Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавра

зі спеціальності 121 – Інженерія програмного забезпечення

На тему: Розробка програмного засобу контролю цілісності файлів на основі розрахунку хеш-сум

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.
Студент гр. ІПЗ–21–2
_____/ Д. О. Лавецький /

Керівник кваліфікаційної роботи _____/ Д. В. Швець /

Завідувач кафедри _____/ А. М. Стрюк /

Кривий Ріг
2025

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Ступінь вищої освіти: бакалавр

Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри

_____ А. М. Стрюк

«____» _____ 2025 р.

ЗАВДАННЯ

на кваліфікаційну роботу

студенту групи ІПЗ–21–2 Лавецькому Даниїлу Олександровичу

1. Тема: Розробка програмного засобу контролю цілісності файлів на основі розрахунку хеш-сум затверджена наказом по КНУ № 200с від «10» квітня 2025 р.

2. Термін подання студентом закінченої роботи: «05» червня 2025 р.

3. Вихідні дані по роботі: програмне забезпечення має надати можливість перевіряти цілісність файлів.

4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): провести аналіз існуючих програмних продуктів, визначити основні функції додатку, спроектувати зазначений програмний засіб, реалізувати програмне забезпечення розроблюваної системи, провести тестування розробленої програми.

5. Перелік ілюстративного матеріалу: функціональна схема, блок–схема алгоритму, скріншоти вікон додатку.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Пошук літературних джерел та огляд інтернет-ресурсів з заданої тематики	15.01.25 – 24.01.25
2	Аналіз існуючих методів вирішення проблеми	27.01.25 – 07.02.25
3	Формулювання актуальності роботи і постановка завдань	10.02.25 – 21.02.25
4	Оформлення матеріалів першого розділу роботи	24.02.25 – 05.03.25
5	Розробка функціональної системи та алгоритму програми	06.03.25 – 20.03.25
6	Оформлення матеріалів другого розділу роботи	21.04.25 – 12.04.25
7	Розробка інтерфейсу програмного забезпечення, програмних модулів	15.04.25 – 02.05.25
8	Оформлення додатків	03.05.25 – 07.05.25
9	Тестування створення системи	08.05.25 – 14.05.25
10	Оформлення пояснювальної записки	15.05.25 – 04.06.25

Дата видачі завдання: «14» січня 2025 р.

Студент: _____ / Д. О. Лавецький /

Керівник роботи: _____ / Д. В. Швець /

РЕФЕРАТ

ХЕШ, ЦІЛІСНІСТЬ, ФАЙЛ, ПЕРЕВІРКА, ХЕШУВАННЯ, КОНТРОЛЬНА СУМА, ХЕШ-СУМА, MD5, SHA-1, SHA-256.

Пояснювальна записка: 52 с., 1 табл., 14 рис., 2 дод., 21 джерело.

Мета кваліфікаційної роботи: розробка програмного засобу контролю цілісності файлів на основі розрахунку хеш-сум.

Об’єкт проектування: програмний засіб контролю цілісності файлів на основі розрахунку хеш-сум.

У теоретичній частині кваліфікаційної роботи проаналізовано наявні на ринку програмні засоби для вирішення зазначеної проблеми. Проведена оцінка існуючих програмних засобів, визначено їх недоліки та переваги. Обґрунтовані актуальність роботи та сформульовані завдання дослідження.

У практичній частині реалізовано функціональну схему розроблюваного програмного продукту, структуру даних, що використовуються в програмному забезпеченні, алгоритм роботи системи. Виконані проектування та розробка баз даних, реалізовано інтерфейс програмного забезпечення та його програмну логіку. Проведено тестування розробленого додатку.

Розроблене програмне забезпечення може бути використане як у професійних колах, так і в повсякденних ситуаціях для забезпечення контролю цілісності файлів.

ABSTRACT

HASH, INTEGRITY, FILE, VERIFICATION, HASHING, CONTROL SUM, HASH SUM, MD5, SHA-1, SHA-256.

Explanatory note: 52 p., 1 table, 14 pics, 2 pp., 21 sources.

The aim of the qualification work: development of a software tool for controlling file integrity based on the calculation of hash sums.

Object of design: a software tool for controlling file integrity based on the calculation of hash sums.

The theoretical part of the qualification work analyzes the software tools available on the market to solve this problem. The existing software tools were evaluated, their disadvantages and advantages were identified. The relevance of the work is substantiated and the research objectives are formulated.

In the practical part, the functional diagram of the developed software product, the structure of the data used in the software, and the algorithm of the system are implemented. The design and development of databases were carried out, the software interface and its program logic were implemented. The developed application was tested.

The developed software can be used both in professional communities and in everyday situations to ensure file integrity control.

ЗМІСТ

ВСТУП

У сьогоднішніх умовах всеохоплюючого використання комп'ютерних систем питання збереження достовірності даних набуває особливої актуальності. Передача файлів через мережу, копіювання між пристроями, зберігання на зовнішніх носіях або у хмарних сервісах пов'язана з ризиком їхнього пошкодження, навмисної чи випадкової модифікації. У результаті можуть виникати серйозні наслідки, починаючи від втрати важливої інформації і закінчуючи порушенням безпеки систем. Тому проблема перевірки цілісності файлів є надзвичайно важливою для забезпечення надійності інформаційних процесів.

Одним із найбільш ефективних і водночас простих методів перевірки цілісності є використання хеш-сум [1] — спеціальних контрольних значень, що обчислюються за допомогою криптографічних алгоритмів на основі вмісту файлу. Будь-яка, навіть найменша зміна у структурі даних призводить до зміни хеш-суми, що дозволяє оперативно виявити пошкодження або модифікацію. Такі алгоритми, як MD5 [2], SHA1 [3], SHA256 [4] та ряд інших, є широко визнаними стандартами у сфері інформаційної безпеки та забезпечують високий рівень надійності при верифікації даних.

Незважаючи на доступність окремих інструментів, користувачі часто стикаються з труднощами в їх використанні через складність інтерфейсів, обмежену функціональність або відсутність підтримки сучасних сценаріїв взаємодії.

У зв'язку з цим виникає необхідність у створенні спеціалізованого програмного забезпечення, яке б поєднувало зручність у користуванні, підтримку популярних алгоритмів хешування, інтуїтивний інтерфейс і можливість інтеграції в повсякденні робочі процеси.

Такий програмний продукт має не лише підвищити ефективність контролю цілісності даних, але й бути доступним для широкого кола

користувачів — від системних адміністраторів до пересічних власників персональних комп’ютерів.

Отже, питання розробки програмного засобу контролю цілісності файлів на основі розрахунку хеш-сум є актуальним завданням, яке потребує розв’язання.

1 АНАЛІЗ ПИТАННЯ КОНТРОЛЮ ЦІЛІСНОСТІ ФАЙЛІВ ТА ЗАСОБІВ ЙОГО РЕАЛІЗАЦІЇ

1.1 Поняття цілісності файлів та даних

Перш за все при аналізі даної теми доцільно розглянути поняття цілісності файлів та даних.

Цілісність файлів [5] - це властивість даних залишатися такими, якими вони були на момент створення або останньої перевірки, без жодних змін чи пошкоджень. Це стосується як структури самого файлу, так і його вмісту. В ідеалі, з моменту запису файлу до нього не повинно бути внесено жодних змін, якщо тільки ці зміни не санкціоновані й не зафіксовані відповідним чином.

Порушення цілісності може відбутися з різних причин. Серед технічних чинників — апаратні збої, зокрема фізичні дефекти накопичувачів, які призводять до втрати або спотворення даних. Іншим можливим джерелом проблем є помилки, що виникають під час передавання файлів через нестабільні канали зв’язку — наприклад, під час завантаження через мережу або резервного копіювання. Також значну роль відіграють збої програмного забезпечення, що можуть призвести до некоректного запису або обробки даних.

Окрім технічних збоїв, загрозою цілісності є навмисне втручання. Зловмисники можуть змінювати вміст файлів, щоб вбудовувати шкідливий код, наприклад, віруси, шпигунські програми або бекдори — спеціальні механізми прихованого доступу до системи.

Такі модифікації, зазвичай, відбуваються непомітно для користувача, тому критично важливо мати інструменти, що дозволяють вчасно виявити навіть незначні зміни.

Для забезпечення контролю за цілісністю використовують обчислення контрольних сум, або хешів. Хеш-функція обробляє вміст файлу та створює унікальний короткий код — хеш [6]. Цей код зберігається як еталон і під час

наступної перевірки файл знову хешується. Якщо новий хеш збігається з початковим, це свідчить про те, що файл не зазнав змін. У випадку розбіжностей робиться висновок, що файл був змінений або пошкоджений. Таким чином, перевірка хешів є ефективним і надійним способом виявлення порушень цілісності.

Цілісність даних є однією з основ інформаційної безпеки. Вона забезпечує впевненість у тому, що дані зберігають свою достовірність, точність та не зазнавали несанкціонованого впливу. Збереження цілісності є критично важливим у таких сферах, як фінансові операції, зберігання медичної інформації, електронне урядування, криміналістика та в багатьох інших галузях, де навіть мінімальна зміна даних може призвести до серйозних наслідків.

1.2 Теоретичні засади контролю цілісності з використанням хеш-функцій

Контроль цілісності даних базується на використанні спеціального математичного механізму, який має назву хеш-функція. Її принцип роботи полягає в тому, що вона здатна перетворювати будь-який набір вхідних даних - незалежно від їхнього розміру чи формату - у фіксоване за довжиною значення, яке зазвичай називають хешем або хеш-сумою. Це значення є своєрідним унікальним «відбитком» даних, що характеризує їхній вміст на момент обчислення. Однією з найважливіших властивостей хеш-функції є надзвичайна чутливість до змін: навіть найменше втручання, наприклад, зміна одного символу або навіть одного біта у файлі, призводить до абсолютно іншого хеш-значення [7]. Така поведінка дозволяє з високою точністю фіксувати будь-які, навіть незначні, відхилення від первісного стану даних [8].

Процедура контролю цілісності передбачає, що після створення або отримання файлу його хеш-значення обчислюється й фіксується як еталонне. Це значення зберігається в захищеному середовищі, окремо від самого файлу або в спеціальній базі. У подальшому, коли виникає потреба перевірити, чи

файл залишився незмінним, відбувається повторне зчитування його вмісту, після чого заново виконується обчислення хешу. Отримане нове хеш-значення порівнюється з еталонним, збереженим раніше. Якщо обидва значення однакові, це означає, що вміст файлу не було змінено, і файл вважається достовірним. Якщо ж хеші не збігаються, це свідчить про те, що файл був або навмисно модифікований, або зазнав пошкоджень внаслідок збоїв чи інших чинників.

Завдяки притаманній хеш-функціям чутливості до змін, а також їхній здатності зводити довільні дані до компактного ідентифікатора, забезпечується надзвичайно ефективний і надійний механізм виявлення порушень цілісності. Це дозволяє не лише своєчасно виявляти втручання у файл, але й запобігати поширенню змінених чи шкідливих версій даних у критично важливих інформаційних системах. Надійність хеш-контролю робить його одним із основних засобів у галузі інформаційної безпеки, цифрової криміналістики, програмного оновлення, резервного копіювання та в багатьох інших сферах, де контроль за достовірністю файлів має вирішальне значення.

1.3 Існуючі програмні рішення для перевірки цілісності файлів на основі хешування

1.3.1 ExactFile

ExactFile [9] — це спеціалізований програмний засіб, призначений для перевірки цілісності файлів у середовищі операційної системи Windows. Програма розроблена з метою забезпечення максимальної точності під час контролю відповідності між оригінальним файлом і його копією, що особливо актуально у випадках, коли навіть найменше відхилення може мати критичні наслідки. Наприклад, ExactFile широко використовується при створенні резервних копій або при копіюванні файлів на оптичні носії, як-от CD чи DVD. Завдяки цьому інструменту користувач може бути впевнений, що кожен байт

даних у скопійованому файлі повністю відповідає оригіналу — такий рівень відповідності часто називають «біт-в-біт».

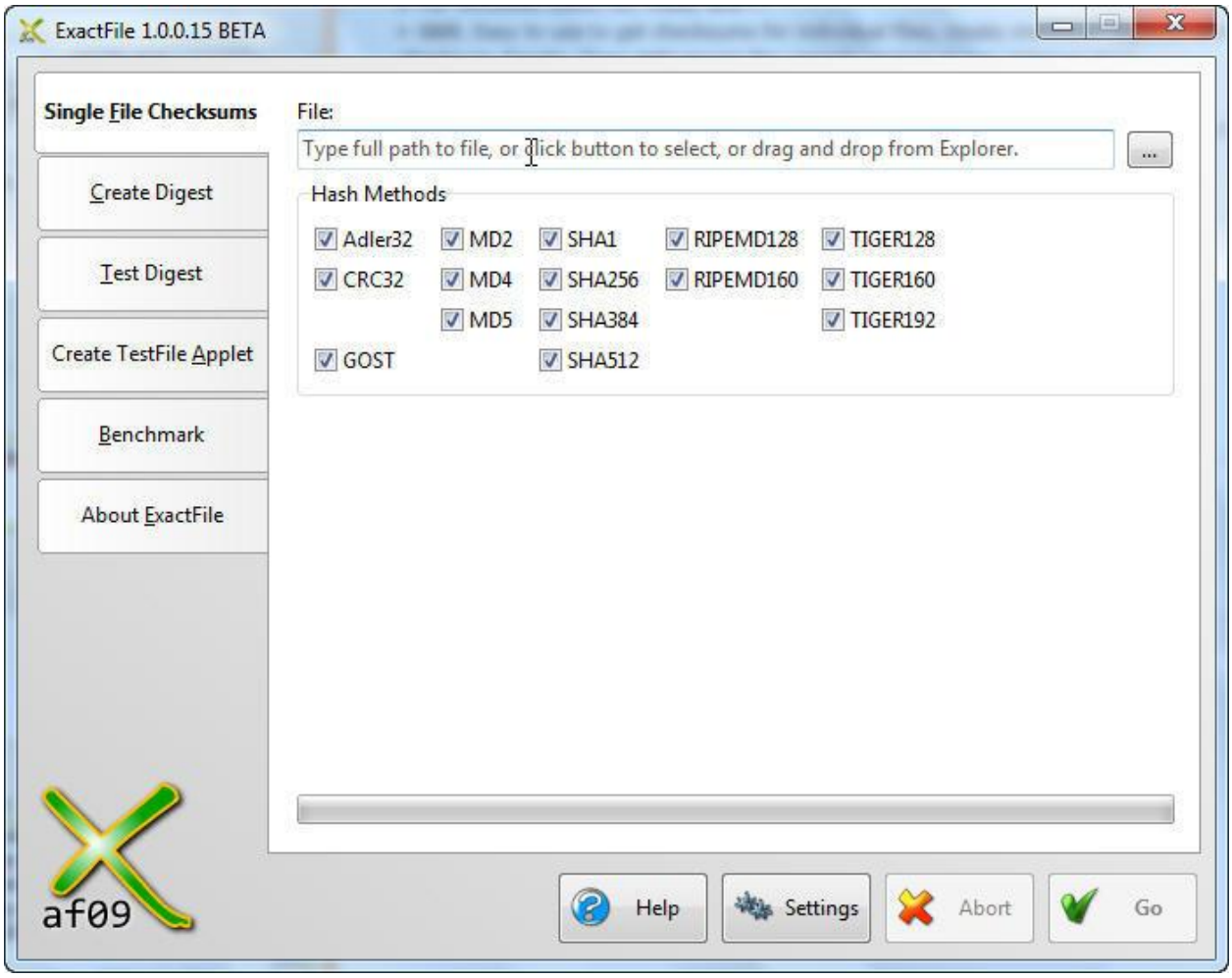

Рисунок 1.1 – Додаток ExactFile [10]

Однією з основних технічних переваг ExactFile є підтримка мультипотокового обчислення хешів. Це означає, що програма може ефективно використовувати ресурси багатоядерного процесора, значно пришвидшуючи обробку великої кількості файлів або файлів з великим обсягом даних. Це особливо корисно в умовах високонавантажених систем або під час роботи з архівами великих обсягів, коли час обробки має принципове значення.

Програма надає користувачу можливість вибору з-поміж кількох популярних алгоритмів хешування, зокрема MD5, SHA-1, CRC32 [11] та

інших. Завдяки цьому можна адаптувати перевірку до конкретних вимог - наприклад, вибрати алгоритм із більшою швидкістю або з вищим рівнем криптографічної стійкості, залежно від контексту використання. Додатковою перевагою є сумісність із різними системами кодування імен файлів, що дозволяє працювати з файлами, які містять символи з різних мовних алфавітів, включно з Unicode [12]. Це забезпечує коректну роботу з файлами з міжнародних джерел, архівів, отриманих з Інтернету, або локалізованих документів.

Ще однією сильною стороною ExactFile є його здатність працювати з дуже великими файлами — як у плані розміру, так і кількості. Це відкриває можливість для використання в середовищах, де обробляються образи дисків, великі мультимедійні файли, наукові дані або масиви логів. Програма дозволяє створювати спеціальні контрольні файли (наприклад, .exf), які можна зберігати разом із даними та використовувати для подальших перевірок у майбутньому.

Водночас варто звернути увагу на те, що ExactFile має і свої обмеження. Найсуттєвішим з них є відсутність активної підтримки та оновлень. Офіційно програма розроблена для сумісності з Windows-версіями, не пізнішими за Windows 7. Хоча в деяких випадках вона може працювати і на новіших системах, жодних гарантій щодо стабільності, безпеки або сумісності не надається. Це означає, що використання ExactFile на сучасних операційних системах, таких як Windows 10 або 11, може супроводжуватися ризиками — наприклад, програмні збої або проблеми з підтримкою нових файлових систем.

Таким чином, ExactFile є зручним інструментом для перевірки цілісності даних, що дозволяє з високою точністю переконатися в автентичності файлів після копіювання, передачі або зберігання. Його багатопотоковість, підтримка декількох алгоритмів та робота з великими обсягами інформації роблять його актуальним для низки задач, попри обмеження, пов’язані з відсутністю подальшої розробки та офіційною підтримкою лише старих версій Windows.

1.3.2 HashCalc

HashCalc [13] — це компактний, безкоштовний програмний засіб, розроблений спеціально для середовища Windows, який надає змогу швидко й просто обчислювати хеш-значення та контрольні суми для широкого спектру вхідних даних. Основна функціональність програми полягає у тому, щоб дозволити користувачу отримати унікальний цифровий підпис для файлу, текстового рядка або навіть шістнадцяткового представлення даних, що може використовуватися для перевірки достовірності чи виявлення змін.

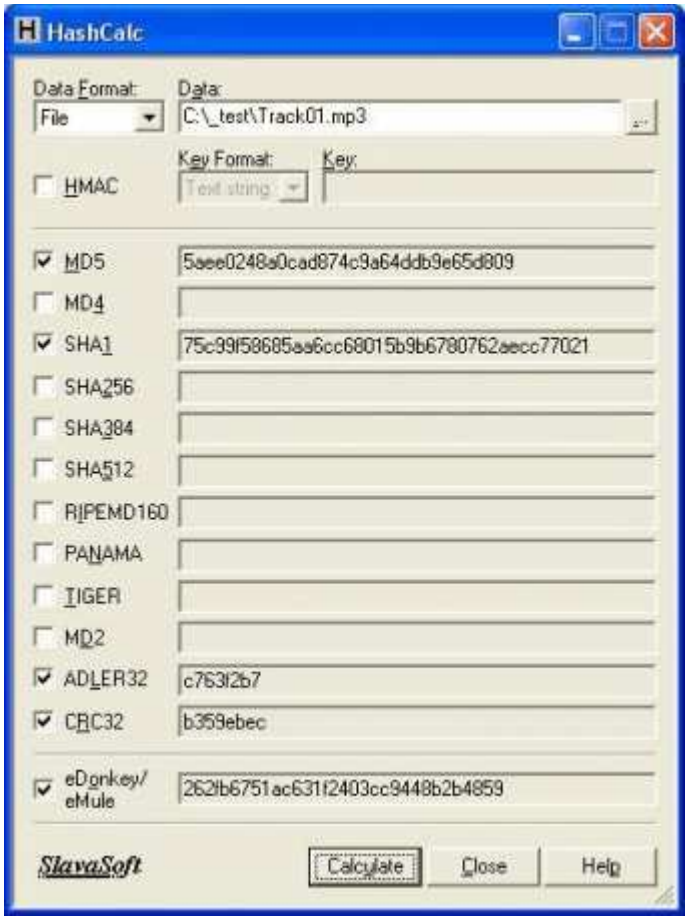

Рисунок 1.2 – Додаток HashCalc [14]

Програма підтримує велику кількість алгоритмів, серед яких можна знайти як класичні, так і менш поширені. До числа доступних хеш-функцій належать MD2 [15], MD4 [16], MD5 — алгоритми, які історично широко використовувалися у перевірці цілісності, хоча зараз вважаються

криптографічно слабкими. Також реалізована підтримка SHA-1 і SHA-2 [17] (включно з SHA-256, SHA-384 та SHA-512), що дозволяє працювати з алгоритмами, які відповідають сучасним стандартам безпеки. Присутні також альтернативні хеші, як-от RIPEMD-160 [18], PANAMA, TIGER, що можуть бути корисними для сумісності з іншими програмами або специфічними системами. Крім того, утиліта дозволяє обчислювати контрольні суми, такі як CRC32 чи ADLER32 [19], які активно використовуються в телекомунікаціях і файлових форматах для виявлення пошкоджених даних.

Відмінною рисою HashCalc є його універсальність у роботі з різними типами вхідних даних. Програма не обмежується лише обробкою файлів з локального диска. Вона також дозволяє вводити довільний текстовий рядок або шістнадцяткове представлення бітів вручну, після чого миттєво генерує відповідні хеші. Це робить її зручною для ситуацій, коли потрібно проаналізувати або порівняти хеш відомого значення без збереження його у файл.

Простий графічний інтерфейс користувача дозволяє майже миттєво здійснити обчислення: достатньо вказати джерело даних, вибрати потрібні алгоритми з переліку доступних, після чого натиснути кнопку запуску. Результати виводяться у вигляді таблиці, в якій можна легко побачити всі відповідні хеші для вхідних даних. Це особливо корисно для ручної перевірки, наприклад, коли необхідно переконатися, що завантажений архів або інсталяційний образ відповідає офіційно опублікованому хешу на сайті розробника.

Разом з тим, варто зважати на обмеження цього програмного засобу. Його архітектура орієнтована передусім на одноразові ручні операції, без можливості пакетної обробки великої кількості файлів. У HashCalc відсутні інструменти автоматизації, зокрема командний рядок, скриптинг чи API, які були б необхідними для інтеграції у більш складні робочі процеси або верифікації великих масивів даних. Крім того, інтерфейс утиліти має досить застарілий вигляд, що може знижувати зручність користування у сучасних

системах з високою роздільною здатністю екрану або нестандартними масштабами інтерфейсу.

У контексті сучасних вимог до безпеки і продуктивності HashCalc радше слід розглядати як інструмент для базових або локальних задач — наприклад, одноразової перевірки контрольної суми інсталяційного пакету чи документа. Його стабільність, швидкість і підтримка великого переліку алгоритмів роблять його ефективним у таких сценаріях. Проте для регулярного моніторингу, автоматизованого тестування чи масового аналізу він не підходить, оскільки не передбачає інтеграції з іншими системами або збереження результатів у зручному для подальшої обробки форматі.

Таким чином, HashCalc залишається корисним рішенням для окремих перевірок цілісності або отримання контрольних сум у ручному режимі, особливо для користувачів, які потребують швидкого результату без складного налаштування.

1.3.3 IDM UltraCompare

IDM UltraCompare [20] - це комерційне програмне забезпечення, розроблене для професійного аналізу відмінностей між файлами, папками та навіть архівами. Утиліта є складовою частиною більшого програмного пакета, відомого як UltraEdit Suite, але також може використовуватися автономно. Основне призначення UltraCompare полягає в тому, щоб надати користувачеві інструменти для детального зіставлення вмісту різних файлів чи директорій з можливістю синхронізації, виправлення та подальшого об'єднання відмінностей. Такий підхід широко застосовується в середовищах розробки програмного забезпечення, при роботі з великими об’ємами документації, резервного копіювання або контролю змін у корпоративних інформаційних системах.

Однією з ключових технічних особливостей UltraCompare є здатність працювати з контрольними сумами та хеш-значеннями, що дозволяє не лише порівнювати вміст на рівні символів чи рядків, але й забезпечує перевірку

цілісності даних на основі обчислених цифрових відбитків. Програма підтримує широкий спектр алгоритмів хешування - серед них CRC, MD5, SHA-1, SHA-256 - що дозволяє здійснювати незалежну перевірку достовірності файлів навіть у тих випадках, коли їхній текстовий або бінарний вміст здається ідентичним. Така функціональність особливо цінна при верифікації даних, перенесених з одного середовища до іншого, або при виявленні прихованих змін у бінарних об'єктах.

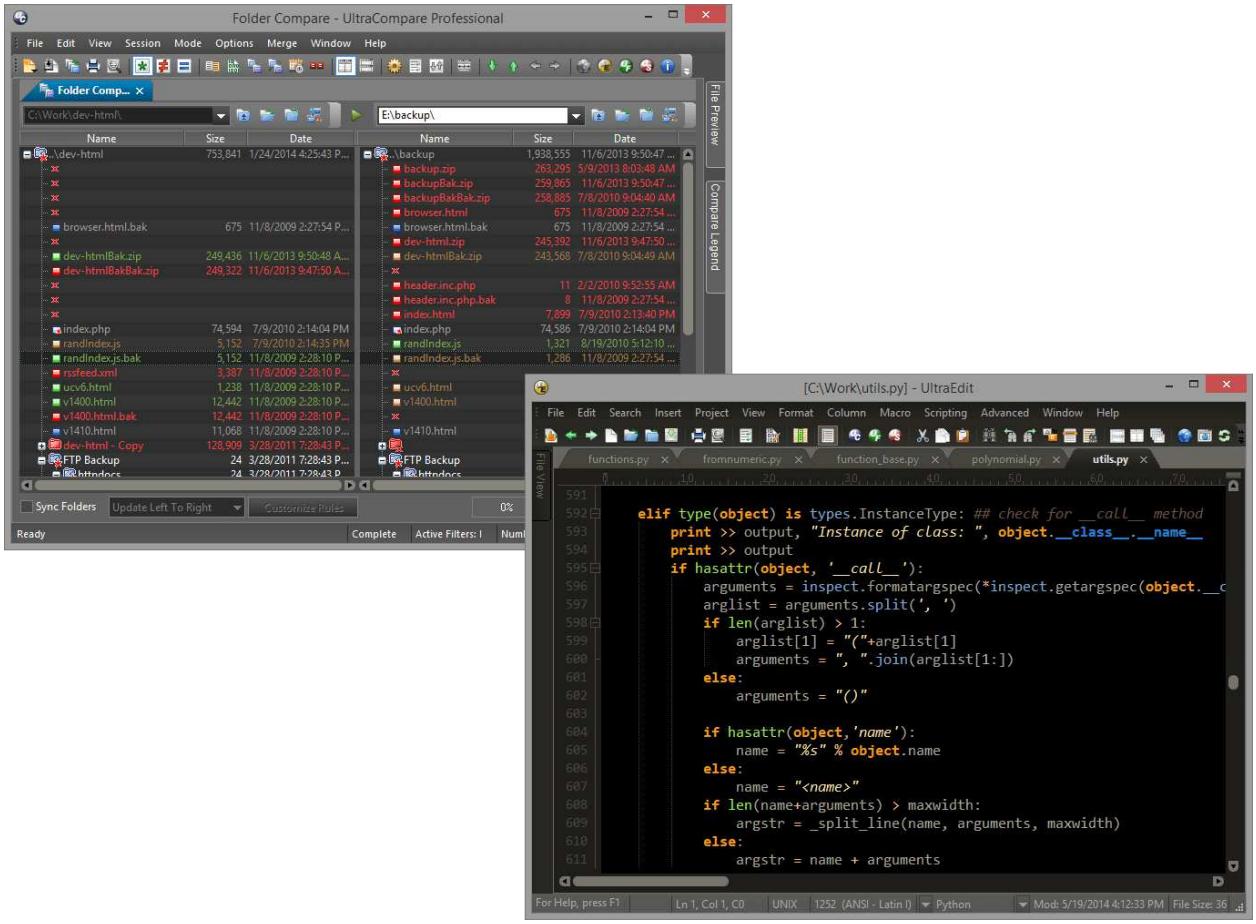

Рисунок 1.3 – Додаток IDM UltraCompare [21]

Важливою перевагою UltraCompare є його інтерактивний інтерфейс, який дозволяє користувачам візуалізувати розбіжності у зручному вигляді. Наприклад, при порівнянні двох текстових документів або скриптів, відмінності виділяються кольором у паралельному поданні, що значно пришвидшує аналіз. У разі роботи з цілими каталогами програма може

показати, які файли були змінені, додані чи видалені, а також пропонує автоматизовану синхронізацію вмісту — як у ручному, так і в автоматичному режимі. Це особливо корисно для IT-фахівців, які займаються обслуговуванням серверів, систем адміністрування або створенням резервних копій.

Ще однією важливою функцією є підтримка порівняння архівованих даних. UltraCompare вміє відкривати файли типу .zip, .rar та інші без потреби попереднього розпакування, що розширює сферу його застосування, зокрема у випадках аналізу архівів з програмним кодом, резервними копіями або інсталяційними пакетами. Крім того, програма працює з FTP- та SFTP-серверами, що дає змогу здійснювати віддалене порівняння та синхронізацію файлів напряму з хмарного або серверного сховища.

Проте UltraCompare не є безкоштовним рішенням. Для повноцінного використання необхідно придбати комерційну ліцензію, яка надається у вигляді щорічної підписки. Вартість підписки на комплект UltraEdit/UltraCompare становить близько 80 доларів США на рік [20], що може бути суттєвим бар'єром для індивідуальних користувачів або для тих, хто шукає лише базовий інструмент для перевірки контрольних сум. Також варто враховувати, що через багатофункціональність програма може здаватися надмірно складною для простих завдань, таких як одноразова перевірка цілісності файлу на основі хешу, без необхідності поглибленого аналізу вмісту чи синхронізації.

Таким чином, UltraCompare - це висококласний інструмент для комплексного аналізу файлів і папок, який поєднує можливості перевірки цілісності, візуального порівняння, синхронізації та роботи з мережевими ресурсами. Він орієнтований передусім на професійне застосування у сфері розробки програмного забезпечення, адміністрування даних або технічного контролю, де глибокий аналіз змін є критично важливим.

Однак у простих сценаріях його функціонал може бути надлишковим, а потреба в оплаті — обмежувальним чинником для широкого кола користувачів.

1.4 Результати огляду програм для оцінки цілісності файлів та висновки

Аналіз сучасних програмних рішень для контролю цілісності файлів виявляє низку значущих недоліків, які суттєво обмежують їх практичну цінність у багатьох прикладних сценаріях.

Комерційні продукти, хоч і надають широкий функціонал, часто виявляються надмірно дорогими для завдань, що потребують лише базової перевірки хеш-значень. Складні умови ліцензування та висока вартість підписки роблять такі інструменти економічно необґрунтованими у випадках, коли не передбачається використання всіх їхніх розширених можливостей.

Ще однією суттєвою проблемою є функціональна перевантаженість. Багато популярних рішень орієнтовані на розробників, адміністраторів або корпоративних користувачів і включають компоненти, які не є необхідними для звичайного порівняння контрольних сум. Замість простої та швидкої перевірки користувачі змушені взаємодіяти зі складним інтерфейсом, витрачати додаткові ресурси системи й проходити навчання, що сповільнює робочий процес.

Окремо варто зазначити недостатню гнучкість таких інструментів. Готові програми часто обмежуються підтримкою лише стандартних алгоритмів і не надають можливостей для розширення чи інтеграції у вузькоспеціалізовані сценарії. Відсутність відкритого API, скриптової підтримки або модульної архітектури унеможливлює використання цих продуктів в автоматизованих системах контролю або під час реалізації нетипових схем обробки даних.

З огляду на зазначене, можна зробити висновок, що існуючі рішення для перевірки цілісності не задовольняють повністю потреби користувачів, які шукають легкий, доступний і налаштовуваний інструмент для виконання простих перевірок.

У зв’язку з цим доцільним виглядає створення власного програмного забезпечення, яке буде орієнтоване саме на мінімалістичність, швидкодію, можливість вибору алгоритму хешування, а також простоту інтеграції у вже існуючі робочі процеси.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАСОБУ КОНТРОЛЮ ЦІЛІСНОСТІ ФАЙЛІВ НА ОСНОВІ РОЗРАХУНКУ ХЕШ-СУМ

2.1 Постановка задачі на розробку програмного забезпечення для перевірки цілісності файлів

Розробка програмного засобу контролю цілісності файлів на основі розрахунку хеш-сум вимагає чіткого визначення його функціональних можливостей, що забезпечують його коректну роботу та виконання поставлених перед ним завдань. У даному розділі сформульовано функціональні вимоги, яким має відповідати програмний засіб, що створюється.

Програмний засіб контролю цілісності файлів повинен надавати користувачеві можливість обчислення та верифікації хеш-сум файлів для перевірки їх цілісності. Основна мета програмного засобу полягає у виявленні змін або пошкоджень файлів шляхом порівняння обчисленої контрольної суми з еталонною.

Програмне забезпечення для перевірки цілісності файлів повинно реалізовувати такі функції:

а) надання користувачу можливості вибору файлу для перевірки хеш-суми за допомогою стандартного діалогу завантаження файлу або шляхом перетягування (використання Drag and Drop);

б) обчислення хеш-сум файлів за допомогою наступних алгоритмів: MD5, SHA-1, SHA-256, SHA-384, SHA-512;

в) можливість швидкої зміни хеш-функції після завантаження файлу для отримання значення хешу;

г) відображення результату обчислення хешу у зручному для копіювання вигляді;

д) введення очікуваного значення хешу вручну та перевірка відповідності обчисленої суми очікуваній;

е) відображення результату порівняння обчисленої хеш-суми з очікуваною;

ж) ведення журналу дій користувача та результатів перевірки (виконання логування);

з) надання можливості відкриття файлу журналу для перегляду з головної форми;

і) автоматичний запуск програми через контекстне меню Windows, якщо файл обрано з Провідника, та розрахунок його хеш-суми «на льоту».

к) можливість швидкої зміни мови інтерфейсу між українською та англійською.

Цей перелік вимог дозволить забезпечити потрібну функціональність створюваного програмного засобу контролю цілісності файлів на основі розрахунку хеш-сум.

2.2 Розробка UML-діаграм програмного забезпечення для контролю цілісності файлів на основі розрахунку хеш-сум

В даному розділі для більш наочної демонстрації взаємозв’язків між елементами програми, що розроблюється, та демонстрації способів її застосування наведено діаграму сценаріїв використання, діаграму послідовності та діаграму станів.

На рисунку 2.1 представлено діаграму сценаріїв використання програми для контролю цілісності файлів.

Діаграма сценаріїв використання для програми перевірки цілісності файлів показує взаємодію між користувачем та системою. Вона допомагає ідентифікувати основні функції, які користувач може виконати, а також взаємодію з іншими компонентами програми.

На діаграмі видно, що користувач може вибрати файл для перевірки, ініціювати процес обчислення хеш-суми, перевірити, чи співпадають хеш-суми, а також переглянути журнал виконаних дій.

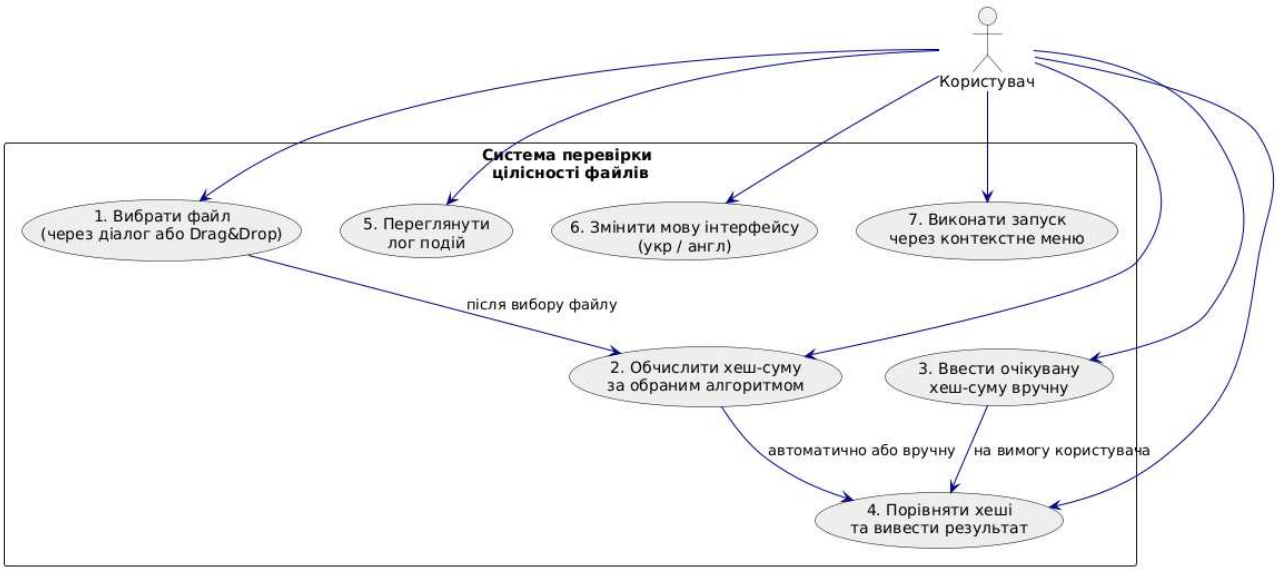

Рисунок 2.1 – Діаграма сценаріїв використання

Перше, що користувач може зробити, — це вибрати файл. Це може бути реалізовано через кнопку для вибору файлу або через функцію перетягування файлу в програму. Користувач вибирає файл, і система починає обчислювати його хеш.

Далі система починає обчислення хеш-суми вибраного файлу. Користувач може вибрати алгоритм хешування з наданого списку. Після цього програма обчислює хеш-суму файлу, використовуючи вибраний алгоритм. Якщо під час обчислення виникає помилка, система повідомляє про це користувача через відповідне повідомлення про помилку.

Після того, як хеш обчислено, програма виводить його на екран, дозволяючи користувачеві перевірити, чи збігається отримана хеш-сума з очікуваною. Користувач може ввести очікувану хеш-суму в спеціальне поле для перевірки. Програма порівнює ці хеші та виводить результат перевірки на екран.

Крім того, користувач може відкрити файл журналу, де зберігаються всі повідомлення про виконані операції. Журнал може містити інформацію про обчислені хеші, вибрані алгоритми, помилки під час обчислення та результати перевірки хешів.

Також користувач може перемикати мову інтерфейсу програми за допомогою кнопки, що дозволяє змінювати мову з української на англійську та навпаки.

Рисунок 2.2 демонструє діаграму послідовності для програми контролю цілісності файлів.

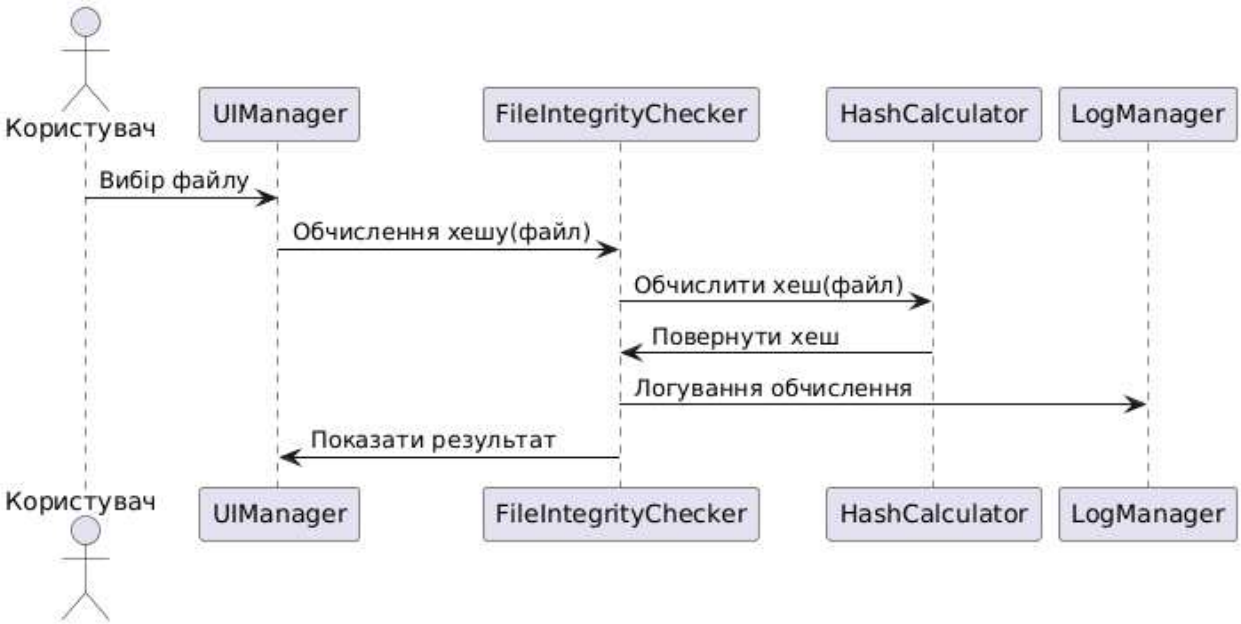

Рисунок 2.2 – Діаграмма послідовності

У згенерованій діаграмі послідовності описано процес взаємодії між компонентами програми під час вибору файлу та обчислення його хешу.

Процес починається з того, що Користувач вибирає файл через інтерфейс користувача (UIManager). Цей крок ініціює подальші дії. UIManager передає запит на обчислення хешу вибраного файлу компоненту FileIntegrityChecker. Далі FileIntegrityChecker запитує обчислення хешу у компонента HashCalculator, який виконує основне завдання – обчислює хеш

для переданого файлу. Після обчислення хешу, HashCalculator повертає результат назад у FileIntegrityChecker.

Після цього FileIntegrityChecker передає інформацію про процес обчислення в LogManager для запису в лог-файл. Логування дозволяє зберігати дані про обчислення, що важливо для подальшого аналізу.

Нарешті, FileIntegrityChecker передає результат обчислення хешу назад у UIManager, який оновлює інтерфейс користувача та відображає результат обчислення, показуючи, наприклад, хеш файлу або повідомлення про помилку, якщо така виникла.

Цей процес завершує цикл роботи програми по обчисленню хешу для вибраного файлу та відображенню результату користувачеві.

На рисунку 2.3 подано діаграму станів програми для контролю цілісності файлів.

Діаграма станів ілюструє логіку переходу між різними станами графічного інтерфейсу користувача залежно від дій користувача та внутрішніх процесів системи. Початковим станом є ситуація, коли жодного файлу не вибрано, а інтерфейс очікує на взаємодію — наприклад, користувач має обрати файл через діалогове вікно або перетягнути його у відповідну область програми. Це базовий стан системи, в якому немає завантаженого файлу і обчислення не проводяться.

Після того як користувач обирає або перетягує файл, система переходить у стан, у якому зберігається інформація про вибраний файл, і запускається процес обчислення хешу. У цьому стані відбувається зчитування вмісту файлу та обчислення контрольної суми за допомогою вибраного алгоритму хешування. Якщо цей процес виконується успішно, система переходить у стан, у якому результат хешування відображається користувачеві. У разі виникнення помилки (наприклад, файл заблокований, пошкоджений або недоступний), програма переходить у стан обробки помилки, в якому користувачеві повідомляється про проблему.

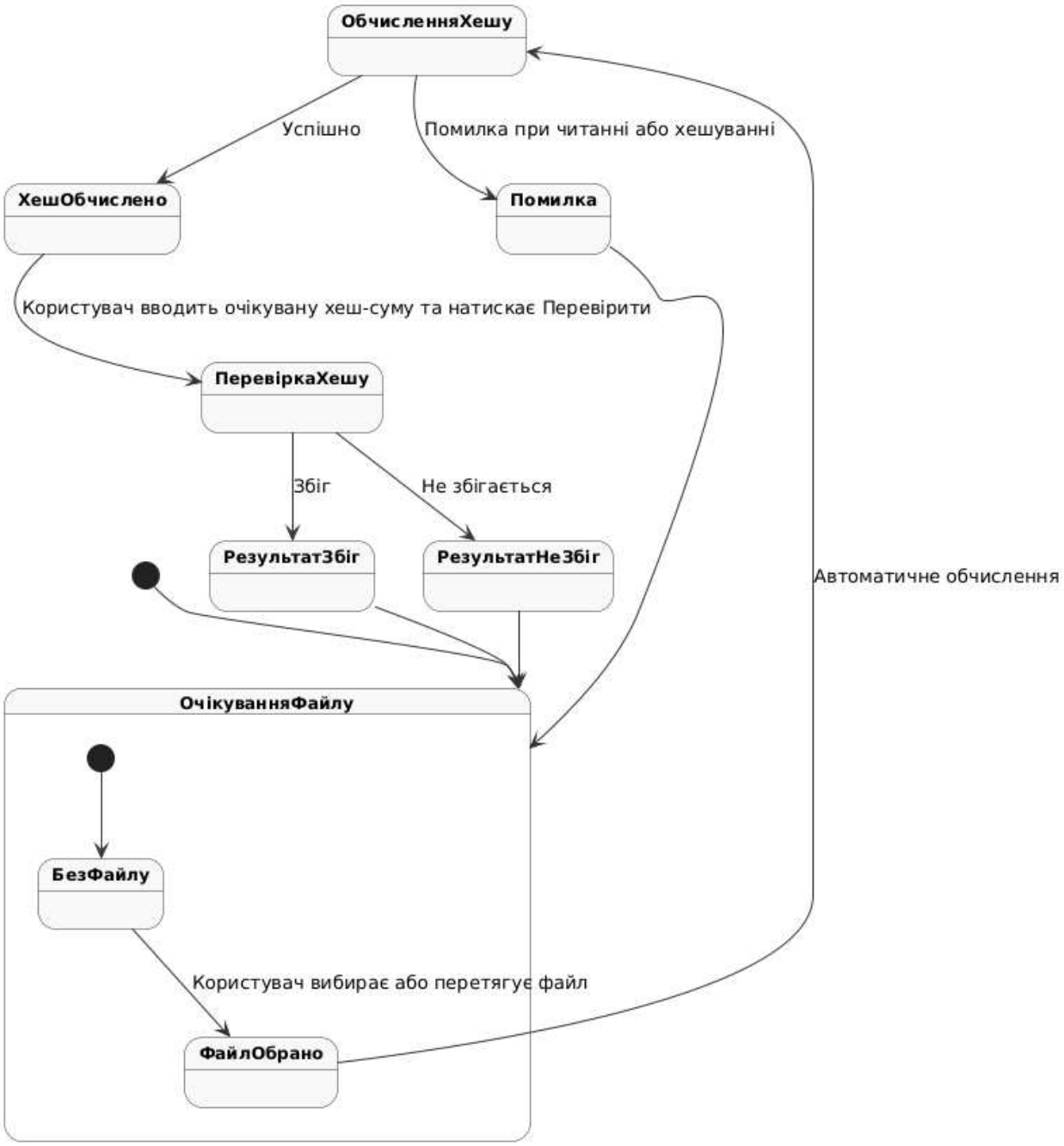

Рисунок 2.3 – Діаграма станів

Після успішного обчислення хешу, користувач має можливість ввести або вставити очікувану хеш-суму й натиснути кнопку перевірки. Це переводить програму в стан перевірки цілісності, де виконується порівняння очікуваної та фактичної хеш-сум. Якщо ці значення збігаються, система переходить у стан позитивного результату перевірки й виводить повідомлення

про збіг хешів. Якщо значення різні, то вона переходить у стан повідомлення про невідповідність.

Після завершення перевірки, незалежно від її результату, програма знову повертається у початковий стан очікування дій користувача — файл може бути змінений або обраний новий, після чого цикл повторюється. Це забезпечує гнучкість і зручність у повторному використанні інтерфейсу для перевірки кількох файлів поспіль. Уся логіка переходів між станами спрямована на мінімізацію дій користувача та швидке виявлення змін у вмісті файлів.

2.3 Визначення системних вимог до програмного засобу контролю цілісності файлів на основі розрахунку хеш-сум

Програма, що створюється, має працювати на обчислювальних машинах, що відповідають наступним мінімальним вимогам.

Таблиця 2.1 - Визначення системних вимог

Комплектуючі	**Найменування**
Процесор	i3, i5, i7 та інші види сучасних процесорів
Тактова частота	2,5 ГГц
Оперативна пам’ять	3 Гб
Обсяг пам’яті на жорсткому диску	20 Мб
Відеокарта	32Мб та вище
Роздільна здатність монітору	1024 * 768
Миша	+
Клавіатура	+
USB-порт	+

Таким чином, запуск програми має відбуватися на комп’ютерах з зазначеними вимогами або на більш продуктивних, хоча успішний запуск програмного засобу контролю цілісності файлів на основі розрахунку хеш-сум можливий і на обчислювальних машинах з меншими потужностями.

3 РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ КОНТРОЛЮ ЦІЛІСНОСТІ ФАЙЛІВ НА ОСНОВІ РОЗРАХУНКУ ХЕШ-СУМ

3.1 Алгоритм роботи програмного засобу контролю цілісності файлів на основі розрахунку хеш-сум

Рисунок 3.1 демонструє алгоритм роботи програмного засобу контролю цілісності файлів на основі розрахунку хеш-сум.

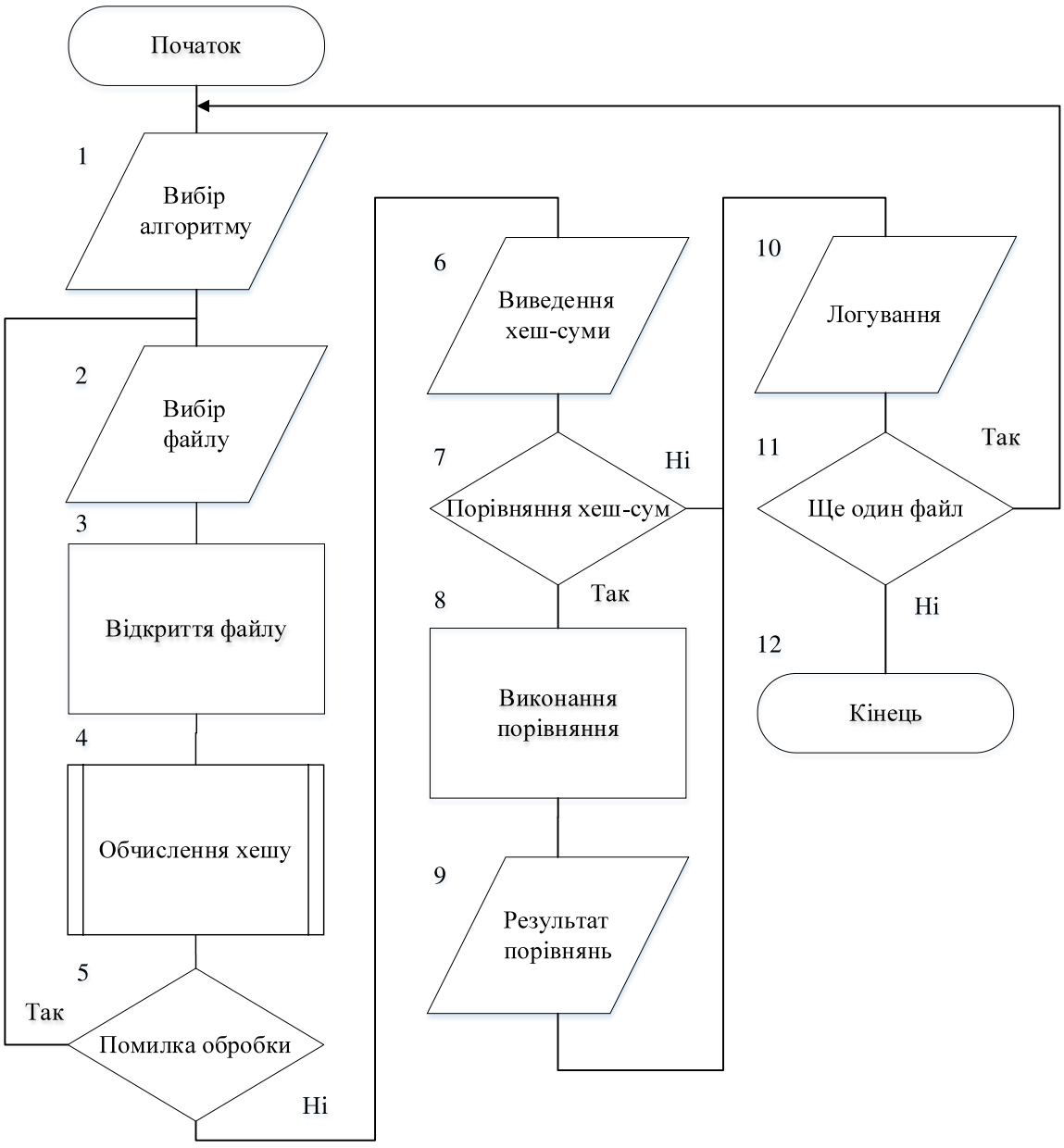

Рисунок 3.1 – Алгоритм роботи програми

Після запуску програми відбувається ініціалізація графічного інтерфейсу користувача та його основних компонентів. Одразу після цього виконується заповнення списку доступних алгоритмів хешування. Паралельно налаштовується підтримка функціональності перетягування файлів за допомогою drag-and-drop на відповідну панель у вікні програми.

Після завершення ініціалізації програма переходить у стан очікування дії користувача. Користувач може перетягнути файл у вікно програми, натиснути кнопку «Обрати» для відкриття діалогу вибору файлу, або запустити програму через контекстне меню провідника Windows із переданим шляхом до файлу як аргументом.

Якщо жодна з цих дій не виконана або файл не було отримано, програма повертається до стану очікування. Якщо ж файл отримано будь-яким способом, система ініціює процедуру обчислення хешу.

Програма зчитує назву обраного користувачем алгоритму з випадаючого списку. Потім відкриває зазначений файл для читання у двійковому форматі та запускає процес хешування згідно з вибраним алгоритмом. Після завершення обчислення хеш значення переводиться у відповідний текстовий формат.

Отримане значення хешу виводиться у визначене текстове поле інтерфейсу для ознайомлення користувача. Програма знову переходить у стан очікування, але цього разу з очікуванням введення хеш-суми, яка буде використана для порівняння.

Коли користувач натискає кнопку «Перевірити», виконується логічне порівняння обчисленої та очікуваної хеш-сум. Якщо обидва значення збігаються з урахуванням регістру, на екрані з'являється підтвердження у вигляді повідомлення про збіг. У протилежному випадку виводиться повідомлення про невідповідність хешів.

Результат перевірки записується до лог-файлу, де зберігається історія перевірок із часовими мітками. Після цього програма знову переходить до

очікування наступної дії користувача. Завершенням роботи є закриття вікна програми, що означає завершення виконання алгоритму.

3.2 Демонстрація роботи програмного засобу контролю цілісності файлів на основі розрахунку хеш-сум

На рисунку 3.2 зображено головне вікно розробленого програмного засобу. Воно містить наступні елементи: панель для перетягування файлів, поле для відображення шляху до файлу, випадаючий список вибору алгоритму хешування, кнопки «Обрати», «Перевірити», «Змінити мову» та «Відкрити лог», а також текстові поля для обчисленої та очікуваної хеш-сум, область відображення результату перевірки.

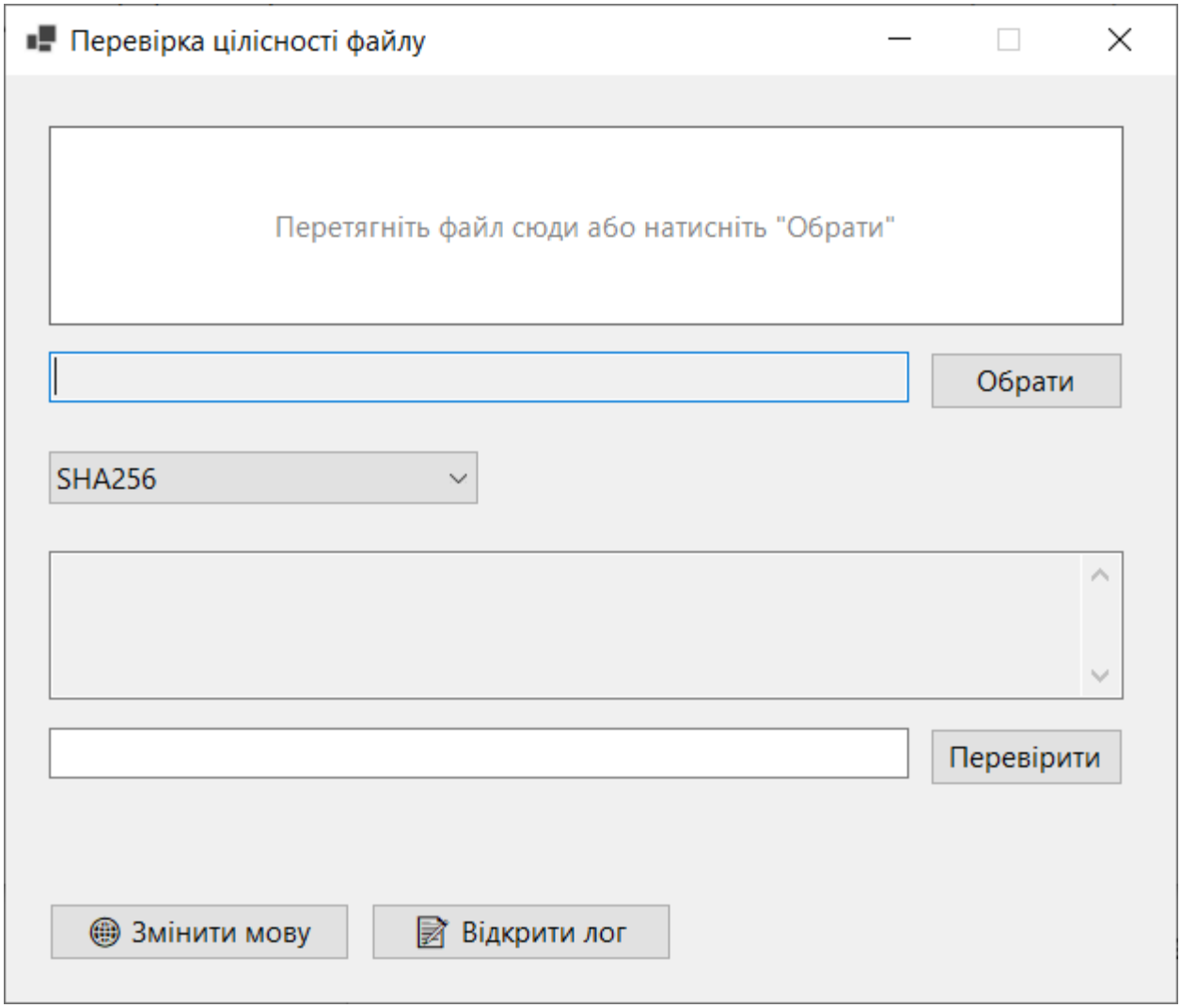

Рисунок 3.2 – Головне вікно програми

Головне призначення програми — перевірка, чи був змінений файл, шляхом порівняння його обчисленої хеш-суми із очікуваною. Для цього користувач має можливість завантажити файл у програму одним із декількох способів: перетягнути його у спеціально відведену панель у вікні програми, скористатися кнопкою «Обрати», що відкриває стандартний файловий діалог Windows, або ж запустити програму через контекстне меню файлу у провіднику Windows, якщо програма була попередньо зареєстрована в системі.

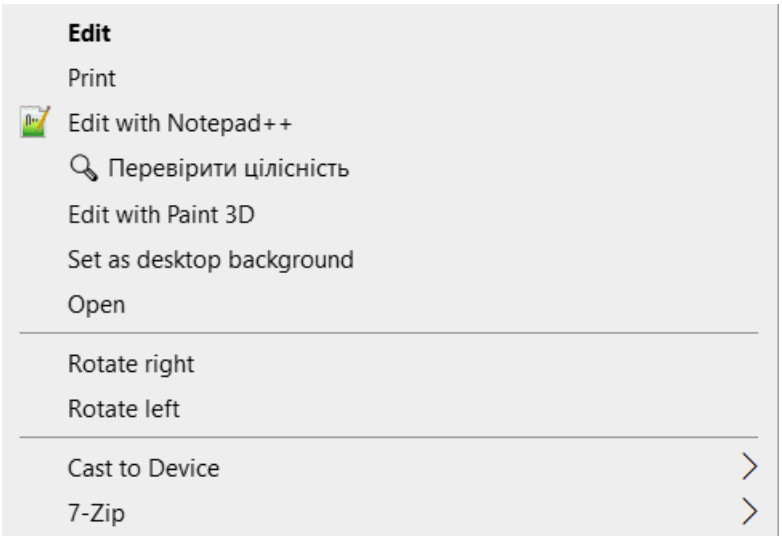

Рисунок 3.3 – Додавання файла в програму для обчислення хеш-суми шляхом використання контекстного меню (пункт «Перевірити цілісність»)

Після завантаження файлу у вікні автоматично відображається його шлях, а також запускається процедура обчислення хешу. Алгоритм для обчислення обирається з випадаючого списку, де за замовчуванням встановлений SHA256. Користувач може змінити алгоритм, після чого обчислення хешу виконається повторно в автоматичному режимі для вже завантаженого файлу. Обчислений хеш виводиться у відповідне текстове поле, де його можна переглянути та скопіювати. Форма з завантаженим файлом та його обчисленою хеш-сумою наведена на рисунку 3.4.

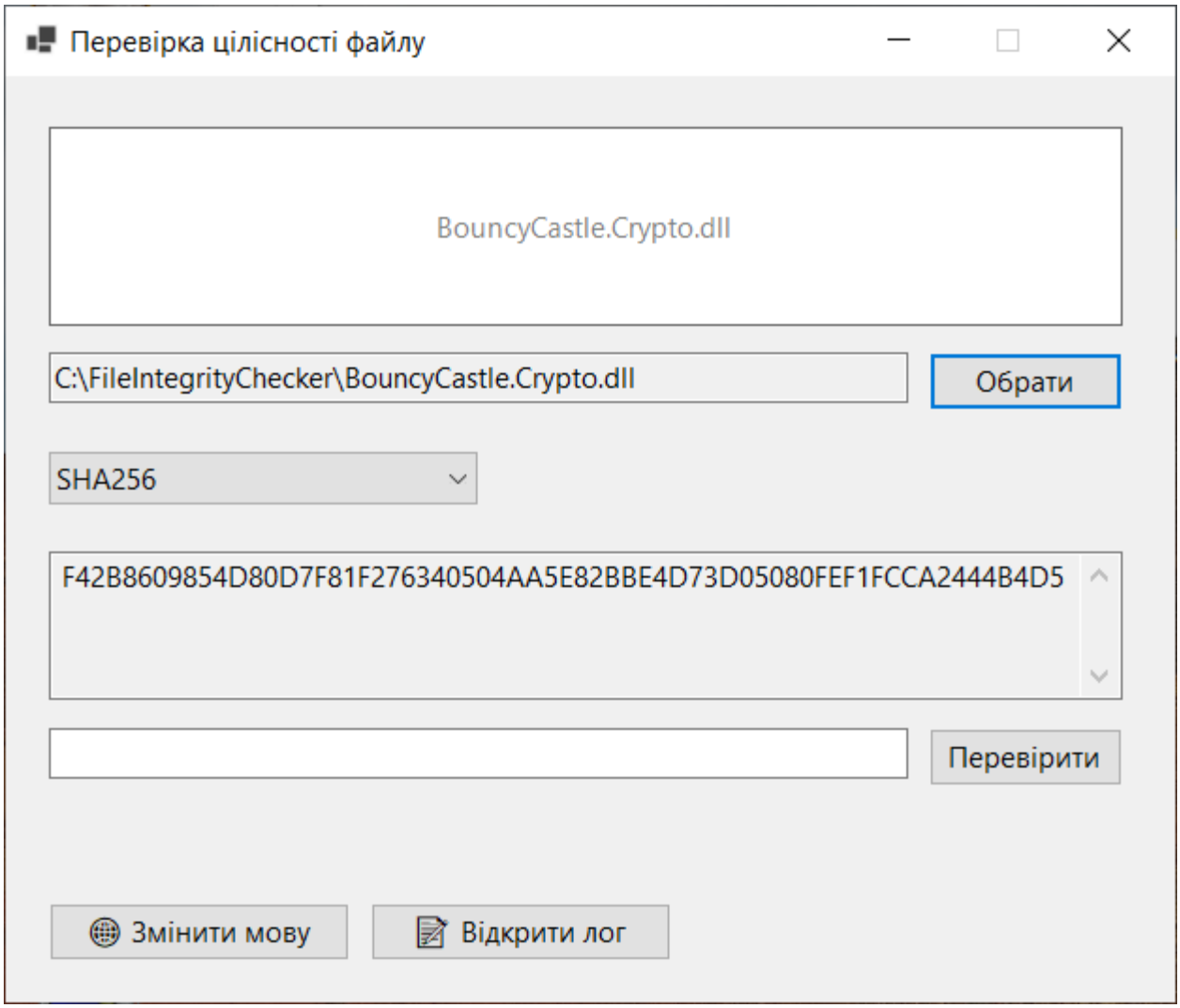

Рисунок 3.4 – Відображення результату обчислення хеш-суми завантаженого файлу

Рисунок 3.5 демонструє перелік підтримуваних програмою хеш-функцій. Для отримання хеш-сум при зміні хеш-функції перезавантажувати файл не потрібно.

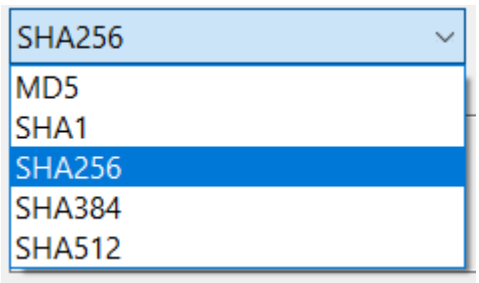

Рисунок 3.5 – Перелік хеш-функцій, що підтримуються в додатку

Наступним кроком є введення очікуваної хеш-суми — наприклад, отриманої з офіційного джерела або з раніше збережених даних. Після натискання кнопки «Перевірити» програма порівнює дві хеш-суми: щойно обчислену та введену вручну. Результат перевірки одразу відображається нижче у вигляді повідомлення з піктограмою: галочка свідчить про збіг, тоді як хрестик означає, що хеш-суми різні.

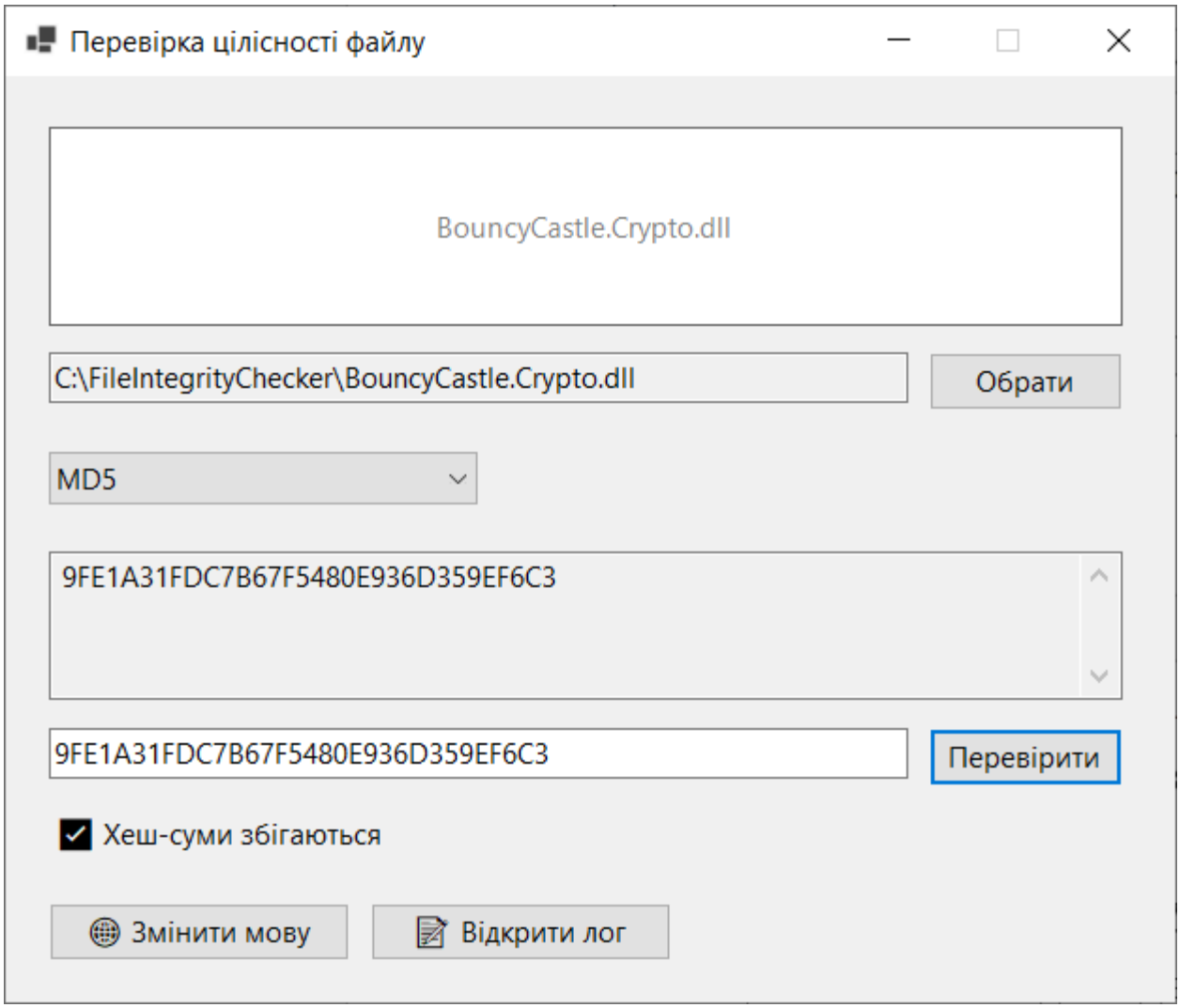

Рисунок 3.6 – Застосування перевірки хеш-сум

Паралельно всі ключові дії користувача, а також результати обчислень та перевірок фіксуються у лог-файл, який можна згодом переглянути за допомогою натискання на кнопку «Відкрити лог».

У разі виникнення помилки, наприклад, якщо файл не вдається відкрити або підтримка обраного алгоритму відсутня, програма виводить відповідне повідомлення українською або англійською мовою залежно від поточного мовного режиму.

Змінити мову інтерфейсу можна у будь-який момент натисканням на кнопку «Змінити мову», що автоматично оновлює всі написи та повідомлення у вікні.

На рисунку 3.7 продемонстровано зміну локалізації, після якої написи на формі відображаються англійською мовою, на рисунку 3.8, своєю чергою, наведено текстовий файл звіту, що відкривається після натискання кнопки «Відкрити лог».

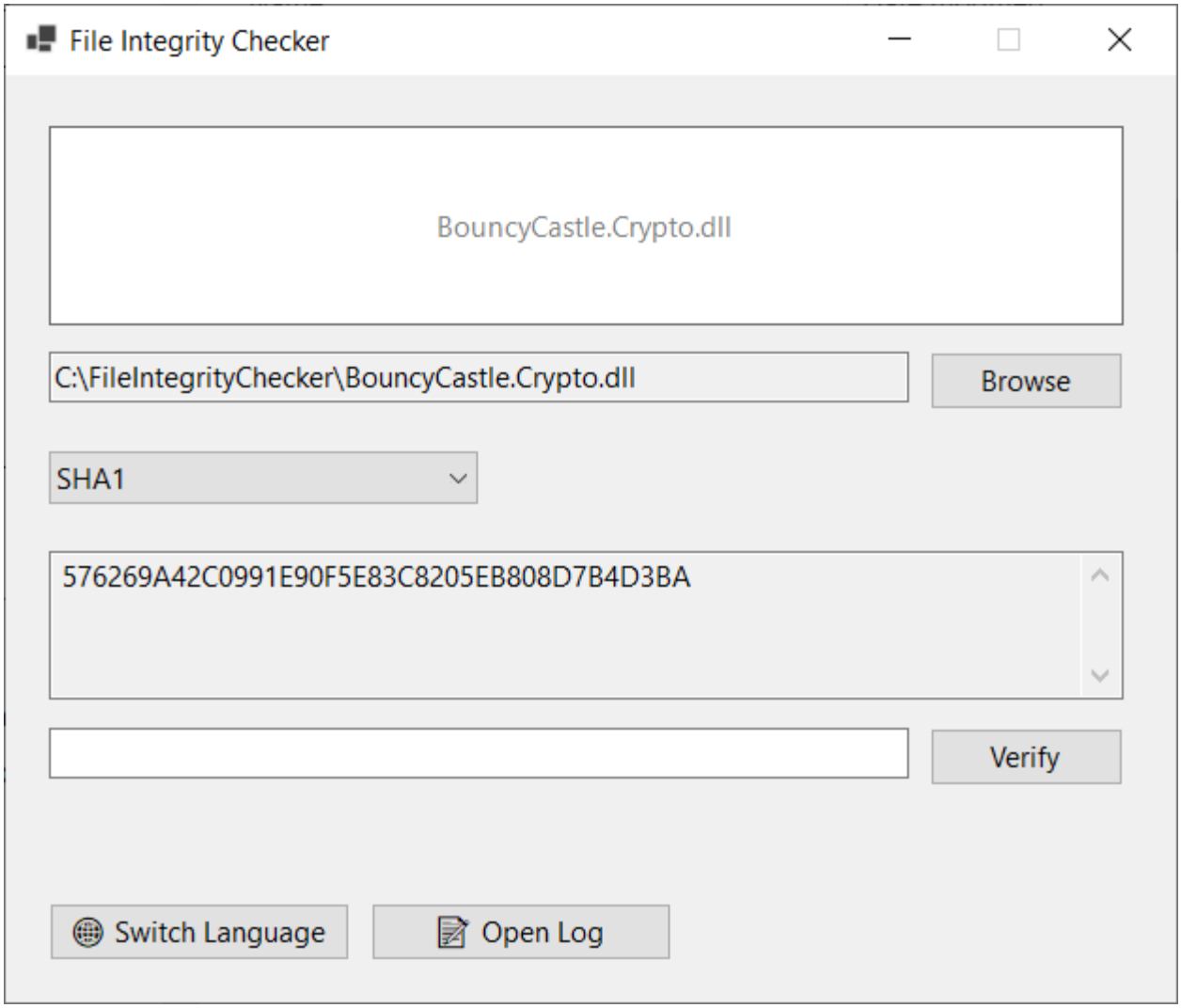

Рисунок 3.7 – Результат зміни локалізації

Рисунок 3.8 – Демонстрація log-файлу

Таким чином, процес використання програми побудований інтуїтивно та не потребує навичок просунутого користувача. Вона забезпечує простий та ефективний механізм перевірки цілісності файлів, та може бути корисна для роботи з архівами, дистрибутивами програм або важливими документами.

ВИСНОВКИ

У ході розробки програмного засобу контролю цілісності файлів на основі розрахунку хеш-сум було створено десктопний застосунок, що дозволяє користувачеві здійснювати обчислення хеш-сум файлів та порівнювати їх із очікуваними значеннями для виявлення можливих змін чи пошкоджень.

Розроблена програма підтримує найпоширеніші алгоритми хешування, зокрема MD5, SHA-1, SHA-256, SHA-384 та SHA-512, що дозволяє адаптувати її під різні вимоги безпеки.

В програмі було реалізовано функціональність drag-and-drop, що спрощує роботу з файлами, а також інтеграцію в контекстне меню провідника Windows для швидкого запуску перевірки прямо з файлової системи.

Інтерфейс програми є двомовним (українська/англійська) та адаптований для зручності кінцевого користувача. Кожна здійснена операція логується у файл, що надає змогу зберігати історію перевірок. Це є корисним для аналізу помилок чи аудиту дій користувача.

Важливою перевагою створеного застосунку є його автономність: програма не потребує підключення до Інтернету та не використовує сторонніх бібліотек, що підвищує надійність її функціонування.

У результаті роботи було досягнуто поставлену мету - розробити простий у використанні і функціональний інструмент для перевірки цілісності файлів на основі хеш-сум, що відповідає актуальним вимогам щодо контролю цілісності даних.

Розроблена програма призначена для широкого кола користувачів, яким необхідно здійснювати перевірку цілісності файлів з метою забезпечення їх автентичності та відстеження відсутності в них змін після передачі, копіювання або зберігання. Передусім, вона буде корисною системним адміністраторам, які щоденно мають справу з обробкою великої кількості

файлів у локальних і мережевих середовищах та повинні гарантувати їхню незмінність під час розгортання програмного забезпечення або оновлень.

Фахівці з інформаційної безпеки можуть використовувати програму для контролю цілісності критичних конфігураційних файлів, звітів або зашифрованих даних, перевіряючи, чи не були вони змінені в результаті несанкціонованого втручання, зловмисних дій або пошкодження носіїв інформації.

Окрім того, програма розрахована й на звичайних користувачів, які не мають глибоких технічних знань, але бажають впевнитися у справжності отриманих файлів (наприклад, драйверів, архівів або оновлень із зовнішніх джерел) або перевірити, чи не було змінено вміст власних документів у процесі збереження або передавання на інші пристрої.

Таким чином, програмний продукт є універсальним інструментом, який може бути використаний як у професійному, так і в повсякденному середовищі для забезпечення контролю цілісності файлів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Хеш-сума [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88-%D1%81%D1%83%D0%BC%D0%B0

2. Function md5() [Електронний ресурс] – Режим доступу до ресурсу: https://www.md5.cz/

3. SHA1 and other hash functions online generator [Електронний ресурс] – Режим доступу до ресурсу: http://www.sha1-online.com/

4. Як визначити хеш SHA-256 файлу для програм безпеки [Електронний ресурс] – Режим доступу до ресурсу: https://www.dell.com/support/kbdoc/uk-ua/000130826/%D1%8F%D0%BA-%D0%B2%D0%B8%D0%B7%D0%BD%D0%B0%D1%87%D0%B8%D1%82%D0%B8-%D1%85%D0%B5%D1%88-sha-256-%D1%84%D0%B0%D0%B9%D0%BB%D1%83-%D0%B4%D0%BB%D1%8F-%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC-%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D0%BA%D0%B8

5. File verification [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/File_verification

6. Математична криптологія [Електронний ресурс] – Режим доступу до ресурсу: https://ami.lnu.edu.ua/wp-content/uploads/2022/06/Cryptology9.pdf

7. What Is Hashing in Cyber Security and Why It Matters [Електронний ресурс] – Режим доступу до ресурсу: https://fidelissecurity.com/cybersecurity-101/learn/what-is-hashing

8. Managing data (Part One): Data integrity with checksums and hashes [Електронний ресурс] – Режим доступу до ресурсу: https://pomfort.com/article/managing-data-part-one-data-integrity-with-checksums-and-hashes

9. ExactFile [Електронний ресурс] – Режим доступу до ресурсу: https://www.exactfile.com/

10. ExactFile 1.0 [Електронний ресурс] – Режим доступу до ресурсу: https://exactfile.software.informer.com/1.0/

11. Циклічний надлишковий код [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%A6%D0%B8%D0%BA%D0%BB%D1%96%D1%87%D0%BD%D0%B8%D0%B9_%D0%BD%D0%B0%D0%B4%D0%BB%D0%B8%D1%88%D0%BA%D0%BE%D0%B2%D0%B8%D0%B9_%D0%BA%D0%BE%D0%B4

12. Unicode – The World Standard for Text and Emoji [Електронний ресурс] – Режим доступу до ресурсу: https://home.unicode.org/

13. Hash, CRC, and HMAC calculator [Електронний ресурс] – Режим доступу до ресурсу: https://hashcalc.en.softonic.com

14. HashCalc 2.02 [Електронний ресурс] – Режим доступу до ресурсу: https://hashcalc.software.informer.com/

15. Muller F. The MD2 Hash Function Is Not One-Way // Advances in Cryptology - ASIACRYPT 2004: 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings / P. J. Lee — Berlin, Heidelberg, New York City, London: Springer Science + Business Media, 2004. — P. 214—229. — 548 p. — ISBN 978-3-540-23975-8 — ISSN 0302-9743; 1611-3349 — doi:10.1007/978-3-540-30539-2_16

16. Kaufman, Charlie; Perlman, Radia; Speciner, Mike (2002). Network security: private communication in a public world. Upper Saddle River, NJ London: Prentice Hall PTR. ISBN 0130460192.

17. SHA-2 [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/SHA-2

18. Dobbertin, Hans; Bosselaers, Antoon; Preneel, Bart. RIPEMD-160: A strengthened version of RIPEMD (PDF). Fast Software Encryption. Third International Workshop. 1996. Cambridge, UK. pp. 71–82. doi:10.1007/3-540-60865-6_44.

19. Theresa C. Maxino, Philip J. Koopman (January 2009). "The Effectiveness of Checksums for Embedded Control Networks" (PDF). IEEE Transactions on Dependable and Secure Computing.
20. UltraEdit 2025: Benefits, Features & Pricing [Електронний ресурс] – Режим доступу до ресурсу: https://www.softwareadvice.com/app-development/ultraedit-profile
21. UltraEdit/UltraCompare Bundle [Електронний ресурс] – Режим доступу до ресурсу: https://www.componentsource.com/zh-hant/product/ultraedit-ultracompare

Додаток А – Програмний код

MainForm.cs

```
using System;
using System.IO;
using System.Security.Cryptography;
using System.Text;
using System.Windows.Forms;

namespace FileIntegrityChecker
{
    public partial class MainForm : Form
    {
        private string currentLanguage = "uk";
        private string droppedFileName = null;

        public MainForm()
        {
            InitializeComponent();
        }

        public MainForm(string[] args) : this()
        {
            InitializeAlgorithmList();
            InitializeDragAndDrop();
            if (args.Length > 0 && File.Exists(args[0]))
            {
                droppedFileName = args[0];
                txtFilePath.Text = droppedFileName;
                dropPanel.Invalidate();
                ComputeHash(droppedFileName);
            }
        }

        private void InitializeAlgorithmList()
        {
            cmbAlgorithm.Items.AddRange(new[] {
                "MD5", "SHA1", "SHA256", "SHA384", "SHA512"
            });
            cmbAlgorithm.SelectedIndex = 2; // За замовчуванням
SHA256
            cmbAlgorithm.SelectedIndexChanged += (s, e) =>
            {
                if   (!string.IsNullOrEmpty(droppedFileName)   &&
File.Exists(droppedFileName))
                {
                    ComputeHash(droppedFileName);
                }
            };
        }
```

```
        private void InitializeDragAndDrop()
        {
            dropPanel.AllowDrop = true;
            dropPanel.DragEnter += (s, e) =>
            {
                if (e.Data.GetDataPresent(DataFormats.FileDrop))
                    e.Effect = DragDropEffects.Copy;
            };

            dropPanel.DragDrop += (s, e) =>
            {
                var                    files                    =
(string[])e.Data.GetData(DataFormats.FileDrop);
                if (files.Length > 0)
                {
                    string file = files[0];
                    droppedFileName = file;
                    txtFilePath.Text = file;
                    dropPanel.Invalidate();    //    Перемальовка
панелі
                    ComputeHash(file);
                }
            };

            dropPanel.Paint += (s, e) =>
            {
                string displayText = droppedFileName == null
                    ? (currentLanguage == "uk" ? "Перетягніть файл
сюди або натисніть \"Обрати\"" : "Drag a file here or click
\"Browse\"")
                    : Path.GetFileName(droppedFileName);

                var font = this.Font;
                var size = e.Graphics.MeasureString(displayText,
font);
                float x = (dropPanel.Width - size.Width) / 2;
                float y = (dropPanel.Height - size.Height) / 2;

                e.Graphics.DrawString(displayText,          font,
System.Drawing.Brushes.Gray, new System.Drawing.PointF(x, y));
            };
        }

        private void ComputeHash(string filePath)
        {
            try
            {
                string                 algoName                 =
cmbAlgorithm.SelectedItem.ToString();
                Log($"[{DateTime.Now}] Starting hash computation
for file: {filePath} with algorithm: {algoName}");
```

```
                using FileStream fs = File.OpenRead(filePath);
                HashAlgorithm hashAlg = algoName switch
                {
                    "MD5" => MD5.Create(),
                    "SHA1" => SHA1.Create(),
                    "SHA256" => SHA256.Create(),
                    "SHA384" => SHA384.Create(),
                    "SHA512" => SHA512.Create(),
                    _ => SHA256.Create(),
                };

                byte[] hashBytes = hashAlg.ComputeHash(fs);
                string hashString =
BitConverter.ToString(hashBytes).Replace("-", "");
                txtComputedHash.Text = hashString;
                Log($"[{DateTime.Now}] {algoName} hash for
'{filePath}': {hashString}");
            }
            catch (Exception ex)
            {
                Log($"[{DateTime.Now}] Hash error:
{ex.ToString()}");
                MessageBox.Show(currentLanguage == "uk" ?
"Помилка під час обчислення хешу." : "Error during hash
computation.");
            }
        }


        private void btnBrowse_Click(object sender, EventArgs e)
        {
            using OpenFileDialog dlg = new OpenFileDialog();
            if (dlg.ShowDialog() == DialogResult.OK)
            {
                droppedFileName = dlg.FileName;
                txtFilePath.Text = dlg.FileName;
                dropPanel.Invalidate(); // Оновити вміст панелі
                ComputeHash(dlg.FileName);
            }
        }

        private void btnVerify_Click(object sender, EventArgs e)
        {
            bool match =
txtComputedHash.Text.Trim().Equals(txtExpectedHash.Text.Trim(),
StringComparison.OrdinalIgnoreCase);
            lblResult.Text = match
                ? (currentLanguage == "uk" ? "✅ Хеш-суми
збігаються" : "✅ Hashes match")
```

```
                : (currentLanguage == "uk" ? "❌ Хеш-суми не
збігаються" : "❌ Hashes do not match");

            Log($"[{DateTime.Now}]      Verification      result:
{lblResult.Text}");
        }

        private   void   btnSwitchLanguage_Click(object   sender,
EventArgs e)
        {
            currentLanguage = currentLanguage == "uk" ? "en" :
"uk";
            UpdateUILanguage();
            dropPanel.Invalidate(); // також оновлюємо текст в
зоні
        }

        private void btnOpenLog_Click(object sender, EventArgs e)
        {
            try
            {
                string                  logPath                 =
Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "log.txt");
                if (File.Exists(logPath))
                {

System.Diagnostics.Process.Start("notepad.exe", logPath);
                }
                else
                {
                    MessageBox.Show(currentLanguage   ==   "uk"   ?
"Файл логу не знайдено." : "Log file not found.");
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show((currentLanguage    ==    "uk"    ?
"Помилка відкриття логу: " : "Error opening log: ") + ex.Message);
            }
        }

        private void UpdateUILanguage()
        {
            if (currentLanguage == "uk")
            {
                btnBrowse.Text = "Обрати";
                btnVerify.Text = "Перевірити";
                btnSwitchLanguage.Text = "🌐 Змінити мову";
                btnOpenLog.Text = "📝 Відкрити лог";
                this.Text = "Перевірка цілісності файлу";
            }
```

```
            else
            {
                btnBrowse.Text = "Browse";
                btnVerify.Text = "Verify";
                btnSwitchLanguage.Text = "🌐 Switch Language";
                btnOpenLog.Text = "📝 Open Log";
                this.Text = "File Integrity Checker";
            }
        }

        private void Log(string message)
        {
            try
            {
                File.AppendAllText("log.txt",      message      +
Environment.NewLine);
            }
            catch
            {

            }
        }


    }

}
```

MainForm.Designer.cs

```
namespace FileIntegrityChecker
{
    partial class MainForm
    {
        private  System.ComponentModel.IContainer  components  =
null;

        // Controls
        private System.Windows.Forms.Panel dropPanel;
        private System.Windows.Forms.TextBox txtFilePath;
        private System.Windows.Forms.Button btnBrowse;
        private System.Windows.Forms.ComboBox cmbAlgorithm;
        private System.Windows.Forms.TextBox txtComputedHash;
        private System.Windows.Forms.TextBox txtExpectedHash;
        private System.Windows.Forms.Button btnVerify;
        private System.Windows.Forms.Label lblResult;
        private System.Windows.Forms.Button btnSwitchLanguage;
        private System.Windows.Forms.Button btnOpenLog;

        protected override void Dispose(bool disposing)
        {
```

```
            if (disposing && (components != null))
                components.Dispose();
            base.Dispose(disposing);
        }

        private void InitializeComponent()
        {
            this.components = new
System.ComponentModel.Container();
            this.dropPanel = new System.Windows.Forms.Panel();
            this.txtFilePath = new
System.Windows.Forms.TextBox();
            this.btnBrowse = new System.Windows.Forms.Button();
            this.cmbAlgorithm = new
System.Windows.Forms.ComboBox();
            this.txtComputedHash = new
System.Windows.Forms.TextBox();
            this.txtExpectedHash = new
System.Windows.Forms.TextBox();
            this.btnVerify = new System.Windows.Forms.Button();
            this.lblResult = new System.Windows.Forms.Label();
            this.btnSwitchLanguage = new
System.Windows.Forms.Button();
            this.btnOpenLog = new System.Windows.Forms.Button();
            this.SuspendLayout();

            // dropPanel
            this.dropPanel.BorderStyle =
System.Windows.Forms.BorderStyle.FixedSingle;
            this.dropPanel.Location = new
System.Drawing.Point(20, 20);
            this.dropPanel.Name = "dropPanel";
            this.dropPanel.Size = new System.Drawing.Size(500,
80);
            this.dropPanel.TabIndex = 0;
            this.dropPanel.AllowDrop = true;
            this.dropPanel.BackColor =
System.Drawing.Color.White;
            //this.dropPanel.Paint += (s, e) => {
                //e.Graphics.DrawString("Перетягніть файл сюди
або натисніть \"Обрати\"",
                    //this.Font, System.Drawing.Brushes.Gray, new
System.Drawing.PointF(10, 30));
            //};

            // txtFilePath
            this.txtFilePath.Location = new
System.Drawing.Point(20, 110);
            this.txtFilePath.Name = "txtFilePath";
            this.txtFilePath.ReadOnly = true;
            this.txtFilePath.Size = new System.Drawing.Size(400,
23);
```

```
            this.txtFilePath.TabIndex = 1;

            // btnBrowse
            this.btnBrowse.Location             =             new
System.Drawing.Point(430, 110);
            this.btnBrowse.Name = "btnBrowse";
            this.btnBrowse.Size   =   new   System.Drawing.Size(90,
23);
            this.btnBrowse.TabIndex = 2;
            this.btnBrowse.Text = "Обрати";
            this.btnBrowse.UseVisualStyleBackColor = true;
            this.btnBrowse.Click              +=              new
System.EventHandler(this.btnBrowse_Click);

            // cmbAlgorithm
            this.cmbAlgorithm.DropDownStyle                      =
System.Windows.Forms.ComboBoxStyle.DropDownList;
            this.cmbAlgorithm.Location           =           new
System.Drawing.Point(20, 150);
            this.cmbAlgorithm.Name = "cmbAlgorithm";
            this.cmbAlgorithm.Size = new System.Drawing.Size(200,
23);
            this.cmbAlgorithm.TabIndex = 3;

            // txtComputedHash
            this.txtComputedHash.Location          =          new
System.Drawing.Point(20, 190);
            this.txtComputedHash.Multiline = true;
            this.txtComputedHash.ReadOnly = true;
            this.txtComputedHash.ScrollBars                       =
System.Windows.Forms.ScrollBars.Vertical;
            this.txtComputedHash.Size            =            new
System.Drawing.Size(500, 60);
            this.txtComputedHash.TabIndex = 4;

            // txtExpectedHash
            this.txtExpectedHash.Location          =          new
System.Drawing.Point(20, 260);
            this.txtExpectedHash.Name = "txtExpectedHash";
            this.txtExpectedHash.Size            =            new
System.Drawing.Size(400, 23);
            this.txtExpectedHash.TabIndex = 5;

            // btnVerify
            this.btnVerify.Location             =             new
System.Drawing.Point(430, 260);
            this.btnVerify.Name = "btnVerify";
            this.btnVerify.Size   =   new   System.Drawing.Size(90,
23);
            this.btnVerify.TabIndex = 6;
            this.btnVerify.Text = "Перевірити";
            this.btnVerify.UseVisualStyleBackColor = true;
```

```
            this.btnVerify.Click              +=              new
System.EventHandler(this.btnVerify_Click);

            // lblResult
            this.lblResult.AutoSize = true;
            this.lblResult.Location             =             new
System.Drawing.Point(20, 295);
            this.lblResult.Name = "lblResult";
            this.lblResult.Size = new System.Drawing.Size(0, 15);
            this.lblResult.TabIndex = 7;

            // btnSwitchLanguage
            this.btnSwitchLanguage.Location         =         new
System.Drawing.Point(20, 330);
            this.btnSwitchLanguage.Name = "btnSwitchLanguage";
            this.btnSwitchLanguage.Size           =           new
System.Drawing.Size(140, 23);
            this.btnSwitchLanguage.TabIndex = 8;
            this.btnSwitchLanguage.Text = "Змінити мову";
            this.btnSwitchLanguage.UseVisualStyleBackColor      =
true;
            this.btnSwitchLanguage.Click          +=          new
System.EventHandler(this.btnSwitchLanguage_Click);

            // btnOpenLog
            this.btnOpenLog.Location             =             new
System.Drawing.Point(170, 330);
            this.btnOpenLog.Name = "btnOpenLog";
            this.btnOpenLog.Size  =  new  System.Drawing.Size(140,
23);
            this.btnOpenLog.TabIndex = 9;
            this.btnOpenLog.Text = "Відкрити лог";
            this.btnOpenLog.UseVisualStyleBackColor = true;
            this.btnOpenLog.Click              +=              new
System.EventHandler(this.btnOpenLog_Click);

            // MainForm
            this.AutoScaleDimensions            =             new
System.Drawing.SizeF(7F, 15F);
            this.AutoScaleMode                                  =
System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(544, 370);
            this.Controls.Add(this.dropPanel);
            this.Controls.Add(this.txtFilePath);
            this.Controls.Add(this.btnBrowse);
            this.Controls.Add(this.cmbAlgorithm);
            this.Controls.Add(this.txtComputedHash);
            this.Controls.Add(this.txtExpectedHash);
            this.Controls.Add(this.btnVerify);
            this.Controls.Add(this.lblResult);
            this.Controls.Add(this.btnSwitchLanguage);
            this.Controls.Add(this.btnOpenLog);
```

```
            this.FormBorderStyle                                    =
System.Windows.Forms.FormBorderStyle.FixedSingle;
            this.MaximizeBox = false;
            this.Name = "MainForm";
            this.Text = "Перевірка цілісності файлу";
            this.ResumeLayout(false);
            this.PerformLayout();
        }
    }
}
```

Додаток Б – Додавання пункту програми в контекстне меню Windows

Для реалізації можливості відправки файла в програму через контекстне меню Windows необхідно створити .reg-файл з наступним вмістом

```
Windows Registry Editor Version 5.00

[HKEY_CLASSES_ROOT\*\shell\FileIntegrityCheck]
@="🔍 Перевірити цілісність"

[HKEY_CLASSES_ROOT\*\shell\FileIntegrityCheck\command]
@="\"C:\\Program
Files\\FileIntegrityChecker\\FileIntegrityChecker.exe\" \"%1\""
```

та запустити його, підтвердивши внесення змін у реєстр. При цьому необхідно задати відповідний шлях до каталогу з програмою та ввести бажану назву пункту меню в зазначених ключах реєстру.