконання.

Інформація про програми, які відслідковуються, зберігається у таблиці «Відслідковуванні застосунки» (табл. 2.5).

Таблиця 2.5 – Атрибути таблиці TrackedProgram

Назва	Тип	Опис
id	INTEGER	Унікальний ідентифікатор програми
name	TEXT	Назва, що відображається
identifier	TEXT	Назва процесу
category	TEXT	Категорія
is_tracked	BOOLEAN	Чи активно відстежується
is_hidden	BOOLEAN	Чи прихована програма в інтерфейсі

Для реалізації зв’язку між завданнями та програмами застосовується таблиця «Завдання-застосунок», що реалізує відношення типу «багато-до-багатьох» (табл. 2.6).

Таблиця 2.6 – Атрибути таблиці TaskProgram

Назва	Тип	Опис
task_id	INTEGER	Зовнішній ключ на завдання
program_id	TEXT	Зовнішній ключ на застосунок

На додаток, для кожної зв’язки зберігається фактична кількість активного часу, що була врахована в рамках певного завдання. Це дозволяє уникати подвійного обліку, якщо одна й та ж програма використовувалась у кількох завданнях (табл. 2.7).

Таблиця 2.7 – Атрибути таблиці TaskProgramUsage

Назва	Тип	Опис
id	INTEGER	Унікальний ідентифікатор запису
task_id	INTEGER	Зовнішній ключ на завдання
program_id	INTEGER	Зовнішній ключ на програму
counted_active_minutes	INTEGER	Зарахований активний час у хвилинах
initial_active_seconds	INTEGER	Час на момент створення завдання

Крім локального часу в завданнях, система також веде глобальну історію використання програм. Таблиця «Архів використання застосунків» (табл. 2.8) зберігає дані про загальний час фокусування та активної взаємодії для кожної програми за кожен день.

Таблиця 2.8 – Атрибути таблиці ProgramUsageLog

Назва	**Тип**	**Опис**
id	INTEGER	Унікальний ідентифікатор запису
program_id	INTEGER	Зовнішній ключ на застосунок
date	TEXT	Дата
counted_active_minutes	INTEGER	Зарахований активний час у хвилинах
total_duration	INTEGER	Загальний час у фокусі
active_duration	INTEGER	Активний час

Щоб забезпечити самоконтроль користувача, система реалізує гнучкий механізм обмежень. Основна таблиця «Обмеження» (табл. 2.9) зберігає загальну інформацію, а таблиця «Правило обмеження» (табл. 2.10) описує конкретні умови.

Таблиця 2.9 – Атрибути таблиці Restriction

Назва	**Тип**	**Опис**
id	INTEGER	Унікальний ідентифікатор
note	TEXT	Примітка або опис обмеження

Таблиця 2.10 – Атрибути таблиці RestrictionRule

Назва	**Тип**	**Опис**
id	INTEGER	Унікальний ідентифікатор правила
restriction_id	INTEGER	Зовнішній ключ на обмеження
rule_type	TEXT	Тип правила
value	INTEGER	Значення параметра

Зв’язок обмежень з програмами реалізується у таблиці «Обмеження-застосунок» (табл. 2.11).

Таблиця 2.11 – Атрибути таблиці RestrictionToProgram

Назва	Тип	Опис
id	INTEGER	Унікальний ідентифікатор запису
restriction_id	INTEGER	Зовнішній ключ на обмеження
program_id	TEXT	Зовнішній ключ на застосунок

На рисунку 2.1 представлено структурну діаграму бази даних застосунку FocusTracker, побудовану за допомогою мови опису DBML. Схема відображає всі основні сутності, їх атрибути та зв’язки між ними.

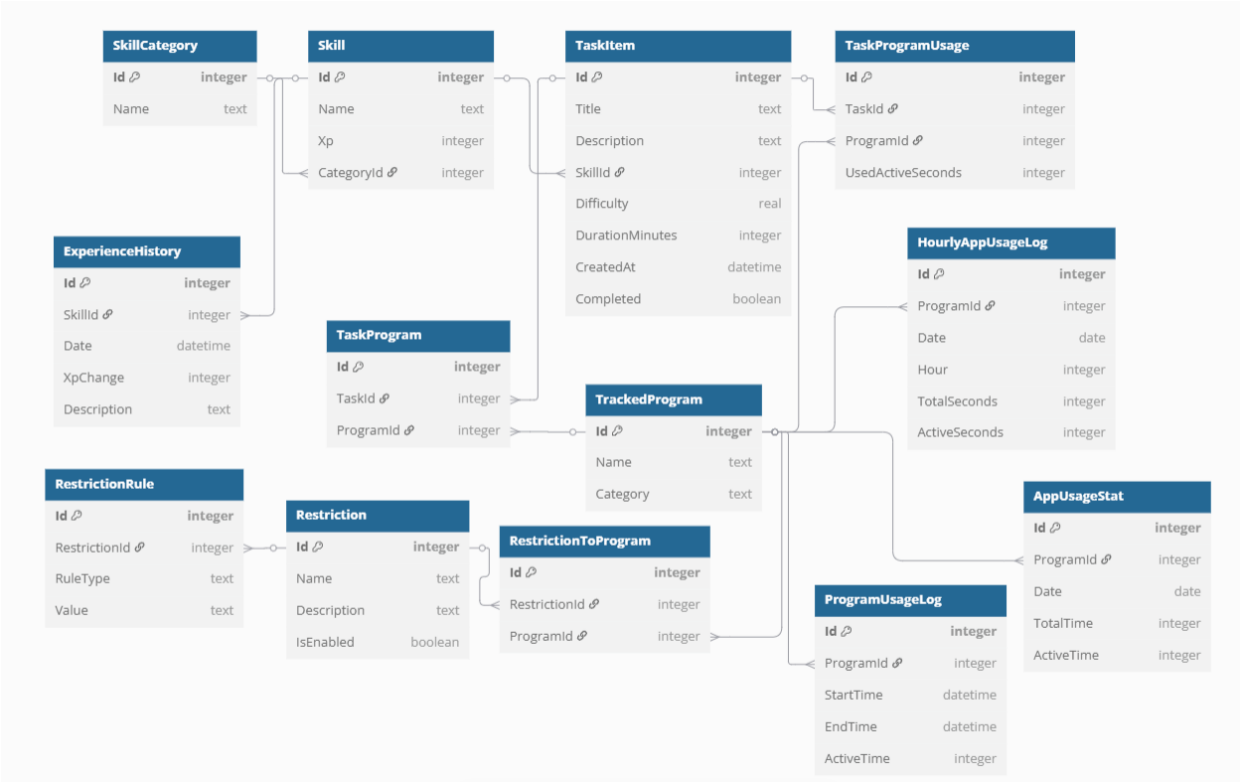

Рисунок 2.1 – Загальна структура бази даних

3 РОЗРОБКА ПРОГРАМНОГО ЗАСТОСУНКУ ДЛЯ КОНТРОЛЮ І КЕРУВАННЯ ДІЯМИ КОРИСТУВАЧА

3.1 Вибір інструментальних засобів для реалізації поставленої задачі

Для реалізації програмного забезпечення було обрано набір інструментальних засобів, що забезпечують ефективну розробку, стабільну роботу застосунку, зручний графічний інтерфейс користувача та надійне зберігання даних. Вибір засобів ґрунтувався на вимогах до продуктивності, простоти підтримки, можливості масштабування та сумісності з сучасними технологіями.

Основною мовою програмування було обрано C#, яка забезпечує високу продуктивність, підтримку об’єктно-орієнтованого підходу, типізацію та широкі можливості для побудови складних застосунків. Завдяки тісній інтеграції з платформою .NET, мова C# дозволяє реалізовувати застосунки різного рівня складності – від консольних до повноцінних графічних інтерфейсів.

Як платформу для розробки графічного інтерфейсу було обрано Windows Presentation Foundation (WPF) – сучасну технологію від Microsoft, що дозволяє створювати адаптивні, функціональні та візуально привабливі настільні додатки для операційної системи Windows. WPF підтримує декларативну розмітку інтерфейсу за допомогою мови XAML, а також забезпечує зручне розділення логіки представлення та бізнес-логіки.

У якості середовища розробки використовувалась Visual Studio 2022, яка є потужним інструментом для програмування на C# та роботи з проєктами на базі .NET. Visual Studio підтримує інтелектуальну підсвітку синтаксису, налагодження, керування залежностями та інші корисні функції, що підвищують ефективність розробки.

Для зберігання структурованих даних застосунку було обрано SQLite – легковагу вбудовану реляційну систему керування базами даних. Вона не

потребує окремого серверного середовища, що робить її ідеальним вибором для настільних програмного забезпечення. SQLite зберігає усі дані в одному локальному файлі, забезпечує швидкий доступ до інформації, підтримує транзакції та дозволяє використовувати стандартний SQL-синтаксис.

Для обробки даних бази та взаємодії з нею було застосовано Entity Framework Core – об’єктно-реляційну ORM-технологію, яка дозволяє розробнику працювати з базою даних через об’єкти мови C#, автоматично створюючи необхідні SQL-запити. Це значно спрощує розробку, зменшує кількість шаблонного коду та підвищує безпеку взаємодії з даними.

Для побудови візуалізацій – графіків продуктивності, динаміки використання програм тощо – було обрано бібліотеку «LiveChartsCore.SkiaSharpView.WPF», яка забезпечує гнучке налаштування компонентів, підтримку інтерактивності та високий рівень продуктивності при роботі з великими обсягами даних.

Для створення і документування структури бази даних використовувався онлайн-інструмент dbdiagram.io, що дозволяє у зручному форматі DBML описувати таблиці та зв’язки між ними, автоматично генеруючи діаграму.

3.2 Етапи розробки застосунку

3.2.1 Модуль завдань та система досвіду

Функціональність управління завданнями у програмі поєднує можливість постановки короткострокових цілей користувача та механізм автоматичного нарахування досвіду за фактичну взаємодію з визначеними застосунками. Завдання є основною одиницею роботи в системі: вони створюються вручну, прив’язуються до певної навички та одного або кількох застосунків, у яких користувач планує працювати.

Під час створення завдання зберігається момент часу, а також фіксується початкове значення активного часу для кожної вибраної програми. Це дозволяє згодом визначити приріст активного часу, який стався після створення завдання, і використовувати лише його для обчислення досвіду. Завдяки цьому враховується виключно нова продуктивна взаємодія, а не загальна історія використання застосунків.

Однак у реальних сценаріях можливе створення кількох завдань, що посилаються на одну й ту ж програму. Щоб уникнути дублювання досвіду, система додатково зберігає інформацію про вже зарахований активний час у рамках попередніх завершених завдань. Таким чином, кожне нове завдання враховує лише той активний час, який ще не був використаний у жодному іншому завданні. Це забезпечує точність і чесність механіки нарахування досвіду [20–22].

Нарахування досвіду виконується лише в момент завершення завдання. Коли користувач натискає кнопку «Завершити», програма виконує таку послідовність дій:

Крок 1. Отримати список усіх програм, прив’язаних до завдання.

Крок 2. Для кожної програми визначити активний час з моменту створення завдання.

Крок 3. Відняти із цього значення ті хвилини, які вже були зараховані раніше у рамках інших завершених завдань, якщо такі булі створені до створення поточної задачі.

Крок 4. Підсумувати активний час по всіх програмах.

Крок 5. Якщо активний час дорівнює нулю – вивести припинити процедуру та не зараховувати виконання завдання.

Крок 6. Обчислити досвід за формулою:

$$\text{досвід} = \text{активні хвилини} \times \text{складність}.$$

Крок 7. Додати обчислений досвід до відповідної навички.

Крок 8. Зберегти використаний активний час для цього завдання, щоб уникнути повторного врахування в майбутньому.

Крок 9. Позначити завдання як завершене.

Це дозволяє уникнути формального виконання завдань – досвід буде нараховано лише за реальну роботу.

Якщо під час обчислення буде досягнуто порогове значення для переходу на новий рівень навички – рівень автоматично підвищується, а прогрес починає накопичуватися до наступного порогу.

Цей підхід дозволяє ефективно пов’язати досягнення користувача з його реальними діями у визначених програмах, забезпечуючи об’єктивність нарахування досвіду. Далі розглядається фрагмент програмного коду, який реалізує механізм завершення завдання та обчислення досвіду. Його подано у лістингу 3.1.

Лістинг 3.1 Логіка завершення завдання та нарахування досвіду:

```
var stats = await _appUsageStatService.GetStatsForDay(task.DateCreated);
var usedMinutes = await
_taskUsageService.GetUsedMinutesForTask(task.Id);

var newMinutes = stats
   .Where(s => task.ProgramIds.Contains(s.ProgramId))
   .Sum(s => Math.Max(0, s.ActiveMinutes -
usedMinutes.GetOrDefault(s.ProgramId)));

if (newMinutes <= 0)
   return Notify("Недостатньо активності для завершення завдання.");

var xp = newMinutes * task.Difficulty;
await _skillService.AddXpAsync(task.SkillId, xp);
await _experienceHistoryService.CreateAsync(task.Id, xp);
```

```
await _taskUsageService.SaveUsedMinutesAsync(task.Id, stats);
task.Completed = true;
```

Таким чином, модуль завдань виконує функції:

– постановку і збереження цілей на день;

– прив’язку завдань до навичок і програм;

– облік реального активного часу після створення завдання;

– захист від дублювання досвіду при паралельних задачах;

– автоматичне нарахування досвіду з урахуванням складності;

– підвищення рівня навичок на основі досягнень.

3.2.2 Модуль фіксування активності в програмах

Ефективність системи самоконтролю значною мірою залежить від здатності точно відстежувати час, проведений користувачем у різних програмах. Для досягнення цієї мети в застосунку реалізовано механізм фонової фіксації активності, який забезпечує облік не лише факту запуску програм, а й реальної взаємодії з ними.

На відміну від класичних таймерів або хронометрів, які фіксують лише загальний час, розроблений модуль збирає два незалежні типи статистики:

– час фокусування – коли вікно програми знаходиться у передньому плані;

– активний час – коли користувач здійснює дії мишкою або клавіатурою.

Цей підхід дозволяє відокремити пасивну присутність програми на екрані (наприклад, відкрите, але не використовуване вікно браузера) від справжньої роботи в ній. Технічно це реалізовано як окремий компонент, що перевіряє поточну активну програму та рівень активності кожні 5 секунд, після чого оновлює відповідні записи в базі даних.

Всі показники зберігаються у вигляді агрегованих добових значень, об’єднаних за назвою програми та датою. Це дозволяє ефективно зберігати історію без надмірного навантаження на базу даних і, водночас, формувати зведені графіки продуктивності.

Користувач сам визначає перелік програм, які повинні бути враховані. Кожній із них присвоюється категорія – продуктивна, суміжна або непродуктивна – що надалі впливає на обчислення підсумкових результатів. Окремо передбачена можливість приховати програму або повністю виключити її з фіксацію, що дозволяє адаптувати систему під індивідуальні потреби.

Зібрані дані не лише використовуються для побудови статистики, а й безпосередньо впливають на інші модулі – наприклад, при завершенні завдань або перевірці активності в рамках обмежень. Завдяки цьому, трекінг програм є центральною частиною архітектури, яка пов’язує всі ключові сценарії взаємодії в системі [23–25].

Цей підхід дозволяє здійснювати фонове оновлення статистики в режимі реального часу, без потреби взаємодії з боку користувача. Нижче наведено фрагмент коду, що відповідає за фіксацію загального та активного часу у програмах. Його подано у лістингу 3.2.

Лістинг 3.2 Фонове оновлення статистики використання програм:

```
var appName = _windowProvider.GetActiveWindowTitle();
var isActive = _userActivityTracker.IsUserActive();

var existing = await _db.AppUsageStats
    .FirstOrDefaultAsync(x => x.AppName == appName && x.Date == 
today);

if (existing == null)
{
```

```
        existing = new AppUsageStat
        {
            AppName = appName,
            Date = today,
            TotalTime = TimeSpan.Zero,
            ActiveTime = TimeSpan.Zero
        };
        _db.AppUsageStats.Add(existing);
    }

    existing.TotalTime += TimeSpan.FromMinutes(1);
    if (isActive)
        existing.ActiveTime += TimeSpan.FromMinutes(1);

    await _db.SaveChangesAsync();
```

Цей код виконується щосекундно у фоновому режимі й оновлює відповідний запис у базі даних. Він відображає як загальну тривалість перебування вікна на екрані, так і активну взаємодію, що дозволяє системі точно враховувати продуктивність користувача.

3.2.3 Модуль обмежень

Для підтримки самоконтролю в застосунку реалізовано систему обмежень, яка дозволяє автоматично блокувати доступ до вибраних програм за заданими умовами. Користувач може створити одне або кілька обмежень, для кожного з яких задаються цільові програми та набір правил. Підтримуються такі типи:

– ліміт добового використання – обмежує загальну тривалість роботи в програмі протягом дня;

– заборонені часові інтервали – визначає періоди доби, в які програма не повинна використовуватись;

– після виконання завдань – дозволяє запуск програм лише після накопичення достатньої кількості активного часу у зарахованих завданнях.

Кожне правило оцінюється окремо, і якщо хоча б одне з них не виконується – доступ до програми блокується негайно. Перевірка виконується не періодично, а одразу при запуску або під час активної роботи програми. Це дозволяє уникнути ситуацій, коли користувач встигає зловживати часом до наступного циклу перевірки.

З технічної точки зору, при кожному зверненні до цільової програми система отримує її назву, перевіряє всі активні обмеження, і порівнює фактичну статистику з умовами. Якщо порушення виявлено – програма автоматично завершується, а користувач бачить повідомлення з коротким поясненням. Завдяки такій схемі блокування працює миттєво, без потреби в зовнішніх інструментах або фонових службах.

Програмна логіка перевірки реалізована в окремому компоненті, який оцінює всі умови в режимі реального часу. Нижче наведено фрагмент відповідної перевірки. Його подано у лістингу 3.3.

Лістинг 3.3 Фонове оновлення статистики використання програм:

```
var restrictions = await
_restrictionService.GetRestrictionsForProgramAsync(programName);

foreach (var restriction in restrictions)
{
   var rules = await
_ruleService.GetRulesForRestrictionAsync(restriction.Id);
   foreach (var rule in rules)
```

```
        {
            if (!await _ruleEvaluator.IsRuleSatisfiedAsync(rule, programName))
            {
                _notifier.Notify($"Обмеження: {restriction.Note}");
                _processKiller.KillProgram(programName);
                return;
            }
        }
    }
```

Система підтримує можливість створення кількох обмежень для одних і тих самих програм. Це дає змогу задавати до кожного застосунку одразу кілька умов – наприклад, одночасно обмежити час використання та заборонити роботу вночі. Така реалізація дозволяє зберегти гнучкість і масштабованість, оскільки обмеження залишаються незалежними один від одного, а їх логіка перевіряється окремо. Це забезпечує простоту конфігурації для користувача і водночас стабільну роботу системи в складних сценаріях.

Загалом, модуль обмежень виконує не лише контролюючу функцію, а й формує корисну звичку дотримуватись встановлених рамок, надаючи користувачу простий і зрозумілий механізм цифрової самодисципліни.

3.2.4 Модуль статистики та візуалізації активності

Для усвідомленого керування часом користувач повинен мати можливість переглядати детальну статистику своєї цифрової активності. У програмі реалізовано модуль, що підсумовує дані використання програм та відображає їх у зручному візуальному форматі.

Дані зберігаються у вигляді добових записів для кожної програми. Така структура дозволяє зберігати повну історію активності без надмірного навантаження на базу, а також формувати зведену аналітику за будь-який день. Для зручності користувача передбачено фільтр за датою.

Усі показники показуються у вигляді горизонтального графіка, що розбито на 24 години. Кожна година відображає загальний обсяг активного часу, розфарбований відповідно до категорій програм:

– зелений – продуктивні;

– жовтий – суміжні;

– червоний – непродуктивні.

Це дозволяє швидко оцінити, коли й на що було витрачено найбільше часу. Додатково реалізовано легенду з поясненням кольорів і підказки при наведенні на кожну колонку.

Нижче графіка розміщується таблиця з усіма відстежуваними програмами за обраний день. Для кожної з них показується загальний активний час і візуальний прогрес-бар. Колір заповнення також відповідає категорії програми, що забезпечує візуальну узгодженість у всьому інтерфейсі.

У статистику включаються лише ті програми, які користувач явно додав до відстеження. Це виключає зайві записи та фокусує увагу лише на важливому. Завдяки цьому користувач може швидко визначити власні зони продуктивності, виявити періоди прокрастинації та коригувати свою поведінку на основі об’єктивних даних [26–28].

3.2.5 Керування відстеженням програм та система сповіщень

Для зручності користувача в застосунку реалізовано модуль управління відстежуваними програмами. Кожна програма, яку система виявляє, може бути окремо налаштована. Основні доступні параметри:

– категорія програми – продуктивна, нейтральна або непродуктивна;

– статус відстеження – користувач може вимкнути фіксування конкретної програми, якщо не хоче включати її до аналізу;

– приховування – програма може бути прихована зі списку без повного вимкнення відстеження.

Ці параметри зберігаються локально та застосовуються автоматично при наступних запусках програми, що дозволяє один раз налаштувати поведінку і надалі не змінювати її вручну.

Ще одним важливим елементом є система сповіщень. Вона повідомляє користувача про наближення до обмежень, надсилаючи повідомлення у таких випадках:

– за 10 хвилин до досягнення ліміту часу;

– за 5 хвилин до початку забороненого інтервалу;

– у разі автоматичного закриття програми через порушення.

Повідомлення реалізовано через локальні діалогові вікна або системні тости залежно від платформи, і можуть бути легко розширені у майбутньому.

Для зручності використання застосунок підтримує режим роботи у фоновому режимі з розміщенням значка в області сповіщень операційної системи. При натисканні на кнопку закриття вікна програма не завершується повністю, а згортається у цю область. Такий підхід дозволяє зберегти активність фонових модулів, зокрема відстеження часу, перевірки обмежень і генерації сповіщень. Повне завершення роботи здійснюється через контекстне меню, яке викликається правою кнопкою миші по іконці застосунку в області сповіщень.

Завдяки цим функціям застосунок не лише відстежує продуктивність, але й адаптується до користувацьких звичок, забезпечуючи комфортне середовище для формування самоконтролю та дисципліни.

3.3 Архітектура проєкту

У процесі розробки програмного забезпечення важливу роль відіграє архітектурне планування – правильне розмежування відповідальностей між компонентами, логічне групування функціоналу та чітке визначення меж між шарами. Такий підхід дозволяє не лише підвищити стабільність коду, але й істотно спрощує його підтримку, масштабування, а також повторне використання в межах інших проєктів.

Розроблений застосунок є настільною програмою з графічним інтерфейсом, створеною на платформі .NET 8 із використанням WPF. В основі архітектурного рішення лежить шаблон MVVM, який дозволяє чітко відокремити логіку обробки даних від графічного інтерфейсу. Це забезпечує гнучкість, тестованість і простоту в розширенні функціональності.

Уся структура програмного забезпечення реалізована у вигляді рішення, що включає чотири окремі проєкти, логічно ізольовані один від одного (рис. 3.1):

– FocusTracker.App – клієнтський застосунок, який містить графічний інтерфейс користувача (XAML), сторінки, ресурси, стилі та моделі подання (ViewModel). Саме в цьому модулі реалізується навігація між сторінками, логіка обробки подій, візуальні компоненти та анімації. Усі дії, які ініціює користувач, надходять у ViewModel, де обробляються відповідними командами та прив’язаними властивостями;

– FocusTracker.Core – логічне ядро застосунку. Тут розміщуються абстрактні сервіси, інтерфейси, базова бізнес-логіка, а також ViewModel-и для кожної сторінки. Цей модуль не має залежностей від користувацького інтерфейсу (User Interface, UI), що дозволяє писати тести та масштабувати функціональність незалежно від візуального інтерфейсу. «Core» взаємодіє з іншими модулями виключно через інтерфейси, що сприяє слабкому зв’язуванню;

– FocusTracker.Data – модуль доступу до даних. У ньому реалізуються сервіси для роботи з базою даних: збереження, вибірка, оновлення об’єктів. Всі класи реалізують інтерфейси, які описані в «Core», а для роботи з даними використовується «ORM Entity Framework Core» з контекстом «FocusTrackerDbContext». Саме тут реалізовано логіку фіксації часу, нарахування досвіду, обмежень, зберігання статистики тощо;

– FocusTracker.Domain – набір моделей предметної області. Цей модуль містить класи-сутності, які описують структуру об’єктів: навичка (Skill), завдання (TaskItem), відслідковувані застосунки (TrackedProgram), обмеження (Restriction) тощо. Вони не залежать ні від логіки, ні від інтерфейсу. Це дозволяє використовувати їх у будь-якому іншому модулі або навіть у зовнішньому API.

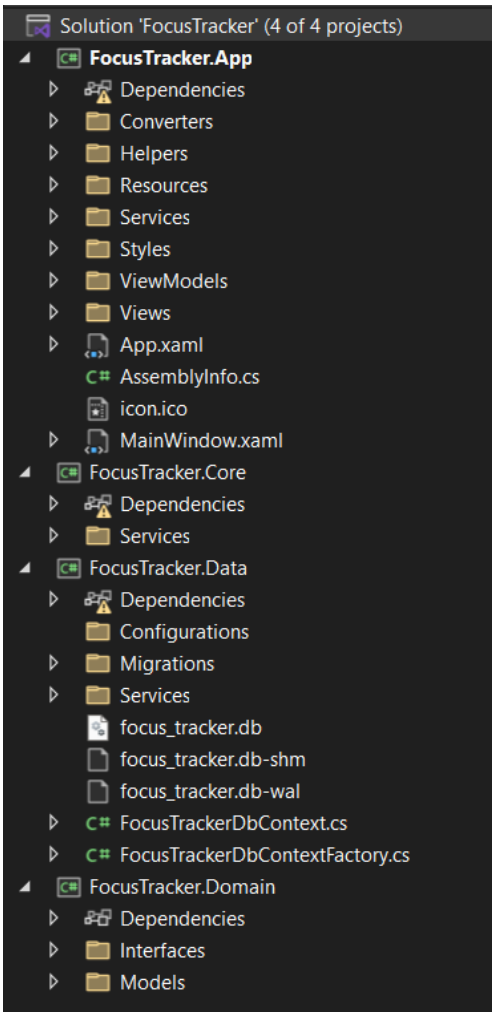

Рисунок 3.1 – Загальна структура каталогів програмного забезпечення

Всі сутності підтримують базові механізми обробки змін (наприклад, «INotifyPropertyChanged»), але не містять жодної бізнес-логіки. Такий підхід до розділення дозволяє:

– ізолювати графічний інтерфейс від обробки даних;
– повторно використовувати логіку або моделі у майбутніх проєктах;
– масштабувати кожен модуль незалежно;
– тестувати логіку без потреби запускати застосунок;
– зберігати гнучкість при зміні технологій зберігання даних або UI.

Кожна сторінка інтерфейсу (наприклад, «Завдання», «Навички», «Статистика», «Обмеження») реалізується у вигляді XAML-компонента (UserControl) і пов’язаної з ним моделі подання (ViewModel). Усі прив’язки (Binding) до кнопок, списків, форм – реалізовані саме в межах ViewModel. Вона, своєю чергою, взаємодіє із сервісами, які отримує через механізм впровадження залежностей (Dependency Injection, DI). Використання шаблону MVVM дає змогу:

– уникнути дублювання коду в UI;
– централізовано зберігати бізнес-логіку в ViewModel;
– швидко змінювати зовнішній вигляд без втрати функціональності;
– спростити підтримку й розширення застосунку.

Для взаємодії між модулями використовується DI. Усі сервіси, необхідні для логіки (наприклад інтерфейс сервісу навичок (ISkillService), інтерфейс сервісу завдань (ITaskItemService), інтерфейс сервісу обмежень (IRestrictionService)), реєструються в App.xaml.cs через метод «ConfigureServices». Таким чином, створення об’єктів делегується фреймворку, що виключає потребу явно створювати залежності у коді.

3.4 Тестування роботи застосунку

Для підтвердження працездатності реалізованого функціоналу було проведено наочне тестування у формі сценаріїв взаємодії з усіма основними модулями системи. Процес тестування передбачав виконання типових дій користувача, фіксацію їх результатів та візуалізацію інтерфейсу за допомогою скріншотів.

Після запуску програми користувач бачить головне вікно з інтерфейсом, який включає навігаційну панель зліва (рис. 3.2). та основну панель змісту справа.

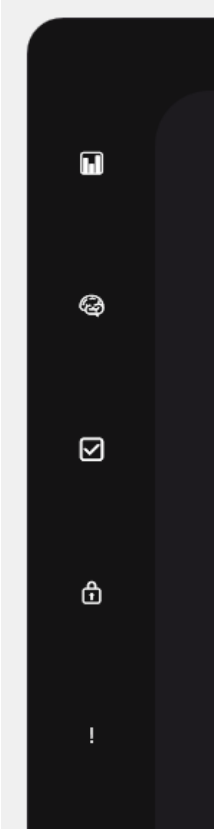

Рисунок 3.2 – Навігаційна панель

При натисканні кнопки закриття вікна, застосунок не завершує роботу повністю, а ховається у системну панель (рис. 3.3), продовжуючи працювати у фоновому режимі. Це дозволяє безперервно фіксувати активність у програмах, перевіряти обмеження та генерувати нагадування.

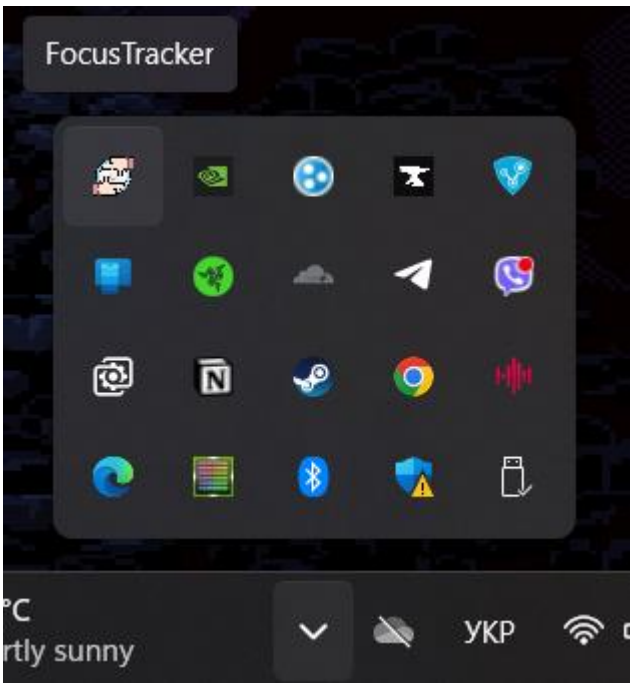

Рисунок 3.3 – Вигляд іконки застосунку в скритій панелі

Одним із перших сценаріїв є створення навичок та категорій. Користувач може додати нову категорію, вказавши її назву, після чого до неї можна додавати конкретні навички. Процес додавання навички зображено на рисунках 3.4–3.6.

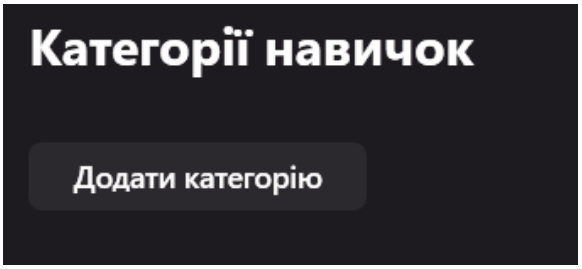

Рисунок 3.4 – Початковий вигляд при додаванні категорії навичок

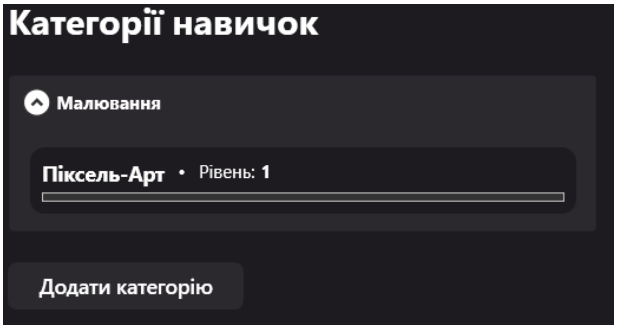

Рисунок 3.5 – Створені категорія та навичка

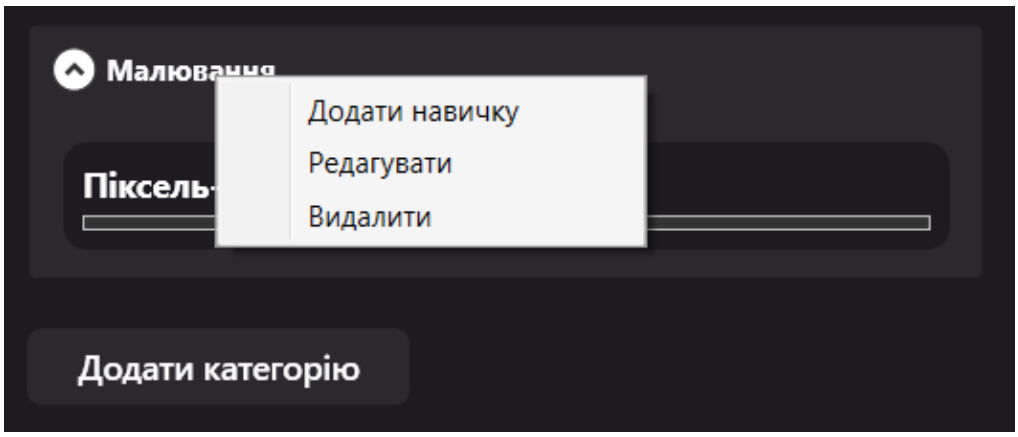

Рисунок 3.6 – Меню для редагування категорії навичок

Далі протестовано створення завдання. У відповідній формі користувач вводить назву, опис, обирає навичку, складність та додає одну або кілька програм, у яких передбачається виконання роботи. У момент створення автоматично фіксується початковий активний час для кожної з програм. Це забезпечує можливість коректно визначити, скільки реального часу було витрачено після створення завдання. Приклад створення завдання представлено на рисунках 3.7–3.9.

Рисунок 3.7 – Форма додавання завдання

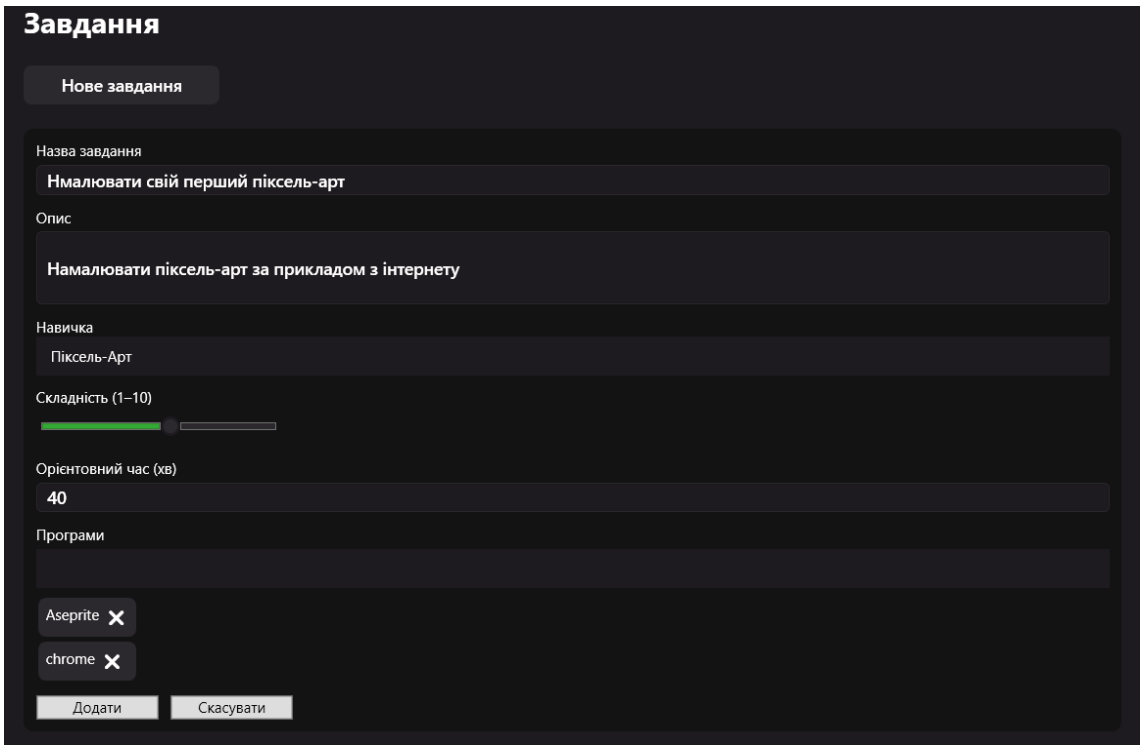

Рисунок 3.8 – Заповнена форма додавання завдання

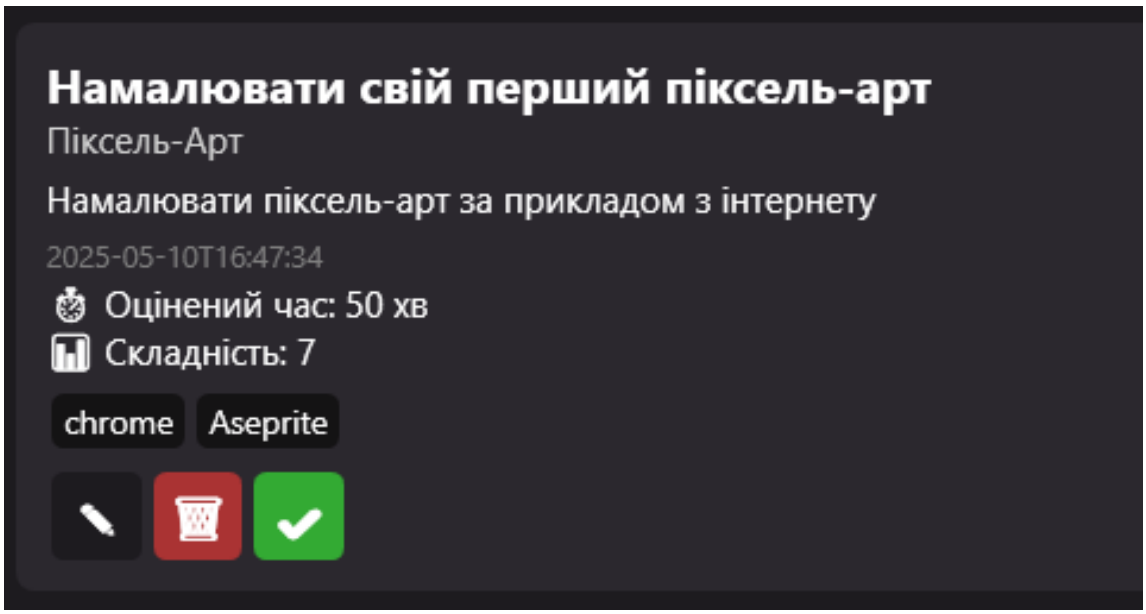

Рисунок 3.9 – Завдання для виконання

Після певного часу взаємодії з обраними застосунками, користувач може вручну завершити завдання. У момент завершення система обчислює приріст активного часу і розраховує досвід відповідно до складності завдання. Якщо досвід достатній – рівень навички підвищується автоматично (рис. 3.10).

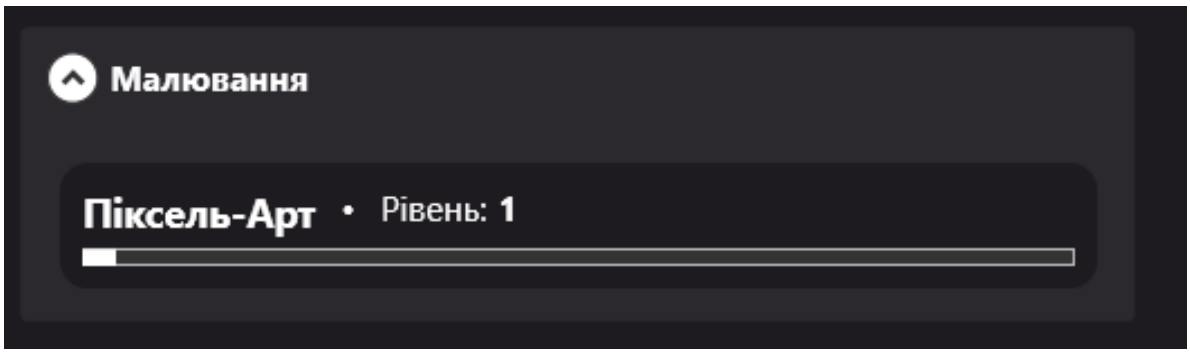

Рисунок 3.10 – Накопичений досвід до навички

Також було протестовано модуль статистики. Користувач може обрати будь-яку дату, щоб переглянути зведену активність. На графіку по годинах відображено обсяг активного часу в кожній категорії програм, розфарбований відповідно до типу: зелений – продуктивний, жовтий – суміжний, червоний – непродуктивний. Це дає змогу оцінити, як розподілявся день. Приклад такого графіка наведено на рисунку 3.11.

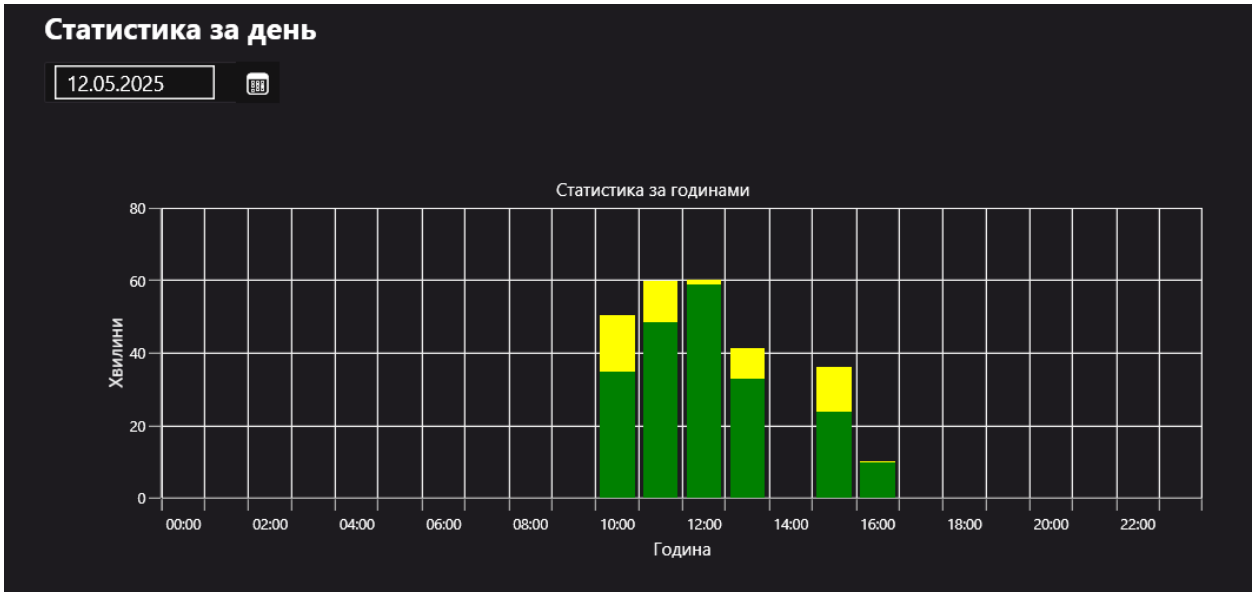

Рисунок 3.11 – Статистика за 12.05.2025 р.

Під графіком виводиться детальна таблиця використання програм за день. Кожна програма супроводжується індикатором активного часу та позначена кольором відповідно до своєї категорії. На рисунку 3.12 зображено таблицю програм зі статистикою.

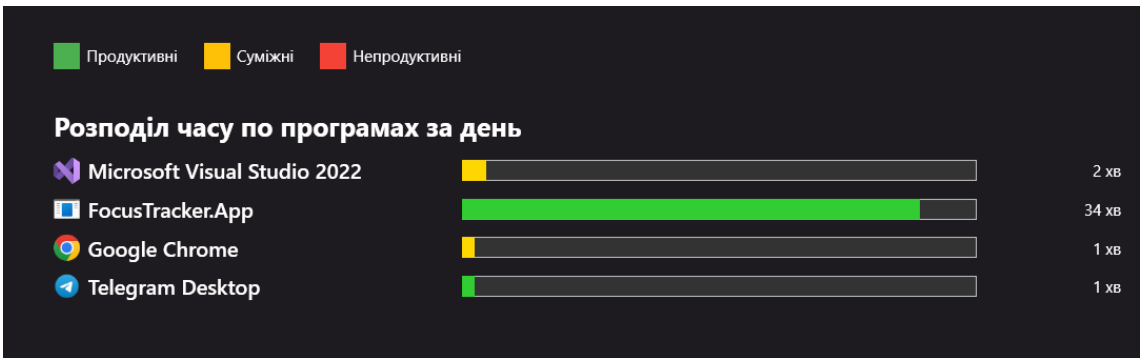

Рисунок 3.12 – Статистика по застосункам

Окремо протестовано модуль обмежень. Користувач може створити обмеження для будь-якої програми, вказавши тип правила – наприклад, максимальний добовий час (рис. 3.13, 3.14). Після досягнення ліміту система автоматично завершує програму, і виводить відповідне повідомлення. Блокування виконується негайно – без затримок, що забезпечує суворе дотримання встановлених рамок. Реалізовано також гнучкі налаштування сповіщень про наближення до ліміту, які дозволяють користувачу отримувати попередження за заданий інтервал до досягнення межі. Крім того, ведеться детальне логування подій обмежень із фіксацією дати, часу та причин блокування, що забезпечує прозорість функціонування та можливість подальшого аналізу користувацької активності. Приклад спрацювання обмеження наведено на рисунках 3.15, 3.16.

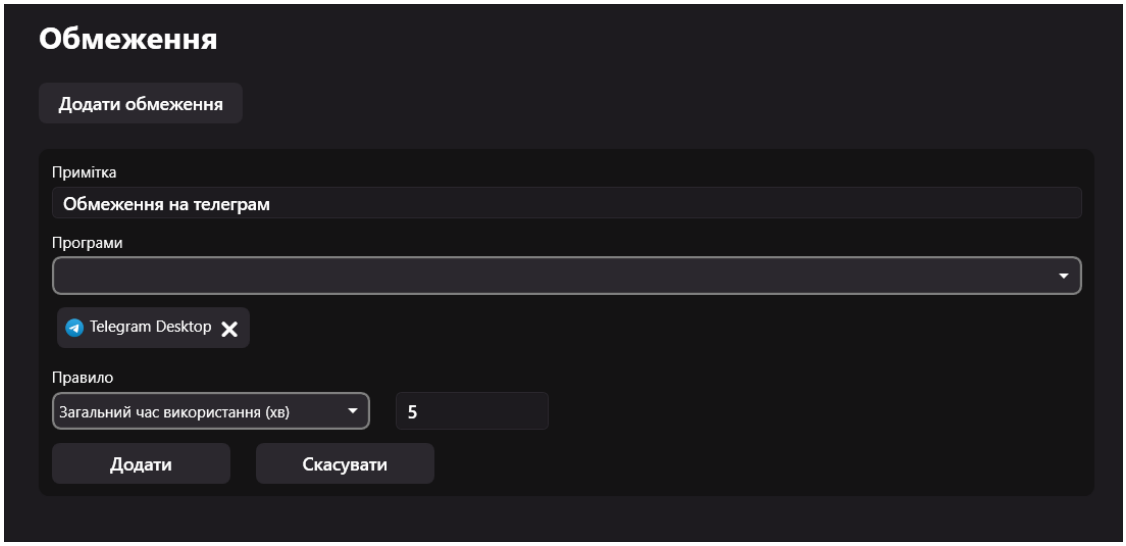

Рисунок 3.13 – Форма створення обммеження

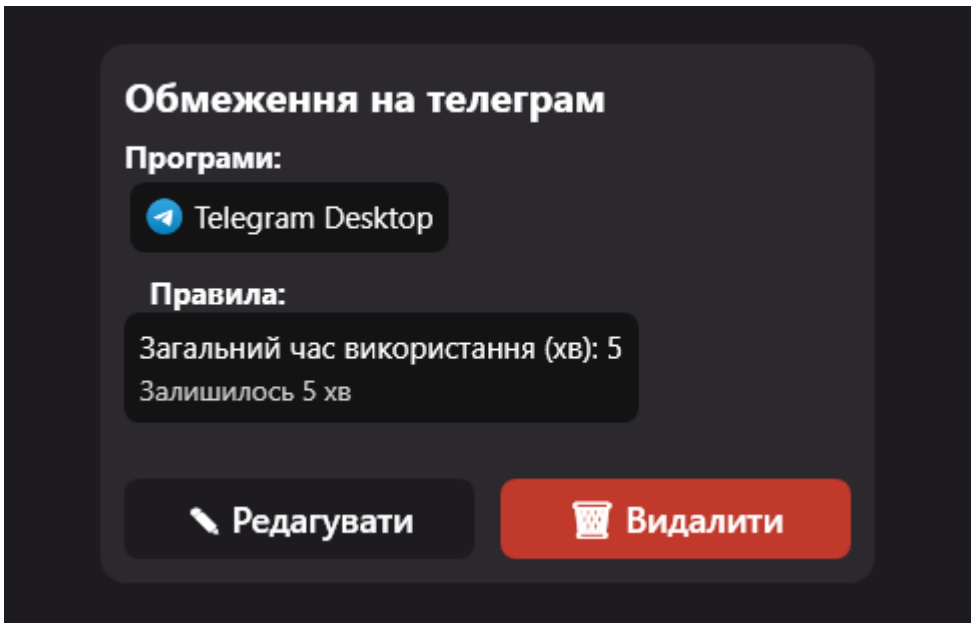

Рисунок 3.14 – Створене обмеження

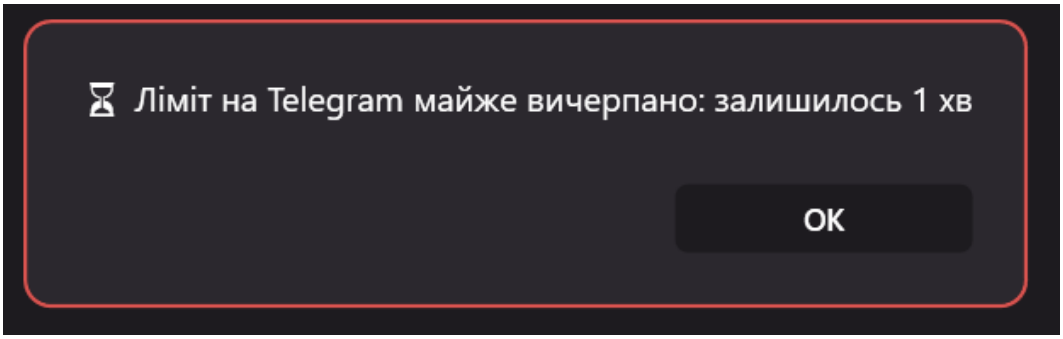

Рисунок 3.15 – Попередження про майже вичерпаний ліміт

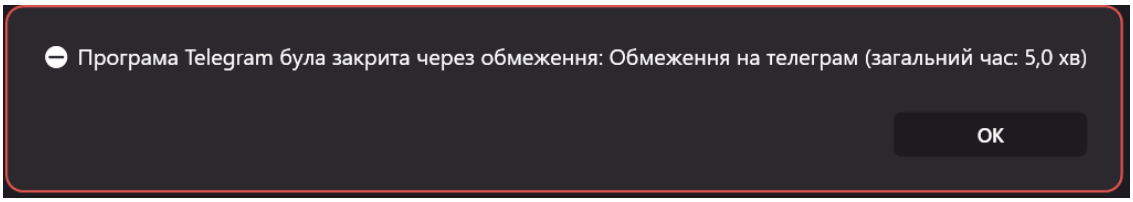

Рисунок 3.16 – Повідомлення про причину закриття застосунку

Також система підтримує показ сповіщень, наприклад, якщо користувач тривалий час не використовує продуктивні програми, пов’язані з активними завданнями. Такі імпульсні нагадування допомагають зберігати фокус. Один із таких прикладів наведено на рисунках 3.17, 3.18.

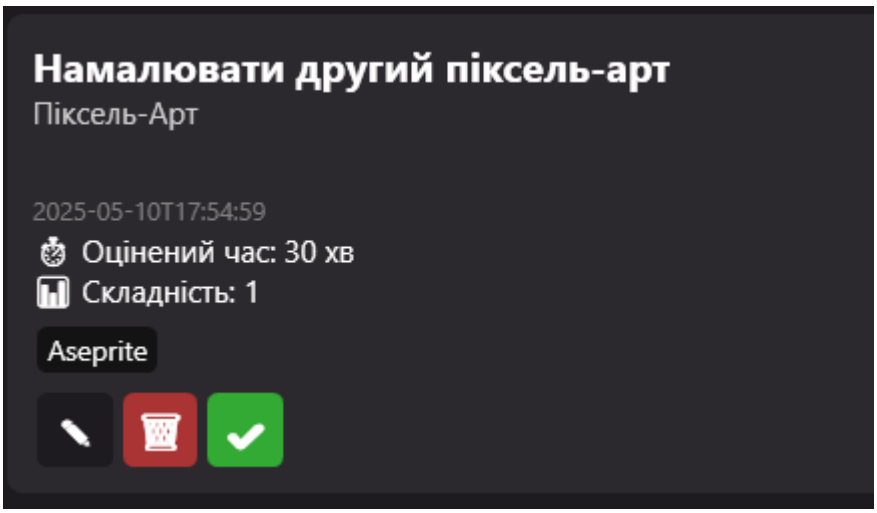

Рисунок 3.17 – Створене завдання

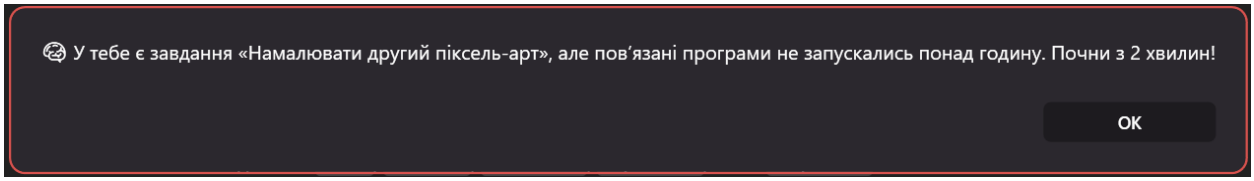

Рисунок 3.18 – Нагадування про завдання

Таким чином, усі ключові функції були протестовані в реальному сценарії використання. Кожен модуль – від створення навичок до автоматичного блокування програм – продемонстрував стабільну та передбачувану поведінку. Це підтверджує, що застосунок готовий до повсякденного використання з метою цифрової самодисципліни, моніторингу та розвитку особистої продуктивності.

3.5 Заходи щодо поліпшення застосунку

Застосунок має потенціал до подальшого вдосконалення. У майбутньому можна реалізувати наступні функціональні доповнення:

– темна/світла тема оформлення: реалізувати можливість перемикання між темною та світлою темами інтерфейсу. Це забезпечить кращий

користувацький досвід у різний час доби та підвищить зручність роботи зі застосунком;

– розширена система досягнень: додати систему винагород за виконання завдань, дотримання обмежень або тривалу продуктивну роботу. Наприклад, досягнення на кшталт «5 днів без непродуктивних програм» чи «10 завершених складних задач»;

– можливість створення довгострокових цілей: реалізувати функцію тижневих і місячних цілей із прогрес-барами, що допоможе користувачу планувати діяльність не лише на день, а й на більші періоди часу;

– інтеграція з хмарним сховищем: забезпечити резервне копіювання бази даних у хмару, що дозволить уникнути втрати даних при перевстановленні програми або зміні комп’ютера;

– глибша статистика з візуалізаціями: впровадити сторінки для перегляду тижневої, місячної та річної статистики з можливістю експорту до PDF, а також діаграми порівняння часу за категоріями;

– підтримка кількох профілів користувача: реалізувати функцію перемикання між обліковими записами, що буде зручно для сімейного або командного використання одного комп’ютера;

– розширення багатомовності: додати підтримку кількох мов (у тому числі англійської), що зробить застосунок доступним для ширшої аудиторії.

Таким чином, реалізація зазначених покращень дозволить не лише підвищити функціональність системи, але й зробити її більш адаптивною, привабливою для користувачів та готовою до масштабування у майбутньому.

ВИСНОВКИ

У межах даної кваліфікаційної роботи було розроблено застосунок для контролю цифрової активності та формування продуктивних звичок. Метою проєкту було створення програмного забезпечення, яке не лише фіксує час використання програм, а й стимулює користувача до саморозвитку через систему гейміфікації, обмежень і завдань.

У процесі реалізації було виконано всі поставлені завдання, а саме:

– проведено аналіз існуючих програм для контролю часу (RescueTime, FocusMe) та методів внутрішньої мотивації, зокрема гейміфікації та методу «двох хвилин», що дозволило сформувати вимоги до власного функціоналу;

– обґрунтовано вибір архітектурного підходу на основі шаблону MVVM та розділення проєкту на чотири окремі модулі (App, Core, Data, Domain), що забезпечує гнучкість, масштабованість та можливість тестування;

– розроблено концептуальну модель даних і реалізовано реляційну базу на основі SQLite, яка дозволяє зберігати дані про навички, завдання, досвід, обмеження та активність користувача (підтверджено схемою та таблицями);

– реалізовано WPF-застосунок з адаптивним інтерфейсом, кастомним дизайном, підтримкою темізації, графіками продуктивності та системою сповіщень;

– впроваджено гейміфікаційний модуль: користувач створює завдання, отримує досвід за активну взаємодію з продуктивними програмами, підвищує рівень навичок та отримує зворотний зв’язок у вигляді прогрес-барів та статистики;

– реалізовано систему обмежень, що дозволяє автоматично блокувати доступ до непродуктивних програм за умовами часу, інтервалів або статусу завдань, забезпечуючи користувачу інструмент цифрової самодисципліни;

– проведено тестування роботи програми у вигляді практичних сценаріїв, що підтвердило стабільність функціоналу: від створення завдань до активації обмежень та формування статистики.

Таким чином, результати бакалаврської атестаційної роботи доводять практичну цінність розробленого застосунку як інструмента цифрового самоконтролю. Він може використовуватись студентами, фахівцями та іншими користувачами, що прагнуть зменшити час у розважальних програмах і зосередитись на досягненні цілей. Гнучка архітектура проєкту дозволяє надалі розширювати функціональність, додавати нові модулі [34, 35], інтегрувати хмарне збереження та реалізовувати багатокористувацьку взаємодію.

Результати роботи апробовано у вигляді тез доповіді під час Міжнародного молодіжного форуму ХНУРЕ «РАДІОЕЛЕКТРОНІКА І МОЛОДЬ У XXI СТОЛІТТІ» [36].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Anderson, M., & Jiang, J. (2018). Teens, social media & technology 2018. *Pew research center*, *31*(2018), 1673-1689.

2. Markowetz, A. (2015). *Digitaler Burnout: warum unsere permanente Smartphone-Nutzung gefährlich ist*. Droemer eBook.

3. Newport, C. (2019). *Digital minimalism: Choosing a focused life in a noisy world*. Penguin.

4. Eikelboom, R. (2017). Irresistible: the rise of addictive technology and the business of keeping us hooked. *Perspectives on Science and Christian Faith*, *69*(4), 253-256.

5. Clear, J. (2018). *Atomic habits: An easy & proven way to build good habits & break bad ones*. Penguin.

6. RescueTime. Automatic Time Tracking. URL: https://www.rescuetime.com/features (дата звернення 15.04.2025).

7. FocusMe. Boost Your Productivity. Try the Best Blocker and Productivity App. URL: https://focusme.com/ (дата звернення 16.04.2025).

8. Newport, C. (2016). *Deep work: Rules for focused success in a distracted world*. Hachette UK.

9. Sussman, S., Lisha, N., & Griffiths, M. (2011). Prevalence of the addictions: a problem of the majority or the minority?. *Evaluation & the health professions*, *34*(1), 3-56.

10. Gorokhovatskyi, V., Chmutov, Y., Tvoroshenko, I., & Kobylin, O. (2025). Reducing computational costs by compressing the structural description in image classification methods. Advanced Information Systems, 9(1), 5–12.

11. Gorokhovatskyi V., Tvoroshenko I., Yakovleva O., and Hudáková M. (2025) Image description compression in classification structural methods, IEEE Access, vol. 13, pp. 43631-43641.

12. Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., and Zeghid M. (2024) Improving the effectiveness of image classification structural methods by compressing the description according to the information content criterion, Computers, Materials & Continua, vol. 80, no. 2, 3085-3106.

13. Гороховатський В.О., Гадецька С.В., Стяглик Н.І. (2019) Вивчення статистичних властивостей моделі блочного подання для множини дескрипторів ключових точок зображень. Радіоелектроніка, інформатика, управління, №2, 100–107.

14. Gorokhovatskyi, O., Peredrii, O., Gorokhovatskyi, V., Vlasenko, N. (2023) Explanation of CNN Image Classifiers with Hiding Parts. In: J. Benois-Pineau, R. Bourqui, D. Petkovic, G. Quenot (eds), Explainable Deep Learning Artificial Intelligence, pp. 125-146, Academic Press, 346 p.

15. Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., Gadetska S., and Al-Dhaifallah M. (2023) Statistical data analysis models for determining the relevance of structural image descriptions, IEEE Access, 11, 126938-126949.

16. Gorokhovatskyi V., Tvoroshenko I. (2024) An effective method for transforming an image description into a compact vector for classification. Information Technology and Implementation (Satellite): Conference Proceedings, November 21, 2024, Kyiv, Ukraine / V. Snytyuk (Editor). – Kyiv: Publishing House «Caravela», 25-28.

17. Pupchenko, D., Gorokhovatskyi, V. (2024) Accelerated filtration of ultrasound images. Information Technology and Implementation (Satellite): Conference Proceedings, November 21, 2024, Kyiv, Ukraine / V. Snytyuk (Editor). – Kyiv: Publishing House «Caravela», 69-72.

18. V. Gorokhovatsky, Y. Putyatin and V. Stolyarov (2017) Research of Effectivenss of Structural Image Classification Methods using Cluster Data Model, Radio Electronics Computer Science Control, vol. 3, no. 42, 78-85.

19. Gorokhovatskyi, V., Vlasenko, N. (2021). Редукція опису зображення у складі множини дескрипторів на основі метричного критерію інформативності. Advanced Information Systems, 5(4), 10-16.

20. Gadetska, S. V., Gorokhovatskyi, V. O., Stiahlyk, N. I., & Vlasenko, N. V. (2021). Statistical data analysis tools in image classification methods based on the description as a set of binary descriptors of key points. *Radio Electronics, Computer Science, Control*, (4), 58-68.

21. Gorokhovatskyi V., Tvoroshenko I., Yakovleva O. (2024) Transforming image descriptions as a set of descriptors to construct classification features, Indonesian Journal of Electrical Engineering and Computer Science, 33 (1), 113-125.

22. Gorokhovatskyi, V., Gadetska, S., & Stiahlyk, N. (2023). Accelerating Image Classification based on a Model for Estimating Descriptor-to-Class Distance. International Journal of Computing, 22(4), 485-492.

23. Gorokhovatskyi V., Tvoroshenko I., Yakovleva O., Hudáková M., and Gorokhovatskyi O. (2024) Application a committee of Kohonen neural networks to training of image classifier based on description of descriptors set, IEEE Access, vol. 12, 73376-73385.

24. Gorokhovatskyi, V., Stiahlyk, N., Mazur, Y., Vechirska, A. (2024) Способи метричної грануляції для опису зображень у задачі класифікації. Системи управління, навігації та зв'язку. Збірник наукових праць, 3(77), 106-112.

25. Gorokhovatskyi, V., Gadetska, S., Stiahlyk, N. (2024) Classification of images based on distance assessment. Information Technology and Implementation (Satellite): Conference Proceedings, November 21, 2024, Kyiv, Ukraine / V. Snytyuk (Editor). – Kyiv: Publishing House «Caravela», 22-24.

26. Gorokhovatskyi V., Gadetska S., Stiahlyk N. (2020) Image structural classification technologies based on statistical analysis of descriptions in the form of bit descriptor set. In CEUR Workshop Proceedings: Computer Modeling and Intelligent Systems (CMIS-2020), 2608, 1027-1039.

27. Gorokhovatsky, V.A., Putyatin, Y.P. (2008) Structural recognition of images on the basis of voting models of attributes of typical points, Data recording, storage and processing, 10(4), 75-85.

28. Gorokhovatskyi V.A. (2018) Image Classification Methods in the Space of Descriptions in the Form of a Set of the Key Point Descriptors. Telecommunications and Radio Engineering, 77 (9), 787-797.

29. Arlow, J., & Neustadt, I. (2005). *UML 2 and the unified process: practical object-oriented analysis and design*. Pearson Education.

30. Microsoft Docs. *GetForegroundWindow function (winuser.h)*. URL: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-getforegroundwindow (дата звернення 17.04.2025).

31. Deterding, S., Dixon, D., Khaled, R., & Nacke, L. (2011). *From Game Design Elements to Gamefulness: Defining "Gamification"*. Proceedings of the 15th International Academic MindTrek Confere.

32. Allen, D. (2015). *Getting things done: The art of stress-free productivity*. Penguin.

33. Hipp, D. R. (2022). *SQLite Documentation*. URL: https://sqlite.org/docs.html. (дата звернення 17.04.2025).

34. Gorokhovatskyi V.A., Zamula A.A. (2016) Employment of Intelligent Technologies in Multiparametric Control Systems. Telecommunications and Radio Engineering. Vol. 75, No 19, 1775–1785.

35. Gorokhovatsky V.A., Putyatin Ye.P. (2009) Image Likelihood Measures of the Basis of the Set of Conformities. Telecommunications and Radio Engineering, 68 (9), pp. 763-778.

36. Гаєвий А.О. (2025) Програмний контроль часу використання інструментальних засобів. *Радіоелектроніка і молодь у XXI столітті: тези доповідей 29-го Міжнародного молодіжного форуму (Харків, 16–19 квітня 2025 р.)*. Харків: ХНУРЕ, 2025. Т. 7. С. 32-33.