

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему СТВОРЕННЯ ПРОГРАМНОЇ СИСТЕМИ ОПРАЦЮВАННЯ
ДАНИХ ДИНАМІЧНОГО ОЦІНЮВАННЯ ФАКТИЧНОГО
РОБОЧОГО ЧАСУ ПРАЦІВНИКІВ
DESIGN OF A DATA PROCESSING SOFTWARE SYSTEM
FOR DYNAMIC ASSESSMENT OF ACTUAL WORKING
HOURS OF EMPLOYEES

Виконав(ла): студент(ка) 4 курсу, групи КНТ-212

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

ХОМЕНКО А.В.

(ПРИЗВИЩЕ та ініціали)

Керівник ЛЬОВКІН В.М.

(ПРИЗВИЩЕ та ініціали)

Рецензент ЗЕЛІК О.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Факультет КНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 122 Комп'ютерні науки
(код і найменування)
 Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2026 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ХОМЕНКО Аліси Валентинівни
(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Створення програмної системи опрацювання даних динамічного оцінювання фактичного робочого часу працівників. Design of a Data Processing Software System for Dynamic Assessment of Actual Working Hours of Employees

керівник проєкту (роботи) к.т.н., доцент, ЛЬОВКІН Валерій Миколайович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 07 ” квітня 2026 року № 139

2. Строк подання студентом проєкту (роботи) 01 червня 2026 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програмної системи. 3. Розробка програмної системи. 4. Експлуатація та тестування програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ЛЬОВКІН В.М., доцент		
Нормоконтроль	БСЛОВА А.В., асистент		

7. Дата видачі завдання « 20 » квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій	2 тиждень	Розділ 2
	розробки.		
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження	5 тиждень	Розділ 4
	програмного забезпечення.		
7	Оформлення пояснювальної записки та документів	6 тиждень	Додатки
	до неї.		
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Аліса ХОМЕНКО
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Валерій ЛЬОВКІН
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 127 с., 16 табл., 41 рис., 2 дод., 13 джерел.

ЦИФРОВА ПРОКРАСТИНАЦІЯ, ПРОДУКТИВНИЙ ЧАС, ДЕСКТОПНА ПРОГРАМНА СИСТЕМА, WORK-LIST, BLACK-LIST, ЧАСОВИЙ БАЛАНС, PYTHON, PYQT6, SQLITE.

Об'єкт дослідження — процес побудови програмних систем обліку та оцінювання фактичного робочого часу працівників у цифровому робочому середовищі.

Предмет дослідження — методи та алгоритми обробки даних про активний ресурс для створення системи визначення фактичного робочого часу.

Мета роботи — підвищення об'єктивності визначення фактичного робочого часу працівників шляхом автоматизованої обробки даних про активний ресурс.

Матеріали, методи та технічні засоби: аналіз предметної області, порівняльний аналіз програмних аналогів, об'єктно-орієнтоване програмування, проектування архітектури програмного забезпечення, функціональне тестування. Для реалізації використано Python 3, PyQt6, SQLite3 та середовище Visual Studio Code.

Результати. Розроблено програмну систему «Honestime», яка визначає активне вікно або вебресурс, перевіряє його належність до Work-List, обліковує фактично зарахований робочий час і формує часовий баланс. У системі реалізовано керування Work-List і Black-List, запуск і завершення робочої сесії, AFK-перевірку, режим Break, збереження історії сесій, перегляд статистики та функцію Reset.

Висновки. Створена програмна система дає змогу відокремлювати загальний час перебування працівника за комп'ютером від фактично зарахованого робочого часу.

Мету роботи досягнуто: підвищено об'єктивність визначення фактичного робочого часу працівників за рахунок автоматизованої обробки даних про активний ресурс, перевірки його належності до Work-List, обліку фактично зарахованого робочого часу та формування часового балансу.

Галузь використання — персональна організація робочого часу працівників, дистанційна робота, навчальна діяльність, фріланс та самоконтроль продуктивності у цифровому середовищі.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 127 pages, 16 tables, 41 figures, 2 appendices, 13 sources.

DIGITAL PROCRASTINATION, PRODUCTIVE TIME, DESKTOP SOFTWARE SYSTEM, WORK-LIST, BLACK-LIST, TIME BALANCE, PYTHON, PYQT6, SQLITE.

Object of research — the process of developing software systems for recording and evaluating the actual working time of employees in a digital work environment.

Subject of research — methods and algorithms for processing data on the active resource in order to create a system for determining actual working time.

Purpose of the work — to increase the objectivity of determining employees' actual working time through automated processing of data on the active resource.

Materials, methods and technical tools: domain analysis, comparative analysis of software analogues, object-oriented programming, software architecture design, and functional testing. Python 3, PyQt6, SQLite3, and the Visual Studio Code environment were used for implementation.

Results. The software system “Honestime” was developed. It determines the active window or web resource, checks whether it belongs to the Work-List, records the actually credited working time, and forms a time balance. The system implements Work-List and Black-List management, starting and ending a work session, AFK checking, Break mode, saving session history, viewing statistics, and a Reset function.

Conclusions. The software system developed makes it possible to distinguish between the total time an employee spends at a computer and the actual working time recorded.

The purpose of the work has been achieved: the objectivity of determining employees' actual working time has been improved through the automated

processing of data on active resources, verification of their inclusion in the Work-List, recording of actual working time and the generation of a time balance.

Field of application — personalised organisation of employees' working time, remote working, educational activities, freelancing and self-monitoring of productivity in a digital environment.

ЗМІСТ

	С.
Вступ.....	10
1 Аналіз предметної області.....	12
1.1 Постановка проблеми досліджуваної предметної області.....	12
1.2 Огляд існуючих програмних засобів для вирішення проблеми.....	15
1.3 Вимоги до програмного забезпечення	21
1.4 Висновки за розділом 1	27
2 Розробка архітектури програмної системи.....	29
2.1 Вибір мови програмування	29
2.2 Проєктування архітектури програмного забезпечення.....	31
2.3 Проєктування структури бази даних.....	36
2.4 Структурна схема програмного проєкту	38
3 Розробка програмної системи	41
3.1 Алгоритм функціонування програмної системи	41
3.2 Опис модулів реалізованої програмної системи.....	42
3.3 Висновки за розділом 3	60
4 Експлуатація та тестування програми	61
4.1 Призначення програми	61
4.2 Умови виконання програми	62
4.3 Виконання програми.....	63
4.4 Тестування програми	67
4.5 Висновки за розділом 4	75
Висновки	76
Перелік джерел посилання	77
Додаток А Текст програми.....	79
Додаток Б Слайди презентації	120

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	— Application Programming Interface;
GUI	— Graphical User Interface;
IDE	— Integrated Development Environment;
Work-List	— перелік робочих ресурсів;
Black-List	— перелік відволікаючих ресурсів;
Break	— режим тимчасової перерви;
Reset	— функція екстреного скидання;
БД	— база даних;
ПЗ	— програмне забезпечення.

ВСТУП

Ефективність роботи працівника в цифровому середовищі залежить не лише від технічних характеристик інформаційних систем, а й від здатності підтримувати концентрацію під час виконання професійних, навчальних або проєктних завдань. Проблема цифрової прокрастинації не завжди може бути пояснена лише браком дисципліни, оскільки вона також пов'язана з особливостями розподілу когнітивних ресурсів під час взаємодії з цифровими інструментами [1].

Повільне мислення, необхідне для виконання складних інтелектуальних завдань, потребує значних когнітивних зусиль. У ситуації виснаження працівник частіше переходить до простіших цифрових дій, які потребують менших зусиль і не завжди пов'язані з виконанням основних завдань. Це призводить до розриву між загальною тривалістю перебування за комп'ютером і фактичною продуктивною активністю. Унаслідок цього знижується результативність роботи, а оцінка власної ефективності може бути спотвореною [2].

У такому стані межа між працею та відновленням розмивається: відпочинок перестає виконувати свою функцію, оскільки супроводжується тривогою, а робота стає фрагментарною. Тому актуальним завданням є розробка програмного забезпечення (ПЗ), яке забезпечує автоматизовану обробку даних активності та дає змогу динамічно оцінювати фактичний робочий час у цифровому середовищі. Це дозволить відокремити загальне перебування за комп'ютером від реально зарахованої роботи та створити умови для більш об'єктивного планування навантаження.

У межах цієї роботи працівник розглядається як користувач цифрового робочого середовища, який виконує професійні, навчальні або проєктні завдання за комп'ютером. Основне призначення розроблюваної системи полягає у персональному обліку фактичного робочого часу та підтримці саморегуляції під час взаємодії з цифровими ресурсами.

Об'єкт дослідження — процес побудови програмних систем обліку та оцінювання фактичного робочого часу працівників у цифровому робочому середовищі.

Предмет дослідження — методи та алгоритми обробки даних про активний ресурс для створення системи визначення фактичного робочого часу.

Мета роботи — підвищення об'єктивності визначення фактичного робочого часу працівників шляхом автоматизованої обробки даних про активний ресурс.

Для досягнення поставленої мети у роботі вирішуються такі завдання:

- проаналізувати предметну область і визначити основні проблеми цифрової прокрастинації;
- провести огляд існуючих програмних рішень для підвищення продуктивності, самоконтролю робочого часу та обмеження цифрових дистракторів;
- сформулювати функціональні та нефункціональні вимоги до системи персонального обліку фактичного робочого часу працівника;
- спроектувати архітектуру програмної системи;
- реалізувати основні модулі програмного забезпечення;
- провести тестування працездатності розробленої програмної системи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка проблеми досліджуваної предметної області

Коли комп'ютер стає основним робочим середовищем працівника, він одночасно є інструментом виконання професійних, навчальних або проєктних завдань і середовищем доступу до цифрових дистракторів. В одному інформаційному просторі поєднуються робочі програми, месенджери, соціальні мережі, розважальні вебресурси, відеоконтент і системні сповіщення. Така концентрація різних цифрових стимулів ускладнює підтримання стабільної уваги на основному завданні. У сучасній інтелектуальній праці цифрові фактори, що відволікають увагу, пов'язують із фрагментацією робочого процесу, зниженням продуктивності та зростанням психологічного напруження [1].

Проблема полягає не лише в наявності сторонніх ресурсів, а й у їх постійній доступності. Працівник може за кілька секунд відкрити повідомлення, вебсторінку, соціальну мережу або інший нецільовий ресурс, що перериває виконання основного робочого завдання. Навіть короткочасне перемикання порушує робочий контекст: після переривання необхідно повернутися до попереднього завдання, відновити логіку його виконання та знову досягти стану зосередженості. Дослідження офісних перерв показують, що їх частота пов'язана із суб'єктивним робочим навантаженням, а складність основної задачі впливає на те, наскільки перерва сприймається як перешкода для подальшої роботи [2].

Окремим джерелом цифрових відволікань є сповіщення від комунікаційних застосунків. Вони забезпечують швидкий обмін інформацією, однак часто переривають діяльність незалежно від готовності користувача реагувати на них. Польове дослідження впливу таких сповіщень показало, що зменшення кількості переривань може позитивно впливати на продуктивність і знижувати рівень напруження [3]. Отже, цифрова прокрастинація пов'язана не лише зі свідомим переходом до розважальних

ресурсів, а й з автоматичними стимулами, які порушують стан зосередженої роботи.

Важливою особливістю проблеми є те, що працівник не завжди помічає момент втрати концентрації та не може точно оцінити обсяг фактично відпрацьованого часу. Сам факт перебування працівника за комп'ютером часто сприймається як робота, навіть якщо частина часу витрачається на дії, не пов'язані з виконанням професійних, навчальних або проєктних завдань. У результаті виникає розрив між загальною активністю за пристроєм і реальною продуктивною роботою. Без об'єктивного обліку складно визначити, скільки часу було реально витрачено на виконання робочих завдань.

У цьому контексті актуальним є не лише блокування небажаних ресурсів, а й уточнення самого поняття фактичного робочого часу працівника. Доцільно враховувати не загальну тривалість перебування за комп'ютером, а взаємодію з інструментами, які справді належать до робочого процесу. Програмні засоби цифрового самоконтролю можуть використовувати самовідстеження, постановку цілей, блокування, гейміфікацію та оцінювання активності [4]. У межах цієї роботи найбільш доцільним є поєднання автоматичного обліку продуктивної активності з механізмом відповідальності за невиконання встановленого плану.

Система, що розробляється в межах роботи, має бути спрямована на розв'язання зазначеної задачі шляхом розмежування загальної активності працівника за комп'ютером та фактично зарахованого робочого часу. Одним із ключових механізмів системи є Work-List — перелік ресурсів, взаємодія з якими вважається робочою активністю працівника. До нього можуть входити робочі застосунки, вебсайти, редактори коду, середовища розробки, офісні програми або інші інструменти, які працівник самостійно визначає як необхідні для виконання своїх завдань.

Якщо активним є ресурс, що не входить до Work-List, відповідний проміжок часу не зараховується як робочий. Такий підхід відрізняється від

повного блокування відволікаючих ресурсів: працівник зберігає можливість перейти до сторонньої активності, однак цей час не включається до фактично зарахованого робочого часу. Це забезпечує гнучкість використання системи, але водночас зменшує ризик суб'єктивної переоцінки фактично відпрацьованого часу.

Для опису моделі програмної системи використовуються такі основні поняття.

Цифрова прокрастинація — поведінка працівника, що полягає у відкладанні пріоритетних завдань через залучення до нецільових активностей у цифровому середовищі [1], [5].

Дистрактори — програми, вебресурси, повідомлення або інші цифрові стимули, що спричиняють перемикання уваги та переривають зосереджену роботу [2], [3].

Work-List — динамічна множина програмних ідентифікаторів і мережевих адрес, взаємодія з якими класифікується системою як робоча активність працівника.

Black-List — сукупність цифрових ресурсів, які працівник самостійно визначає як відволікаючі та доступ до яких може бути обмежений системою за умовами часового балансу.

Часовий баланс — показник, що відображає профіцит або дефіцит фактично зарахованого робочого часу працівника відносно встановленого плану.

Після завершення робочої сесії система порівнює фактично зарахований час із денним нормативом. Якщо працівник має менший фактично зарахований робочий час, ніж встановлений план, формується від'ємний часовий баланс, що відображає невиконану частину встановленого робочого нормативу. Якщо фактичний робочий час перевищує норматив, утворюється позитивний баланс, що може компенсувати попередні або майбутні відхилення. За наявності від'ємного балансу система обмежує

доступ до ресурсів із Black-List і забороняє видалення елементів із Work-List та Black-List до моменту компенсації боргу.

Додатковим компонентом «Honestime» є статистика, яка відображає накопичений робочий час, поточний баланс та історію сесій. Самовідстеження є одним із поширених механізмів цифрового самоконтролю [4], а текстове й графічне подання результатів може використовуватися як інструмент рефлексії над прокрастинаційною поведінкою [5]. У цьому контексті наявність статистики сама по собі не забезпечує усунення прокрастинаційної поведінки, однак надає користувачу основу для аналізу власних звичок і подальшого планування роботи.

Отже, проблема досліджуваної предметної області полягає у відсутності об'єктивного уявлення про фактично відпрацьований час у цифровому середовищі. У межах даної роботи необхідно розробити програмне забезпечення, яке дає змогу працівнику оцінювати не загальну тривалість перебування за комп'ютером, а фактичний робочий час, визначений за взаємодією з робочими цифровими ресурсами.

1.2 Огляд існуючих програмних засобів для вирішення проблеми

Проблема цифрової прокрастинації та неефективного використання робочого часу частково вирішується за допомогою різних програмних засобів. Такі рішення можуть застосовувати соціальний контроль, блокування відволікаючих ресурсів, ручне планування задач, гейміфікацію або статистику активності користувача. Дослідження застосунків, спрямованих на зменшення цифрового відволікання, показує, що подібні системи можуть використовувати різні групи функцій: обмеження доступу, постановку цілей, самовідстеження, нагадування, винагороди та покарання [4].

Для аналізу було обрано три програмні продукти, які реалізують різні підходи до підвищення продуктивності: Focusmate, Cold Turkey Blocker та

Habitica. Їх порівняння з розроблюваною програмною системою «Honestime» дозволяє визначити переваги та обмеження наявних рішень, а також обґрунтувати необхідність створення власної системи обліку фактичного робочого часу.

1.2.1 Аналіз платформи Focusmate

Платформа Focusmate реалізує підхід зовнішньої відповідальності. Її основна ідея полягає не у технічному блокуванні відволікаючих ресурсів, а у створенні робочої ситуації, в якій користувач виконує завдання в умовах соціальної присутності іншого учасника. Сервіс поєднує двох користувачів у відеосесії, під час якої кожен учасник працює над власною задачею, але присутність іншої людини створює додатковий соціальний контроль [6].

На початку сесії користувач формулює ціль роботи, після чого протягом визначеного проміжку часу виконує власну задачу. Focusmate підтримує сесії тривалістю 25, 50 або 75 хвилин, що дозволяє використовувати платформу як для коротких робочих інтервалів, так і для довших періодів зосередженої роботи [7]. Приклад використання платформи Focusmate зображено на рис. 1.1.

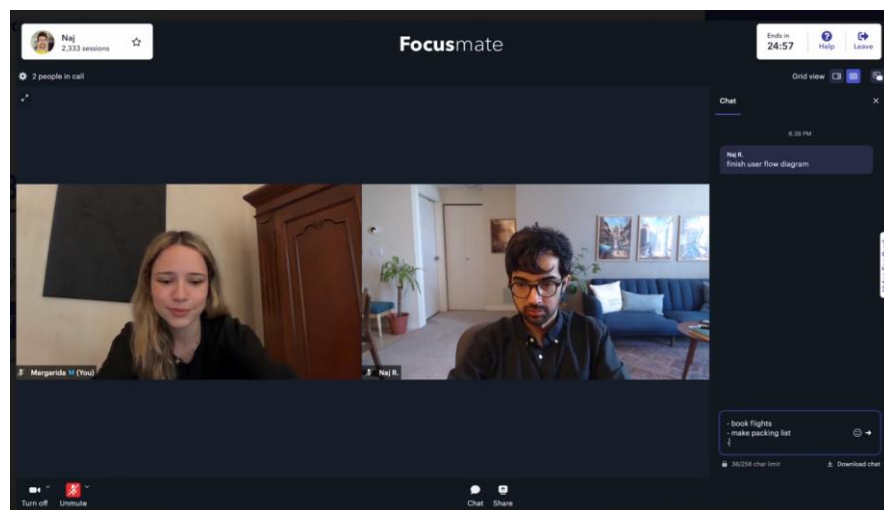


Рисунок 1.1 — Відеоконференція «Focusmate»

Сильна сторона цього підходу полягає у тому, що користувач отримує зовнішній стимул розпочати роботу у визначений час. Для людей, яким складно самотійно перейти від наміру до дії, присутність партнера може виконувати функцію зовнішнього мотиваційного чинника. Водночас платформа не перевіряє, з якими саме програмами або вебресурсами взаємодіє користувач під час сесії. Тобто Focusmate допомагає організувати робочий інтервал, але не визначає фактичний робочий час на рівні активного вікна або конкретного ресурсу.

Для задачі даної роботи цього недостатньо, оскільки «Honestime» має працювати без зовнішнього партнера та автоматично відрізняти продуктивну активність від сторонньої. У розроблюваній системі контроль базується не на соціальній присутності, а на перевірці активного ресурсу за переліком Work-List.

1.2.2 Застосунок Cold Turkey Blocker

Застосунок Cold Turkey Blocker представляє інший підхід — примусове обмеження доступу до відволікаючих ресурсів. Його основна функція полягає у блокуванні сайтів, застосунків, ігор, відеосервісів або навіть усього Інтернету, окрім визначених винятків [8]. Така модель орієнтована на ситуації, коли користувач не хоче залишати собі можливість швидко перейти до розважального ресурсу.

На відміну від Focusmate, де контроль створюється через іншу людину, Cold Turkey Blocker переносить контроль у програмне середовище. Користувач самотійно задає список ресурсів, які потрібно заблокувати, після чого система обмежує доступ до них. Приклад інтерфейсу застосунку Cold Turkey Blocker наведено на рис. 1.2.

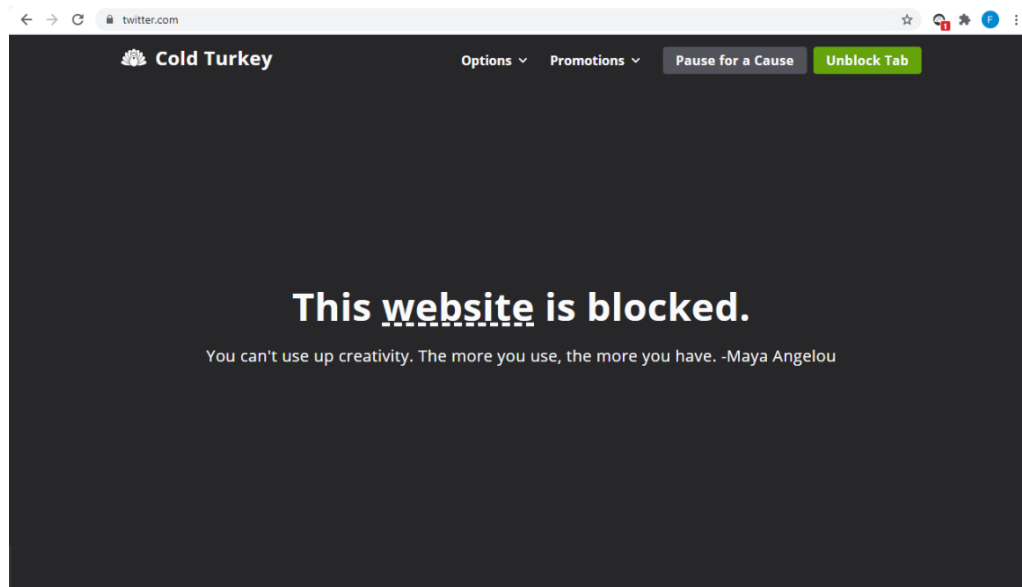


Рисунок 1.2 — Приклад заблокованого вебсайту з Cold Turkey Blocker

Перевагою такого рішення є високий рівень примусового обмеження доступу до визначених ресурсів. Якщо користувач визначив певний сайт або застосунок як дистрактор, Cold Turkey Blocker може зробити його недоступним на заданий час. Це корисно у випадках, коли основна проблема полягає саме у неконтрольованому переході до конкретних ресурсів.

Однак для поставленої задачі простого блокування недостатньо. Відсутність доступу до розважального сайту ще не означає, що користувач працює. Він може залишатися за комп'ютером, але не виконувати цільову задачу. Крім того, модель прямого блокування не враховує накопичувальний робочий баланс і не розрізняє, скільки часу фактично було проведено у робочому середовищі. Саме тому у «Honestime» Black-List не є єдиною основою контролю. Основний облік виконується через Work-List, а обмеження відволікаючих ресурсів розглядається як допоміжний механізм, пов'язаний із часовим балансом.

1.2.3 Застосунок Habitica

Habitica використовує гейміфікацію як спосіб підвищення мотивації. На відміну від Cold Turkey Blocker, цей застосунок не зосереджується на

технічному обмеженні доступу до ресурсів, а використовує ігрові механіки для підвищення мотивації до виконання завдань. Habitica позиціонується як застосунок для звичок і продуктивності, у якому реальне життя користувача подається у формі гри [9].

Користувач створює персонажа, додає звички, щоденні задачі та разові завдання. За виконання дій він отримує ігрові винагороди, а за пропуски може втрачати ігрові показники. Приклад інтерфейсу застосунку Habitica зображено на рис. 1.3.

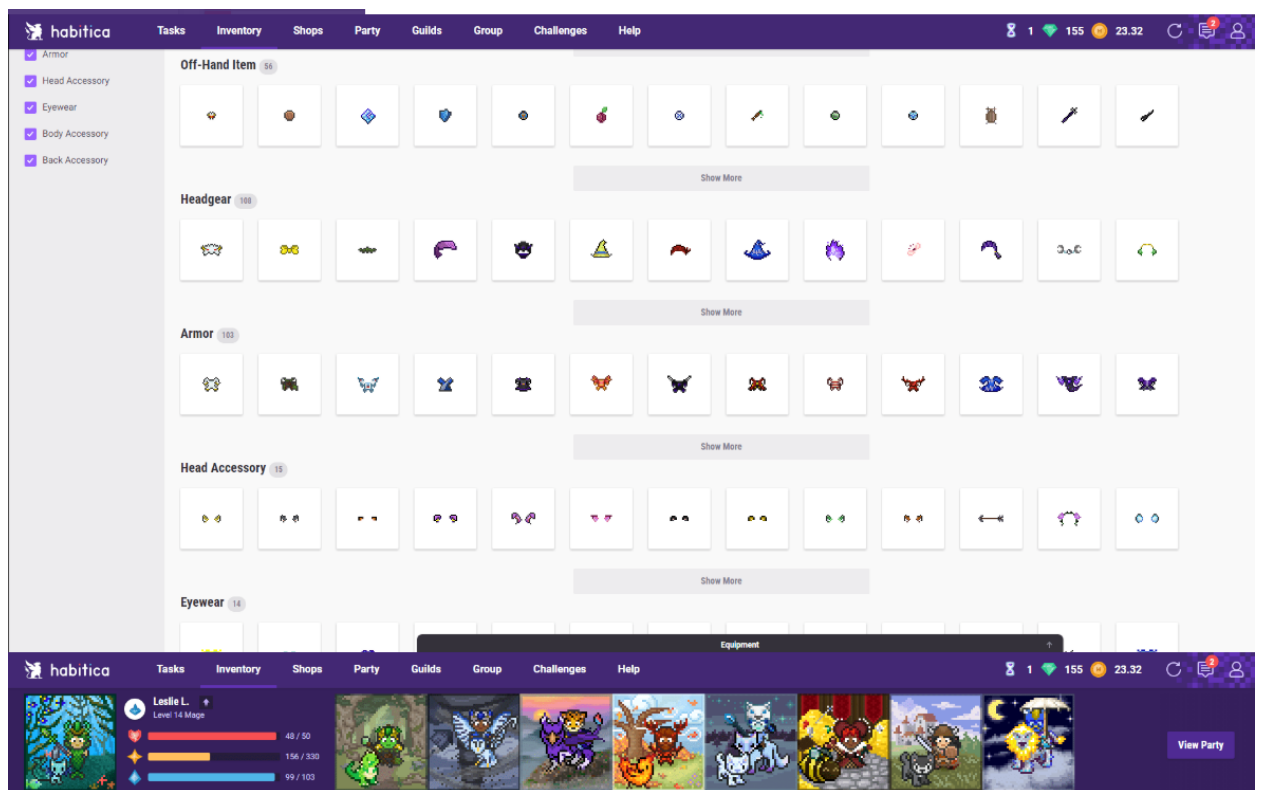


Рисунок 1.3 — Приклад сторінки у Habitica

Перевагою Habitica є те, що вона намагається підвищити залученість користувача до процесу самоконтролю. Для довгострокового формування звичок або планування побутових, навчальних і особистих задач такий підхід може бути зручним.

Водночас Habitica переважно спирається на ручне введення та ручне підтвердження результатів. Користувач сам позначає виконання задачі, тому

система не може перевірити, чи дійсно робота виконувалася у потрібному програмному середовищі. Для «Honestime» це є ключовою відмінністю: застосунок має автоматично визначати фактичну робочу активність через активне вікно та Work-List, а не покладатися лише на самооцінку користувача.

1.2.4 Порівняльна характеристика існуючих аналогів та розроблюваного ПЗ

Проведений огляд показує, що розглянуті застосунки мають різну природу контролю. Focusmate формує відповідальність через присутність іншої людини, Cold Turkey Blocker створює технічну заборону доступу до дистракторів, а Habitica перетворює виконання задач на ігровий процес. Жоден із цих підходів сам по собі не вирішує задачу автоматичного обліку фактичного робочого часу на основі активного ресурсу.

Розроблювана програмна система «Honestime» відрізняється тим, що не ототожнює перебування за комп'ютером із продуктивною роботою. Система зараховує час лише тоді, коли активний ресурс належить до Work-List, а Black-List використовується як допоміжний механізм обмеження дистракторів за умовами часового балансу. Порівняльний аналіз розглянутих програмних продуктів наведено у табл. 1.1.

Таблиця 1.1 — Порівняльний аналіз програмних продуктів

Критерії порівняння	Focusmate	Cold Turkey Blocker	Habitica	Honestime
1	2	3	4	5
Платформа	Web	Desktop	Web / Mobile	Desktop
Метод контролю	Візуальний нагляд	Системне перехоплення	Ручне відмічання	Моніторинг активного вікна

Кінець таблиці 1.1

1	2	3	4	5
Об'єкт обмеження	Відсутній	Сайти, додатки, ПК	Відсутній	Ресурси з Black-List за умовами балансу
Автоматизація	Низька	Середня (за розкладом)	Відсутня	Висока (Work-List tracking)
Механіка часового балансу	Присутність партнера	Неможливість доступу до заблокованих ресурсів	Тільки втрата НР	Часовий баланс, борг і обмеження видалення зі списків
Анти-кризові дії	Відсутні	Відсутні	Відсутні	Екстрене скидання Reset
Збереження даних	Хмарне (Cloud)	Локальне	Хмарне	Локальне (SQLite)

Отже, розглянуті аналоги реалізують окремі підходи до самоконтролю, однак не поєднують автоматичну перевірку активного ресурсу, локальний облік продуктивного часу та механізм часового балансу. Саме ці обмеження враховано під час формування вимог до програмної системи «Honestime».

1.3 Вимоги до програмного забезпечення

На основі аналізу предметної області та порівняння наявних аналогів сформовано вимоги до програмної системи «Honestime». Основним

призначенням системи є автоматизований облік та динамічне оцінювання фактичного робочого часу працівника на основі активного ресурсу, що належить до Work-List, а також підтримка механізму часового балансу.

Ключовими елементами системи є Work-List, Black-List, моніторинг активного ресурсу, часовий баланс, історія робочих сесій і статистика. На основі цих елементів система повинна визначати, які проміжки активності можуть бути зараховані як робочий час, а які не належать до продуктивної діяльності.

Після завершення робочої сесії система повинна порівнювати фактично зарахований час із денним нормативом і на цій основі оновлювати часовий баланс. Додатне або від'ємне значення балансу використовується для визначення подальшого режиму роботи зі списками ресурсів та обмеженнями.

Формальна постановка задачі має такий вигляд.

Вхідними даними є:

- перелік продуктивних ресурсів Work-List;
- перелік відволікаючих ресурсів Black-List;
- денний норматив робочого часу;
- інформація про активне вікно або активний вебресурс користувача;
- поточний стан робочої сесії;
- поточне значення часового балансу.

Вихідними даними є:

- фактично зарахований робочий час;
- оновлене значення часового балансу;
- історія робочих сесій;
- статистика продуктивної активності;
- стан доступу до ресурсів із Black-List;
- стан можливості редагування Work-List і Black-List.

Основним критерієм коректності роботи системи є відповідність зарахованого часу фактичній взаємодії користувача з ресурсами,

визначеними у Work-List. Додатковими критеріями є збереження даних після перезапуску застосунку, коректне оновлення балансу після завершення сесії, блокування Black-List за від'ємного балансу та недопущення видалення ресурсів зі списків у заборонених станах.

1.3.1 Визначення функціональних вимог

Функціональні вимоги описують дії, які система повинна виконувати для реалізації основного сценарію роботи користувача. Для «Honestime» такими діями є налаштування списків ресурсів, запуск і супровід робочої сесії, облік продуктивного часу, розрахунок балансу та відображення статистики. Функціональні вимоги до застосунку наведено у табл. 1.2.

Таблиця 1.2 — Функціональні вимоги до програмної системи «Honestime».

Позначення	Функціональна вимога
1	2
F1	Система повинна дозволяти користувачу формувати та редагувати Work-List та Black-List
F2	Система повинна дозволяти додавання нових ресурсів до Work-List і Black-List у головному меню та під час активної робочої сесії
F3	Система повинна дозволяти видалення ресурсів із Work-List і Black-List лише у головному меню за умови невід'ємного балансу
F4	Система повинна забороняти видалення ресурсів із Work-List і Black-List під час активної сесії
F5	Система повинна забороняти видалення ресурсів із Work-List і Black-List, якщо часовий баланс користувача є від'ємним

Кінець таблиці 1.2

1	2
F6	Система повинна дозволяти встановлення денного нормативу робочого часу
F7	Система повинна автоматично визначати активне вікно або активний вебресурс користувача
F8	Система повинна зараховувати робочий час лише для ресурсів, що належать до Work-List
F9	Система повинна не зараховувати час, проведений поза Work-List, як продуктивний
F10	Система повинна обчислювати часовий баланс на основі денного нормативу та фактично відпрацьованого часу
F11	Система повинна блокувати ресурси з Black-List, якщо часовий баланс користувача є від'ємним
F12	Система повинна зберігати налаштування, історію сесій та статистику роботи користувача
F13	Система повинна працювати у фоновому режимі та продовжувати моніторинг активного ресурсу під час робочої сесії
F14	Система повинна надавати користувачу можливість тимчасово призупинити облік часу в режимі Break
F15	Система повинна надавати функцію Reset для аварійного завершення поточного обмеження із фіксацією події в історії роботи та встановленням штрафного балансу —1 година.
F16	Система повинна відображати користувачу поточний стан сесії, зарахований робочий час, денний норматив і значення часового балансу

Наведені вимоги використовуються як основа для подальшого проєктування та тестування.

1.3.2 Нефункціональні вимоги

Нефункціональні вимоги визначають якісні характеристики застосунку, обмеження його роботи та очікувані властивості програмного забезпечення (табл. 1.3). Для «Honestime» важливими є стабільність фоновий моніторингу, локальне збереження даних, прозорість роботи для користувача, помірне використання системних ресурсів і коректна робота в межах обраної операційної системи.

Таблиця 1.3 — Нефункціональні вимоги до програмної системи «Honestime»

Категорія	Вимога
1	2
Продуктивність	Застосунок повинен виконувати фоновий моніторинг активного ресурсу без помітного сповільнення роботи операційної системи та інших користувацьких програм
Частота оновлення	Стан активного ресурсу та робочого таймера повинен оновлюватися з інтервалом, достатнім для коректного обліку робочого часу
Надійність	Значення часового балансу, налаштування, списки ресурсів та історія сесій повинні зберігатися після закриття або перезапуску застосунку
Зручність	Інтерфейс повинен надавати користувачу доступ до основних дій: запуску сесії, завершення роботи, перегляду балансу, редагування списків і перегляду статистики
Прозорість	Користувач повинен бачити, чи зараховується поточний час як робочий, який ресурс активний, скільки часу вже зараховано та який поточний баланс

Кінець таблиці 1.3

1	2
Безпека даних	Дані користувача повинні зберігатися локально та не передаватися стороннім сервісам
Сумісність	У межах цієї роботи застосунок повинен коректно працювати в операційній системі «Windows»

1.3.3 Обмеження системи

Під час формування вимог також визначено межі відповідальності системи. «Honestime» не є корпоративною системою стеження, планувальником задач або універсальним блокувальником відволікаючих ресурсів. Його призначення — персональний облік і динамічне оцінювання фактичного робочого часу.

Система не аналізує зміст виконуваної роботи. Час зараховується лише тоді, коли активний ресурс належить до Work-List, тому точність обліку залежить від коректності його налаштування користувачем.

Black-List у межах розроблюваної системи не використовується для визначення продуктивності. Його призначення полягає у допоміжному обмеженні доступу до ресурсів, які працівник самостійно визначив як відволікаючі. Таке обмеження застосовується у випадку, коли часовий баланс користувача є від’ємним.

Окреме обмеження стосується редагування списків. У головному меню користувач може додавати ресурси до Work-List і Black-List незалежно від стану балансу. Видалення елементів дозволяється лише за умови невід’ємного балансу. Якщо баланс є від’ємним, ця дія блокується до моменту компенсації часового боргу.

Таке правило потрібне для збереження логіки часового боргу. Оскільки від’ємний баланс формується на основі встановленого плану та

поточних списків, користувач не повинен змінювати умови відпрацювання після фіксації невиконаного плану.

Система також не гарантує повного усунення прокрастинації. Вона надає інструменти для об'єктивного обліку часу, контролю доступу до визначених ресурсів і аналізу статистики, однак кінцева ефективність залежить від налаштувань користувача та його готовності дотримуватися обраного режиму роботи.

1.4 Висновки за розділом 1

У першому розділі було проаналізовано проблему обліку робочого часу в цифровому середовищі. Визначено, що загальна тривалість перебування за комп'ютером не завжди відповідає фактичній продуктивній діяльності, оскільки користувач може перемикатися між робочими та сторонніми ресурсами.

Було розглянуто застосунки Focusmate, Cold Turkey Blocker та Habitica. Встановлено, що вони використовують різні підходи до зменшення прокрастинації, однак не повністю відповідають задачі автоматизованого оцінювання фактичного робочого часу на основі активного ресурсу.

Сформовано функціональні та нефункціональні вимоги до системи «Honestime», визначено вхідні й вихідні дані системи, критерії коректності роботи та основні обмеження. Обґрунтовано доцільність розробки локальної десктопної системи для персонального обліку фактично зарахованого робочого часу.

Завдання роботи:

- проаналізувати предметну область і визначити основні проблеми цифрової прокрастинації;

- провести огляд існуючих програмних рішень для підвищення продуктивності, самоконтролю робочого часу та обмеження цифрових дистракторів;

- сформулювати функціональні та нефункціональні вимоги до системи персонального обліку фактичного робочого часу працівника;
- спроектувати архітектуру програмної системи;
- реалізувати основні модулі програмного забезпечення;
- провести тестування працездатності розробленої програмної системи.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМНОЇ СИСТЕМИ

2.1 Вибір мови програмування

Для реалізації програмної системи «Honestime» було обрано мову програмування Python 3. Вибір здійснювався з урахуванням специфіки проєкту: необхідності швидкої розробки десктопної програми, підтримки графічного інтерфейсу, роботи у фоновому режимі, взаємодії з локальною базою даних та реалізації логіки обліку фактичного робочого часу.

У межах цієї роботи система реалізується для операційної системи Windows. Такий вибір пов'язаний із необхідністю отримання інформації про активне вікно, назву процесу, системний ідентифікатор вікна та роботу системи у фоновому режимі. Для цього використовуються системні інтерфейси програмування застосунків (API) та бібліотеки Python. Використання Python дає змогу реалізувати ці функції з використанням готових бібліотек і водночас зберегти просту структуру коду [10].

Для обґрунтування вибору було розглянуто три технологічні варіанти: Python 3 з PyQt6, C++ з Qt та C# з WPF. Порівняльну характеристику мов програмування та засобів розробки наведено у табл. 2.1.

Таблиця 2.1 — Порівняльна характеристика мов програмування

Критерій порівняння	Python 3 + PyQt6	C++ + Qt	C# (WPF/.NET)
1	2	3	4
Швидкість розробки	Висока завдяки лаконічному синтаксису та великій кількості бібліотек	Нижча через складнішу мову та більшу кількість службового коду	Середня, зручна для Windows-застосунків

Кінець таблиці 2.1

1	2	3	4
Графічний інтерфейс	PyQt6 надає засоби для створення повноцінного десктопного інтерфейсу	Qt забезпечує широкі можливості GUI, але потребує складнішої реалізації	WPF забезпечує розвинений інтерфейс для Windows
Робота з локальною БД	Підтримується через SQLite та стандартні бібліотеки Python	Підтримується, але потребує більшої кількості коду	Підтримується через .NET-інструменти
Робота з фоновими процесами	Реалізується через механізми Python і PyQt6	Має високу продуктивність, але складніша в реалізації	Добре підтримується в середовищі Windows
Взаємодія з ОС	Можлива через системні бібліотеки та Python-обгортки	Висока завдяки Qt, але збірка складніша	Зручна для Windows-середовища
Складність підтримки коду	Низька або середня	Вища через складність мови	Середня
Доцільність для «Honestime»	Висока: достатня функціональність і швидка розробка	Надлишкова для задач проєкту	Доцільна для Windows, але менш гнучка для швидкої розробки

Аналіз табл. 2.1 показує, що C++ з Qt забезпечує високу продуктивність і широкі можливості для створення десктопних застосунків,

однак складність розробки та супроводу є надлишковою для задач цього проєкту. C# з WPF є зручним варіантом для Windows-застосунків, проте його використання сильніше прив'язує програму до екосистеми .NET [11].

Python 3 у поєднанні з PyQt6 є доцільним вибором для «Honestime», оскільки забезпечує достатню функціональність для створення графічного інтерфейсу користувача (GUI), реалізації фонові роботи, взаємодії з локальною базою даних і швидкої розробки прикладної логіки. Для створення графічного інтерфейсу обрано PyQt6 — набір прив'язок Python до Qt v6 [12]. Цей фреймворк дозволяє реалізувати головне вікно, вікно робочої сесії, діалогові вікна, системний трей, таймери та обробку подій користувача.

Для локального збереження даних обрано базу даних (БД) SQLite3. SQLite є самодостатньою безсерверною БД, що не потребує окремого налаштування серверного процесу [13]. Такий підхід відповідає вимогам системи, оскільки Work-List, Black-List, історія сесій, налаштування, статистика та часовий баланс повинні зберігатися локально на комп'ютері користувача.

Основним інтегрованим середовищем розробки (IDE) обрано Visual Studio Code. Його використання доцільне завдяки підтримці Python, інтеграції з системами контролю версій, засобам налагодження та зручності роботи з проєктами невеликого й середнього розміру.

Отже, для реалізації «Honestime» обрано зв'язку Python 3, PyQt6 та SQLite3. Цей набір засобів забезпечує швидку розробку, підтримку графічного інтерфейсу, фонову роботу та локальне збереження даних, що відповідає сформованим функціональним і нефункціональним вимогам.

2.2 Проєктування архітектури програмного забезпечення

Архітектура системи «Honestime» побудована за модульним принципом із розмежуванням функцій між інтерфейсним рівнем, керувальною логікою, модулем моніторингу активного ресурсу, модулем

роботи зі списками ресурсів і модулем локального збереження даних. Такий підхід забезпечує логічну відокремленість основних частин програмного забезпечення, спрощує супровід коду, тестування окремих компонентів і подальше розширення функціональності.

Центральним компонентом архітектури є FocusController, який координує виконання робочої сесії та забезпечує взаємодію між іншими модулями системи. Його функції охоплюють обробку інформації про активне вікно, перевірку ресурсу за списками Work-List і Black-List, оновлення фактично зарахованого часу, зміну часового балансу та передавання актуальних даних до інтерфейсу користувача. Отримання низькорівневої інформації про активне вікно винесено до компонента WindowMonitor, робота зі списками ресурсів реалізована через ResourceManager, а взаємодія з локальною базою даних SQLite інкапсульована у компоненті DatabaseManager.

2.2.1 Діаграма прецедентів

На рис. 2.1 зображено діаграму прецедентів системи «Honestime». Основним актором є користувач, який налаштовує Work-List і Black-List, встановлює денний норматив, запускає робочу сесію, переглядає статистику та завершує роботу.

Після запуску сесії система переходить у режим фонового моніторингу: визначає активне вікно або вебресурс, перевіряє його належність до Work-List і зараховує час лише у разі відповідності заданим правилам. Для тимчасового призупинення обліку передбачено режим Break.

Під час завершення роботи система фіксує результат сесії, порівнює фактично зарахований час із денним нормативом і оновлює часовий баланс. Якщо баланс стає від'ємним, система блокує ресурси з Black-List і забороняє видалення елементів зі списків до моменту компенсації часового боргу.

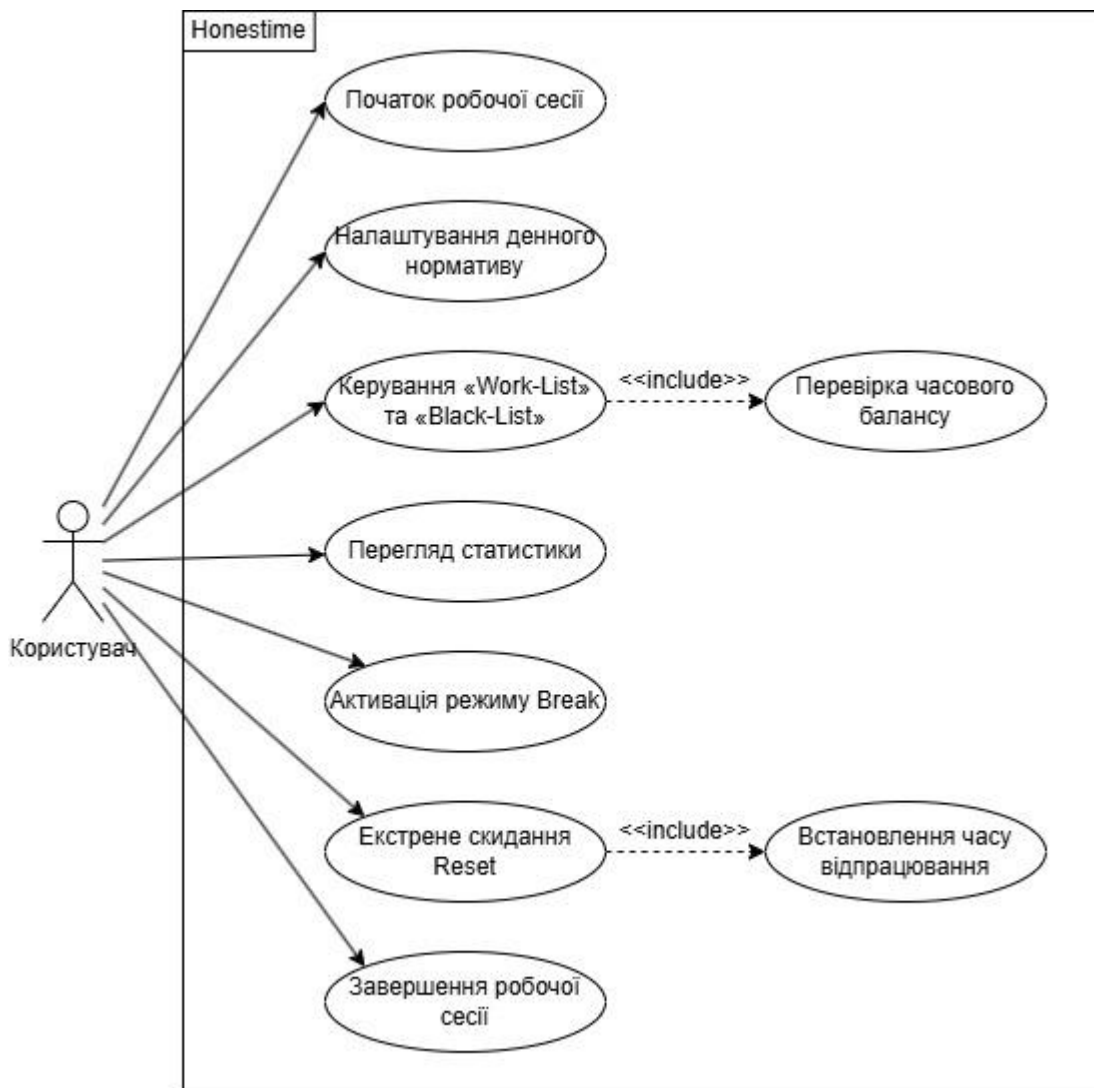


Рисунок 2.1 — Діаграма прецедентів системи «Honestime»

2.2.2 Діаграма класів

На рис. 2.2 представлено діаграму класів системи «Honestime», яка відображає основні програмні сутності системи, їхні атрибути, методи та зв'язки між компонентами. Архітектура побудована навколо класу `FocusController`, який координує робочу сесію, моніторинг активного ресурсу, взаємодію зі списками та збереження даних.

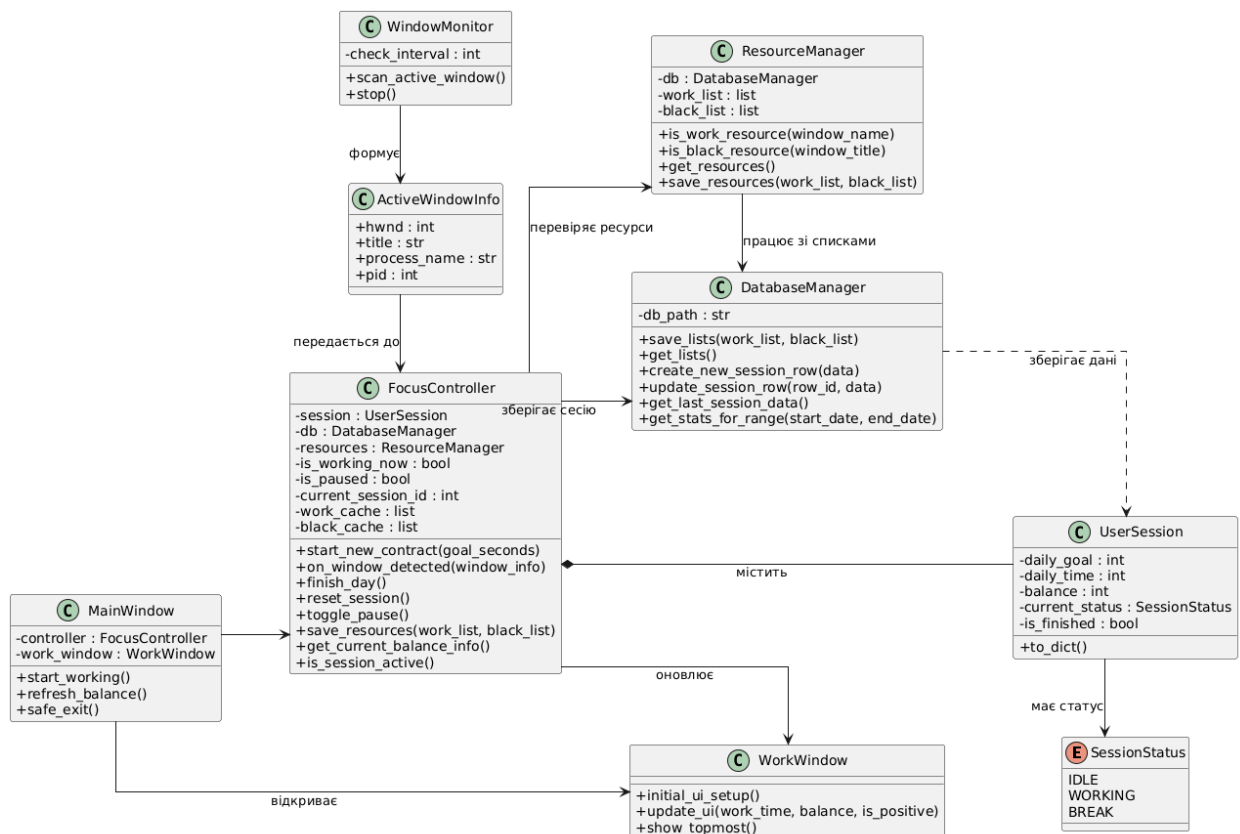


Рисунок 2.2 — Діаграма класів системи «Honestime»

Клас `MainWindow` належить до інтерфейсного рівня системи та реалізує головне вікно системи. Він забезпечує запуск робочої сесії, оновлення поточного балансу, безпечне завершення роботи, а також відкриття вікон налаштувань, статистики та діалогу скидання.

Клас `WorkWindow` відповідає за відображення стану активної робочої сесії. Через нього користувач системи бачить фактично зарахований робочий час, поточний баланс і стан виконання робочого плану. Оновлення цього вікна виконується на основі даних, які формує `FocusController`.

Центральним керувальним класом є `FocusController`. Він зберігає інформацію про поточну сесію, обробляє дані про активне вікно, перевіряє активний ресурс через `ResourceManager`, оновлює часовий баланс, керує режимом паузи, завершує робочий день і передає дані для оновлення інтерфейсу.

Клас `WindowMonitor` відповідає за періодичне визначення активного вікна операційної системи. Отримані дані передаються до `FocusController` у

вигляді об'єкта `ActiveWindowInfo`, який містить ідентифікатор вікна, заголовок, назву процесу та ідентифікатор процесу.

Клас `ResourceManager` забезпечує роботу зі списками `Work-List` і `Black-List`. Він завантажує списки з бази даних, перевіряє належність активного ресурсу до робочих або обмежених ресурсів і зберігає оновлені списки.

Клас `DatabaseManager` відповідає за доступ до локального сховища даних. Через нього виконуються операції збереження списків ресурсів, створення та оновлення записів робочих сесій, збереження статистичних даних і отримання статистики за вибраний період.

Клас `UserSession` описує поточний стан робочої сесії: денний норматив, фактично зарахований робочий час, часовий баланс, статус сесії та ознаку завершення. Перелік `SessionStatus` формалізує можливі стани сесії: очікування, робота та перерва.

Отже, діаграма класів показує взаємодію інтерфейсного рівня, керувального компонента, модуля моніторингу активного вікна, модуля роботи з ресурсами, локальної бази даних і моделі робочої сесії.

2.2.3 Модульна структура програмної системи

На рис. 2.3 представлено модульну структуру програмної системи «Honestime», яка побудована навколо керування робочою сесією. Центральною частиною системи є `FocusController`, який взаємодіє з інтерфейсом користувача, отримує дані про активний ресурс, перевіряє його за `Work-List` і `Black-List` та передає результати для збереження.

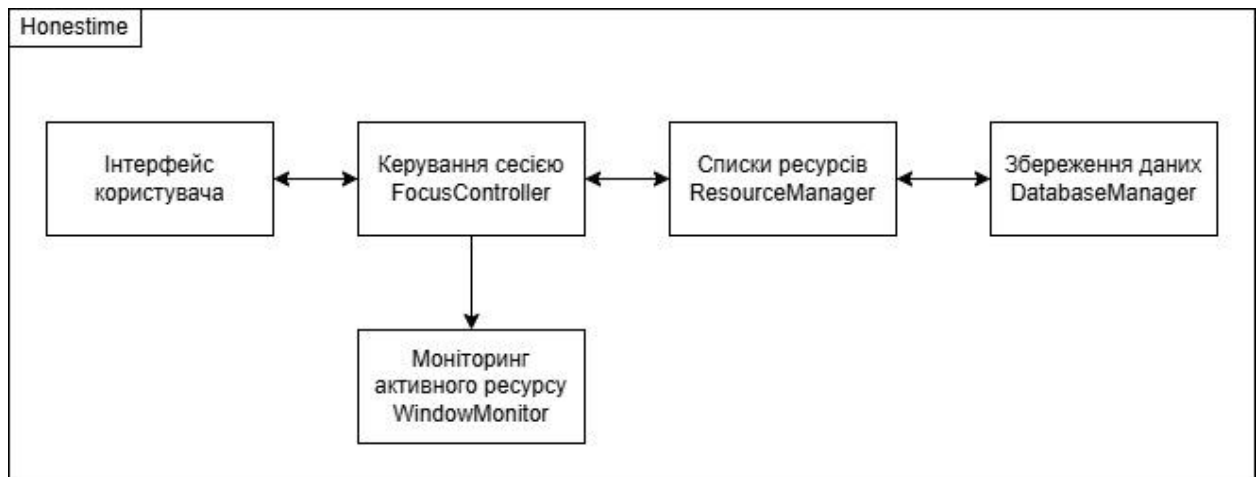


Рисунок 2.3 — Модульна структура програмної системи «Honestime»

Інтерфейс користувача забезпечує введення денного нормативу, запуск і завершення робочої сесії, перегляд поточного стану, статистики та редагування списків. Моніторинг активного ресурсу, представлений класом WindowMonitor, визначає активне вікно або вебресурс і передає ці дані до FocusController.

Частина системи, пов'язана зі списками ресурсів, представлена класом ResourceManager. Вона забезпечує роботу з Work-List і Black-List, тобто з переліками робочих і відволікаючих ресурсів. Збереження даних, представлене класом DatabaseManager, відповідає за локальне зберігання списків ресурсів, історії робочих сесій, статистики та значення часового балансу.

2.3 Проєктування структури бази даних

Для локального збереження даних системи «Honestime» використовується база даних SQLite. Такий вибір відповідає призначенню програми, оскільки програмна система є персональним десктопним інструментом і не потребує розгортання окремої серверної частини.

На логічному рівні база даних містить дві групи даних: списки ресурсів користувача та історію робочих сесій. До списків ресурсів належать

Work-List і Black-List, а записи сесій зберігають параметри виконання робочого плану.

Для збереження цих даних передбачено дві основні таблиці: resources та sessions. Структури наведено у табл. 2.2 та табл. 2.3.

Таблиця 2.2 — Структура таблиці resources

Поле	Тип даних	Призначення
id	INTEGER	Первинний ключ запису, що автоматично збільшується
name	TEXT	Назва ресурсу або ідентифікатор активного вікна
type	TEXT	Тип списку, до якого належить ресурс: work або black

Таблиця 2.3 — Структура таблиці sessions

Поле	Тип даних	Призначення
id	INTEGER	Первинний ключ запису, що автоматично збільшується
date	TEXT	Дата та час створення запису сесії
daily_goal	INTEGER	Денний норматив робочого часу
daily_time	INTEGER	Фактично зарахований робочий час у межах сесії
balance	INTEGER	Поточне значення часового балансу

Таблиця resources використовується для збереження двох списків ресурсів: Work-List і Black-List. Поле name містить назву ресурсу або фрагмент назви вікна чи процесу, а поле type визначає належність ресурсу до робочих або обмежуваних елементів.

Така структура дає змогу зберігати правила класифікації ресурсів у єдиній таблиці та розрізняти їх за типом. Детальний опис програмних операцій читання й запису списків наведено у підрозділі 3.2.5.

Таблиця sessions використовується для збереження історії роботи користувача. Кожний запис містить дату створення, встановлений денний норматив, фактично зарахований робочий час і значення часового балансу. Наявність таблиці дає змогу відновлювати основні параметри після повторного запуску програми та формувати статистику роботи за вибраний період.

Програмну реалізацію операцій роботи з базою даних наведено у підрозділі 3.2.5.

2.4 Структурна схема програмного проєкту

Структура програмного проєкту «Honestime» організована за модульним принципом. Основний програмний код розміщено в каталозі src, що дає змогу відокремити вихідні файли системи від службових каталогів, зібраних файлів і ресурсів проєкту. Структуру програмного проєкту наведено на рис. 2.4.

У межах каталогу виділено пакет database для роботи з локальною базою даних і пакет ui для модулів графічного інтерфейсу користувача. Також у структурі проєкту розміщено окремі програмні модулі, що відповідають за запуск системи, керування робочою логікою, моніторинг активного вікна, обробку ресурсів, службовий контроль процесу та допоміжні функції.

Каталоги build і dist є службовими та використовуються під час збирання системи у виконуваний файл. Вони не містять основної логіки програми, а призначені для підготовки фінальної версії програмного продукту.

Пакет database призначений для роботи з локальною базою даних SQLite. Файл database.py реалізує створення структури бази даних, збереження списків ресурсів, створення й оновлення записів робочих сесій, а також отримання статистичних даних. Файл honestime.db є локальним файлом бази даних, у якому зберігаються списки ресурсів, історія сесій і значення часового балансу.

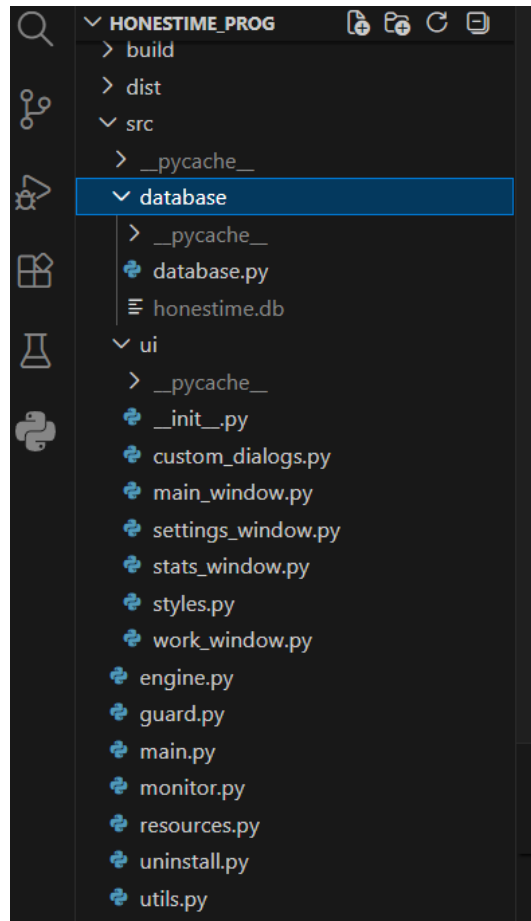


Рисунок 2.4 — Структура програмного проєкту «Honestime»

Пакет ui містить модулі графічного інтерфейсу: головне вікно, робоче вікно активної сесії, вікно налаштувань, статистику, користувацькі діалоги та стилі інтерфейсу.

Файл main.py є точкою входу до системи. Він створює основні компоненти системи, ініціалізує базу даних, менеджер ресурсів, контролер фокусування, монітор активного вікна та головне вікно інтерфейсу.

Модулі `engine.py`, `monitor.py` і `resources.py` реалізують основну логіку системи: керування робочою сесією, визначення активного вікна та перевірку ресурсів за `Work-List` і `Black-List`.

Службові модулі `guard.py`, `utils.py` та `uninstall.py` використовуються для допоміжних операцій: контролю роботи системи, роботи зі шляхами до ресурсів, службових дій інтерфейсу та очищення компонентів програми.

Таким чином, структура проєкту відображає архітектурний поділ системи на рівень інтерфейсу, логіку керування, моніторинг активності, збереження даних і службові засоби. Детальний опис реалізації кожного модуля наведено у розділі 3.

3 РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

3.1 Алгоритм функціонування програмної системи

У цьому підрозділі наведено загальний алгоритм роботи програми під час запуску робочої сесії, обліку часу, оновлення балансу та завершення роботи. Загальну блок-схему функціонування системи наведено на рис. 3.1.

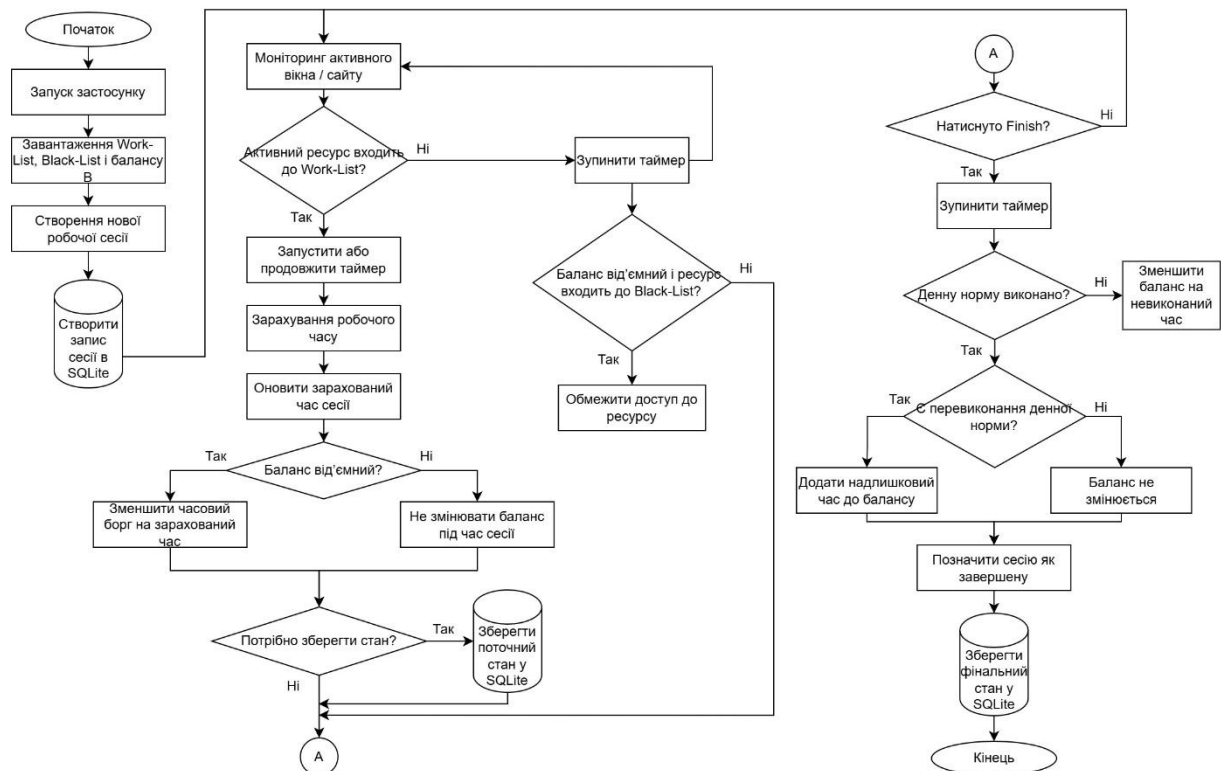


Рисунок 3.1 — Алгоритм функціонування системи «Honestime»

Перевірка активного ресурсу виконується за заголовком вікна та назвою процесу. Це дозволяє визначати робочі й відволікаючі ресурси не лише за назвою вікна, а й за запущеним процесом. Умови запуску та зупинки таймера наведено в табл. 3.1.

Таблиця 3.1 — Умови запуску та зупинки таймера

Умова	Дія системи
Активне вікно входить до Work-List	Таймер запускається
Активне вікно не входить до Work-List	Таймер зупиняється
Увімкнено режим Break	Таймер зупиняється
Робочу сесію завершено	Таймер не запускається
Користувач неактивний понад 3 хвилини	Таймер зупиняється
Баланс від’ємний і активний ресурс входить до Black-List	Активне вікно згортається, таймер зупиняється

Оновлення балансу відбувається під час зарахування робочого часу. Якщо баланс є від’ємним, зарахований час використовується для погашення боргу. Якщо денний норматив уже виконано, додатковий робочий час додається до позитивного балансу. Під час завершення сесії система додатково перевіряє виконання денного нормативу: якщо норматив не виконано, невиконаний час віднімається від балансу.

Функція Reset використовується для екстреного скидання поточного стану. Вона не обнуляє баланс, а встановлює штрафне значення, після чого сесія позначається завершеною.

У результаті алгоритм забезпечує облік лише фактично підтвердженого робочого часу, контроль бездіяльності користувача, періодичне збереження сесії та застосування Black-List-обмежень за від’ємного балансу.

3.2 Опис модулів реалізованої програмної системи

Оскільки загальну архітектуру системи та взаємодію компонентів

наведено в розділі 2, у цьому підрозділі описано програмні модулі, які реалізують алгоритм функціонування, поданий у підрозділі 3.1.

Програмне забезпечення «Honestime» реалізовано за модульним принципом. Такий поділ спрощує підтримку програми, оскільки логіка обліку часу не змішується з інтерфейсом користувача або операціями з базою даних. Загальне призначення основних модулів наведено в табл. 3.2.

Таблиця 3.2 — Призначення модулів системи «Honestime»

Модуль	Призначення
main.py	Точка входу до системи, створення основних об'єктів і запуск графічного інтерфейсу
engine.py	Реалізація логіки робочої сесії: таймер, баланс, Break, Reset, AFK-перевірка, Black-List-обмеження
monitor.py	Визначення активного вікна, його заголовка, процесу та ідентифікатора
resources.py	Завантаження, збереження та перевірка списків Work-List і Black-List
database.py	Збереження списків ресурсів, створення та оновлення записів робочих сесій
ui	Реалізація графічного інтерфейсу користувача
guard.py	Допоміжний службовий модуль контролю роботи системи
utils.py	Допоміжні функції для службових операцій
styles.py	Стилі графічного інтерфейсу

Основна взаємодія модулів відбувається через FocusController. Модуль monitor.py передає дані про активне вікно, engine.py обробляє ці дані та визначає, чи потрібно запускати або зупиняти таймер. Модуль database.py зберігає стан сесії та списки ресурсів, а resources.py використовується для отримання й перевірки Work-List і Black-List.

У наступних підрозділах наведено опис кожного модуля реалізованої програми. Спочатку розглядається модуль запуску системи, після чого описуються модулі керування сесією, моніторингу активного вікна, роботи з ресурсами, базою даних, графічним інтерфейсом і службовими компонентами.

3.2.1 Модуль запуску та ініціалізації системи

Модуль `main.py` є точкою входу в систему «Honestime». Його завдання полягає у початковій підготовці програми до роботи: перевірці умов запуску, створенні основних об'єктів і запуску головного вікна.

Під час запуску `main.py` виконує такі дії:

- перевіряє, чи не запущено інший екземпляр системи;
- додає службовий процес до автозапуску та перевіряє його роботу;
- створює об'єкт `QApplication`;
- застосовує глобальний стиль та іконку програми;
- створює об'єкти бази даних, ресурсів, сесії, контролера та моніторингу;
- підключає сигнал монітора активного вікна до контролера сесії;
- створює та відображає головне вікно.

Найважливішим у цьому модулі є формування зв'язку між технічним моніторингом і логікою робочої сесії. Перший компонент отримує дані про активне вікно, а другий використовує ці дані для прийняття рішення щодо обліку часу. Фрагмент програмної реалізації модуля запуску системи наведено в додатку А.1.

3.2.2 Модуль керування робочою сесією

Модуль engine.py містить основну логіку роботи системи. У ньому визначено стани сесії, модель UserSession та клас FocusController. Контролер запускає нову робочу сесію, обробляє дані активного вікна, запускає або зупиняє таймер, оновлює зарахований час і баланс, виконує AFK-перевірку, обробляє режим Break, завершення сесії та Reset. Фрагмент програмної реалізації модуля керування робочою сесією наведено в додатку А.2.

3.2.2.1 Структура робочої сесії

У модулі використано три основні програмні елементи:

- SessionStatus — перелік станів робочої сесії;
- UserSession — об'єкт поточного стану сесії;
- FocusController — контролер керування робочим процесом.

UserSession зберігає основні значення, які змінюються під час роботи: денний норматив, фактично зарахований час, часовий баланс, поточний стан і ознаку завершення сесії.

Об'єкт UserSession передається до FocusController, який змінює його значення під час роботи: збільшує зарахований час, оновлює баланс, змінює стан сесії та фіксує її завершення. Для збереження цих даних у базі використовується перетворення стану сесії у словник.

3.2.2.2 Облік робочого часу

Облік часу не починається автоматично після запуску сесії. Спочатку контролер отримує дані про активне вікно та перевіряє, чи належить воно до Work-List. Якщо активний ресурс є робочим, облік часу запускається або продовжується. Якщо користувач переходить до іншого ресурсу, облік зупиняється.

3.2.2.3 Розрахунок часового балансу

Часовий баланс змінюється під час роботи та після завершення сесії. Якщо користувач має борг, зарахований робочий час спочатку використовується для його компенсації. Якщо денний норматив уже виконано, подальша робота може збільшувати позитивний баланс. Правила оновлення балансу наведено в табл. 3.3.

Таблиця 3.3 — Правила оновлення часового балансу

Умова	Дія
Баланс має від’ємне значення	Зарахований робочий час компенсує борг
Денний норматив виконано, боргу немає	Подальша робота збільшує позитивний баланс
Сесію завершено до виконання нормативу	Різниця між планом і невиконаним часом віднімається від балансу
Денний норматив виконано	Баланс не зменшується

3.2.2.4 Обробка перерви та бездіяльності

Під час роботи контролер використовує завантажені списки Work-List і Black-List. Work-List визначає, коли час можна зараховувати як робочий. Black-List використовується як механізм обмеження, якщо часовий баланс користувача має від’ємне значення.

Порядок обробки активного вікна такий:

- перевіряється, чи сесія активна;
- перевіряється, чи не завершена сесія;
- визначається належність активного ресурсу до Work-List;
- визначається належність активного ресурсу до Black-List;
- перевіряється бездіяльність користувача;

- приймається рішення щодо обліку часу або обмеження вікна.

Якщо баланс має від’ємне значення і активне вікно відповідає Black-List, воно примусово згортається. При цьому стороння система не завершується, але користувач не може комфортно продовжувати взаємодію з відволікаючим ресурсом до компенсації боргу.

Бездіяльність користувача обробляється окремо. Якщо протягом визначеного часу не фіксується взаємодія з клавіатурою або мишею, облік часу зупиняється навіть тоді, коли активним залишається робочий ресурс.

3.2.2.5 Завершення, пауза та скидання сесії

Користувач може тимчасово призупинити облік часу. У цьому випадку сесія не завершується, але робочий час не збільшується до відновлення роботи.

Після завершення сесії контролер порівнює фактично зарахований час із денним нормативом. Якщо норматив не виконано, різниця впливає на часовий баланс як борг. Якщо норматив виконано, підсумкові значення просто фіксуються.

Скидання сесії використовується як окремий аварійний сценарій. У цьому випадку поточний прогрес обнуляється, а до балансу застосовується штрафне значення. Додаткові сценарії керування робочою сесією наведено у табл. 3.4.

Таблиця 3.4 — Додаткові сценарії керування сесією

Сценарій	Результат
1	2
Пауза	Тимчасова зупинка обліку часу
Завершення сесії	Фіксація результату та оновлення балансу

Кінець таблиці 3.4

1	2
Скидання сесії	Обнулення поточного прогресу і застосування штрафу
Зміна списків ресурсів	Оновлення списків, які використовує контролер

3.2.3 Модуль моніторингу активного вікна

Модуль `monitor.py` забезпечує технічну основу для автоматичного визначення робочої активності користувача. Його завдання — отримати інформацію про поточне активне вікно та передати її до контролера робочої сесії.

У модулі реалізовано дві основні сутності:

- `ActiveWindowInfo` — структура даних про активне вікно, які передаються до `FocusController` через сигнал `window_detected` (табл. 3.5);
- `WindowMonitor` — клас, який періодично виконує перевірку активного вікна.

Таблиця 3.5 — Структура `ActiveWindowInfo`

Поле	Призначення
<code>hwnd</code>	Системний ідентифікатор активного вікна
<code>title</code>	Заголовок активного вікна
<code>process_name</code>	Назва процесу, якому належить вікно
<code>pid</code>	Ідентифікатор процесу

Моніторинг виконується з інтервалом 800 мс. Під час кожної перевірки модуль отримує активне вікно, визначає його заголовок,

ідентифікатор процесу та назву процесу. Після цього формується об'єкт `ActiveWindowInfo`, який передається через сигнал `window_detected`.

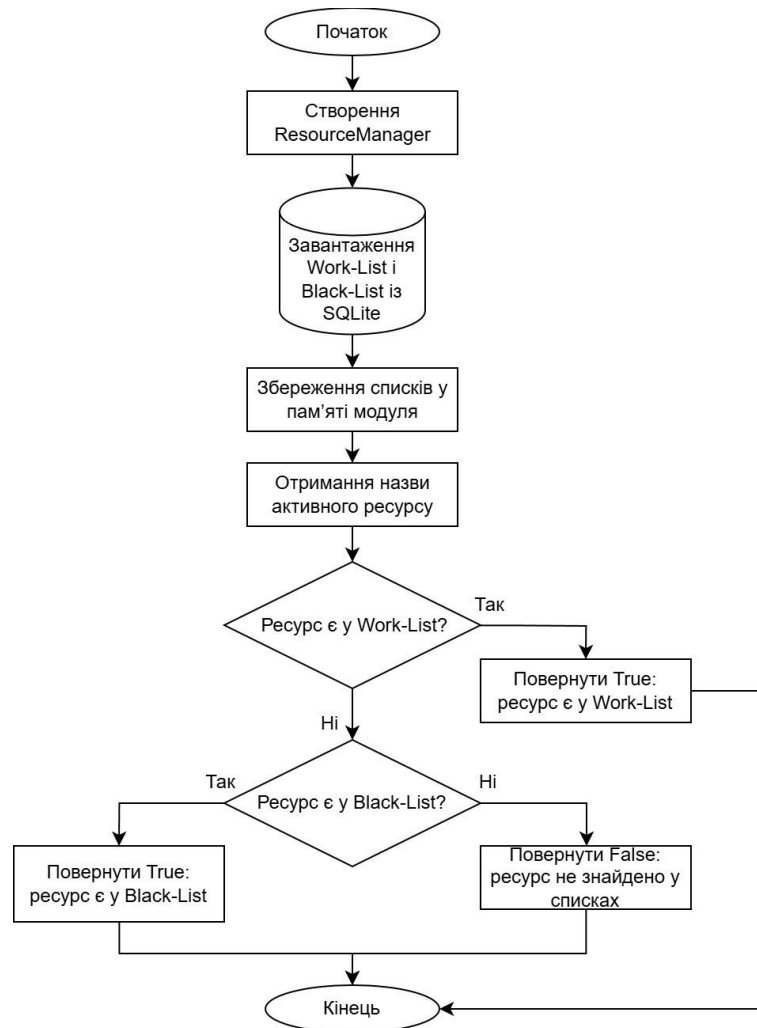
Завдяки цьому контролер сесії працює вже не з низькорівневими системними викликами, а з готовим об'єктом, який містить усю необхідну інформацію про активне вікно. Фрагмент програмної реалізації модуля моніторингу активного вікна наведено в додатку А.3.

3.2.4 Модуль роботи зі списками ресурсів

У модулі використовується клас `ResourceManager`. Під час створення він завантажує з бази даних два списки:

- `Work_list` — ресурси, що вважаються робочими;
- `Black_list` — ресурси, які можуть бути обмежені при від'ємному балансі.

Основна логіка модуля полягає у перевірці назви активного вікна за цими списками. Якщо назва або її частина збігається з елементом `Work-List`, ресурс вважається робочим. Якщо збіг знайдено у `Black-List`, модуль лише повертає результат перевірки. Рішення щодо зарахування часу, зупинки таймера або згортання `Black-List`-вікна приймає не `ResourceManager`, а `FocusController`. Загальний принцип роботи модуля наведено на рис. 3.2.

Рисунок 3.2 — Принцип роботи модуля `resources.py`

Модуль `resources.py` реалізує клас `ResourceManager`, який відповідає за роботу зі списками ресурсів системи. Під час створення об'єкта модуль отримує доступ до менеджера бази даних, завантажує з `SQLite` списки `Work-List` і `Black-List` та зберігає їх у пам'яті.

Під час перевірки активного ресурсу назва вікна порівнюється з елементами `Work-List` і `Black-List`. Якщо ресурс знайдено у `Work-List`, метод повертає значення `True` для робочого ресурсу. Якщо ресурс знайдено у `Black-List`, метод повертає значення `True` для обмежуваного ресурсу. Якщо збігів не виявлено, повертається значення `False`. Отриманий результат надалі використовується контролером робочої сесії для запуску або зупинки таймера та застосування обмежень. Фрагмент програмної реалізації модуля роботи з ресурсами наведено в додатку А.4.

3.2.5 Модуль локального збереження даних

Модуль database.py реалізує доступ до локальної бази даних системи. Схему бази даних було розглянуто у розділі 2, тому в цьому підрозділі наведено лише особливості її використання в реалізованій програмі.

Основним класом модуля є DatabaseManager. Через нього інші частини системи отримують і зберігають дані, не звертаючись до SQL-запитів напряму.

Модуль використовується для таких операцій:

- завантаження останнього збереженого балансу та денного нормативу;
- збереження Work-List і Black-List;
- створення нового запису робочої сесії;
- оновлення фактично зарахованого часу та балансу;
- отримання даних для побудови статистики.

Під час запуску системи з бази береться останній баланс і денний норматив. Робочий час нової сесії не переноситься, оскільки він має накопичуватися заново.

Під час активної роботи модуль оновлює поточний запис сесії. Для статистики він повертає дані за вибраний період, групуючи робочий час за днями.

Фрагмент програмної реалізації модуля роботи з базою даних наведено в додатку А.5.

3.2.6 Модулі графічного інтерфейсу

Модулі графічного інтерфейсу реалізовано у пакеті ui. Вони відповідають за відображення станів системи, зміну візуальних елементів залежно від балансу та режиму сесії, а також за передавання дій користувача до контролера програми.

3.2.6.1 Початковий стан головного вікна

Початковий стан головного вікна відображається після запуску системи. У цьому стані користувач бачить поле встановлення денного нормативу, поточний часовий баланс, кнопку запуску сесії та елементи переходу до статистики й налаштувань.

Значення балансу виводиться окремим індикатором. Якщо баланс додатний, індикатор має зелений колір (рис. 3.3); якщо баланс від’ємний — червоний (рис. 3.4). Це дозволяє відразу визначити, чи має користувач накопичений резерв часу або борг. Фрагмент програмної реалізації головного вікна системи наведено в додатку А.6.

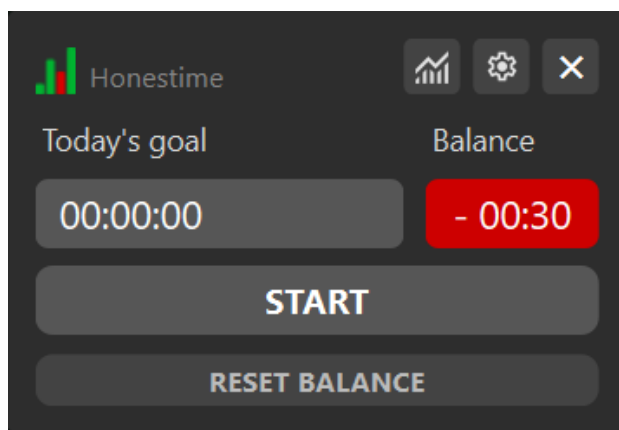


Рисунок 3.3 — Початковий стан головного вікна з додатним балансом

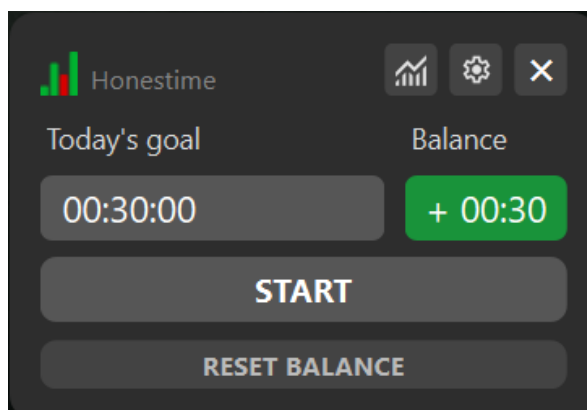


Рисунок 3.4 — Початковий стан головного вікна з від’ємним балансом

3.2.6.2 Стан активної робочої сесії

Після запуску сесії головне вікно приховується, а на екрані відображається робоче вікно. У цьому стані показуються поточний баланс, фактично зарахований робочий час і встановлена мета.

Показник балансу змінює колір за тим самим принципом, що й у головному вікні: зелений колір відповідає додатному балансу, червоний — від’ємному. Робочий час оновлюється під час активної сесії (рис. 3.5, рис. 3.6). Фрагмент програмної реалізації вікна робочої сесії наведено в додатку А.7.

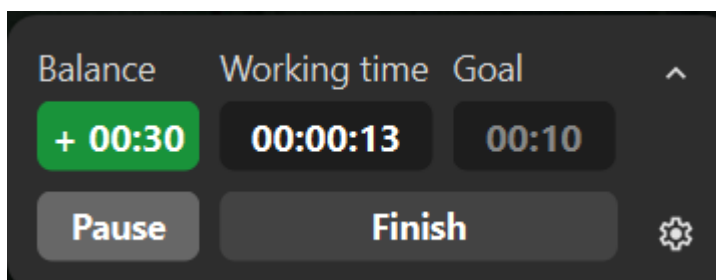


Рисунок 3.5 — Активна робоча сесія з додатним балансом

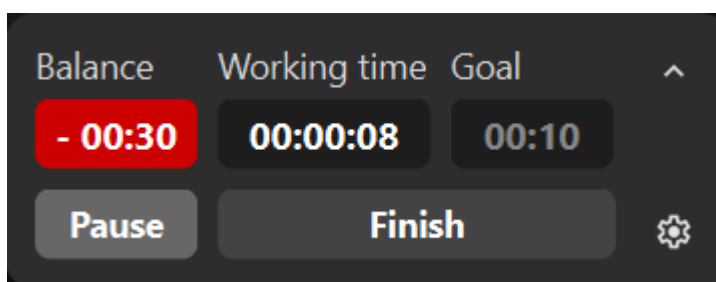


Рисунок 3.6 — Активна робоча сесія з від’ємним балансом

3.2.6.3 Режим перерви

Режим перерви використовується для тимчасового призупинення робочої сесії без її завершення. У цьому стані кнопка «Pause» змінюється на «Continue», що показує можливість повернення до роботи (рис. 3.7).

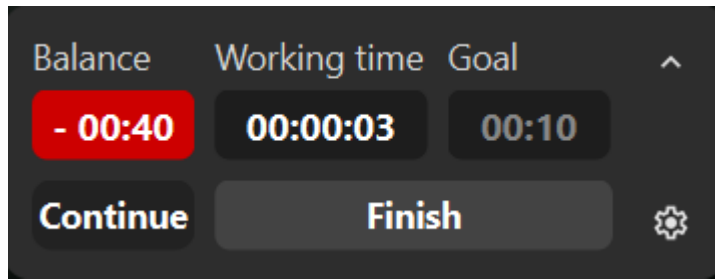


Рисунок 3.7 — Режим перерви під час робочої сесії

3.2.6.4 Компактний режим робочого вікна

Компактний режим призначений для зменшення розміру робочого вікна (рис. 3.8). У цьому стані приховуються кнопки керування сесією, але залишаються видимими основні часові показники.

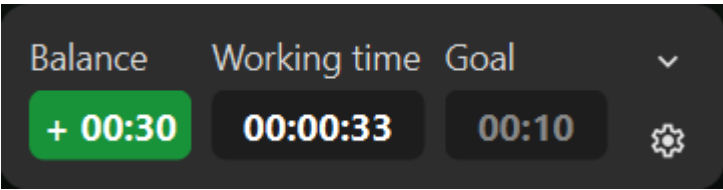


Рисунок 3.8 — Компактний режим робочого вікна

3.2.6.5 Стан налаштування Work-List і Black-List

Вікно налаштувань відкривається з головного або робочого вікна системи. У ньому відображаються два списки ресурсів: Work resources і Distractions (рис. 3.9). Перший список використовується для визначення робочих ресурсів, другий — для відволікаючих ресурсів.

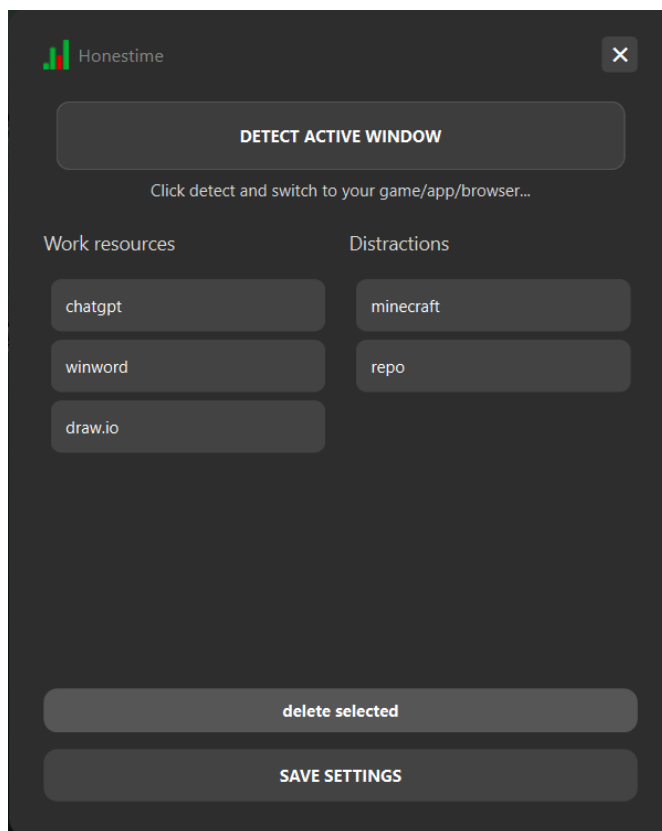


Рисунок 3.9 — Налаштування Work-List і Black-List

3.2.6.6 Стан швидкого додавання знайденого ресурсу

Під час визначення активного вікна система фіксує назву активного процесу або вебресурсу (рис. 3.10). Після цього в інтерфейсі відображається знайдений ресурс і кнопки для швидкого додавання його до Work-List або Black-List.

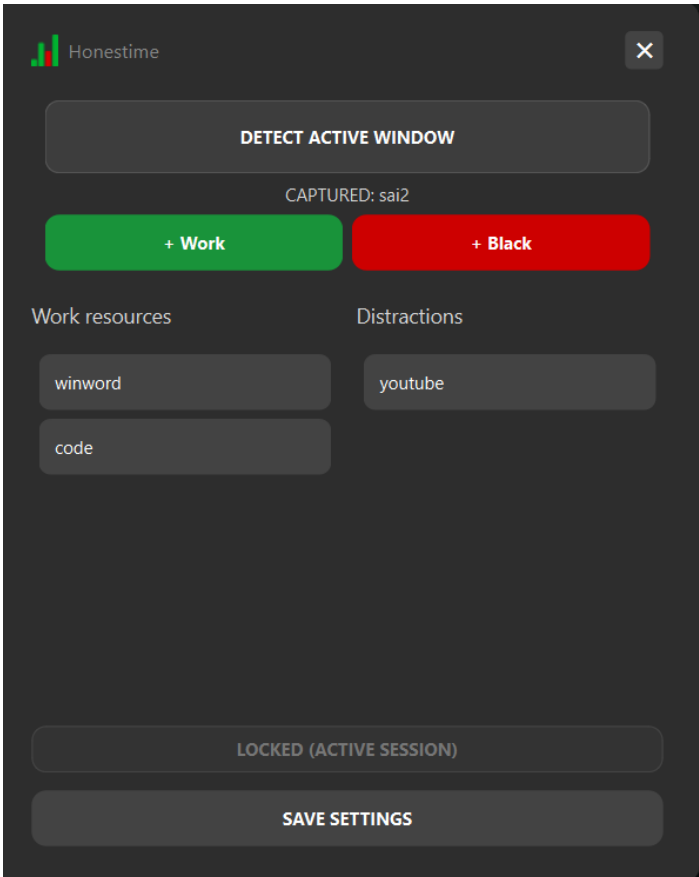


Рисунок 3.10 — Швидке додавання знайденого ресурсу

3.2.6.7 Заблокований стан редагування списків

За наявності від’ємного балансу редагування списків обмежується. У такому стані кнопка видалення елементів стає недоступною, а її напис змінюється на «LOCKED (NEGATIVE BALANCE)» (рис. 3.11). Це показує, що користувач не може змінити списки до компенсації боргу.

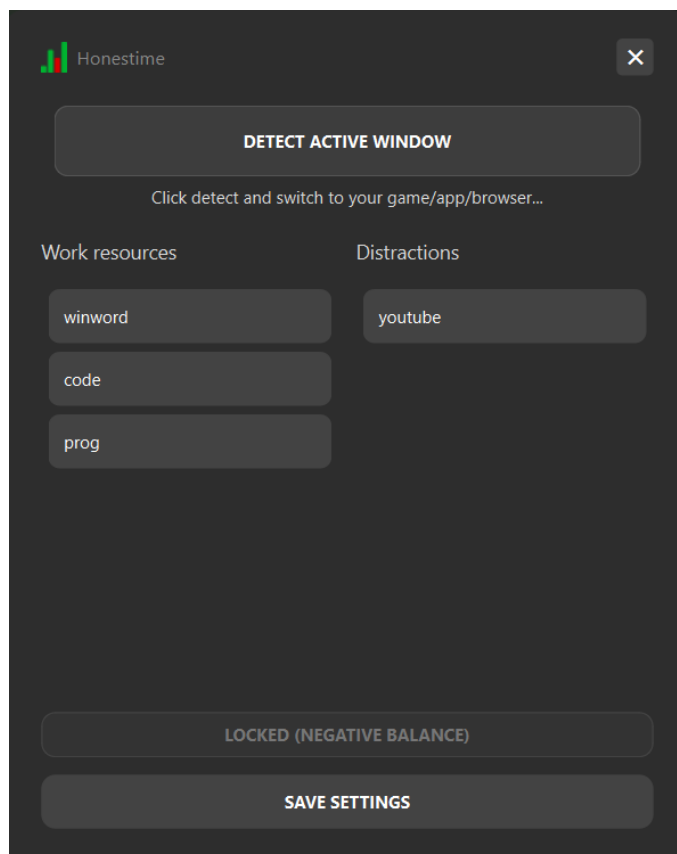


Рисунок 3.11 — Заблоковане редагування списків при від’ємному балансі

3.2.6.8 Стан перегляду статистики

Вікно статистики відображає результати попередніх робочих сесій. У ньому доступні режими перегляду за тиждень і за місяць, графік зарахованого часу, загальний робочий час, середній робочий час і показники балансу (рис. 3.12, рис. 3.13).



Рисунок 3.12 — Перегляд статистики за неділю

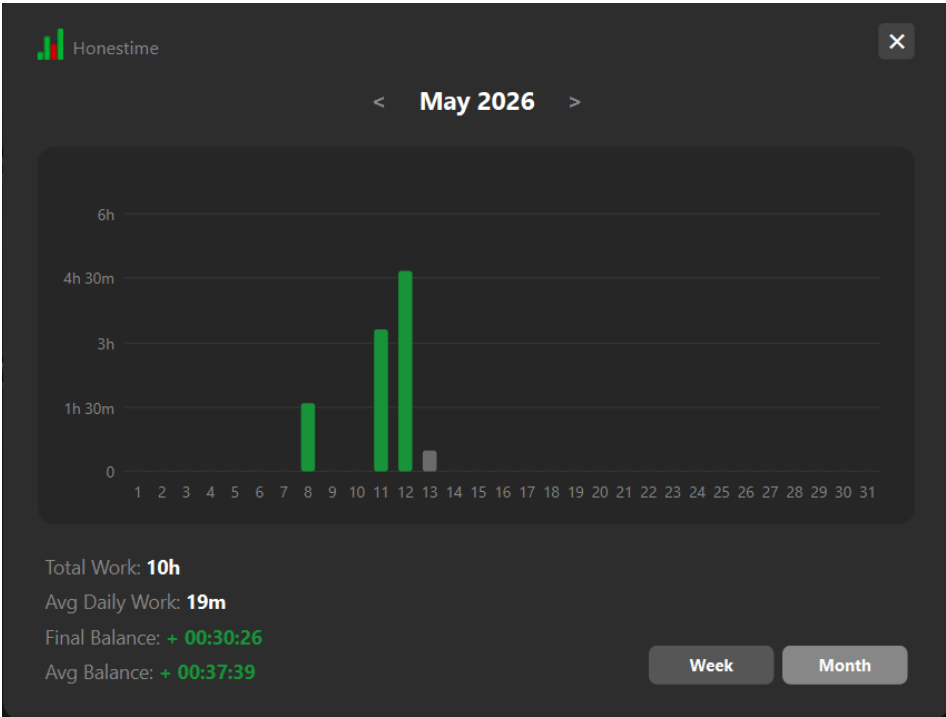


Рисунок 3.13 — Перегляд статистики за місяць

3.2.6.9 Службові повідомлення

Службові повідомлення використовуються для відображення помилок, попереджень і підтвердження важливих дій. Наприклад, система показує повідомлення при спробі запуску сесії без доданих робочих ресурсів або при спробі повного виходу з програми з від’ємним балансом (рис. 3.14).

Окремо реалізовано діалог підтвердження скидання сесії. Він належить до головного вікна та використовується перед виконанням Reset (рис. 3.15).

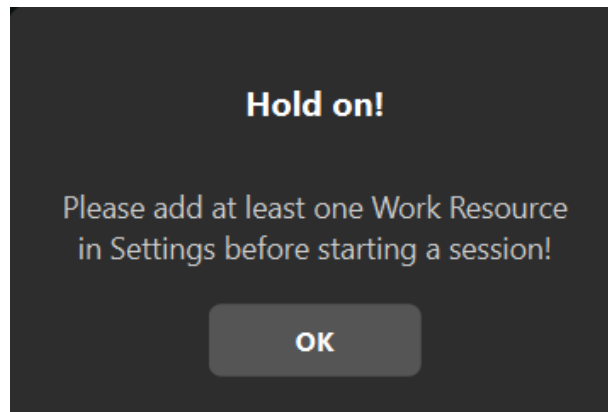


Рисунок 3.14 — Службове повідомлення системи

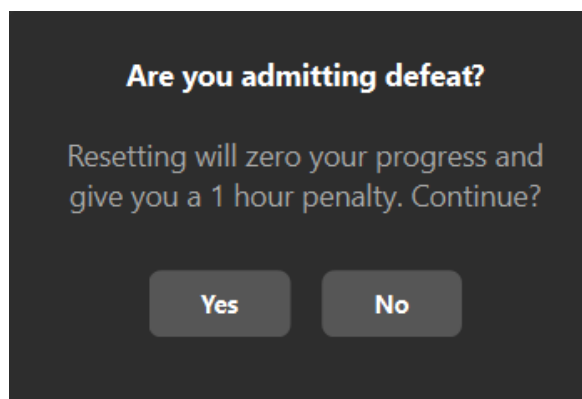


Рисунок 3.15 — Діалог підтвердження скидання сесії

3.2.7 Службові модулі системи

До службових модулів програмної системи належать `guard.py`, `utils.py` і `styles.py`.

Модуль `guard.py` реалізує службовий процес, який контролює наявність основного процесу `Honestime.exe`. Якщо систему було випадково завершено, службовий процес може повторно його запустити. Коректне завершення виконується через службовий файл «`exit.flag`»: після його появи `guard.py` припиняє роботу і не перезапускає основну програму.

Модуль `utils.py` містить допоміжні функції, які використовуються в різних частинах системи:

- визначення правильного шляху до файлів ресурсів;
- додавання службового процесу до автозапуску Windows;
- налаштування зовнішнього вигляду безрамкових вікон.

Модуль `styles.py` містить стилі графічного інтерфейсу. Винесення стилів в окремий файл дає змогу відокремити оформлення від логіки роботи вікон.

3.3 Висновки за розділом 3

У третьому розділі було описано реалізацію програмного забезпечення «Honestime». Окремо описано реалізацію обліку часу за `Work-List`, обмеження `Black-List` за від'ємного балансу, `AFK`-перевірку, періодичне збереження сесії та механізм `Reset`. Розглянуто алгоритм функціонування програми та основні модулі, що забезпечують запуск системи, керування робочою сесією, моніторинг активного вікна, роботу зі списками ресурсів, збереження даних і графічний інтерфейс.

Таким чином, у результаті розробки отримано працездатну десктопну систему, структура якого відповідає спроектованій архітектурі. Подальший розділ присвячено експлуатації та тестуванню програми.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення програми

Програмна система «Honestime» призначений для індивідуального використання під час роботи за персональним комп'ютером. Основне призначення програми полягає у наданні користувачу об'єктивної інформації про фактично зарахований робочий час, поточний часовий баланс та результати виконаних робочих сесій. Функціональні вимоги, на основі яких реалізовано ці можливості, наведено у підрозділі 1.3.

Цільовою аудиторією програми є користувачі, які самостійно організовують свій робочий час у цифровому середовищі: студенти, фрилансери, віддалені працівники, розробники програмного забезпечення, автори навчальних або творчих проєктів. Для таких користувачів важливим є не лише планування часу, а й отримання фактичних даних про виконання запланованої роботи.

Програмна система може застосовуватися під час виконання навчальних завдань, роботи над програмними проєктами, написання текстових матеріалів, підготовки звітів, самостійного навчання або іншої діяльності, що виконується за комп'ютером. Результатом використання програми є сформована історія робочих сесій, поточний часовий баланс і статистика, які можуть бути використані користувачем для оцінювання власної продуктивності.

Програма орієнтована на персональний самоконтроль і не призначений для прихованого спостереження, корпоративного моніторингу працівників або передавання даних стороннім особам. Дані, необхідні для роботи програми, зберігаються локально на комп'ютері користувача.

4.2 Умови виконання програми

Для коректного виконання програмної системи «Honestime» необхідне персональне комп'ютерне середовище, у якому підтримується робота графічного інтерфейсу, фонових процесів та локального збереження даних. Мінімальні вимоги до апаратного та програмного забезпечення наведено в табл. 4.1.

Таблиця 4.1 — Мінімальні вимоги до середовища виконання програми

Параметр	Мінімальна вимога
Операційна система	Windows 10 або Windows 11
Процесор	1,5 ггц або вище
Оперативна пам'ять	4 ГБ або більше
Вільне місце на диску	200 МБ або більше
Роздільна здатність екрана	1366 × 768 або вище
Пристрої введення	Клавіатура та миша або інший маніпулятор
Права доступу	Права звичайного користувача; права адміністратора для окремих службових дій
Підключення до Інтернету	Не є обов'язковим для базових функцій
Збереження даних	Локальне збереження на комп'ютері користувача

Система призначена для роботи у середовищі операційної системи Windows. Для базових функцій програми підключення до мережі Інтернет не є обов'язковим, оскільки дані робочих сесій, списків ресурсів, статистики та часового балансу зберігаються локально на комп'ютері користувача.

Для виконання окремих службових дій програмі можуть бути потрібні розширені права доступу. Це стосується ситуацій, у яких необхідно забезпечити фонову роботу програми, коректне визначення активного вікна або виконання обмежень для ресурсів із Black-List. Водночас звичайний запуск програми, перегляд статистики та налаштування списків можуть виконуватися у межах прав користувача.

Під час експлуатації програми користувач повинен враховувати, що коректність обліку часу залежить від попереднього налаштування робочого середовища. Умови запуску та зупинки таймера наведено у підрозділі 3.1.

4.3 Виконання програми

Приклад роботи користувача з програмою розглянуто на основі типового сценарію використання системи «Honestime» під час виконання завдання. У межах сценарію користувач планує працювати над звітом робочого проєкту протягом 60 хвилин. Початкове значення часового балансу становить 0 хвилин.

Після запуску програми користувач відкриває головне вікно програми, у якому відображаються поточний часовий баланс, кнопка запуску робочої сесії, Work-List, Black-List та блок статистики. Приклад головного вікна програми наведено на рис. 4.1.

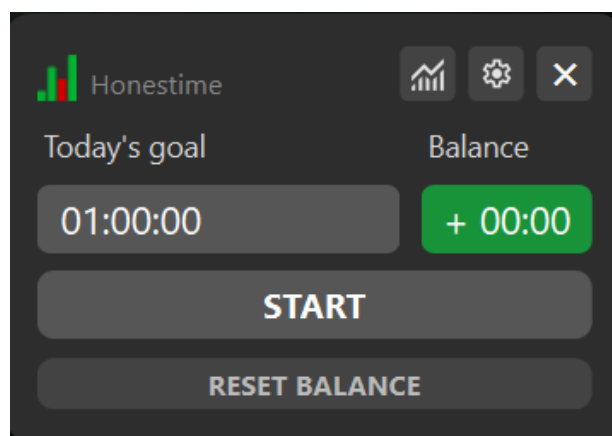


Рисунок 4.1 — Головне вікно програми «Honestime»

На першому етапі користувач налаштовує списки ресурсів. До Work-List додаються програми та вебресурси, які використовуються для виконання роботи: Microsoft Word та Visual Studio Code. До Black-List додаються ресурси, які можуть відволікати від роботи: YouTube, для прикладу. Приклад налаштування списків наведено на рис. 4.2.

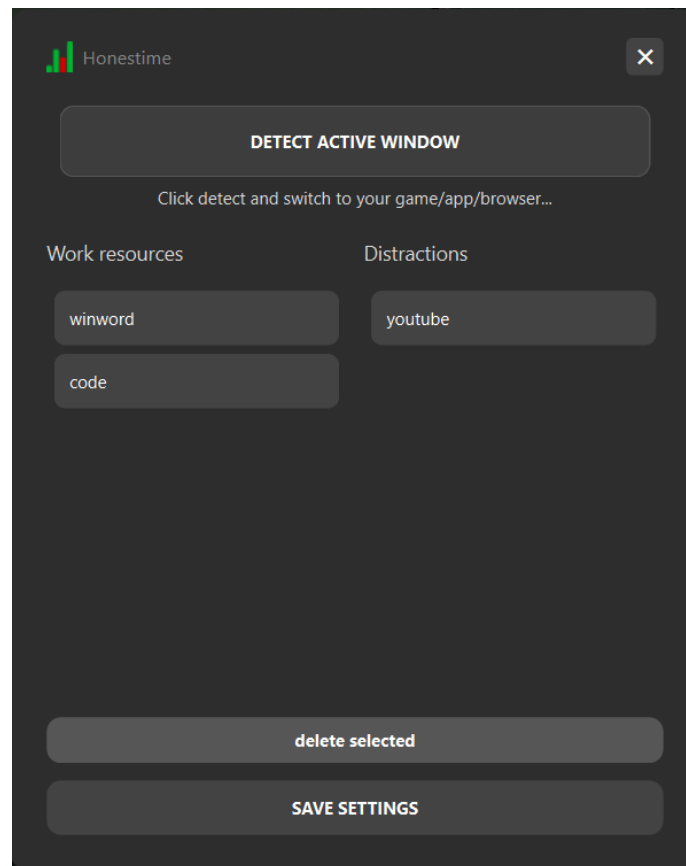


Рисунок 4.2 — Налаштування Work-List і Black-List

Після налаштування списків користувач встановлює денний норматив роботи — 60 хвилин — і запускає робочу сесію. З цього моменту система починає відстежувати активний ресурс і зараховує час лише тоді, коли користувач працює з елементами Work-List. Приклад запущеної робочої сесії наведено на рис. 4.3.

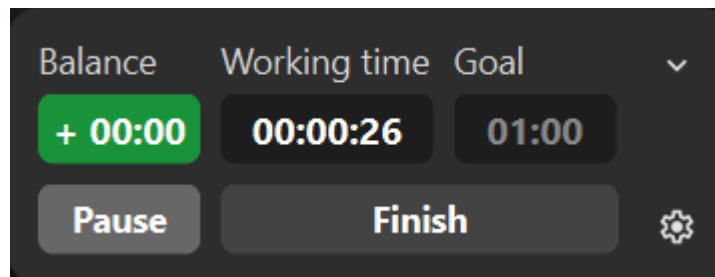


Рисунок 4.3 — Запуск робочої сесії

Під час сесії користувач відкриває Microsoft Word і редагує текст звіту. Оскільки цей застосунок входить до Work-List, час роботи зараховується як продуктивний. Якщо користувач переходить до Google Docs або Visual Studio Code, облік часу також продовжується, оскільки ці ресурси теж належать до робочого списку.

Для перевірки реакції системи користувач відкриває ресурс із Black-List, наприклад YouTube. У цьому випадку час перебування на сторонньому ресурсі не зараховується до робочого часу. Якщо у користувача є від’ємний часовий баланс, доступ до такого ресурсу може бути обмежений.

Після повернення до ресурсу з Work-List облік робочого часу відновлюється. Таким чином, система відокремлює фактичну продуктивну активність від загального часу перебування за комп’ютером.

Після певного часу роботи користувач вирішує достроково завершити робочу сесію. Програмна система підсумовує фактично зарахований робочий час і порівнює його з установленим денним нормативом. У розглянутому прикладі користувач запланував 60 хвилин роботи, але до моменту завершення сесії система зарахувала лише 30 хвилин продуктивної активності. У результаті формується від’ємний часовий баланс, тобто борг у розмірі 30 хвилин. Результат завершеної сесії наведено на рис. 4.4.

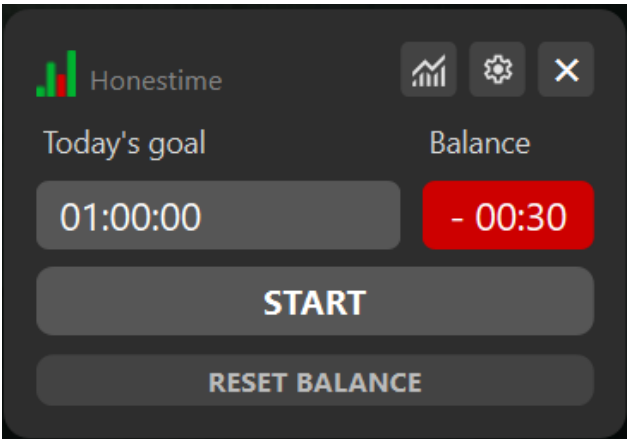


Рисунок 4.4 — Результат завершення робочої сесії

Сформований від’ємний баланс означає, що користувач має компенсувати 30 хвилин роботи. До цього моменту система може обмежувати доступ до ресурсів із Black-List і забороняти видалення елементів зі списків. Після додаткової роботи з ресурсами Work-List борг зменшується або переходить у позитивний баланс.

Користувач також може переглянути історію сесій і статистику роботи. Приклад перегляду статистики наведено на рис. 4.5.

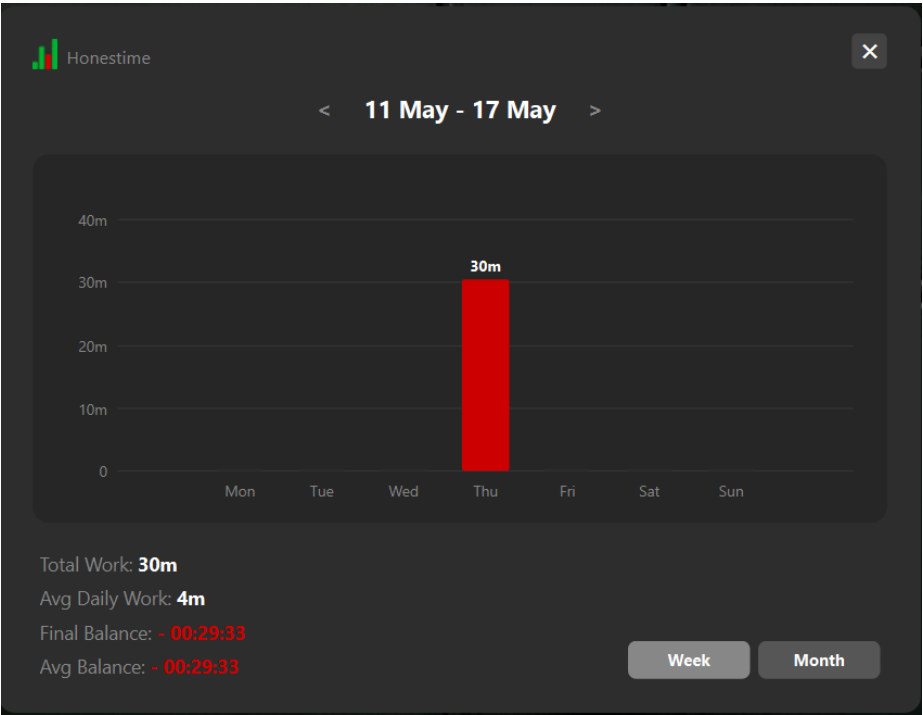


Рисунок 4.5 — Перегляд статистики робочих сесій

Отже, наведений сценарій демонструє основний принцип роботи програмної системи «Honestime»: система зараховує лише час взаємодії з ресурсами Work-List, не враховує сторонню активність, формує часовий баланс і застосовує обмеження для ресурсів із Black-List у разі невиконання встановленого плану.

4.4 Тестування програми

Тестування програмної системи «Honestime» проводилося з метою перевірки відповідності реалізованого програмного забезпечення функціональним вимогам, визначеним у попередніх розділах роботи. Основну увагу було приділено перевірці коректності обліку робочого часу, роботи списків Work-List і Black-List, формування часового балансу, збереження даних та поведінки програми у типових сценаріях використання.

Тестування виконувалося на персональному комп'ютері з операційною системою Windows. Як тестові робочі ресурси використовувалися текстовий редактор, середовище розробки та вебресурси навчального призначення. Як тестові відволікаючі ресурси використовувалися розважальні вебсайти та застосунки, додані до Black-List.

4.4.1 Методика тестування

Для перевірки працездатності програми було використано функціональне тестування. Для кожної основної функції системи було сформовано тестовий сценарій, що містить дію користувача, очікуваний результат і фактичний результат виконання.

Тестування проводилося вручну, оскільки система орієнтована на взаємодію з активним вікном операційної системи, користувацькими списками ресурсів і зміною стану робочої сесії. Такий підхід дозволив

перевірити поведінку програми в умовах, наближених до реального використання.

Основними критеріями успішного проходження тестування були:

- коректне додавання ресурсів до Work-List і Black-List;
- зарахування часу лише під час роботи з ресурсами Work-List;
- незарахування часу під час переходу до сторонніх ресурсів;
- правильне формування позитивного або від’ємного часового балансу;
- обмеження ресурсів із Black-List за наявності боргу;
- збереження статистики після завершення сесії;
- стабільна робота програми у фоновому режимі.

4.4.2 Функціональне тестування

Функціональне тестування було спрямоване на перевірку основних можливостей системи з погляду користувача. Результати перевірки наведено в табл. 4.2.

Таблиця 4.2 — Результати функціонального тестування системи «Honestime»

№	Функція для перевірки	Дії користувача	Очікуваний результат	Фактичний результат	Статус
1	2	3	4	5	6
ТС-01	Додавання ресурсу до Work-List	Додати робочий застосунок до Work-List	Ресурс з'являється у списку робочих ресурсів	Ресурс успішно додано	Пройдено
ТС-02	Додавання ресурсу до Black-List	Додати відволікаючий ресурс до Black-List	Ресурс з'являється у списку обмежень	Ресурс успішно додано	Пройдено

Продовження таблиці 4.2

1	2	3	4	5	6
ТС-03	Запуск робочої сесії	Натиснути кнопку запуску сесії	Сесія переходить в активний стан	Сесію запущено	Пройдено
ТС-04	Зарахування робочого часу	Відкрити ресурс із Work- List	Таймер зараховує робочий час	Час зараховується	Пройдено
ТС-05	Перехід до стороннього ресурсу	Відкрити ресурс, якого немає у Work- List	Робочий час не зараховується	Облік часу призупинено	Пройдено
ТС-06	Перехід до ресурсу з Black-List	Відкрити ресурс із Black- List	Ресурс не зараховується як робочий, за потреби обмежується	Ресурс оброблено коректно	Пройдено
ТС-07	Завершення сесії	Натиснути кнопку завершення сесії	Сесія завершується, результат зберігається	Сесію завершено	Пройдено
ТС-08	Перерахунок часового балансу	Завершити сесію з часом меншим за норматив	Різниця між фактичним часом і нормативом віднімається від поточного балансу	Баланс перераховано відповідно до результату сесії	Пройдено
ТС-09	Перегляд статистики	Відкрити розділ статистики	Відображається інформація про сесії	Статистика відображається	Пройдено
ТС-10	Фонова робота	Згорнути програму під час сесії	Моніторинг активного ресурсу продовжується	Система працює у фоні	Пройдено

Кінець таблиці 4.2

1	2	3	4	5	6
ТС-11	Заборона зміни списків при боргу	Спробувати видалити ресурс при від'ємному балансі	Видалення недоступне	Видалення заблоковано	Пройдено
ТС-12	Екстрене скидання	Виконати Reset	Поточне обмеження скидається, баланс набуває штрафного від'ємного значення	Баланс очищено та нараховано штраф –1 година	Пройдено

За результатами функціонального тестування встановлено, що основні функції системи працюють відповідно до очікуваної логіки. Система коректно обробляє дії користувача, розрізняє робочі та сторонні ресурси, перераховує часовий баланс, зберігає результати сесій і застосовує обмеження відповідно до поточного стану користувача.

4.4.3 Тестування розрахунку часового балансу

Окремо було перевірено правильність перерахунку часового балансу (табл. 4.3). У системі баланс не формується заново після кожної сесії, а змінюється з урахуванням попереднього значення. Після завершення сесії система визначає різницю між фактично зарахованим робочим часом і встановленим нормативом, після чого додає цю різницю до поточного балансу.

Таблиця 4.3 — Перевірка перерахунку часового балансу

№	Початковий баланс, хв	Денний нормати в, хв	Зарахований час, хв	Різниця, хв	Очікуваний баланс, хв	Фактичний баланс, хв	Статус
1	2	3	4	5	6	7	8
1	0	60	60	0	0	0	Пройдено
2	0	60	45	-15	-15	-15	Пройдено
3	0	60	75	+15	+15	+15	Пройдено
4	+30	60	45	-15	+15	+15	Пройдено
5	+10	120	95	-25	-15	-15	Пройдено
6	-20	60	90	+30	+10	+10	Пройдено

Результати тестування показали, що система коректно перераховує часовий баланс з урахуванням його попереднього значення. Якщо фактично зарахований час менший за встановлений норматив, баланс зменшується. Якщо фактичний час перевищує норматив, баланс збільшується. Від’ємний баланс формується лише тоді, коли після перерахунку підсумкове значення стає меншим за нуль.

4.4.4 Тестування обмежень Black-List

Для перевірки роботи механізму Black-List було створено тестові ситуації з різними значеннями часового балансу. Метою перевірки було визначити, чи залежить доступ до відволікаючих ресурсів і можливість редагування списків від поточного стану балансу. Результати тестування наведено в табл. 4.4.

Таблиця 4.4 — Перевірка роботи обмежень Black-List

№	Умова	Дія користувача	Очікуваний результат	Фактичний результат	Статус
1	2	3	4	5	6
1	Баланс дорівнює 0	Відкрити ресурс із Black-List поза активною сесією	Доступ обробляється згідно з поточними налаштуваннями програми	Поведінка відповідає налаштуванням	Пройдено
2	Баланс менше 0	Відкрити ресурс із Black-List	Доступ до ресурсу обмежується	Ресурс заблоковано	Пройдено
3	Баланс менше 0	Видалити ресурс із Black-List	Видалення недоступне до компенсації боргу	Видалення заблоковано	Пройдено
4	Баланс менше 0	Видалити ресурс із Work-List	Видалення недоступне до компенсації боргу	Видалення заблоковано	Пройдено
5	Баланс більше або дорівнює 0	Видалити ресурс зі списку	Видалення доступне	Ресурс видалено	Пройдено

Проведене тестування підтвердило, що механізм Black-List працює у зв'язку з часовим балансом. Якщо користувач має від'ємний баланс, програма обмежує доступ до ресурсів із Black-List і не дозволяє змінювати списки таким чином, щоб обійти встановлені правила. Після компенсації боргу можливість редагування списків відновлюється.

4.4.5 Тестування збереження даних і фоновій роботі

Оскільки програма має працювати не лише у відкритому вікні, а й у фоновому режимі, окремо було перевірено збереження даних після завершення сесії та поведінку програми після згортання. Результати перевірки наведено в табл. 4.5.

Таблиця 4.5 — Перевірка збереження даних і фоновій роботі

№	Перевірювана умова	Дія користувача	Очікуваний результат	Фактичний результат	Статус
1	2	3	4	5	6
1	Збереження Work-List	Додати ресурс до Work-List і перезапустити програму	Доданий ресурс залишається у списку	Ресурс збережено	Пройдено
2	Збереження Black-List	Додати ресурс до Black-List і перезапустити програму	Доданий ресурс залишається у списку	Ресурс збережено	Пройдено
3	Збереження результату сесії	Завершити робочу сесію та відкрити статистику	Завершена сесія відображається в історії	Запис збережено	Пройдено
4	Збереження балансу	Перезапустити програму після зміни балансу	Поточний баланс зберігається	Баланс відновлено після запуску	Пройдено
5	Фонova робота	Згорнути програму під час активної сесії	Облік активного ресурсу продовжується	Моніторинг продовжується	Пройдено

Кінець таблиці 4.5

1	2	3	4	5	6
6	Робота з трею	Закрити вікно програми кнопкою X	Система не завершується повністю, а переходить у фоновий режим	Система продовжує роботу у треї	Пройдено

Результати тестування показали, що програма коректно зберігає основні дані користувача: списки ресурсів, історію сесій і поточний часовий баланс. Також підтверджено, що після згортання програма продовжує виконувати моніторинг активного ресурсу, що є необхідною умовою для коректного обліку робочого часу.

4.4.6 Аналіз результатів тестування

Система коректно виконує облік продуктивного часу, враховує лише ресурси з Work-List, не зараховує сторонню активність, перераховує часовий баланс з урахуванням його попереднього значення та застосовує обмеження до ресурсів із Black-List у разі наявності від'ємного балансу.

Критичних помилок під час тестування не виявлено. Виявлені особливості роботи пов'язані з необхідністю коректного початкового налаштування списків користувачем. Якщо користувач не додасть потрібний робочий ресурс до Work-List, час роботи з ним не буде зараховано. Тому перед початком використання програми необхідно уважно сформувати перелік робочих і відволікаючих ресурсів.

Також було підтверджено коректну роботу функції Reset. Вона не виконує повне анулювання наслідків роботи програми, а встановлює штрафний баланс –1 година. Це дозволяє використовувати Reset як аварійний механізм, але не перетворює його на спосіб простого обходу правил.

Таким чином, результати тестування підтверджують, що система відповідає поставленим функціональним вимогам і може використовуватися для обліку фактичного робочого часу користувача в цифровому середовищі.

4.5 Висновки за розділом 4

У четвертому розділі розглянуто умови виконання програми «Honestime», наведено приклад її практичного використання та проведено тестування основних функцій.

Під час експлуатаційної перевірки показано типовий сценарій роботи: налаштування Work-List і Black-List, запуск сесії, автоматичний облік продуктивної активності, завершення роботи та перегляд результатів. Це дозволило продемонструвати відмінність між загальним перебуванням за комп'ютером і фактично зарахованим часом.

За результатами тестування встановлено, що система коректно обробляє списки ресурсів, визначає активне вікно, перераховує часовий баланс, зберігає статистику та продовжує моніторинг у фоновому режимі. Також підтверджено роботу обмежень для ресурсів із Black-List за наявності боргу.

Отже, проведена перевірка підтвердила працездатність реалізованого програмного забезпечення та його відповідність поставленим функціональним вимогам.

ВИСНОВКИ

У результаті виконання дипломної роботи розроблено програмну систему «Honestime» для обліку та динамічного оцінювання фактичного робочого часу працівників у цифровому робочому середовищі.

У роботі проаналізовано проблему цифрової прокрастинації та розриву між загальним часом перебування працівника за комп'ютером і фактично зарахованим робочим часом. Розглянуто існуючі програмні засоби для самоконтролю продуктивності, обмеження цифрових дистракторів та організації робочого часу. На основі проведеного аналізу сформовано вимоги до програмної системи «Honestime».

Сформовано функціональні та нефункціональні вимоги до програмної системи. Спроектовано архітектуру програмної системи, структуру бази даних та алгоритм її функціонування.

Реалізовано основні модулі програмного забезпечення: моніторинг активного ресурсу, керування Work-List і Black-List, облік фактично зарахованого робочого часу, формування часового балансу, збереження історії сесій, перегляд статистики, режим Break, AFK-перевірку та функцію Reset.

Проведено тестування працездатності програмної системи. Перевірка підтвердила коректність визначення активного ресурсу, обліку робочого часу, оновлення часового балансу, збереження статистики та роботи обмежень для ресурсів із Black-List.

Мету роботи досягнуто: підвищено об'єктивність визначення фактичного робочого часу працівників за рахунок автоматизованої обробки даних про активний ресурс, перевірки його належності до Work-List, обліку лише фактично зарахованого робочого часу та формування часового балансу. Розроблена програмна система дає змогу відокремлювати загальний час перебування працівника за комп'ютером від фактичної роботи у визначених робочих ресурсах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Jarrahi M. H., Blyth D. L., Goray C. Mindful work and mindful technology: Redressing digital distraction in knowledge work. *Digital Business*. 2023. Vol. 3, Iss. 1. Article 100051. DOI: 10.1016/j.digbus.2022.100051. URL: <https://www.sciencedirect.com/science/article/pii/S266695442200031X> (accessed: 28.03.2026).
2. Rick V. B., Brandl C., Mertens A., Nitsch V. Work interruptions of office workers: The influence of the complexity of primary work tasks on the perception of interruptions. *Work*. 2024. Vol. 77, Iss. 1. P. 185—196. URL: <https://publications.rwth-aachen.de/record/962254> (accessed: 28.03.2026).
3. Ohly S., Bastin L. Effects of task interruptions caused by notifications from communication applications on strain and performance. *Journal of Occupational Health*. 2023. Vol. 65, Iss. 1. Article e12408. DOI: 10.1002/1348-9585.12408. URL: https://www.jstage.jst.go.jp/browse/joh/65/1/_contents/-char/en (accessed: 28.03.2026).
4. Rahmillah F. I., Tariq A., King M., Oviedo-Trespalacios O. Evaluating the effectiveness of apps designed to reduce mobile phone use and prevent maladaptive mobile phone use: Multimethod study. *Journal of Medical Internet Research*. 2023. Vol. 25. Article e42541. DOI: 10.2196/42541. URL: <https://www.jmir.org/2023/1/e42541/> (accessed: 01.04.2026).
5. Murad M., Collins K. StudyTracker self-tracking app and its relationship to university student procrastination. *Canadian Journal of Learning and Technology*. 2025. Vol. 50, Iss. 2. P. 1—23. DOI: 10.21432/cjlt28644. URL: <https://cjlt.ca/index.php/cjlt/article/view/28644> (accessed: 01.04.2026).
6. How It Works. *Focusmate*. URL: <https://www.focusmate.com/how-it-works/> (accessed: 03.04.2026).
7. Focusmate. Virtual body doubling for getting anything done. URL: <https://www.focusmate.com/> (accessed: 03.04.2026).
8. Cold Turkey Blocker. The toughest website & app blocker on the

Internet. URL: <https://getcoldturkey.com/> (accessed: 03.04.2026).

9. Habitica. Gamify Your Life. URL: <https://habitica.com/> (accessed: 03.04.2026).

10. Python 3.14.4. Python.org. URL: <https://www.python.org/downloads/release/python-3144/> (accessed: 05.05.2026).

11. What is Windows Presentation Foundation. *Microsoft Learn*. URL: <https://learn.microsoft.com/en-us/dotnet/desktop/wpf/overview/> (accessed: 05.05.2026).

12. PyQt6. *Python Package Index*. URL: <https://pypi.org/project/PyQt6/> (accessed: 05.05.2026).

13. About SQLite. *SQLite*. URL: <https://sqlite.org/about.html> (accessed: 05.05.2026).

ДОДАТОК А
Текст програми

A.1 Файл main.py

```

import sys
import signal
import subprocess
import psutil
import ctypes
import os
import traceback
import time

from PyQt6.QtWidgets import QApplication
from PyQt6.QtCore import QTimer
from PyQt6.QtGui import QIcon

from utils import (
    resource_path,
    add_to_startup,
    get_app_dir,
    get_guard_path,
    get_exit_flag_path,
)

from database.database import DatabaseManager
from resources import ResourceManager
from ui.main_window import MainWindow
from engine import FocusController, UserSession
from monitor import WindowMonitor
from ui.styles import GLOBAL_STYLE

signal.signal(signal.SIGINT, signal.SIG_DFL)

GUARD_NAME = "guard.exe"

_app_mutex = None

def setup_windows_app_id():
    if sys.platform != "win32":
        return

```

```

try:
    myappid = "honestime.app.v1"
    ctypes.windll.shell32.SetCurrentProcessExplicitAppUserModelID(myappid)
except Exception:
    pass

def prevent_multiple_instances():
    """
    Забороняє запуск другої копії програми.
    """
    global _app_mutex

    if sys.platform != "win32":
        return

    _app_mutex = ctypes.windll.kernel32.CreateMutexW(
        None,
        False,
        "Honestime_Single_Instance_Mutex"
    )

    if ctypes.windll.kernel32.GetLastError() == 183:
        sys.exit(0)

def is_fresh_exit_flag(max_age_seconds=60):
    flag_path = get_exit_flag_path()

    if not os.path.exists(flag_path):
        return False

    try:
        age = time.time() - os.path.getmtime(flag_path)
        return age <= max_age_seconds
    except Exception:
        return True

def remove_stale_exit_flag():
    """
    Видаляє лише старий exit.flag.

    Новий exit.flag залишаємо без змін, щоб guard встиг його побачити
    після натискання «Log out completely».

```

```

"""
flag_path = get_exit_flag_path()

if not os.path.exists(flag_path):
    return

if is_fresh_exit_flag():
    return

try:
    os.remove(flag_path)
    print(">>> Старий exit.flag видален")
except Exception as e:
    print(f'!!! Не вдалося видалити старий файл exit.flag: {e}')

def is_guard_running():
    guard_path = os.path.normcase(os.path.abspath(get_guard_path()))

    for proc in psutil.process_iter(["name", "exe"]):
        try:
            name = proc.info.get("name")
            exe = proc.info.get("exe")

            if not name or name.lower() != GUARD_NAME.lower():
                continue

            if exe:
                proc_path = os.path.normcase(os.path.abspath(exe))
                if proc_path == guard_path:
                    return True

        return True

    except (psutil.NoSuchProcess, psutil.AccessDenied, psutil.ZombieProcess):
        continue
    except Exception:
        continue

    return False

def launch_guard_process():
    try:
        guard_path = get_guard_path()

```

```

    if not os.path.exists(guard_path):
        print(f'!!! guard.exe не знайдено: {guard_path}')
        return False

    creationflags = subprocess.CREATE_NO_WINDOW if sys.platform ==
"win32" else 0

    subprocess.Popen(
        [guard_path],
        cwd=get_app_dir(),
        creationflags=creationflags
    )

    print(">>> guard.exe запущено")
    return True

except Exception as e:
    print(f'!!! Ошибка запуска guard.exe: {e}')
    return False

def start_watchdog():
    """
    Запускає guard.exe під час запуску програми.
    """
    remove_stale_exit_flag()

    if not getattr(sys, "frozen", False):
        return

    if is_fresh_exit_flag():
        return

    if not is_guard_running():
        launch_guard_process()

def monitor_guard():
    """
    Стежить, щоб guard.exe працював.
    """
    if not getattr(sys, "frozen", False):
        return

```

```

if is_fresh_exit_flag():
    return

if not is_guard_running():
    print("print >>> guard.exe не найдено. Перезапускаемо...")
    launch_guard_process()

def main():
    try:
        setup_windows_app_id()
        prevent_multiple_instances()

        start_hidden = "--background" in sys.argv or "--hidden" in sys.argv

        add_to_startup()
        start_watchdog()

        app = QApplication(sys.argv)
        app.setQuitOnLastWindowClosed(False)

        app.setStyleSheet(GLOBAL_STYLE)
        app.setWindowIcon(QIcon(resource_path("assets/logo.png")))

        db_manager = DatabaseManager()
        resource_manager = ResourceManager(db_manager)
        last_data = db_manager.get_last_session_data()

        session = UserSession(
            daily_goal=last_data.get("daily_goal", 0),
            daily_time=last_data.get("daily_time", 0),
            balance=last_data.get("balance", 0)
        )

        controller = FocusController(session, db_manager, resource_manager)

        monitor = WindowMonitor(check_interval=800)
        monitor.window_detected.connect(controller.on_window_detected)

        window = MainWindow(controller)
        controller.main_window = window

        if start_hidden:
            window.hide()
        else:

```

```

        window.show()

    app.monitor = monitor

    guard_timer = QTimer()
    guard_timer.timeout.connect(monitor_guard)
    guard_timer.start(5000)
    app.guard_timer = guard_timer

    sys.exit(app.exec())

except Exception:
    print(f'!!! ПОМИЛКА:\n{traceback.format_exc()}')

if __name__ == "__main__":
    main()

```

A.2 Файл engine.py

```

from enum import Enum
from PyQt6.QtCore import QObject, QTimer, pyqtSignal
import pygetwindow as gw
from typing import TYPE_CHECKING
import psutil
import pygetwindow as gw
import win32process
import ctypes
import psutil, win32gui, win32con

PENALTY = -3600
# Якщо True — формат ММ:СС (для тестів). Якщо False — ГГ:ММ (для
релізу).
IS_TEST_MODE = False
NORM_AFK = 180

if TYPE_CHECKING:
    from src.database.database import DatabaseManager
    from src.resources import ResourceManager

class SessionStatus(Enum):
    IDLE = "Очікування"
    WORKING = "Працює"

```

```
BREAK = "Перерва"
OVERTIME = "Овертайм"
```

```
class LASTINPUTINFO(ctypes.Structure):
```

```
    _fields_ = [
        ('cbSize', ctypes.c_uint),
        ('dwTime', ctypes.c_uint),
    ]
```

```
class UserSession:
```

```
    def __init__(self, daily_goal=0, daily_time=0, balance=0):
        self.daily_goal = daily_goal
        self.daily_time = daily_time
        self.balance = balance
        self.current_status = SessionStatus.IDLE
        self.is_finished = False
```

```
    def to_dict(self):
```

```
        return {
            'daily_goal': self.daily_goal,
            'daily_time': self.daily_time,
            'balance': self.balance
        }
```

```
class FocusController(QObject):
```

```
    tick_signal = pyqtSignal(str, str, bool)
```

```
    def __init__(self, user_session, db_manager, resource_manager):
```

```
        super().__init__()
        self.session = user_session
        self.db = db_manager
        self.resources = resource_manager
```

```
        self.is_working_now = False
        self.is_paused = False
        self.current_session_id = None
        self._last_save_time = 0
```

```
        self.timer = QTimer()
        self.timer.timeout.connect(self._on_tick)
```

```
        self.work_cache = []
        self.black_cache = []
        self.update_cache() # Завантажуємо дані з БД один раз при старті
```

```

self.main_window = None

def _get_idle_time(self) -> float:
    """Повертає час бездіяльності клавіатури/миші у секундах"""
    lii = LASTINPUTINFO()
    lii.cbSize = ctypes.sizeof(LASTINPUTINFO)
    if ctypes.windll.user32.GetLastInputInfo(ctypes.byref(lii)):
        millis = ctypes.windll.kernel32.GetTickCount() - lii.dwTime
        return millis / 1000.0
    return 0.0

def update_cache(self):
    """Завантажує списки з БД в оперативну пам'ять"""
    lists = self.db.get_lists()
    self.work_cache = [str(r).lower() for r in lists.get('work', [])]
    self.black_cache = [str(r).lower() for r in lists.get('black', [])]
    print(">>> Кеш ресурсів завантажено!")

def _on_tick(self):
    """
    Викликається тільки тоді, коли активне вікно входить до Work-List.
    Тому кожен tick = 1 секунда підтвердженого робочого часу.
    """

    # Робочий час завжди зростає, бо таймер працює тільки на Work-List.
    self.session.daily_time += 1

    # Якщо є борг — робочий час погашає борг.
    if self.session.balance < 0:
        self.session.balance += 1

    # Якщо боргу немає і користувач перевиконав норму — росте плюс-
    баланс.
    elif self.session.daily_goal > 0 and self.session.daily_time >
self.session.daily_goal:
        self.session.balance += 1

    if not hasattr(self, '_tick_counter'):
        self._tick_counter = 0

    self._tick_counter += 1

    if self._tick_counter >= 30:
        self._quick_save()
        self._tick_counter = 0

```

```

work_str = self._format_time(self.session.daily_time, for_work_time=True)
bal_str, is_pos = self._format_balance(self.session.balance)
self.tick_signal.emit(work_str, bal_str, is_pos)

if self.session.daily_goal > 0 and self.session.daily_time >=
self.session.daily_goal:
    try:
        if hasattr(self, "work_window"):
            self.work_window.apply_white_goal_style()
    except Exception as e:
        print(f">>> [DEBUG] ПОМИЛКА: {e}")

def _stop_working_process(self):
    """Викликається при паузі або зміні вікна"""
    if self.is_working_now:
        self.is_working_now = False
        self.timer.stop()
        self.session.current_status = SessionStatus.BREAK
        self._quick_save()

def finish_day(self):
    if self.session.is_finished: return

    self._stop_working_process()

    if self.session.daily_time < self.session.daily_goal:
        deficit = self.session.daily_goal - self.session.daily_time
        self.session.balance -= deficit
        print(f">>> Штраф за недопрацювання: віднято {deficit} секунд.")

    self.session.is_finished = True
    self._quick_save()

    work_str = self._format_time(self.session.daily_time, for_work_time=True)
    bal_str, is_pos = self._format_balance(self.session.balance)
    self.tick_signal.emit(work_str, bal_str, is_pos)

    self.is_paused = False

    print(f">>> КОНТРАКТ ЗАВЕРШЕНО. Фінальний баланс:
{self.session.balance}c")

def _quick_save(self):
    if not self.current_session_id:

```

```

    print("!!! Запис неможливий: ID сесії відсутній")
    return

    data = self.session.to_dict()
    print(f"DEBUG: Пишемо в ID {self.current_session_id} дані: {data}")

    try:
        self.db.update_session_row(self.current_session_id, data)
    except Exception as e:
        print(f"DB Error: {e}")

def _format_balance(self, seconds):
    is_pos = seconds >= 0
    sign = "+" if is_pos else "-"

    abs_seconds = abs(int(seconds))

    if is_pos:
        minutes = abs_seconds // 60
    else:
        minutes = (abs_seconds + 59) // 60

    hours = minutes // 60
    mins = minutes % 60

    time_str = f"{hours:02d}:{mins:02d}"

    return f"{sign} {time_str}", is_pos

def _security_check(self):
    import pygetwindow as gw
    window = gw.getActiveWindow()
    if window:
        self._universal_logic(window.title)

def on_window_detected(self, window_info):
    try:
        title_low = (window_info.title or "").lower()
        process_name = (window_info.process_name or "").lower()
        hwnd = window_info.hwnd

        is_work = any(r in title_low or r in process_name for r in self.work_cache)
        is_black = any(r in title_low or r in process_name for r in self.black_cache)

        if self.session.balance < 0 and is_black:

```

```

    if self.is_working_now:
        self._stop_working_process()

    safe_processes = ["explorer.exe", "taskmgr.exe", "honestime", "python"]

    is_safe = any(p in process_name or p in title_low for p in
safe_processes)

    if not is_safe:
        if not win32gui.IsIconic(hwnd):
            print(f">>> БЛОК ЧОРНОГО СПИСКУ: {window_info.title}")
            win32gui.ShowWindow(hwnd, win32con.SW_FORCEMINIMIZE)

    return

    if self.is_paused or self.current_session_id is None or
self.session.is_finished:
        return

    idle_seconds = self._get_idle_time()
    if idle_seconds > NORM_AFK:
        if self.is_working_now:
            print(f">>> AFK DETECTED ({int(idle_seconds)}s). Зупиняю
таймер.")
            self._stop_working_process()
        return

    if is_work:
        if not self.is_working_now:
            self.is_working_now = True
            self.timer.start(1000)
            self.session.current_status = SessionStatus.WORKING
        else:
            if self.is_working_now:
                self._stop_working_process()

    except Exception as e:
        print(f"!!! ПОМИЛКА В ДЕТЕКТОРІ ВІКОН: {e}")

def _format_time(self, seconds, for_work_time=False):
    """Універсальний форматувальник"""
    if for_work_time:
        abs_seconds = abs(int(seconds))
        h, r = divmod(abs_seconds, 3600)
        m, s = divmod(r, 60)

```

```

        return f"{h:02d}:{m:02d}:{s:02d}"

    if seconds < 0:
        abs_seconds = abs(int(seconds))
        minutes = (abs_seconds + 59) // 60

        hours = minutes // 60
        mins = minutes % 60
        return f"{hours:02d}:{mins:02d}"
    else:
        minutes = int(seconds) // 60
        hours = minutes // 60
        mins = minutes % 60
        return f"{hours:02d}:{mins:02d}"

    def get_current_balance_info(self):
        bal_str, is_pos = self._format_balance(self.session.balance)
        return bal_str, is_pos

    def start_new_contract(self, goal_input_seconds: int):
        self.session.daily_goal = goal_input_seconds
        self.session.daily_time = 0
        self.session.is_finished = False

        self.is_paused = False
        self.is_working_now = False

        self.current_session_id =
self.db.create_new_session_row(self.session.to_dict())

        if self.current_session_id:
            print(f">>> БАЗА: Створено запис №{self.current_session_id}")
        else:
            print("!!! ПОМИЛКА: БД не повернула ID нової сесії")

    def toggle_pause(self):
        self.is_paused = not self.is_paused
        if self.is_paused:
            self._stop_working_process()
            print(">>> ПАУЗА УВІМКЕНА")
        else:
            print(">>> ПАУЗА ВИМКЕНА. Чекаю на робоче вікно...")

    def _block_window(self, window_title):
        try:

```

```

    windows = gw.getWindowsWithTitle(window_title)
    for win in windows:
        if not win.isMinimized:
            win.minimize()
except:
    pass

def reset_session(self):
    self.is_working_now = False
    self.timer.stop()

    self.session.daily_goal = 0
    self.session.daily_time = 0
    self.session.balance = PENALTY # штраф
    self.session.is_finished = True

    self.current_session_id =
self.db.create_new_session_row(self.session.to_dict())
    bal_str, is_pos = self._format_balance(PENALTY)

    self.tick_signal.emit("00:00:00", bal_str, is_pos)

    print(f">>> RESET: Penalty applied. New balance: {PENALTY}s")

def get_resources(self):
    return self.db.get_lists()

def save_resources(self, work_list, black_list):
    self.db.save_lists(work_list, black_list)
    self.update_cache() # Оновлюємо кеш після збереження в БД
    print(">>> Ресурси оновлено та збережено в БД")

def is_session_active(self):
    """
    Повертає True, якщо робочий контракт створений і ще не завершений.
    Таймер може бути тимчасово зупинений, але сесія все одно активна.
    """
    return self.current_session_id is not None and not self.session.is_finished

```

A.3 Файл monitor.py

```

import pygetwindow as gw
import psutil

```

```
import win32process
from dataclasses import dataclass
from PyQt6.QtCore import QObject, pyqtSignal, QTimer
```

```
@dataclass
```

```
class ActiveWindowInfo:
```

```
    hwnd: int
```

```
    title: str
```

```
    process_name: str
```

```
    pid: int | None = None
```

```
class WindowMonitor(QObject):
```

```
    window_detected = pyqtSignal(object)
```

```
    def __init__(self, check_interval=800):
```

```
        super().__init__()
```

```
        self.timer = QTimer()
```

```
        self.timer.timeout.connect(self._scan_active_window)
```

```
        self.timer.start(check_interval)
```

```
    def _scan_active_window(self):
```

```
        try:
```

```
            window = gw.getActiveWindow()
```

```
            if not window or window._hWnd == 0:
```

```
                return
```

```
            hwnd = window._hWnd
```

```
            title = window.title or ""
```

```
            pid = None
```

```
            process_name = ""
```

```
        try:
```

```
            _, pid = win32process.GetWindowThreadProcessId(hwnd)
```

```
            process_name = psutil.Process(pid).name()
```

```
        except Exception:
```

```
            pass
```

```
        info = ActiveWindowInfo(
```

```
            hwnd=hwnd,
```

```
            title=title,
```

```
            process_name=process_name,
```

```

        pid=pid
    )

    self.window_detected.emit(info)

except Exception:
    pass

def stop(self):
    self.timer.stop()

```

A.4 Файл resources.py

```

class ResourceManager:
    def __init__(self, db_manager):
        self.db = db_manager

        lists = self.db.get_lists()
        self.work_list = lists.get("work", [])
        self.black_list = lists.get("black", [])

    def is_work_resource(self, window_name: str) -> bool:
        name = window_name.lower()
        for work_item in self.work_list:
            if work_item.lower() in name:
                print(f"Так вікно {name} є робочим ресурсом")
                return True
        return False

    def is_black_resource(self, window_title: str) -> bool:
        """Перевіряє, чи є назва вікна в чорному списку (розваги)"""
        for black_item in self.black_list:
            if black_item.lower() in window_title.lower():
                return True
        return False

    def get_resources(self):
        """Отримує списки сайтів/програм із бази даних"""
        return self.db.get_lists()

    def save_resources(self, work_list, black_list):
        """Зберігає оновлені списки у базу даних"""
        self.db.save_lists(work_list, black_list)

```

```

self.work_list = work_list
self.black_list = black_list
print(">>> Ресурси оновлено та збережено в БД")

```

A.5 Файл database/database.py

```

import os
import sys
import sqlite3
from datetime import datetime
from typing import List, Dict

if getattr(sys, 'frozen', False):
    BASE_DIR = os.path.dirname(sys.executable)
else:
    BASE_DIR = os.path.dirname(os.path.abspath(__file__))

ABSOLUTE_DB_PATH = os.path.join(BASE_DIR, "honestime.db")

class DatabaseManager:
    def __init__(self, db_path: str = ABSOLUTE_DB_PATH):
        self.db_path = db_path
        self._init_db()

    def _init_db(self):
        """Ініціалізація структури бази даних"""
        with sqlite3.connect(self.db_path) as conn:
            cursor = conn.cursor()

            cursor.execute("""
                CREATE TABLE IF NOT EXISTS resources (
                    id INTEGER PRIMARY KEY AUTOINCREMENT,
                    name TEXT NOT NULL,
                    type TEXT NOT NULL
                )
            """)

            cursor.execute("""
                CREATE TABLE IF NOT EXISTS sessions (
                    id INTEGER PRIMARY KEY AUTOINCREMENT,
                    date TEXT,
                    daily_goal INTEGER,
                    daily_time INTEGER,

```

```

        balance INTEGER
    )
    """
    conn.commit()

def save_lists(self, work_list: List[str], black_list: List[str]):
    """Оновлює списки дозволених та заборонених вікон"""
    with sqlite3.connect(self.db_path) as conn:
        cursor = conn.cursor()
        cursor.execute("DELETE FROM resources")
        for item in work_list:
            cursor.execute("INSERT INTO resources (name, type) VALUES
(?, 'work')", (item,))
        for item in black_list:
            cursor.execute("INSERT INTO resources (name, type) VALUES
(?, 'black')", (item,))
        conn.commit()

def get_lists(self) -> Dict[str, List[str]]:
    """Повертає словник зі списками ресурсів"""
    with sqlite3.connect(self.db_path) as conn:
        cursor = conn.cursor()
        cursor.execute("SELECT name, type FROM resources")
        rows = cursor.fetchall()
        data = {"work": [], "black": []}
        for name, r_type in rows:
            data[r_type].append(name)
        return data

def create_new_session_row(self, data: dict) -> int:
    """
    Створює новий запис у таблиці при кожному СТАРТІ або
ПРОДОВЖЕННІ роботи.
    Повертає ID цього запису, щоб ми могли його оновлювати.
    """
    now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    with sqlite3.connect(self.db_path) as conn:
        cursor = conn.cursor()
        cursor.execute("""
            INSERT INTO sessions (date, daily_goal, daily_time, balance)
            VALUES (?, ?, ?, ?)
        """, (
            now,
            data.get('daily_goal', 0),
            data.get('daily_time', 0),

```

```

        data.get('balance', 0)
    ))
    conn.commit()
    return cursor.lastrowid # Повертає ID створеного рядка

def update_session_row(self, row_id: int, data: dict):
    """
    Максимально швидко оновлення поточної активної сесії за її ID.
    Викликається кожні 10 сек або при ПАУЗІ.
    """
    if row_id is None:
        return

    with sqlite3.connect(self.db_path) as conn:
        cursor = conn.cursor()
        cursor.execute("""
            UPDATE sessions
            SET daily_time = ?, balance = ?
            WHERE id = ?
        """, (
            data.get('daily_time', 0),
            data.get('balance', 0),
            row_id
        ))
        conn.commit()

def get_last_session_data(self) -> dict:
    """
    Завантажує стан для старту нової сесії.

    daily_time завжди 0, бо він належить тільки поточній сесії.
    balance переноситься з останнього запису.
    daily_goal також береться з останнього запису.
    """
    try:
        with sqlite3.connect(self.db_path) as conn:
            conn.row_factory = sqlite3.Row
            cursor = conn.cursor()

            cursor.execute("""
                SELECT daily_goal, balance
                FROM sessions
                ORDER BY id DESC
                LIMIT 1
            """)

```

```

row = cursor.fetchone()

if not row:
    return {
        "daily_goal": 0,
        "daily_time": 0,
        "balance": 0
    }

return {
    "daily_goal": row["daily_goal"] or 0,
    "daily_time": 0,
    "balance": row["balance"] or 0
}

except Exception as e:
    print(f"Помилка завантаження останньої сесії: {e}")
    return {
        "daily_goal": 0,
        "daily_time": 0,
        "balance": 0
    }

def get_stats_for_range(self, start_date: str, end_date: str,
exclude_session_id: int | None = None) -> dict:
    """
    Отримує статистику за діапазон дат.

    daily_time у БД — це час окремої сесії.
    Тому для графіка за день треба SUM(daily_time).

    Якщо exclude_session_id передано, ця сесія не враховується в
    SQL.

    Це потрібно для live-графіка сьогодні:
    БД бере завершені/збережені сесії, а поточна додається з пам'яті.
    """
    try:
        with sqlite3.connect(self.db_path) as conn:
            conn.row_factory = sqlite3.Row
            cursor = conn.cursor()

            params = [start_date, end_date]

            exclude_sql = ""
            if exclude_session_id is not None:

```

```

exclude_sql = "AND id != ?"
params.append(exclude_session_id)

query = f"""
    WITH filtered AS (
        SELECT
            id,
            substr(date, 1, 10) AS day,
            daily_goal,
            daily_time,
            balance
        FROM sessions
        WHERE substr(date, 1, 10) BETWEEN ? AND ?
        {exclude_sql}
    ),
    totals AS (
        SELECT
            day,
            SUM(COALESCE(daily_time, 0)) AS total_time,
            MAX(id) AS last_id
        FROM filtered
        GROUP BY day
    )
    SELECT
        totals.day AS day,
        totals.total_time AS total_time,
        last_row.daily_goal AS daily_goal,
        last_row.balance AS last_balance
    FROM totals
    JOIN sessions AS last_row
        ON last_row.id = totals.last_id
    ORDER BY totals.day ASC
    """

cursor.execute(query, params)
rows = cursor.fetchall()

return {
    row["day"]: {
        "time": row["total_time"] or 0,
        "goal": row["daily_goal"] or 0,
        "balance": row["last_balance"] or 0
    }
    for row in rows
}

```

```

except Exception as e:
    print(f'DB Stats Error: {e}')
    return {}

```

A.6 Файл ui/main_window.py

```

from PyQt6.QtWidgets import QMainWindow, QWidget, QVBoxLayout,
QHBoxLayout, QLabel, QPushButton, QFrame, QTimeEdit, QApplication
from PyQt6.QtCore import Qt, QSize, QTimer, QTime
from PyQt6.QtGui import QIcon, QApplication
from .styles import *
from .work_window import WorkWindow
from PyQt6.QtWidgets import QDialog, QVBoxLayout, QLabel,
QHBoxLayout, QPushButton
from PyQt6.QtGui import QIcon, QAction
from PyQt6.QtWidgets import QSystemTrayIcon, QMenu
from ui.settings_window import SettingsWindow
from ui.stats_window import StatsWindow
import os
import sys
from PyQt6.QtGui import QIcon
from PyQt6.QtGui import QPixmap
sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
)))
from utils import resource_path
from PyQt6.QtWidgets import QLineEdit
from ui.custom_dialogs import CustomAlertDialog
from utils import resource_path, remove_window_border,
get_exit_flag_path
IS_TEST_MODE = False

class MainWindow(QMainWindow):
    def __init__(self, controller):
        super().__init__()
        self.controller = controller

        self.setWindowFlags(
            Qt.WindowType.Window |
            Qt.WindowType.FramelessWindowHint |
            Qt.WindowType.WindowSystemMenuHint |
            Qt.WindowType.WindowMinimizeButtonHint
        )
        self.setAttribute(Qt.WidgetAttribute.WA_TranslucentBackground)

```

```

self.setWindowTitle("Honestime")

self.init_ui()

self.stick_to_corner()
self.refresh_balance()

self.init_tray()

QTimer.singleShot(0, self.apply_window_effects)

def safe_exit(self):
    """Дозволено повністю вийти з програми через трей."""
    try:
        current_balance = self.controller.session.balance

        if current_balance < 0:

            bal_str, _ = self.controller._format_balance(current_balance)

            dialog = CustomAlertDialog(
                "Hold on!",
                f'Your balance is {bal_str}. Finish your work first!',
                self
            )
            dialog.exec()
            return False

        flag_path = get_exit_flag_path()

        with open(flag_path, "w", encoding="utf-8") as f:
            f.write("stop")

        print(f">>> exit.flag created: {flag_path}")
        print(">>> App closing...")

        if hasattr(self, "tray_icon"):
            self.tray_icon.hide()

        QTimer.singleShot(1200, QApplication.quit)

    return True

```

```

except Exception as e:
    print(f"Critical Error during exit: {e}")
    return False

def init_tray(self):
    self.tray_icon = QSystemTrayIcon(self)
    self.tray_icon.setToolTip("Honestime")
    self.tray_icon.setIcon(QApplication.instance().windowIcon())

    tray_menu = QMenu()

    show_action = QAction("Show Honestime", self)
    quit_action = QAction("Log out completely", self)

    show_action.triggered.connect(self.show_from_tray)
    quit_action.triggered.connect(self.on_exit_triggered)

    tray_menu.addAction(show_action)
    tray_menu.addSeparator()
    tray_menu.addAction(quit_action)

    self.tray_icon.setContextMenu(tray_menu)
    self.tray_icon.activated.connect(self.on_tray_activated)
    self.tray_icon.show()

def apply_window_effects(self):
    """
    Застосовує режим поверх інших вікон, а потім прибирає системну
рамку.
    """
    if self.isVisible() and not self.isMinimized():
        self.set_topmost(True)

    QTimer.singleShot(0, lambda: remove_window_border(self))

def set_topmost(self, enabled: bool):
    """
    Вмикає або вимикає режим поверх інших вікон.
    Після зміни flags повторно прибирає системну рамку.
    """
    was_visible = self.isVisible()

    flags = self.windowFlags()

    if enabled:

```

```

        flags |= Qt.WindowType.WindowStaysOnTopHint
    else:
        flags &= ~Qt.WindowType.WindowStaysOnTopHint

    self.setWindowFlags(flags)

    if was_visible:
        self.show()
        self.raise_()
        self.activateWindow()

    QTimer.singleShot(0, lambda: remove_window_border(self))

def show_topmost(self):
    """
    Показує вікно, вмикає режим поверх інших вікон і прибирає
рамку.
    """
    self.set_topmost(True)
    self.showNormal()
    self.raise_()
    self.activateWindow()

    QTimer.singleShot(0, lambda: remove_window_border(self))

def hide_to_tray(self):
    """
    Ховає вікно у системний трей.
    """
    self.set_topmost(False)
    self.hide()

def show_from_tray(self):
    """
    Показує актуальне вікно з трея.
    Під час активної робочої сесії головне меню не відкривається.
    """
    if self.controller.is_session_active():
        work_window = getattr(self.controller, "work_window", None)

        if work_window is not None:
            if hasattr(work_window, "show_topmost"):
                work_window.show_topmost()
            else:

```

```

        work_window.showNormal()
        work_window.raise_()
        work_window.activateWindow()

    return

self.show_topmost()

def on_tray_activated(self, reason):
    if reason == QSystemTrayIcon.ActivationReason.Trigger:
        self.show_from_tray()

def closeEvent(self, event):
    """
    Системне закриття вікна ховає систему у трей.
    Повний вихід виконується тільки через меню трея.
    """
    event.ignore()
    self.hide_to_tray()

def event(self, event):
    if event.type() == event.Type.WindowStateChange:
        QTimer.singleShot(50, self._restore_topmost_after_state_change)

    return super().event(event)

def _restore_topmost_after_state_change(self):
    if self.isVisible() and not self.isMinimized():
        self.set_topmost(True)
        QTimer.singleShot(0, lambda: remove_window_border(self))

def on_exit_triggered(self):
    success = self.safe_exit()

    if not success:
        print("Exit failed: user still has negative balance.")

def init_ui(self):
    self.main_widget = QFrame()
    self.main_widget.setObjectName("MainContainer")
    self.setCentralWidget(self.main_widget)

```

```

self.main_layout = QVBoxLayout(self.main_widget)
self.main_layout.setContentsMargins(15, 15, 15, 15)
self.main_layout.setSpacing(0)

# Header

self.logo_label = QLabel()
logo_pix = QPixmap(resource_path("assets/logo.png"))
self.logo_label.setPixmap(logo_pix.scaled(22,
Qt.AspectRatioMode.KeepAspectRatio,
Qt.TransformationMode.SmoothTransformation))
self.logo_label.setFixedSize(22, 22)
icon_chart = QIcon(resource_path("assets/chart_icon.svg"))

self.btn_chart = QPushButton()
self.btn_chart.setIcon(icon_chart)
self.btn_chart.setIconSize(QSize(20, 20))
self.btn_chart.setFixedSize(26, 26)
self.btn_chart.setProperty("class", "ControlBtn")
self.btn_chart.clicked.connect(self.open_stats)

self.quote_label = QLabel("Honestime")
self.quote_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
self.quote_label.setObjectName("QuoteLabel")

icon_set = QIcon(resource_path("assets/settings_icon.svg"))
self.btn_settings = QPushButton()
self.btn_settings.setIcon(icon_set)
self.btn_settings.setFixedSize(26, 26)
self.btn_settings.setProperty("class", "ControlBtn")
self.btn_settings.clicked.connect(self.open_settings)

self.btn_close = QPushButton("X")
self.btn_close.setFixedSize(26, 26)
self.btn_close.setProperty("class", "ControlBtn")
self.btn_close.setContentsMargins(5, 0, 0, 0)
self.btn_close.clicked.connect(self.hide_to_tray)

header_layout = QHBoxLayout()
header_layout.setAlignment(Qt.AlignmentFlag.AlignVCenter)
header_layout.addWidget(self.logo_label,
alignment=Qt.AlignmentFlag.AlignBottom)
header_layout.addSpacing(3)

```

```

        header_layout.addWidget(self.quote_label,
alignment=Qt.AlignmentFlag.AlignBottom)

        header_layout.addStretch() # Відштовхуємо кнопки вправо
        header_layout.addSpacing(10)
        header_layout.addWidget(self.btn_chart)
        header_layout.addSpacing(6)
        header_layout.addWidget(self.btn_settings)
        header_layout.addSpacing(6)
        header_layout.addWidget(self.btn_close)

        # Info
        self.goal_title = QLabel("Today's goal")
        self.goal_title.setFixedHeight(20)
        self.goal_title.setObjectName("InfoTitle")

        self.goal_input = QTimeEdit()
        self.goal_input.setObjectName("GoalInput")
        self.goal_input.setFixedHeight(BLOCK_HEIGHT)
        self.goal_input.setDisplayFormat("HH:mm:ss")
        self.goal_input.timeChanged.connect(self.lock_seconds)

self.goal_input.setButtonSymbols(QTimeEdit.ButtonSymbols.NoButtons)

        self.balance_title = QLabel("Balance")
        self.balance_title.setFixedHeight(20)
        self.balance_title.setObjectName("InfoTitle")

        self.balance_val = QLabel("+ 00:00")
        self.balance_val.setObjectName("BalanceVal")
        self.balance_val.setAlignment(Qt.AlignmentFlag.AlignCenter)
        self.balance_val.setFixedSize(BLOCK_WEIGHT_MIN,
BLOCK_HEIGHT)

        info_layout = QHBoxLayout()
        info_layout.setSpacing(10)
        goal_col = QVBoxLayout()
        goal_col.addWidget(self.goal_title);
goal_col.addWidget(self.goal_input)
        balance_col = QVBoxLayout()
        balance_col.addWidget(self.balance_title);
balance_col.addWidget(self.balance_val)
        info_layout.addLayout(goal_col); info_layout.addLayout(balance_col)

        # Buttons

```

```

self.btn_start = QPushButton("START")
self.btn_start.setObjectName("StartBtn")
self.btn_start.setFixedHeight(BLOCK_HEIGHT)
self.btn_start.clicked.connect(self.start_working)

self.btn_reset = QPushButton("RESET BALANCE")
self.btn_reset.setObjectName("ResetBtn")
self.btn_reset.setFixedHeight(24)
self.btn_reset.clicked.connect(self.handle_reset_click)

self.main_layout.addLayout(header_layout)
self.main_layout.addStretch(1)
self.main_layout.addLayout(info_layout)
self.main_layout.addStretch(1)
self.main_layout.addWidget(self.btn_start)
self.main_layout.addStretch(1)
self.main_layout.addWidget(self.btn_reset)

def lock_seconds(self, time: QTime):
    """
    Фіксує секунди на рівні 00.
    Дозволяє користувачеві крутити лише години та хвилини.
    """
    if time.second() != 0:
        # Створюємо новий об'єкт часу, де секунди примусово
        дорівнюють 0
        fixed_time = QTime(time.hour(), time.minute(), 0)

        # Блокуємо сигнали, щоб не викликати цю саму функцію
        нескінченно!
        self.goal_input.blockSignals(True)
        self.goal_input.setTime(fixed_time)
        self.goal_input.blockSignals(False)

def stick_to_corner(self):
    screen = QApplication.primaryScreen().availableGeometry()
    self.setFixedSize(290, 200)
    x = screen.x() + screen.width() - self.width()
    y = screen.y() + screen.height() - self.height()
    self.move(x, y)

def start_working(self):
    time_obj = self.goal_input.time()
    total_seconds = time_obj.minute()

```

```

        if not self.controller.work_cache:
            dialog = CustomAlertDialog(
                "Hold on!",
                "Please add at least one Work Resource in Settings before
starting a session!",
                self
            )
            dialog.exec()
            return

        if IS_TEST_MODE:
            total_seconds = time_obj.hour() * 60 + time_obj.minute()
        else:
            total_seconds = time_obj.hour() * 3600 + time_obj.minute() * 60

        self.controller.main_window = self

        if not hasattr(self, 'work_window') or self.work_window is None:
            self.work_window = WorkWindow(self.controller)
            self.controller.tick_signal.connect(self.work_window.update_ui)

        self.controller.work_window = self.work_window

        self.controller.start_new_contract(total_seconds)

        self.work_window.initial_ui_setup()

        goal_display = self.controller._format_time(total_seconds)
        self.work_window.lbl_goal_time.setText(goal_display)

        self.hide_to_tray()
        self.work_window.show_topmost()

    def mousePressEvent(self, event):
        if event.button() == Qt.MouseButton.LeftButton:
            self.drag_pos = event.globalPosition().toPoint()

    def mouseMoveEvent(self, event):
        if event.buttons() == Qt.MouseButton.LeftButton:
            self.move(self.pos() + event.globalPosition().toPoint() -
self.drag_pos)
            self.drag_pos = event.globalPosition().toPoint()

    def update_ui(self, work_time, balance, is_positive):
        self.balance_val.setText(balance)

```

```

        color = C_ACCENT_GRN if is_positive else C_ACCENT_RED
        # Динамічні стилі залишаємо тут
        self.balance_val.setStyleSheet(f"background-color: {color}; color:
white; border-radius: 6px;")

```

```

def refresh_balance(self):
    bal_str, is_pos = self.controller.get_current_balance_info()
    self.balance_val.setText(bal_str)
    color = C_ACCENT_GRN if is_pos else C_ACCENT_RED
    self.balance_val.setStyleSheet(f"""
        background-color: {color};
        color: white;
        border-radius: 6px;
        font-size: 18px;
    """)

```

```

def handle_reset_click(self):
    if not self.controller.work_cache:
        dialog = CustomAlertDialog(
            "Hold on!",
            "Please add at least one Work Resource in Settings before
resetting a session!",
            self
        )
        dialog.exec()
        return

```

```

        dialog = ResetDialog(self)

```

```

    if dialog.exec() == QDialog.DialogCode.Accepted:
        self.controller.reset_session()
        self.refresh_balance()

```

```

def open_settings(self):
    dialog = SettingsWindow(self.controller, self)
    if self.controller.session.balance < 0:
        dialog.lock_editing()

```

```

    screen = QApplication.primaryScreen().availableGeometry()
    dialog.move(
        (screen.width() - dialog.width()) // 2,
        (screen.height() - dialog.height()) // 2
    )
    dialog.exec()

```

```

def open_stats(self):
    dialog = StatsWindow(self.controller, self)

    screen = QtGuiApplication.primaryScreen().availableGeometry()
    dialog.move(
        (screen.width() - dialog.width()) // 2,
        (screen.height() - dialog.height()) // 2
    )
    dialog.exec()

class ResetDialog(QDialog):
    def __init__(self, parent=None):
        super().__init__(parent)

        self.setWindowFlags(Qt.WindowType.FramelessWindowHint |
Qt.WindowType.Dialog)
        self.setWindowModality(Qt.WindowModality.WindowModal)
        self.setAttribute(Qt.WidgetAttribute.WA_TranslucentBackground)
        if parent:
            self.setFixedSize(parent.size())
        else:
            self.setFixedSize(290, 200)

        self.main_layout = QVBoxLayout(self)
        self.main_layout.setContentsMargins(0, 0, 0, 0)

        self.bg_frame = QFrame(self)
        self.bg_frame.setObjectName("ResetBgFrame")
        self.main_layout.addWidget(self.bg_frame)

        self.content_layout = QVBoxLayout(self.bg_frame)
        self.content_layout.setContentsMargins(25, 25, 25, 20)
        self.content_layout.setSpacing(10)

        label = QLabel()
        label.setObjectName("ResetDialogText")
        text = (
            "<b><span style='color: #FFFFFF;*>Are you admitting
defeat?</b><br><br>"
            "<span style='color: #aaaaaa;*>Resetting will zero your progress "
            "and give you a 1 hour penalty. Continue?</span>"
        )
        label.setText(text)
        label.setWordWrap(True)
        label.setAlignment(Qt.AlignmentFlag.AlignCenter)

```

```

self.content_layout.addWidget(label)

btn_layout = QHBoxLayout()
btn_layout.setSpacing(15)
btn_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)

self.btn_yes = QPushButton("Yes")
self.btn_yes.setObjectName("YesBtn")
self.btn_yes.setProperty("class", "DialogBtn")

self.btn_no = QPushButton("No")
self.btn_no.setProperty("class", "DialogBtn")

btn_layout.addWidget(self.btn_yes)
btn_layout.addWidget(self.btn_no)
self.content_layout.addLayout(btn_layout)

self.btn_yes.clicked.connect(self.accept)
self.btn_no.clicked.connect(self.reject)

def showEvent(self, event):
    super().showEvent(event)

    if self.parent():
        self.move(self.parent().mapToGlobal(self.parent().rect().topLeft()))

```

A.7 Файл ui/work_window.py

```

from PyQt6.QtWidgets import QMainWindow, QVBoxLayout,
    QHBoxLayout, QLabel, QPushButton, QFrame, QApplication
from PyQt6.QtCore import Qt, QTimer
from .styles import *
from PyQt6.QtGui import QIcon, QGuiApplication
from utils import resource_path
from utils import resource_path, remove_window_border

MAX_HEIGHT = 110
MAX_WEIGHT = 290

SMALL_W = 65
WORK_W = 85
BADGE_H = 28
GAP = 8

```

```

class WorkWindow(QMainWindow):
    def __init__(self, controller):
        super().__init__()
        self.controller = controller

        self.setWindowFlags(
            Qt.WindowType.Window |
            Qt.WindowType.FramelessWindowHint |
            Qt.WindowType.WindowSystemMenuHint |
            Qt.WindowType.WindowMinimizeButtonHint
        )
        self.setAttribute(Qt.WidgetAttribute.WA_TranslucentBackground)

        self.init_ui()

        self.initial_ui_setup()

        self.setFixedHeight(MAX_HEIGHT)
        self.setFixedWidth(MAX_WEIGHT)
        self.stick_to_corner()
        self.setWindowTitle("Honestime")
        QTimer.singleShot(0, self.apply_window_effects)

    def initial_ui_setup(self):
        bal_str, is_pos = self.controller.get_current_balance_info()
        self.lbl_balance.setText(bal_str)

        color = C_ACCENT_GRN if is_pos else C_ACCENT_RED
        self.lbl_balance.setStyleSheet(f"background-color: {color}; color:
white; border-radius: 6px;")

        self.lbl_work_time.setText("00:00:00")

        self.lbl_goal_time.setObjectName("GoalTimeBadge")

        self.lbl_goal_time.setStyleSheet("")

        self.lbl_goal_time.style().unpolish(self.lbl_goal_time)
        self.lbl_goal_time.style().polish(self.lbl_goal_time)

    def apply_window_effects(self):
        """
        Застосовує режим поверх інших вікон, а потім прибирає системну
рамку.

```

```

"""
if self.isVisible() and not self.isMinimized():
    self.set_topmost(True)

QTimer.singleShot(0, lambda: remove_window_border(self))

```

```

def set_topmost(self, enabled: bool):
    """
    Вмикає або вимикає режим поверх інших вікон.
    Після зміни flags повторно прибирає системну рамку.
    """
    was_visible = self.isVisible()

    flags = self.windowFlags()

    if enabled:
        flags |= Qt.WindowType.WindowStaysOnTopHint
    else:
        flags &= ~Qt.WindowType.WindowStaysOnTopHint

    self.setWindowFlags(flags)

    if was_visible:
        self.show()
        self.raise_()
        self.activateWindow()

    QTimer.singleShot(0, lambda: remove_window_border(self))

```

```

def show_topmost(self):
    """
    Показує вікно, вмикає режим поверх інших вікон і прибирає
рамку.
    """
    self.set_topmost(True)
    self.showNormal()
    self.raise_()
    self.activateWindow()

    QTimer.singleShot(0, lambda: remove_window_border(self))

```

```

def hide_to_tray(self):

```

```

"""
Ховає робоче вікно.
"""
self.set_topmost(False)
self.hide()

def closeEvent(self, event):
    """
    Системне закриття робочого вікна ховає його, але не завершує
    сесію.
    """
    event.ignore()
    self.hide_to_tray()

def event(self, event):
    if event.type() == event.Type.WindowStateChange:
        QTimer.singleShot(50, self._restore_topmost_after_state_change)

    return super().event(event)

def _restore_topmost_after_state_change(self):
    if self.isVisible() and not self.isMinimized():
        self.set_topmost(True)
        QTimer.singleShot(0, lambda: remove_window_border(self))

def init_ui(self):
    self.main_widget = QFrame()
    self.main_widget.setObjectName("WorkContainer")
    self.setCentralWidget(self.main_widget)
    self.main_widget.setStyleSheet(WORK_WINDOW_STYLE)

    full_window_layout = QHBoxLayout(self.main_widget)
    full_window_layout.setContentsMargins(12, 12, 12, 12)
    full_window_layout.setSpacing(12)

    content_layout = QVBoxLayout()
    content_layout.setSpacing(10)

    top_row = QHBoxLayout()
    top_row.setSpacing(0)
    # TUT: Вирівнюємо весь верхній ряд по самому верху
    top_row.setAlignment(Qt.AlignmentFlag.AlignTop)

```

```

balance_col = QVBoxLayout()
balance_col.setSpacing(4)
lbl_b_title = QLabel("Balance")
lbl_b_title.setFixedHeight(18) # ТУТ: Жорстка висота
self.lbl_balance = QLabel("+ 00:00")
self.lbl_balance.setProperty("class", "TimeBadge")
self.lbl_balance.setObjectName("BalanceBadge")
self.lbl_balance.setFixedSize(SMALL_W, BADGE_H)
self.lbl_balance.setAlignment(Qt.AlignmentFlag.AlignCenter)
balance_col.addWidget(lbl_b_title)
balance_col.addWidget(self.lbl_balance)
top_row.addLayout(balance_col)

top_row.addSpacing(GAP)

times_row = QHBoxLayout()
times_row.setSpacing(GAP)

work_col = QVBoxLayout()
work_col.setSpacing(4)

lbl_w_title = QLabel("Working time")
lbl_w_title.setFixedHeight(18)

self.lbl_work_time = QLabel("00:00:00")
self.lbl_work_time.setProperty("class", "TimeBadge")
self.lbl_work_time.setFixedSize(WORK_W, BADGE_H)
self.lbl_work_time.setAlignment(Qt.AlignmentFlag.AlignCenter)

work_col.addWidget(lbl_w_title)
work_col.addWidget(self.lbl_work_time)

goal_col = QVBoxLayout()
goal_col.setSpacing(4)

lbl_g_title = QLabel("Goal")
lbl_g_title.setFixedHeight(18)

self.lbl_goal_time = QLabel("00:00")
self.lbl_goal_time.setProperty("class", "TimeBadge")
self.lbl_goal_time.setObjectName("GoalTimeBadge")
self.lbl_goal_time.setFixedSize(SMALL_W, BADGE_H)
self.lbl_goal_time.setAlignment(Qt.AlignmentFlag.AlignCenter)

```

```

goal_col.addWidget(lbl_g_title)
goal_col.addWidget(self.lbl_goal_time)

times_row.addLayout(work_col)
times_row.addLayout(goal_col)

top_row.addLayout(times_row)

content_layout.addLayout(top_row)

self.buttons_container = QFrame()
btns_layout = QHBoxLayout(self.buttons_container)
btns_layout.setContentsMargins(0, 0, 0, 0)
btns_layout.setSpacing(GAP)

self.btn_pause = QPushButton("Pause")
self.btn_pause.setFixedSize(SMALL_W, BADGE_H)
self.btn_pause.setObjectName("PauseBtn")
self.btn_pause.setStyleSheet(PAUSE_STATE_STYLE)
self.btn_pause.clicked.connect(self.handle_pause_click)

self.btn_finish = QPushButton("Finish")
self.btn_finish.setObjectName("FinishBtn")
self.btn_finish.setFixedHeight(28)
self.btn_finish.clicked.connect(self.finish_working)

btns_layout.addWidget(self.btn_pause)
btns_layout.addWidget(self.btn_finish)
content_layout.addWidget(self.buttons_container)

full_window_layout.addLayout(content_layout, stretch=1)

# 1. Створюємо вертикальну колонку
right_col = QVBoxLayout()

right_col.setContentsMargins(0, 0, 0, 0)

icon_arrow = QIcon(resource_path("assets/arrow_down.svg"))
self.btn_min = QPushButton()
self.btn_min.setIcon(icon_arrow)
self.btn_min.setFixedSize(22, 22)
self.btn_min.setProperty("class", "SideBtn")
self.btn_min.clicked.connect(self.toggle_compact_mode)

# --- Кнопка налаштувань ---

```

```

        icon_set = QIcon(resource_path("assets/settings_icon.svg"))
        self.btn_settings = QPushButton()
        self.btn_settings.setIcon(icon_set)
        self.btn_settings.setFixedSize(22, 22)
        self.btn_settings.setProperty("class", "SideBtn")
        self.btn_settings.clicked.connect(self.open_settings)

        # 2. ПОЗМИЦУЄМО ПО КРАЯХ
        right_col.addWidget(self.btn_min,
alignment=Qt.AlignmentFlag.AlignTop)

        right_col.addStretch()

        right_col.addWidget(self.btn_settings,
alignment=Qt.AlignmentFlag.AlignBottom)

        full_window_layout.addLayout(right_col)

    def finish_working(self):
        self.btn_pause.setText("Pause")
        self.btn_pause.setStyleSheet(PAUSE_STATE_STYLE)
        if self.btn_pause.isCheckable():
            self.btn_pause.setChecked(False)
        self.controller.finish_day()
        self.hide()

        if      hasattr(self.controller,      'main_window')      and
self.controller.main_window:
            main_win = self.controller.main_window
            main_win.refresh_balance()

            if hasattr(main_win, "show_topmost"):
                main_win.show_topmost()
            else:
                main_win.show()
                main_win.raise_()
                main_win.activateWindow()
        else:
            print("Error: Main window not found in controller!")

    def stick_to_corner(self):
        screen = QApplication.primaryScreen().availableGeometry()
        x = screen.x() + screen.width() - self.width()
        y = screen.y() + screen.height() - self.height()
        self.move(x, y)

```

```

def toggle_compact_mode(self):
    is_full_mode = self.btn_pause.isVisible()
    current_bottom = self.geometry().bottom()
    icon_set_down = QIcon(resource_path("assets/arrow_down.svg"))
    icon_set_up = QIcon(resource_path("assets/arrow_up.svg"))

    if is_full_mode:
        self.btn_pause.hide()
        self.btn_finish.hide()
        self.btn_min.setIcon(icon_set_down)
        self.setFixedHeight(74)
        new_y = current_bottom - 74 + 1
        self.move(self.x(), new_y)
    else:
        self.btn_pause.show()
        self.btn_finish.show()
        self.btn_min.setIcon(icon_set_up)
        self.setFixedHeight(MAX_HEIGHT)
        new_y = current_bottom - MAX_HEIGHT + 1
        self.move(self.x(), new_y)

def mousePressEvent(self, event):
    if event.button() == Qt.MouseButton.LeftButton:
        self.drag_pos = event.globalPosition().toPoint()

def mouseMoveEvent(self, event):
    if event.buttons() == Qt.MouseButton.LeftButton:
        self.move(self.pos() + event.globalPosition().toPoint() -
self.drag_pos)
        self.drag_pos = event.globalPosition().toPoint()

def update_ui(self, work_time, balance, is_positive):
    self.lbl_work_time.setText(work_time)
    self.lbl_balance.setText(balance)

    # Міняємо колір залежно від того, чи ми в плюсі
    if is_positive:
        # Зелений (профіцит)
        self.lbl_balance.setStyleSheet(f"background-color:
{C_ACCENT_GRN}; color: white; border-radius: 6px;")
    else:
        # Червоний (борг)

```

```

        self.lbl_balance.setStyleSheet(f"background-color:
{C_ACCENT_RED}); color: white; border-radius: 6px;")

    def handle_pause_click(self):
        self.controller.toggle_pause()

        if self.controller.is_paused:
            self.btn_pause.setText("Continue")
            self.btn_pause.setStyleSheet(CONTINUE_STATE_STYLE)
        else:
            self.btn_pause.setText("Pause")
            self.btn_pause.setStyleSheet(PAUSE_STATE_STYLE)

    def apply_white_goal_style(self):
        print(">>> [DEBUG] СИГНАЛ НА ЗМІНУ КОЛЬОРУ
ОТРИМАНО!")

        self.lbl_goal_time.setStyleSheet("color:    #00B330;    font-weight:
bold;")

        self.lbl_goal_time.style().unpolish(self.lbl_goal_time)
        self.lbl_goal_time.style().polish(self.lbl_goal_time)

    def open_settings(self):
        from ui.settings_window import SettingsWindow

        dialog = SettingsWindow(self.controller, self)

        # Передаємо причину "session", щоб кнопка написала правильний
текст!
        dialog.lock_editing(reason="session")

        screen = QGuiApplication.primaryScreen().availableGeometry()
        dialog.move(
            (screen.width() - dialog.width()) // 2,
            (screen.height() - dialog.height()) // 2
        )
        dialog.exec()

```

ДОДАТОК Б
Слайди презентації

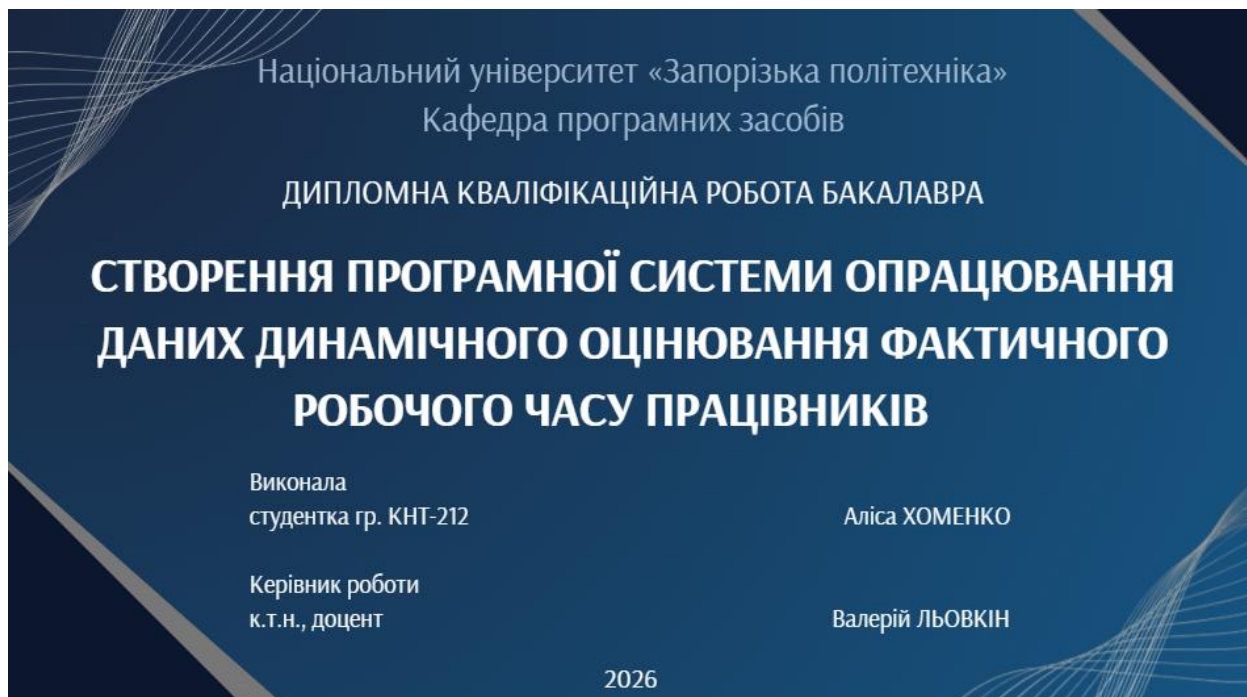


Рисунок Б.1 – Слайд 1

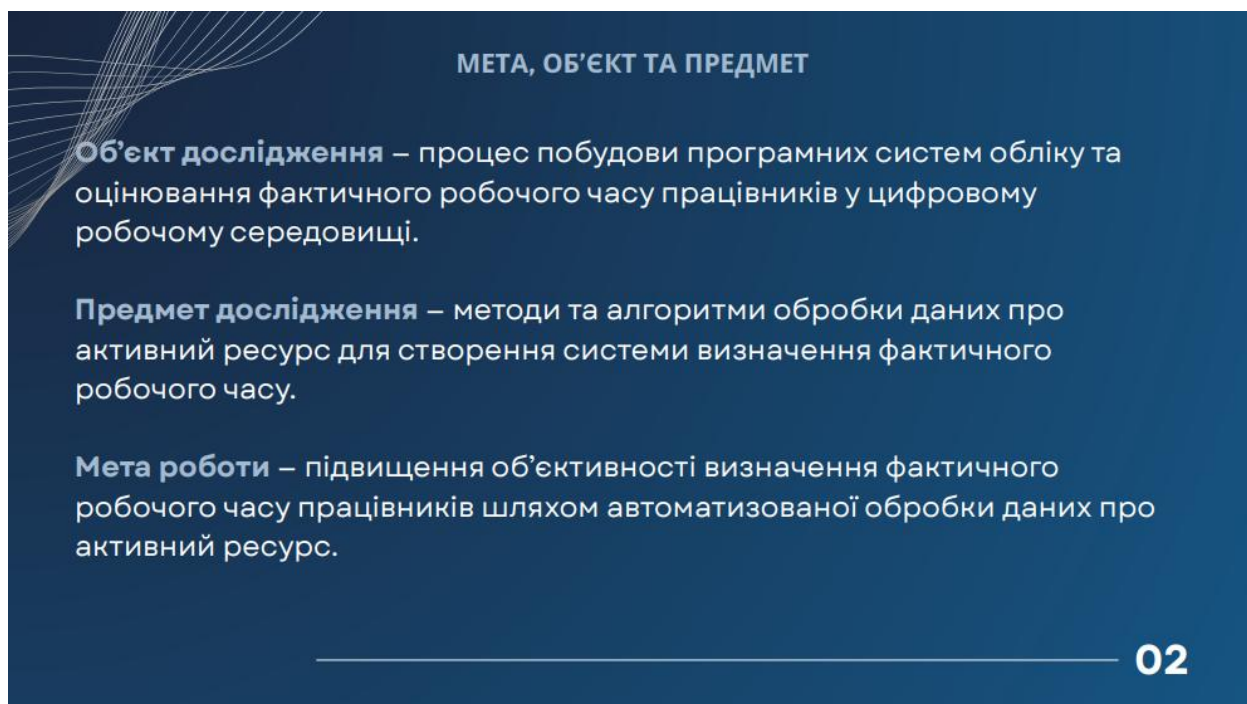


Рисунок Б.2 – Слайд 2

ЗАВДАННЯ РОБОТИ

Завдання роботи:

1. проаналізувати предметну область і визначити основні проблеми цифрової прокрастинації;
2. провести огляд існуючих програмних рішень для підвищення продуктивності, самоконтролю робочого часу та обмеження цифрових дистракторів;
3. сформулювати функціональні та нефункціональні вимоги до системи персонального обліку фактичного робочого часу працівника;
4. спроектувати архітектуру програмної системи;
5. реалізувати основні модулі програмного забезпечення;
6. провести тестування працездатності розробленої програмної системи.

03

Рисунок Б.3 – Слайд 3

ПОРІВНЯЛЬНИЙ АНАЛІЗ ПРОГРАМНИХ АНАЛОГІВ

Критерій	Focusmate	Cold Turkey Blocker	Habitica	Honestime
Платформа	Web	Desktop	Web / Mobile	Desktop
Метод контролю	Соціальна присутність	Блокування ресурсів	Гейміфікація	Моніторинг активного вікна
Автоматичний облік фактичного часу	Ні	Ні	Ні	Так
Механіка часового балансу	Ні	Ні	Частково, через HP	Так
Збереження даних	Хмарне	Локальне	Хмарне	Локальне, SQLite

04

Рисунок Б.4 – Слайд 4

ВИБІР МОВИ ПРОГРАМУВАННЯ ТА ІНСТРУМЕНТАРІЮ

Критерій	Python + PyQt6	C++ + Qt	C# + WPF
Швидкість розробки	Висока	Нижча	Середня
Графічний інтерфейс	PyQt6 / Qt v6	Qt	WPF
Локальна БД	Проста	Потребує більше коду	Підтримується
Робота з активним вікном	Через готові бібліотеки	Низькорівневий доступ	Через WinAPI
Фонові роботи	Підтримується	Підтримується	Підтримується
Читабельність коду	Висока	Нижча	Середня

05

Рисунок Б.5 – Слайд 5

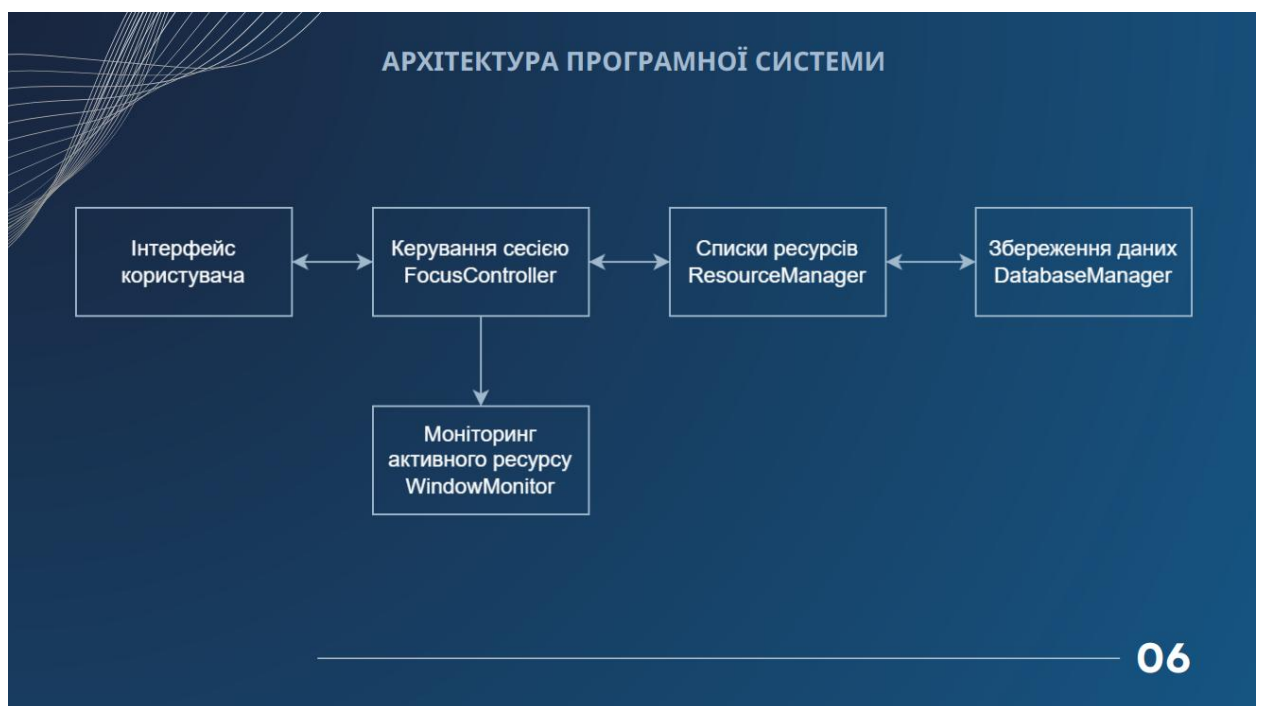


Рисунок Б.6 – Слайд 6



Рисунок Б.7 – Слайд 7



Рисунок Б.8 – Слайд 8

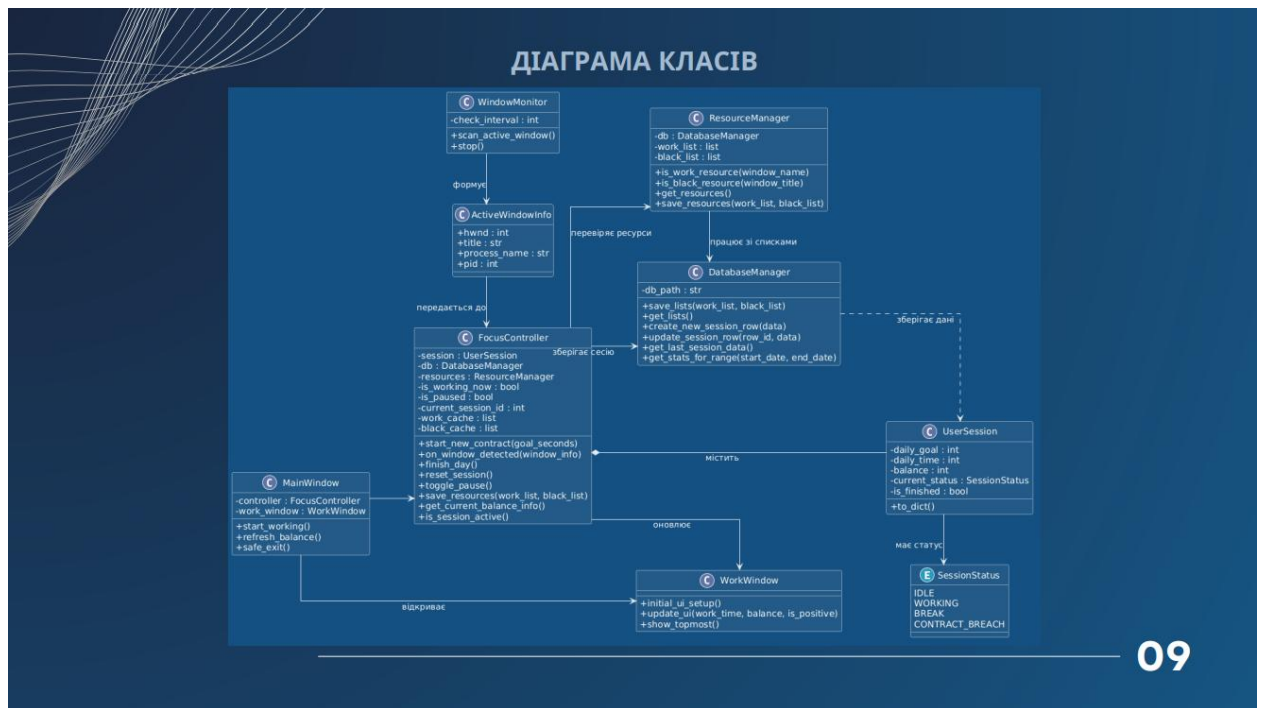


Рисунок Б.9 – Слайд 9

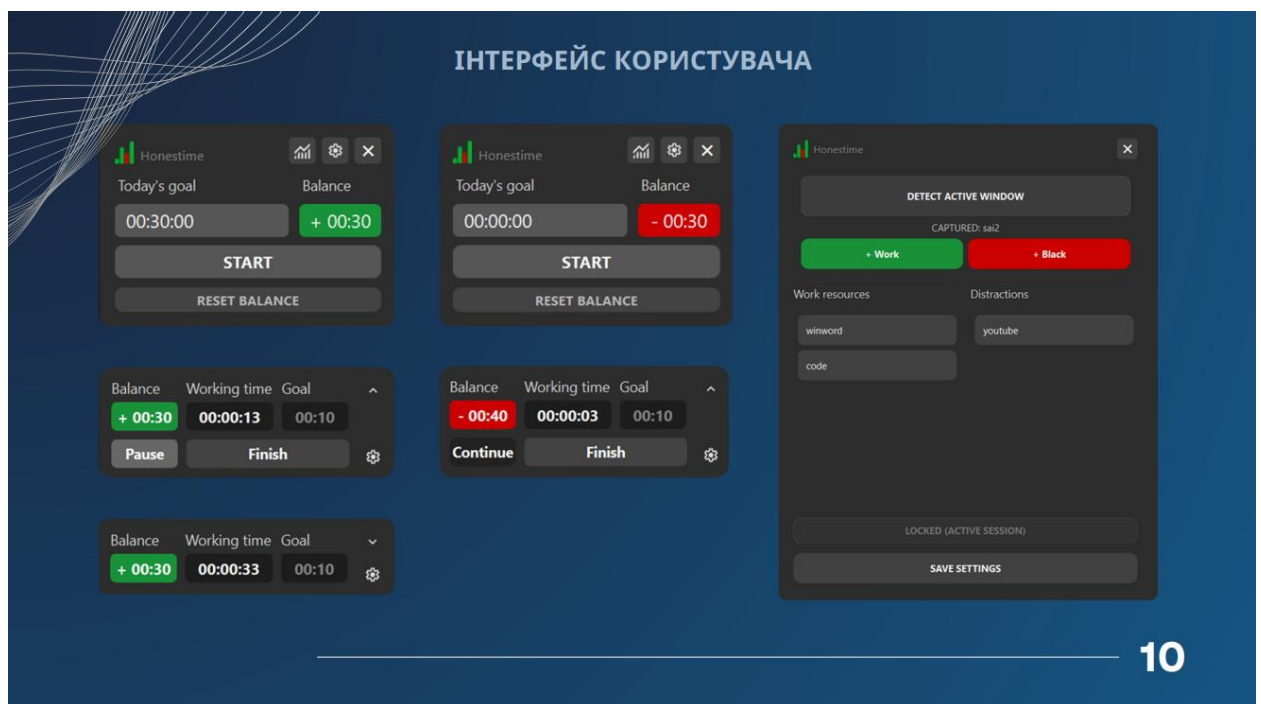


Рисунок Б.10 – Слайд 10

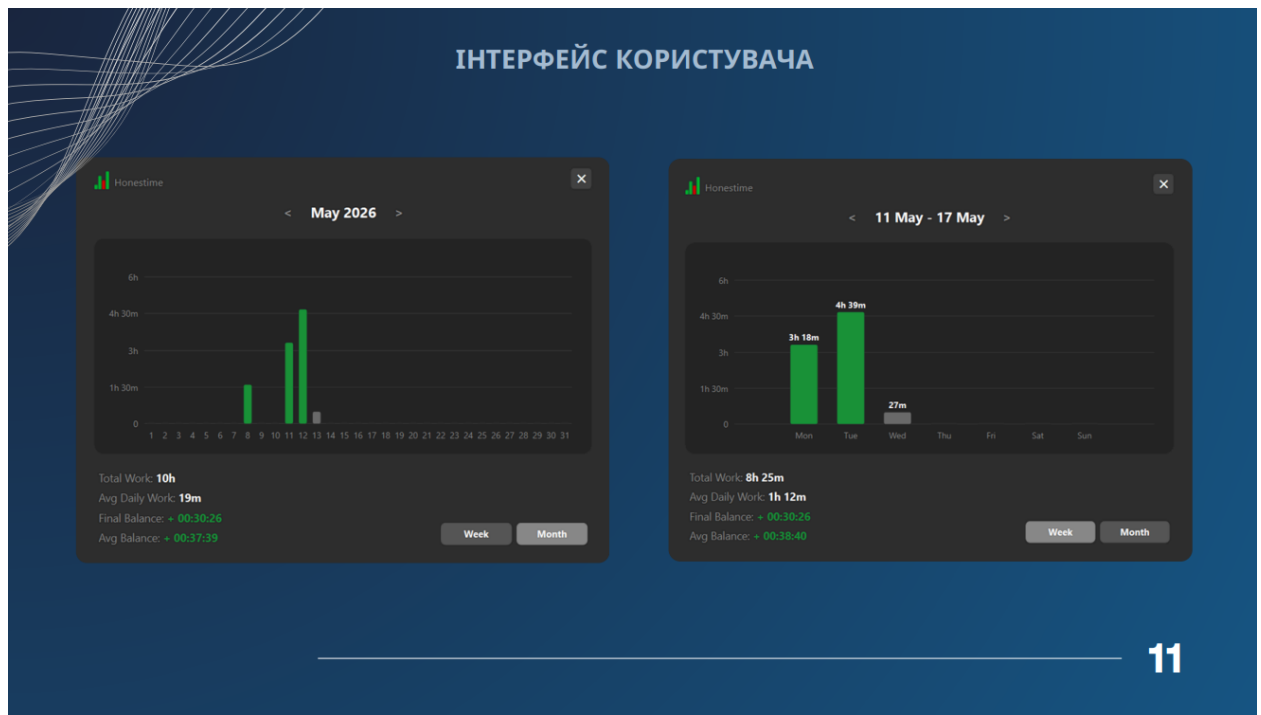


Рисунок Б.11 – Слайд 11

РЕЗУЛЬТАТИ ТЕСТУВАННЯ

№	Перевірка	Очікуваний результат	Статус
1	Запуск системи	Система відкривається, дані завантажуються	Виконано
2	Додавання ресурсу до Work-List	Ресурс зберігається в БД	Виконано
3	Додавання ресурсу до Black-List	Ресурс зберігається в БД	Виконано
4	Запуск робочої сесії	Починається моніторинг активного ресурсу	Виконано
5	Активний ресурс є у Work-List	Час зараховується як робочий	Виконано
6	Активний ресурс не у Work-List	Час не зараховується	Виконано
7	Завершення сесії	Оновлюється баланс і запис у sessions	Виконано
8	Режим Break / Reset	Стан системи змінюється відповідно до сценарію	Виконано

12

Рисунок Б.12 – Слайд 12

ВИСНОВКИ

- Проаналізовано проблему обліку фактичного робочого часу у цифровому середовищі.
- Спроектвано архітектуру, структуру бази даних та алгоритм роботи системи.
- Реалізовано програмну систему «Honestime» для моніторингу активного ресурсу, обліку часу та формування часового балансу.
- Проведено тестування основних сценаріїв роботи системи.
- Мету досягнуто: підвищено об'єктивність визначення фактичного робочого часу шляхом автоматизованої обробки даних про активний ресурс.

13

Рисунок Б.13 – Слайд 13

ПЕРЕЛІК ПОСИЛАНЬ

1. Пархоменко І. А., Хоменко А. В., Дубровін В. І. Методи та засоби розробки інтелектуального асистента для планування «ощадливого» ІТ-стартапу. Інформаційні технології і автоматизація – 2025 : зб. доп. XVIII Міжнар. наук.-практ. конф. Одеса, 2025. С. 1019–1020.
2. Інформаційні технології і автоматизація – 2025 : програма XVIII Міжнар. наук.-практ. конф., м. Одеса, 30–31 жовт. 2025 р. Одеса, 2025. URL: <https://ontu.edu.ua/download/konfi/2025/Program-Information-Technology-and-Automation-Program-2025.pdf> (дата звернення: 25.05.2026).
3. Mobility program supported by DAAD – InduTwin : information page on the academic mobility programme and TOGAF Block Week participation. URL: <https://go-study-europe.de/mobility-programm-supported-by-daad-indutwin/> (accessed: 25.05.2026).



Тези ITIA-2025



Програма ITIA-2025



DAAD / TOGAF

14

Рисунок Б.14 – Слайд 14