МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ
Циклова комісія (кафедра) комп’ютерної інженерії та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему

ЗАСОБИ СТВОРЕННЯ СЦЕНАРІЇВ ДЛЯ АВТОМАТИЗАЦІЇ РОБОТИ СИСТЕМИ

Виконав: студент групи 1П-21
Спеціальності
121 Інженерія програмного забезпечення
Максим ЗАЄЦЬ
Керівник:
Маргарита МЕДОЛИЗ

Черкаси 2025

АНОТАЦІЯ

Кваліфікаційна робота присвячена розробці програмного засобу для автоматизації резервного копіювання даних. Робота включає аналіз сучасних підходів до автоматизації задач системного адміністрування та порівняння основних інструментів для створення сценаріїв, таких як Bash, PowerShell та Python.

На основі проведеного аналізу обґрунтовано вибір Python як оптимального інструменту для розробки кросплатформеного рішення. Практична частина роботи полягає у реалізації двокомпонентної архітектури скрипта. Цей скрипт забезпечує автономне резервне копіювання файлів на локальний диск або в хмарне сховище Google Drive.

Метою роботи є підвищення надійності збереження даних шляхом створення гнучкого та простого у використанні програмного засобу. Для цього було реалізовано механізм автентифікації з Google Drive API та розроблено модуль для збереження конфігурації користувача у форматі JSON.

Розроблений програмний засіб має значне практичне значення як готове до використання рішення. Робота демонструє ефективне застосування мови Python та її бібліотек для вирішення прикладних задач автоматизації.

Ключові слова: РЕЗЕРВНЕ КОПІЮВАННЯ, АВТОМАТИЗАЦІЯ, PYTHON, СИСТЕМНЕ АДМІНІСТРУВАННЯ, GOOGLE DRIVE API, TKINTER, СЦЕНАРІЙ.

ANNOTATION

The qualification work is dedicated to the development of a software tool for automating data backup. The work includes an analysis of modern approaches to automating system administration tasks and a comparison of the main scripting tools such as Bash, PowerShell and Python.

Based on this analysis, the choice of Python as the optimal tool for developing a cross-platform solution is justified. The practical part of the work consists in the implementation of a two-component script architecture. This script provides offline backup of files to a local disc or to Google Drive cloud storage.

The aim of the work is to improve the reliability of data storage by creating a flexible and easy-to-use software tool. To do this, we implemented an authentication mechanism using the Google Drive API and developed a module for saving user configuration in JSON format.

The developed software tool has significant practical value as a ready-to-use solution. The work demonstrates the effective use of Python and its libraries for solving applied automation tasks.

Keywords: BACKUP, AUTOMATION, PYTHON, SYSTEM ADMINISTRATION, GOOGLE DRIVE API, TKINTER, SCRIPT.

ЗМІСТ

ВСТУП

У сучасній цифровій економіці, де інформація перетворилася на найцінніший актив, а технології є фундаментом будь-якого успішного підприємства, надійність, безпека та ефективність IT-інфраструктури стають критично важливими. Втрата чи компрометація корпоративних або персональних даних може мати руйнівні наслідки, виражені не тільки у прямих фінансових збитках, але й у довгострокових репутаційних втратах. Загрози для даних є різноманітними: від збоїв апаратного забезпечення та помилок програмного забезпечення до кібератак, таких як програми-вимагачі (ransomware), що стали справжньою епідемією останніх років, та банального людського фактора.

В цих умовах, створення надійної системи резервного копіювання даних перестає бути просто рекомендованою практикою і перетворюється на абсолютно необхідний елемент стратегії інформаційної безпеки. Однак ручне виконання резервного копіювання є неефективним, схильним до помилок та не гарантує регулярність, необхідну для захисту від сучасних загроз. Саме тому автоматизація цього процесу набуває особливої актуальності. Розробка гнучкого, доступного та простого у використанні інструменту, який дозволяє автоматизувати резервне копіювання як на локальні носії, так і в хмарні сховища, є важливою та практично значущою задачею як для окремих користувачів, так і для малого бізнесу, що не завжди має ресурси на дорогі комерційні рішення.

Об'єктом дослідження є процес автоматизації задач системного адміністрування, зокрема, процес створення резервних копій даних користувача.

Предметом дослідження є мови сценаріїв та методи їх застосування для розробки програмного засобу резервного копіювання з графічним інтерфейсом користувача та інтеграцією з хмарними сервісами на прикладі Google Drive API.

Метою кваліфікаційної роботи є підвищення надійності збереження даних користувача шляхом розробки та реалізації програмного засобу для

автоматизованого резервного копіювання файлів з гнучкими налаштуваннями місця збереження.

Для досягнення поставленої мети було визначено наступні завдання:

- провести аналіз теоретичних основ автоматизації в системному адмініструванні;
- порівняти основні мови сценаріїв (Bash, PowerShell, Python) та обґрунтувати вибір інструменту для реалізації проєкту;
- спроєктувати архітектуру програмного засобу, що передбачає розділення модуля конфігурації та виконавчого модуля;
- розробити модуль конфігурації з графічним інтерфейсом користувача на базі бібліотеки Tkinter;
- реалізувати виконавчий модуль, що забезпечує логіку локального копіювання та завантаження файлів на Google Drive;
- реалізувати механізм автентифікації з Google Drive API за допомогою протоколу OAuth 2.0;
- провести тестування розробленого програмного засобу для перевірки його функціональності та стабільності.

Практичне значення отриманих результатів полягає в тому розроблений програмний засіб є готовим до використання інструментом, що надає простий та безкоштовний спосіб автоматизації резервного копіювання для широкого кола користувачів, включаючи індивідуальних користувачів та малий бізнес. Модульна архітектура програми дозволяє легко розширювати її функціонал, наприклад, додаючи підтримку інших хмарних сервісів.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ АВТОМАТИЗАЦІЇ В СИСТЕМНОМУ АДМІНІСТРУВАННІ

1.1 Поняття системного адміністрування та його задач

Системне адміністрування є ключовою дисципліною в галузі інформаційних технологій, що відповідає за надійне, безпечне та ефективне функціонування IT-інфраструктури організації. Воно охоплює широкий спектр завдань, від підтримки окремих комп'ютерів до управління складними розподіленими системами, що включають сервери, мережеве обладнання та хмарні ресурси. Основна мета системного адміністрування — забезпечити безперебійний доступ користувачів до технологічних ресурсів, необхідних для виконання їхніх робочих завдань, а також захист і цілісність корпоративних даних. Сьогодні інформація є одним з найцінніших активів, її втрата може призвести до значних втрат для будь-якої організації. Тому роль системного адміністратора виходить за межі простої технічної підтримки і стає стратегічно важливою для бізнесу [1].

Діяльність системного адміністратора можна умовно поділити на кілька основних напрямків, кожен з яких вимагає специфічних знань та навичок.

Управління інфраструктурою та обладнанням - це фундаментальний рівень, що включає встановлення, налаштування та підтримку фізичного та віртуального обладнання. Сюди належить монтаж серверів у стійки, підключення мережевих кабелів, конфігурація комутаторів та маршрутизаторів, а також розгортання та адміністрування віртуальних машин на гіпервізорах (VMware, Hyper-V, KVM). Адміністратор повинен розуміти принципи роботи апаратного забезпечення, щоб правильно планувати навантаження та забезпечувати відмовостійкість. З розвитком хмарних технологій цей напрямок також включає управління віртуальними ресурсами в хмарі (IaaS - Infrastructure as a Service), такими як віртуальні машини, дискові сховища та мережі в

середовищах AWS, Microsoft Azure або Google Cloud. Це вимагає нових навичок, пов'язаних з управлінням хмарними бюджетами та оптимізацією витрат.

Адміністрування операційних систем - системний адміністратор відповідає за встановлення, конфігурацію та підтримку операційних систем на серверах та робочих станціях. Це включає управління оновленнями безпеки (patch management), налаштування системних параметрів для оптимізації продуктивності (performance tuning), моніторинг використання ресурсів (процесора, пам'яті, дискового простору) та вирішення проблем, пов'язаних з роботою ОС. Знання як Windows Server, так і дистрибутивів Linux (наприклад, Ubuntu Server, CentOS, Red Hat Enterprise Linux) є частою вимогою, оскільки сучасні інфраструктури є гетерогенними. Важливою частиною цього процесу є "зміцнення" (hardening) системи – відключення непотрібних сервісів та налаштування параметрів безпеки для мінімізації поверхні атаки.

Управління користувачами та правами доступу – це забезпечення того, щоб кожен користувач мав доступ лише до тих ресурсів, які необхідні йому для роботи, є критично важливим для безпеки. Системний адміністратор створює та видаляє облікові записи, керує групами користувачів, налаштовує політики доступу до файлів, папок та мережевих ресурсів, часто з використанням служб каталогів, таких як Active Directory або LDAP. Це вимагає ретельного планування та дотримання принципу найменших привілеїв (Principle of Least Privilege), згідно з яким користувачеві надається мінімально необхідний рівень доступу для виконання його завдань. Також сюди входить впровадження політик паролів та багатофакторної автентифікації, що значно підвищує рівень безпеки.

Моніторинг та забезпечення безперебійної роботи - проактивний підхід до адміністрування передбачає постійний моніторинг стану системи. Це включає відстеження навантаження на сервери, доступності мережевих сервісів, стану жорстких дисків та інших критичних компонентів. Використання систем моніторингу дозволяє завчасно виявляти потенційні проблеми (наприклад, закінчення дискового простору чи аномальне зростання навантаження на процесор) та запобігати збоям, що могли б призвести до простою сервісів.

Сучасний підхід також включає концепцію спостережуваності (observability), яка, на відміну від простого моніторингу, який показує, що система не працює, намагається пояснити, чому вона не працює, збираючи та аналізуючи метрики, логи та трасування для глибинного розуміння поведінки системи.

Забезпечення інформаційної безпеки - адміністратор відіграє ключову роль у захисті інфраструктури від зовнішніх та внутрішніх загроз. Його обов'язки включають налаштування міжмережевих екранів (firewalls), встановлення та оновлення антивірусного програмного забезпечення, виявлення та запобігання вторгненням (Intrusion Detection/Prevention Systems), а також впровадження політик безпеки, таких як політики паролів, шифрування даних та управління вразливостями. Це вимагає постійного відстеження нових загроз, таких як програми-вимагачі, фішинг та інші види атак, і своєчасного оновлення захисних механізмів [3].

Резервне копіювання та відновлення даних - одна з найважливіших задач системного адміністрування. Втрата даних через технічний збій, кібератаку, людську помилку чи природний катаклізм може мати катастрофічні наслідки для бізнесу. Тому системний адміністратор повинен розробити та впровадити надійну стратегію резервного копіювання, яка включає регулярне створення копій критично важливих даних та періодичне тестування процедури відновлення, щоб переконатися в її працездатності. Стратегія має визначати, які дані копіювати, як часто RPO - Recovery Point Objective, тобто точка відновлення, що визначає максимальний допустимий обсяг втрати даних, та як швидко їх можна відновити RTO - Recovery Time Objective, тобто час відновлення, що визначає максимальний допустимий час простою. Також стратегія повинна включати правило "3-2-1": мати щонайменше три копії даних, на двох різних типах носіїв, і одна з копій має зберігатися поза основним місцем роботи (off-site), наприклад, у хмарному сховищі, для захисту від локальних катастроф.

У контексті цих задач, скрипти та автоматизація виступають як потужний інструмент, що дозволяє перетворити рутинні, повторювані операції на

ефективні, надійні та автоматизовані процеси, підвищуючи загальну продуктивність та стабільність IT-інфраструктури.

1.2 Роль автоматизації в забезпеченні ефективності адміністрування

Автоматизація в системному адмініструванні – це процес використання програмних засобів та скриптів для виконання завдань, які раніше виконувалися вручну. Роль автоматизації важко переоцінити, оскільки вона кардинально змінює підхід до управління IT-інфраструктурою, перетворюючи його з реактивного (реагування на проблеми, що вже виникли) на проактивний (запобігання проблемам) та підвищуючи загальну ефективність. В сучасних умовах, коли складність та масштаби IT-систем стрімко зростають, ручне адміністрування стає не просто неефективним, а практично неможливим (див. рис. 1.1).

Впровадження автоматизованих рішень дозволяє досягти значних переваг. По-перше, мінімізуються помилки, оскільки автоматизація рутинних завдань, таких як копіювання файлів за певним шляхом чи конфігурування параметрів, суттєво знижує ризик, обумовлений людським фактором. Машина виконує заданий алгоритм точно і послідовно, що є недосяжним при ручному виконанні, особливо при великій кількості повторень. Наприклад, помилка в одному символі при ручному введенні команди може призвести до збою в роботі критичного сервісу. По-друге, досягається оптимізація часу: операції, що регулярно повторюються, виконуються значно швидше, вивільняючи час адміністратора для вирішення більш складних та пріоритетних завдань, таких як аналіз безпеки чи планування розвитку інфраструктури. По-третє, забезпечується відтворюваність – автоматизовані процеси гарантують, що завдання щоразу виконуватимуться однаково за заданим алгоритмом. Це критично важливо для стабільності та передбачуваності системи, особливо при розгортанні нових серверів або відновленні після збоїв. Нарешті, підвищується надійність, оскільки автоматичне виконання критично важливих завдань, таких

як резервне копіювання гарантує їх регулярність та своєчасність, на відміну від ручних процесів, які можуть бути пропущені або виконані з запізненням, що напряму впливає на дотримання угод про рівень обслуговування (SLA) [3].

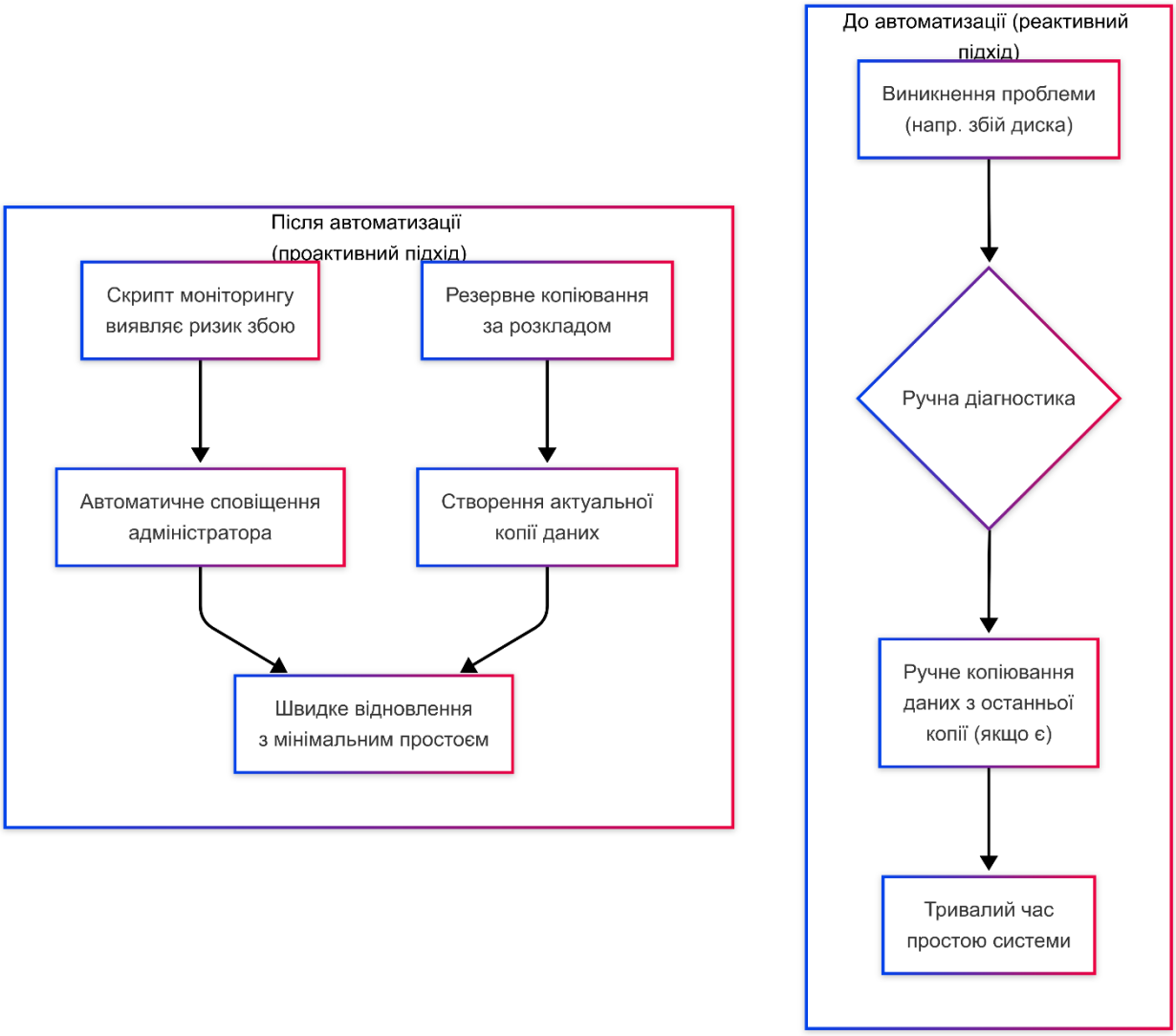

Рисунок 1.1 – Вплив автоматизації на роботу системного адміністратора

Одним з найяскравіших прикладів необхідності автоматизації є резервне копіювання даних. Втрата інформації через технічні збої, кібератаки, помилки програмного забезпечення чи людський фактор може мати катастрофічні наслідки. Ручне копіювання є неефективним та ненадійним. Автоматизація цього процесу дозволяє гарантувати регулярне створення копій, знизити ризики втрати даних та забезпечити можливість їх швидкого відновлення. Наприклад, замість того, щоб щовечора вручну копіювати базу даних, адміністратор може написати

скрипт, який буде робити це автоматично о 2-й годині ночі, перевіряти цілісність копії (наприклад, порівнюючи контрольні суми), стискати архів для економії місця, завантажувати його в хмарне сховище та надсилати звіт на електронну пошту. Такий підхід не тільки виключає можливість "забути" зробити копію, але й дозволяє реалізувати більш складні стратегії, наприклад, ротацію резервних копій (збереження денних, тижневих та місячних копій) для можливості відновлення на будь-яку дату.

1.3 Огляд мов сценаріїв

Для написання скриптів автоматизації використовується широкий спектр мов програмування, які умовно можна класифікувати за їх основним призначенням. Вибір конкретної мови залежить від операційної системи, специфіки завдання та навичок адміністратора.

До першої групи належать консольні (командні) оболонки, тісно інтегровані з операційною системою. Найвідомішими представниками є Bash для Unix-подібних систем: Linux, macOS та PowerShell для Windows. Ці мови ідеально підходять для маніпуляцій з файловою системою та управління процесами операційних систем [4].

Bash є потужним інструментом для роботи з файлами, текстом та керування процесами, його філософія базується на роботі з текстовими потоками, що робить його незамінним для аналізу логів та обробки текстових даних за допомогою таких утиліт як sed, awk та grep. PowerShell, розроблена Microsoft, натомість оперує об'єктами .NET, що надає широкі можливості для керування компонентами Windows, наприклад: Active Directory, Exchange Server та іншими продуктами Microsoft, дозволяючи працювати з їхніми властивостями та методами напряму. Це робить його надзвичайно ефективним для глибокого адміністрування Windows-середовищ [5].

Друга група – це мови загального призначення, що завдяки гнучкості та наявності бібліотек чудово підходять для складних скриптів; яскравими

представниками тут є Python, Perl та Ruby. Python здобув величезну популярність в автоматизації завдяки своєму простому синтаксису, кросплатформеності та величезній екосистемі бібліотек, що дозволяють вирішувати практично будь-які завдання, від роботи з мережевими протоколами, приклад, бібліотека socket, до взаємодії з хмарними API, наприклад, бібліотека boto3 для AWS, google-api-python-client для Google Cloud. Perl історично був дуже популярним для обробки тексту та CGI-скриптів, але його популярність дещо знизилась з ростом Python, хоча він і досі використовується в деяких системах. Ruby, як і Python, є потужною об'єктно-орієнтованою мовою, що часто використовується в інфраструктурних інструментах (наприклад, Chef та Puppet). Ці мови дозволяють створювати більш структурований та підтримуваний код порівняно з командними оболонками, що є важливим для великих та складних скриптів [6].

Третя категорія – це вбудовувані мови, призначені для розширення функціоналу інших програм, наприклад, JavaScript у веб-браузерах або VBA в додатках Microsoft Office. Вони зазвичай використовуються для автоматизації завдань у межах конкретного застосунку і рідше застосовуються для загального системного адміністрування, хоча Node.js (серверна платформа для JavaScript) іноді використовується для написання інструментів автоматизації.

1.4 Класифікація інструментів автоматизації

Окрім мов сценаріїв, існує кілька класів інструментів, що використовуються для автоматизації в системному адмініструванні. Їх можна класифікувати за рівнем абстракції та підходом до вирішення завдань (див. рис. 1.2).

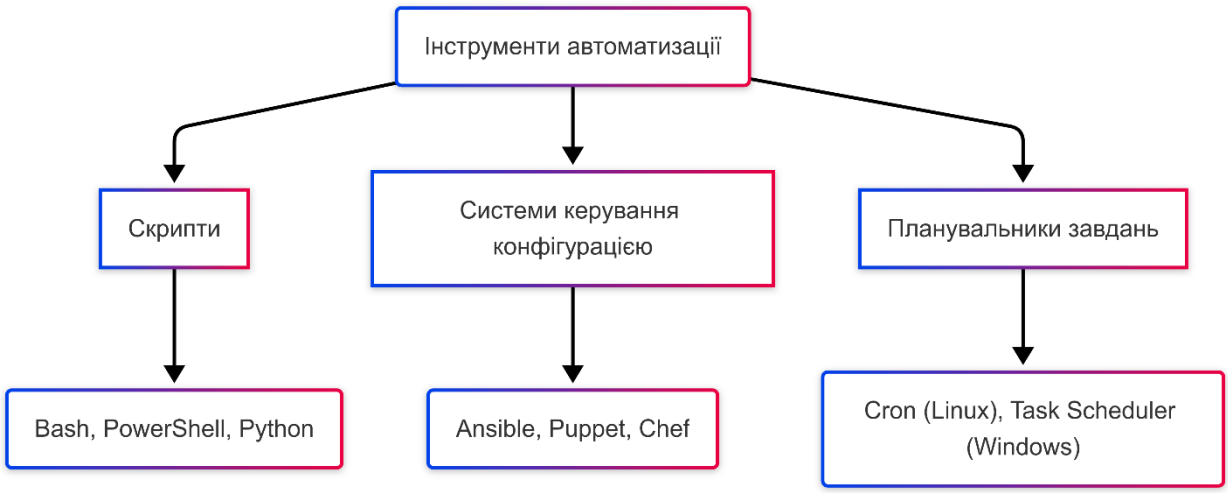

Рисунок 1.2 – Класифікація інструментів автоматизації

Як було зазначено, це найпростіший та найгнучкіший інструмент, що являє собою послідовність команд. Вони ідеально підходять для вирішення специфічних, унікальних завдань, як-от резервне копіювання за унікально заданими правилами, міграція даних між системами, або виконання послідовності дій для розгортання простого додатку. Їх головна перевага – повний контроль над логікою виконання (імперативний підхід, що описує, як виконати завдання). Це дозволяє реалізувати будь-який, навіть найскладніший, алгоритм, не обмежуючись можливостями готових фреймворків. Однак, при управлінні великою кількістю систем, підтримка та масштабування імперативних скриптів може стати складним завданням.

Системи керування конфігурацією (Configuration Management Tools) – це більш складні системи, наприклад, Ansible, Puppet, Chef, SaltStack, призначені для декларативного опису бажаного стану серверів та автоматичного приведення їх до цього стану. Замість того, щоб писати послідовність команд (імперативний підхід), адміністратор описує, який стан має бути досягнутий: який пакет має бути встановлений, який сервіс має бути запущений, і який вміст має бути у конфігураційному файлі. Інструмент сам визначає, які кроки потрібно виконати. Цей підхід, відомий як "Infrastructure as Code" (IaC), є стандартом для керування великими парками серверів, оскільки він забезпечує ідемпотентність – повторне

застосування конфігурації не змінює стан системи, якщо вона вже відповідає опису, та легкість масштабування. Ці інструменти ідеально підходять для підтримки консистентності конфігурацій на десятках або сотнях машин [3].

Планувальники завдань (Task Schedulers) – це системні утиліти, що дозволяють налаштувати автоматичний запуск скриптів або програм за певним розкладом або при настанні певної події. Вони є сполучною ланкою, яка перетворює просто скрипт на повністю автоматизований процес. До них належать стандартний для Unix-систем Cron, який дозволяє налаштовувати запуск з точністю до хвилини, використовуючи спеціальний синтаксис у файлі crontab. Вбудований в ОС Windows Task Scheduler має графічний інтерфейс та дозволяє налаштовувати більш складні тригери, такі як запуск при вході користувача, при бездіяльності системи або при записі певного повідомлення в системний журнал подій [5].

У даній кваліфікаційній роботі буде розглядатися створення скрипту на мові Python, який є імперативним інструментом для вирішення специфічного завдання резервного копіювання. В подальшому цей скрипт може бути інтегрований з планувальником завдань операційної системи для забезпечення повної автоматизації процесу. Такий підхід є оптимальним для вирішення конкретного завдання, як ось резервне копіювання файлів користувача, оскільки він забезпечує необхідну гнучкість, не вимагаючи при цьому розгортання складної системи керування конфігурацією, яка була б надлишковою для даної задачі.

РОЗДІЛ 2
АНАЛІЗ ТА ПОРІВНЯННЯ ЗАСОБІВ СТВОРЕННЯ СЦЕНАРІЇВ

2.1 Bash-скрипти в середовищі Linux: особливості та приклади

Bash (Bourne Again Shell) є не просто командною оболонкою, а повноцінним середовищем для написання сценаріїв, яке стало стандартом для переважної більшості Unix-подібних операційних систем, включаючи всі популярні дистрибутиви Linux та macOS. Його роль в адмініструванні цих систем важко переоцінити, оскільки він надає прямий доступ до ядра системи та всіх її утиліт, дозволяючи автоматизувати практично будь-яке завдання. Розуміння принципів роботи Bash є фундаментальною навичкою для будь-якого системного адміністратора, що працює з Linux [4].

В основі роботи Bash та всього Unix-середовища лежить філософія "все є файл". Це означає, що пристрої, сокети, процеси та, звичайно, самі файли представлені у файловій системі як об'єкти, з якими можна взаємодіяти за допомогою стандартних утиліт. Ця концепція, у поєднанні з механізмом стандартних потоків вводу/виводу, створює надзвичайно потужне середовище для автоматизації.

Основною особливістю Bash є його тісна інтеграція з операційною системою. Скрипт на Bash може безпосередньо викликати будь-які системні утиліти, такі як cp (копіювання), mv (переміщення), rm (видалення), rsync (синхронізація файлів), а також потужні інструменти для роботи з текстом, як-от grep (пошук за патерном), sed (потоковий текстовий редактор) та awk (мова обробки патернів).

Іншою фундаментальною концепцією є використання конвеєрів які дозволяють передавати стандартний вивід (stdout) однієї команди на стандартний ввід (stdin) іншої. Це дає змогу будувати складні ланцюжки команд для обробки даних "на льоту". Наприклад, команда:

```
ps aux | grep 'httpd' | wc -l
```

послідовно виводить список всіх процесів, фільтрує ті, що містять 'httpd', та підраховує їх кількість, і все це однією строкою.

Також важливим є механізм перенаправлення вводу/виводу. Оператори > та >> дозволяють перенаправити вивід команди у файл перезаписуючи його або доповнюючи, а оператор < дозволяє передати вміст файлу на вхід команди. Це є основою для роботи з файлами та створення логів.

Скрипт на Bash зазвичай починається з так званого "шебангу" (#!/bin/bash), який вказує системі, який інтерпретатор слід використовувати для виконання файлу.

Синтаксис Bash включає всі необхідні конструкції для процедурного програмування:

– Змінні визначаються простим присвоєнням, наприклад, MY_VAR="value", і викликаються зі знаком долара, наприклад, $MY_VAR. Важливо пам'ятати, що Bash не є строго типізованою мовою, і всі змінні за замовчуванням є рядками.

– Умовні конструкції реалізуються за допомогою if-elif-else-fi. Умови перевіряються за допомогою команди test (або її синоніму [ ]) для перевірки властивостей файлів, порівняння рядків та чисел.

– Bash підтримує цикли for , while та until, що дозволяє реалізовувати складні алгоритми обробки даних.

– Функції дозволяють групувати команди у логічні блоки для повторного використання, що покращує структуру та читабельність скрипту.

Незважаючи на потужність у своєму середовищі, Bash має суттєві недоліки, що обмежують його застосування. Його синтаксис для роботи зі структурами даних, окрім простих масивів, обробки помилок та виконання комплексних логічних операцій може бути досить складним та неінтуїтивним. Робота з веб-API вимагає використання зовнішніх утиліт на кшталт curl або wget та ручного парсингу відповідей, часто у форматі JSON або XML, що є значно

складнішим порівняно з мовами загального призначення, які мають для цього вбудовані бібліотеки.

Найголовнішим недоліком є відсутність нативної підтримки у Windows, що робить Bash непридатним для створення універсальних, кросплатформених рішень. Хоча існують способи запустити Bash на Windows, наприклад, через Windows Subsystem for Linux (WSL) або Cygwin, це потребує додаткових налаштувань і не є нативним досвідом, що може створювати проблеми із сумісністю та доступом до специфічних функцій Windows.

Таким чином, Bash є незамінним інструментом для швидкої та ефективної автоматизації в середовищі Linux, але для створення складних, кросплатформених додатків з графічним інтерфейсом та інтеграцією з веб-сервісами доцільніше обирати мови загального призначення, такі як Python.

2.2 PowerShell у Windows: функціональність і структура сценаріїв

PowerShell, розроблений корпорацією Microsoft, є відповіддю на командні оболонки Unix і на сьогодні є стандартним та найпотужнішим інструментом для автоматизації й адміністрування в екосистемі Windows. Він був створений для подолання обмежень традиційної командної оболонки Windows (cmd.exe) і надання адміністраторам сучасного, гнучкого та потужного середовища для керування всіма аспектами операційної системи та серверних продуктів Microsoft [5].

Головна відмінність та перевага PowerShell полягає в його об'єктній моделі. На відміну від Bash, який передає між командами текстові потоки, PowerShell оперує повноцінними об'єктами платформи .NET. Це означає, що результат виконання однієї команди, яка називається командлет (cmdlet), є не просто текстом, а структурованим об'єктом зі своїми властивостями та методами. Ці об'єкти передаються далі по конвеєру (|) до наступного командлета, який може безпосередньо звертатися до цих властивостей. Такий підхід усуває необхідність складного та схильного до помилок парсингу тексту, як це часто доводиться

робити в Bash за допомогою grep, awk, sed. Це робить скрипти більш надійними, читабельними та легкими в підтримці.

Командлети мають стандартизовану структуру іменування "Дієслово-Іменник", наприклад, Get-Process, Set-Item, Start-Service, що робить їх імена інтуїтивно зрозумілими та легко прогнозованими.

Завдяки глибокій інтеграції з .NET Framework та Windows Management Instrumentation (WMI), PowerShell надає повний програмний доступ до будь-якого компонента операційної системи Windows. Адміністратор може програмно керувати службами, процесами, реєстром, мережевими налаштуваннями, користувачами Active Directory, поштовими скриньками Exchange та багатьма іншими елементами.

Ще однією важливою концепцією є провайдери (Providers). PowerShell надає уніфікований спосіб доступу до різних сховищ даних, представляючи їх у вигляді ієрархічної структури, схожої на файлову систему. Наприклад, існують провайдери для файлової системи (FileSystem), системного реєстру (Registry), сховища сертифікатів (Certificate) та інших. Це дозволяє використовувати однакові командлети, наприклад, Get-Item, Set-Item, Get-ChildItem для навігації та маніпуляції даними в абсолютно різних сховищах, що значно спрощує адміністрування.

Сценарії PowerShell є текстовими файлами з розширенням .ps1. Вони підтримують всі стандартні програмні конструкції: змінні (з префіксом $), масиви, хеш-таблиці, умовні оператори (if-elseif-else), цикли (foreach, for, while) та функції.

Хоча PowerShell Core, зараз просто PowerShell 7+, є кросплатформеним і може працювати на Linux та macOS, його основна сила та найбільша кількість модулів, особливо для роботи з продуктами Microsoft, залишаються орієнтованими на Windows. Використання PowerShell для адміністрування Linux-систем є можливим, але часто менш зручним та ефективним порівняно з нативними інструментами, як-от Bash. Його синтаксис, хоча і є послідовним,

може здаватися надто "багатослівним" для адміністраторів, що звикли до лаконічності Bash.

Таким чином, PowerShell є незамінним інструментом для глибокої автоматизації в Windows-орієнтованих інфраструктурах, але його універсальність для гетерогенних середовищ є обмеженою.

2.3 Python як універсальний інструмент для кросплатформної автоматизації

На фоні спеціалізованих інструментів, таких як Bash та PowerShell, що тісно пов'язані з певними операційними системами, Python виступає як універсальне рішення, яке поєднує простоту, потужність та, що найважливіше, не прив'язане до конкретної платформи. Саме ці якості стали вирішальними при виборі інструменту для реалізації даної кваліфікаційної роботи, оскільки задача розробки гнучкого засобу резервного копіювання вимагала поєднання роботи з файловою системою, створення графічного інтерфейсу та інтеграції з хмарними сервісами.

Кросплатформеність є фундаментальною перевагою Python у контексті системного адміністрування. Скрипт, написаний на Python, буде працювати практично однаково на Windows, Linux та macOS з мінімальними змінами або взагалі без них. Це досягається завдяки тому, що стандартна бібліотека Python містить модулі, що абстрагують відмінності операційних систем. Наприклад, модуль os.path надає функції для роботи зі шляхами файлів (os.path.join, os.path.basename), які автоматично використовують правильні роздільники (\ для Windows, / для Unix), роблячи код портативним. Аналогічно, модуль shutil, використаний у даній роботі для локального копіювання, надає високорівневі функції, що працюють однаково на всіх платформах. Це дозволяє розробляти один інструмент, який може бути розгорнутий у гетерогенному середовищі без необхідності переписувати логіку під кожну ОС.

Python має низький поріг входження, що дозволяє швидко розробляти та тестувати функціональність. Його синтаксис, який часто порівнюють з псевдокодом, є чистим та лаконічним, що спрощує підтримку та модифікацію скриптів у довгостроковій перспективі. Це особливо важливо для інструментів автоматизації, які можуть використовуватися та змінюватися різними людьми протягом тривалого часу. На відміну від складних конструкцій Bash або багатослівних командлетів PowerShell, код на Python часто є більш інтуїтивно зрозумілим, що зменшує ймовірність помилок.

Філософія Python "batteries included" ("батарейки в комплекті") означає, що його стандартна бібліотека є надзвичайно багатою і надає модулі для вирішення більшості стандартних завдань. У контексті даної роботи були використані[6]:

Tkinter - стандартна бібліотека для створення графічного інтерфейсу користувача. Хоча існують більш потужні фреймворки, Tkinter порівняно з PyQt, Kivy, wxPython не потребує встановлення жодних зовнішніх залежностей, що робить розповсюдження програми максимально простим. Для задачі створення простого вікна налаштувань його функціоналу є цілком достатньо.

Стандартні модулі os, shutil, json надають весь необхідний функціонал для роботи з файловою системою, копіювання файлів та обробки конфігураційних файлів у форматі JSON.

Окрім стандартної бібліотеки, існує центральний репозиторій сторонніх пакетів PyPI, наприклад Python Package Index, що налічує сотні тисяч бібліотек для будь-яких потреб. Саме завдяки цьому для даної роботи була обрана бібліотека PyDrive. Вона є високорівневою обгорткою над Google Drive API, яка інкапсулює всю складну логіку автентифікації через OAuth 2.0, включаючи обробку токенів, та надає простий і зрозумілий об'єктний інтерфейс для завантаження файлів. Це дозволило реалізувати функціонал хмарного резервного копіювання з мінімальними зусиллями, не вдаючись у деталі HTTP-запитів та протоколу OAuth 2.0 [7].

Сучасна автоматизація часто вимагає взаємодії з веб-сервісами та API. Python є фактично стандартом для такої роботи. Його вбудовані структури даних,

такі як словники та списки ідеально відповідають структурі формату JSON, який є основним форматом обміну даними у більшості сучасних REST API. Вбудований модуль json дозволяє легко перетворювати JSON-рядки у Python-об'єкти і навпаки. Це робить процес роботи з API, включаючи відправку запитів та парсинг відповідей, значно простішим порівняно з Bash, де для цього доводиться використовувати зовнішні утиліти та інструменти для обробки тексту [8].

Таким чином, Python дозволяє реалізувати всю необхідну логіку – від створення кросплатформеного GUI до взаємодії з хмарним API – в рамках однієї мови та екосистеми. Це усуває необхідність комбінувати різні інструменти, спрощує процес розробки та робить кінцеве рішення більш надійним, підтримуваним та масштабованим. Саме тому він був обраний як оптимальний інструмент для реалізації програмного засобу резервного копіювання в рамках даної кваліфікаційної роботи.

2.4 Порівняльна характеристика інструментів

Вибір інструменту для автоматизації є ключовим рішенням, яке впливає на швидкість розробки, надійність, масштабованість та можливість підтримки розробленого рішення. Як було показано в попередніх підрозділах, кожна мова сценаріїв має свої сильні та слабкі сторони, а також свою нішу застосування (див. табл. 2.1). Для того, щоб зробити обґрунтований висновок щодо вибору Python для даної кваліфікаційної роботи, необхідно провести детальний порівняльний аналіз Bash, PowerShell та Python за низкою критеріїв, що є критично важливими для поставленої задачі.

Це один з найважливіших критеріїв для сучасних IT-середовищ, які часто є гетерогенними. Bash є нативним та ідеальним інструментом для Linux, але його використання на Windows є ускладненим. Аналогічно, PowerShell, незважаючи на кросплатформеність нової версії, залишається в першу чергу інструментом для Windows. Python, натомість, є повністю кросплатформеним. Це означає, що

розроблений програмний засіб для резервного копіювання може бути легко адаптований для роботи на будь-якій основній ОС без необхідності переписувати код, що є значною перевагою[4, 5].

Таблиця 2.1 – Порівняльна характеристика засобів створення сценаріїв

Критерій	Bash	PowerShell	Python
Основна платформа	Linux, macOS (Unix-подібні)	Windows	Кросплатформений (Windows, Linux, macOS)
Синтаксис	Компактний, але може бути складним	Вербозний, структурований, об'єктний	Простий, чистий, легко читабельний
Робота з API/Веб	Обмежена (потребує curl, wget)	Добра, інтеграція з .NET	Відмінна (бібліотеки requests, PyDrive)
Графічний інтерфейс (GUI)	Складна (потребує спец. утиліт)	Можлива (через .NET)	Легка (бібліотеки Tkinter, PyQt, Kivy)
Основні переваги	Швидкість для простих файлових операцій, нативність для Linux	Потужне керування Windows, об'єктна модель	Універсальність, кросплатформеність, велика кількість бібліотек
Основні недоліки	Обмежена функціональність поза Unix, складність для великих проєктів	Орієнтованість на Windows, менш поширений на інших ОС	Нижча швидкість виконання порівняно з компільованими мовами
Сфери застосування	Адміністрування Linux, обробка тексту	Адміністрування Windows, керування продуктами Microsoft	Веб-розробка, аналіз даних, машинне навчання, кросплатформена автоматизація

Синтаксис Bash, хоч і потужний, є досить архаїчним і може бути складним для розуміння та відлагодження, особливо для великих скриптів. PowerShell пропонує більш структурований, але водночас багатослівний підхід. Python вирізняється своїм чистим, мінімалістичним та інтуїтивно зрозумілим синтаксисом. Це не тільки прискорює процес розробки, але й значно спрощує подальшу підтримку та модифікацію коду, що є важливим фактором при виборі [5].

Сучасна автоматизація вимагає інтеграції з різноманітними сервісами, особливо хмарними. Для реалізації резервного копіювання на Google Drive необхідно взаємодіяти з його API. У Python для цього існують готові,

високорівневі бібліотеки, як-от PyDrive. Реалізація аналогічного функціоналу в Bash або PowerShell вимагала б значно більше зусиль.

Однією з ключових вимог до розробленого засобу була наявність зручного інтерфейсу для налаштування. Python має вбудовану бібліотеку Tkinter, яка дозволяє легко і швидко створювати прості графічні інтерфейси, що працюють на всіх платформах. Створення GUI за допомогою Bash є вкрай нетривіальним завданням і зазвичай вимагає використання зовнішніх інструментів. У PowerShell створення GUI можливе через Windows Forms або WPF, але це знову ж таки прив'язує рішення до платформи Windows [7].

Проведений аналіз показує, що хоча Bash та PowerShell є потужними інструментами у своїх нішах адміністрування Linux та Windows відповідно, вони мають суттєві обмеження, коли мова йде про створення кросплатформених додатків зі складною логікою, графічним інтерфейсом та інтеграцією з веб-API. Python, завдяки своїй універсальності, великій екосистемі бібліотек, простоті синтаксису та кросплатформеності, виявився найбільш збалансованим та оптимальним вибором для вирішення задач, поставлених у даній кваліфікаційній роботі. Він дозволив реалізувати весь необхідний функціонал в рамках єдиної технологічної платформи, що забезпечило швидкість розробки та високу якість кінцевого продукту.

РОЗДІЛ 3
ПРАКТИЧНА РЕАЛІЗАЦІЯ СЦЕНАРІЇВ АВТОМАТИЗАЦІЇ

3.1 Постановка задач для автоматизації

Переходячи від теоретичного аналізу до практичної реалізації, була сформульована головна задача даної кваліфікаційної роботи: створення програмного засобу для автоматизації процесу резервного копіювання, який би задовольняв вимоги гнучкості, надійності та простоти використання для кінцевого користувача. На основі аналізу, проведеного в попередніх розділах, ця глобальна задача була декомпозована на низку конкретних технічних завдань, що визначають функціональні та нефункціональні вимоги до розроблюваного програмного забезпечення.

Першочерговою задачею була реалізація інтуїтивно зрозумілого інструменту, за допомогою якого користувач міг би легко налаштовувати параметри резервного копіювання. Це завдання включало наступні вимоги:

- Реалізувати можливість вибору одного або декількох файлів для резервного копіювання через стандартний системний діалог.
- Надати користувачеві вибір місця призначення резервних копій: локальний диск або хмарне сховище Google Drive.
- Забезпечити механізм для вказання конкретних параметрів призначення: шляху до локальної папки для локального збереження та ідентифікатора (ID) цільової папки для збереження на Google Drive.
- Реалізувати функцію збереження всієї введеної конфігурації у зовнішній файл для подальшого використання.
- Забезпечити завантаження раніше збережених налаштувань при повторному запуску програми для зручності редагування.

Другою ключовою задачею було створення окремого скрипта, здатного працювати автономно, без втручання користувача. Вимоги до цього модуля були наступними:

– Скрипт повинен зчитувати збережені налаштування з конфігураційного файлу.

– На основі цих налаштувань скрипт має автоматично виконувати процес копіювання файлів на локальний диск або завантаження в хмарне сховище.

– При роботі з Google Drive скрипт повинен підтримувати процес автентифікації користувача через протокол OAuth 2.0, забезпечуючи безпечний доступ до хмарного сховища.

– Скрипт повинен надавати інформативний звіт про свою роботу у вигляді повідомлень в консолі, включаючи інформацію про оброблені файли, успішне завершення операцій та можливі помилки.

Важливою архітектурною вимогою було чітке розділення функціоналу на два незалежні компоненти: графічний конфігуратор та консольний виконувач.

Така архітектура дозволяє користувачеві один раз налаштувати програму через GUI, а потім налаштувати автоматичний запуск лише виконавчого скрипта, який не потребує графічного середовища для своєї роботи.

Для спрощення використання програми кінцевим користувачем, який може не мати встановленого середовища Python [3], необхідно було передбачити можливість компіляції виконавчого скрипта у самостійний виконуваний файл. Це, в свою чергу, дозволило б легко налаштувати автоматичний запуск програми за допомогою стандартних засобів операційної системи, таких як папка "Автозавантаження" або "Планувальник завдань".

Таким чином, постановка задач охоплювала повний цикл розробки програмного продукту: від проєктування інтерфейсу та реалізації основної логіки до забезпечення можливості його зручного розгортання та інтеграції в робоче середовище користувача.

3.2 Розробка та опис скриптів для резервного копіювання

Для вирішення поставлених задач було розроблено два основні скрипти на мові Python - це config_gui.py, що реалізує модуль конфігурації, та runner.py, що є виконавчим модулем.

Розроблений програмний засіб має двокомпонентну архітектуру, яка є ключовою для досягнення гнучкості та можливості автоматизації. Ця архітектура чітко розділяє логіку налаштування, яка вимагає взаємодії з користувачем, та логіку безпосереднього виконання резервного копіювання, яка має працювати автономно.

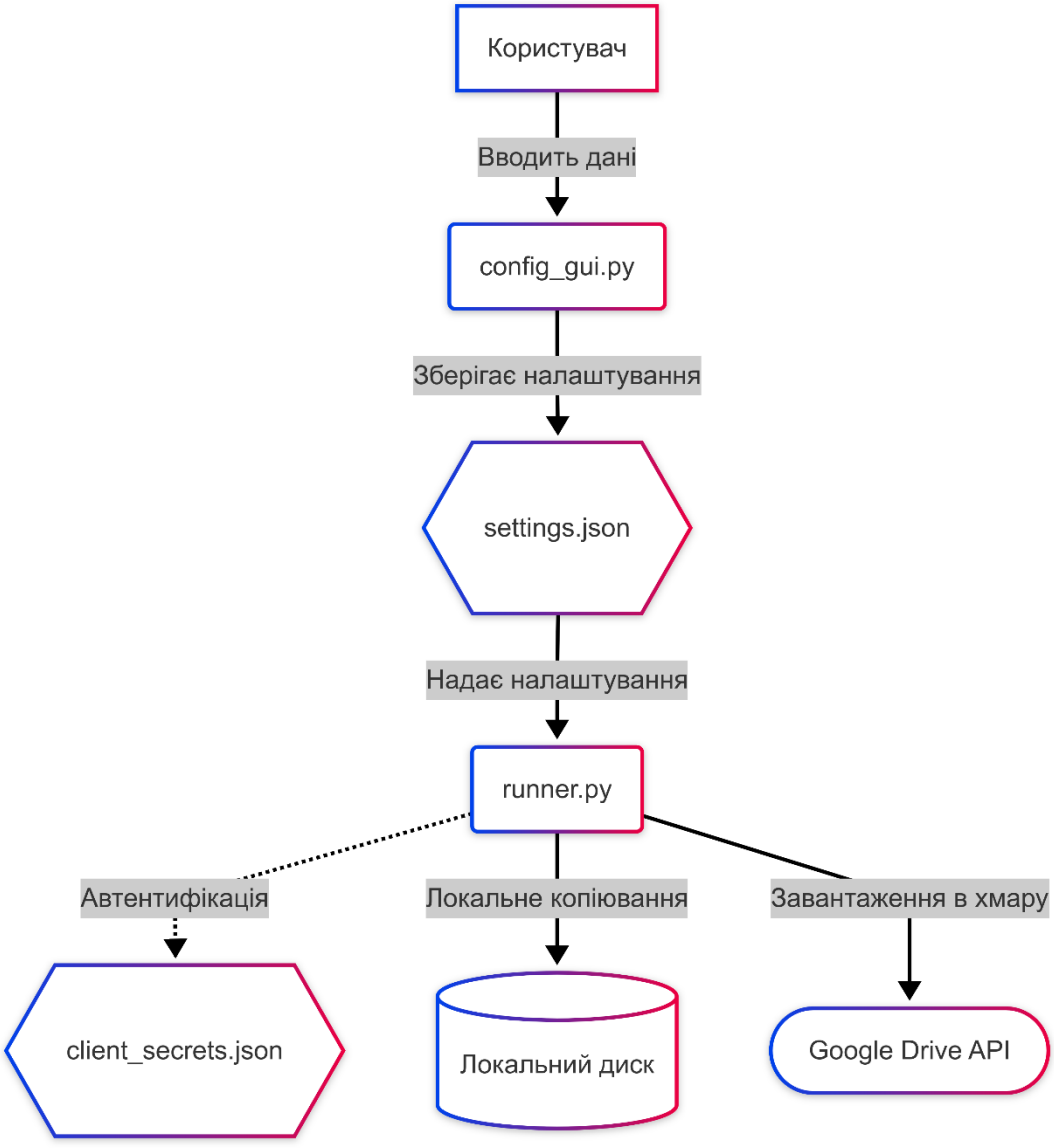

Рисунок 3.1 – Загальна схема системи резервного копіювання

Як показано на схематичній діаграмі (див. рис. 3.1), користувач взаємодіє виключно з модулем config_gui.py. Цей модуль надає графічний інтерфейс для введення всіх необхідних параметрів. Після збереження, вся конфігурація записується у файл settings.json. Цей файл слугує "мостом" між двома компонентами системи.

Виконавчий модуль runner.py, який призначений для автоматичного запуску, не має власного інтерфейсу. При старті він зчитує всю необхідну інформацію з файлу settings.json. Якщо налаштовано резервне копіювання на Google Drive, runner.py через бібліотеку PyDrive неявно використовує файл client_secrets.json для проходження процесу автентифікації з Google Drive API. Після цього, залежно від налаштувань, він виконує копіювання файлів на локальний диск або завантажує їх на Google Drive [9].

Така архітектура дозволяє досягти кількох важливих переваг:

- Розділення відповідальності (Separation of Concerns) [9,10], коли кожен модуль виконує чітко визначену функцію, що спрощує розробку, тестування та подальшу підтримку коду .
- Гнучкість використання, коли користувач може запускати config_gui.py лише тоді, коли потрібно змінити налаштування. Процес резервного копіювання може виконуватися незалежно за розкладом.
- Виконавчий модуль runner.py не залежить від графічного середовища, що дозволяє запускати його на серверах або в контейнерах, де графічний інтерфейс відсутній.

3.2.1 Модуль конфігурації (config_gui.py)

Цей модуль реалізує графічний інтерфейс за допомогою бібліотеки Tkinter і відповідає за всю взаємодію з користувачем на етапі налаштування.

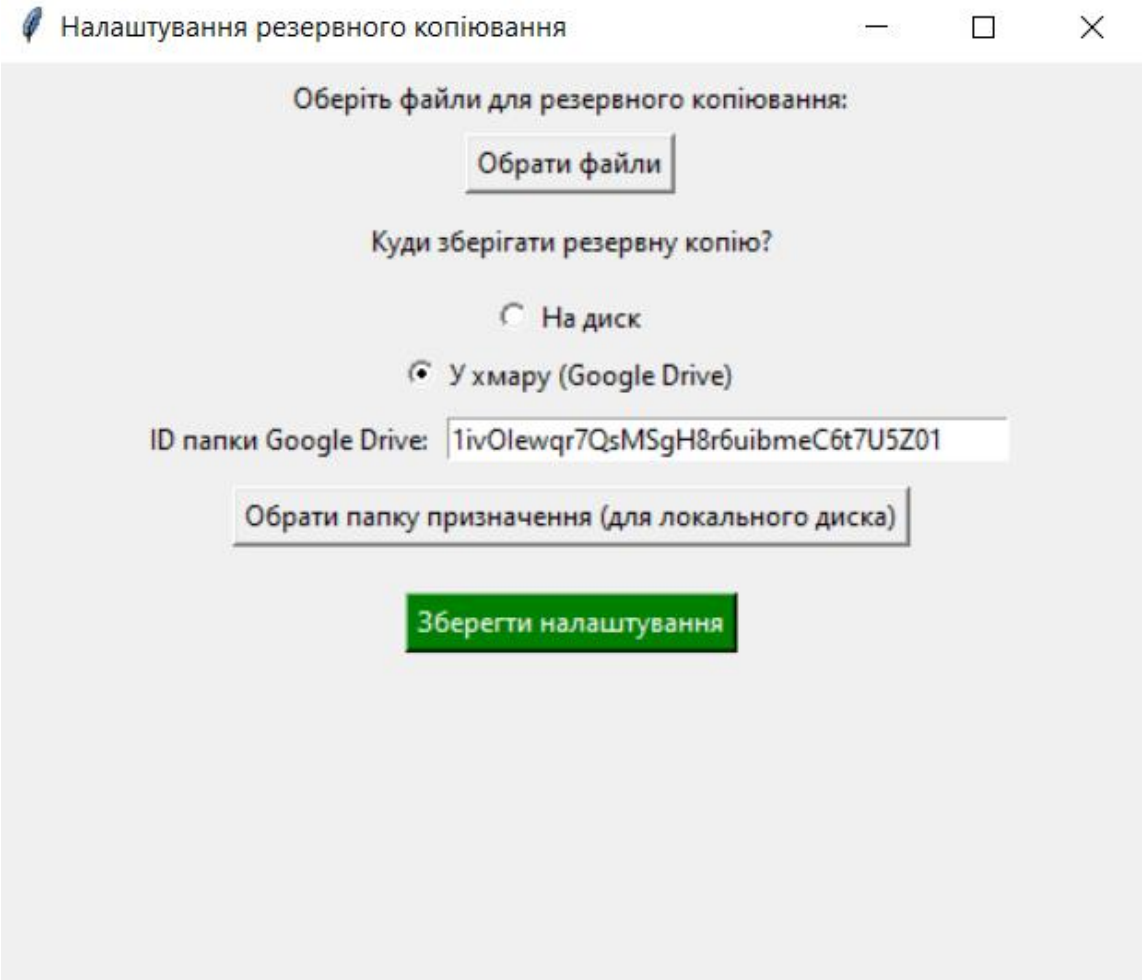

Рисунок 3.2 – Графічний інтерфейс модуля конфігурації

Модуль config_gui.py є ключовим компонентом розробленого програмного засобу, що відповідає за всю взаємодію з користувачем на етапі налаштування процесу резервного копіювання. Його основне призначення – надати простий, інтуїтивно зрозумілий та функціональний графічний інтерфейс (GUI), за допомогою якого користувач, навіть не маючи глибоких технічних знань, може легко визначити всі необхідні параметри для створення резервних копій. Цей модуль є "пультом керування" всією системою, оскільки саме тут генерується конфігурація, яка в подальшому використовується виконавчим модулем runner.py для автономної роботи.

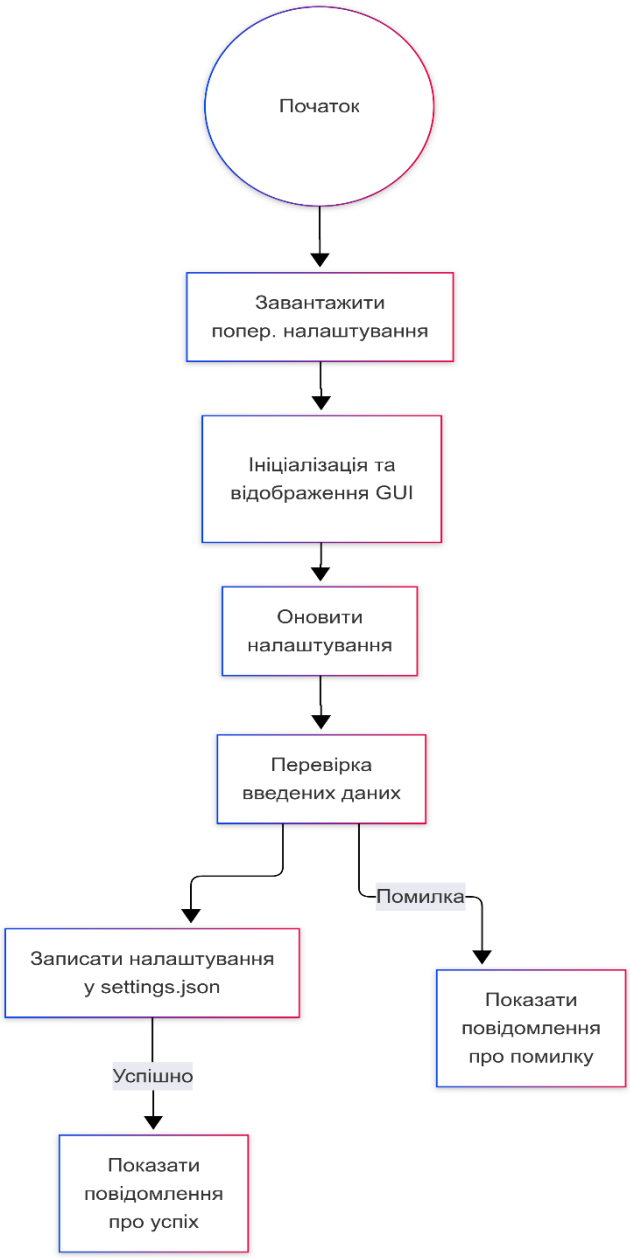

Рисунок 3.3 – Блок-схема логіки роботи модуля config_gui

Весь код модуля інкапсульовано у клас ConfigApp, що відповідає принципам об'єктно-орієнтованого програмування. Такий підхід дозволяє логічно згрупувати дані (атрибути класу, такі як список файлів, шлях призначення) та функції, що ними оперують (методи класу), в єдину сутність.

Для забезпечення функціональності на початку файлу імпортуються всі необхідні модулі. Основний модуль tkinter під псевдонімом tk використовується для створення елементів GUI, тоді як його підмодулі filedialog та messagebox надають доступ до стандартних системних діалогових вікон для вибору файлів,

директорій та відображення повідомлень. Для роботи з конфігураційними даними залучено модуль json, а для взаємодії з операційною системою, зокрема для перевірки існування файлів за допомогою os.path.exists, використовується модуль os.

Конструктор класу __init__(self, root) відповідає за ініціалізацію головного вікна програми та всіх його компонентів. Окрім встановлення заголовка та розмірів вікна, він ініціалізує ключові змінні стану, які зберігатимуть вибір користувача. Атрибут self.files створюється як порожній список для шляхів до файлів, а self.destination – як порожній рядок для шляху до локальної папки. Для відстеження вибору типу збереження використовується спеціальна змінна Tkinter tk.BooleanVar(), яка пов'язана з відповідними радіокнопками. Аналогічно, tk.StringVar() використовується для динамічного зв'язку з полем введення ID папки Google Drive. Застосування цих спеціальних змінних є рекомендованою практикою в Tkinter, оскільки вони дозволяють автоматично оновлювати стан віджетів при програмній зміні значення змінної, і навпаки.

Побудова самого графічного інтерфейсу відбувається послідовно за допомогою менеджера геометрії .pack(), який розміщує віджети один за одним по вертикалі. Спочатку створюється блок для вибору файлів, що складається з мітки та кнопки, прив'язаної до методу self.select_files. Далі йде блок вибору типу збереження, де дві радіокнопки пов'язані з однією змінною self.backup_to_drive, але мають різні значення (False для локального диска та True для Google Drive). Важливо, що кожна радіокнопка має прив'язану команду self.toggle_gdrive_id_input, що забезпечує динамічну зміну інтерфейсу. Після них розміщується блок введення ID папки Google Drive, реалізований за допомогою tk.Frame для зручного керування видимістю. Наступними елементами є кнопка для вибору локальної папки призначення та ключовий елемент керування – кнопка збереження налаштувань. Завершує композицію статусна мітка в нижній частині вікна, яка надає користувачеві текстовий зворотний зв'язок про його дії.

3.2.2 Логіка роботи методів

Метод load_initial_settings(self) реалізує важливу функцію для покращення досвіду користувача – збереження стану програми між запусками. При старті програми він перевіряє, чи існує файл settings.json. Якщо так, він намагається його прочитати та десеріалізувати дані з JSON у словник Python. Після цього значення зі словника використовуються для встановлення початкового стану віджетів GUI. Наприклад, self.backup_to_drive.set(...) встановлює, яка з радіокнопок буде обрана за замовчуванням, а self.gdrive_folder_id_var.set(...) заповнює поле введення ID папки раніше збереженим значенням. Це дозволяє користувачеві не вводити всі налаштування заново, а лише редагувати існуючі. Метод також містить обробку помилок на випадок, якщо файл конфігурації пошкоджений або має некоректний формат.

Метод toggle_gdrive_id_input(self) відповідає за динамічність інтерфейсу. Він викликається щоразу, коли користувач натискає на одну з радіокнопок вибору типу збереження. Метод перевіряє поточне значення змінної self.backup_to_drive. Якщо воно True (обрано Google Drive), він робить видимим фрейм з полем для введення ID папки, використовуючи self.gdrive_id_frame.pack(). Якщо ж значення False (обрано локальний диск), він приховує цей фрейм за допомогою self.gdrive_id_frame.pack_forget(). Такий підхід робить інтерфейс більш чистим та зрозумілим, показуючи користувачеві лише ті поля, які є релевантними для його поточного вибору.

Методи select_files(self) та select_destination(self) використовують підмодуль tkinter.filedialog для взаємодії з нативною файловою системою операційної системи. Метод filedialog.askopenfilenames() відкриває стандартне діалогове вікно, що дозволяє користувачеві обрати один або декілька файлів, і повертає їхні шляхи. Метод filedialog.askdirectory() дозволяє обрати одну директорію. Після успішного вибору обидва методи оновлюють текст статусної мітки, інформуючи користувача про результат його дії (наприклад, "Обрано 5 файлів" або "Папка призначення: D:\Backups").

Ключовий метод save_settings(self) що виконується при натисканні кнопки "Зберегти налаштування". Він виконує дві основні задачі: валідацію введених даних та їх збереження.

Процес валідації починається з перевірки, чи було обрано хоча б один файл для копіювання. Далі, залежно від обраного типу збереження, перевіряється наявність ID папки Google Drive або шляху до локальної папки призначення. Якщо будь-яка з цих перевірок не проходить, за допомогою messagebox.showerror() виводиться діалогове вікно з відповідним повідомленням про помилку, і процес збереження переривається. Якщо валідація пройшла успішно, програма переходить до формування та збереження даних. Створюється словник Python, куди записуються всі налаштування, після чого цей словник серіалізується у формат JSON та записується у файл settings.json. Використання конструкції with open(...) гарантує коректне закриття файлу, а параметр encoding="utf-8" забезпечує правильну роботу з нелатинськими символами у шляхах. Після успішного збереження користувач отримує підтвердження через інформаційне діалогове вікно.

Блок if __name__ == "__main__" - це стандартна конструкція Python забезпечує, що код для запуску програми буде виконано лише тоді, коли файл config_gui.py запускається як основний скрипт. Він створює кореневе вікно tk.Tk(), створює екземпляр класу ConfigApp, передаючи йому це вікно, і запускає головний цикл обробки подій root.mainloop(). Цей цикл очікує на дії користувача, наприклад, натискання кнопок, введення тексту, і реагує на них, викликаючи відповідні методи-обробники.

Таким чином, модуль config_gui.py є самодостатнім, добре структурованим застосунком, який ефективно вирішує задачу надання користувачеві зручного інструменту для конфігурації процесу резервного копіювання, та є невід'ємною частиною розробленої системи.

Виконавчий модуль runner.py є основним робочим компонентом. Якщо модуль config_gui.py є "пультом керування" системи, то runner.py – це її "двигун". Цей скрипт є повністю автономним, не має графічного інтерфейсу і

призначений для виконання основної задачі – резервного копіювання файлів згідно з раніше збереженою конфігурацією. Його архітектура дозволяє запускати його автоматично при старті системи, що є ключовим для реалізації принципу "налаштуй і забудь".

Робота скрипта runner.py побудована на послідовному виконанні кількох логічних кроків, інкапсульованих у окремі функції. Основний процес керується функцією main(), яка виступає в ролі диспетчера. Для забезпечення функціональності скрипт імпортує необхідні бібліотеки: json для роботи з файлом налаштувань, os та shutil для операцій з файловою системою, а також GoogleAuth та GoogleDrive з бібліотеки PyDrive для взаємодії з Google Drive API.

Функція load_settings() є точкою входу для виконавчого модуля. Її завдання – прочитати файл settings.json та надати решті програми конфігураційні дані. Функція реалізує надійний механізм читання, що включає перевірку існування файлу за допомогою os.path.exists(). Якщо файл не знайдено, або якщо він пошкоджений і не може бути десеріалізований (обробка винятку json.JSONDecodeError), функція повертає None і виводить інформативне повідомлення в консоль, що дозволяє коректно зупинити подальше виконання скрипта.

Логіку локального резервного копіювання реалізує функція copy_locally(files, destination). Функція приймає список шляхів до файлів та шлях до папки призначення. Перед початком копіювання виконується низка перевірок: чи існує папка призначення, і якщо ні, робиться спроба її створити за допомогою os.makedirs(). Далі, у циклі перебирається кожен файл зі списку, перевіряється його існування через os.path.isfile(), після чого за допомогою shutil.copy2() файл копіюється до папки призначення. Використання shutil.copy2() є перевагою, оскільки ця функція намагається зберегти метадані файлу, такі як час модифікації. Процес супроводжується виведенням повідомлень у консоль, а лічильник successful_copies підраховує кількість успішно скопійованих файлів для фінального звіту.

Функція upload_to_drive(files, gdrive_folder_id) - це найбільш складна частина модуля, що відповідає за завантаження файлів на Google Drive. Вона демонструє взаємодію з зовнішнім API. Першим кроком є автентифікація. Виклик gauth = GoogleAuth() та gauth.LocalWebserverAuth() ініціює процес OAuth 2.0. Бібліотека PyDrive неявно використовує файл client_secrets.json для отримання облікових даних додатку, після чого відкриває браузер для отримання дозволу від користувача. При наступних запусках PyDrive може використовувати збережені токени, уникаючи повторної автентифікації. Після успішної автентифікації створюється об'єкт GoogleDrive, який використовується для взаємодії з файлами. Далі, аналогічно до локального копіювання, скрипт ітерує по списку файлів. Для кожного файлу за допомогою drive.CreateFile() створюється об'єкт файлу на Google Drive, якому в якості "батька" (parents) присвоюється ID папки, вказаний користувачем. Після цього встановлюється вміст файлу та виконується його завантаження. Цей процес також супроводжується логуванням та підрахунком успішних операцій.

Функція main() – функція, що є центральним координатором. Вона викликає load_settings() для отримання конфігурації, розбирає отриманий словник на окремі змінні files, use_drive, destination тощо та, на основі значення use_drive, приймає рішення, яку саме функцію викликати – copy_locally чи upload_to_drive. Також у main() реалізовані фінальні перевірки, наприклад, чи не порожній список файлів і чи вказано папку призначення для відповідного типу копіювання. Це забезпечує коректну та передбачувану поведінку програми.

Завдяки такій структурі, runner.py є надійним та самодостатнім модулем, готовим до інтеграції з системними планувальниками завдань для повної автоматизації процесу резервного копіювання.

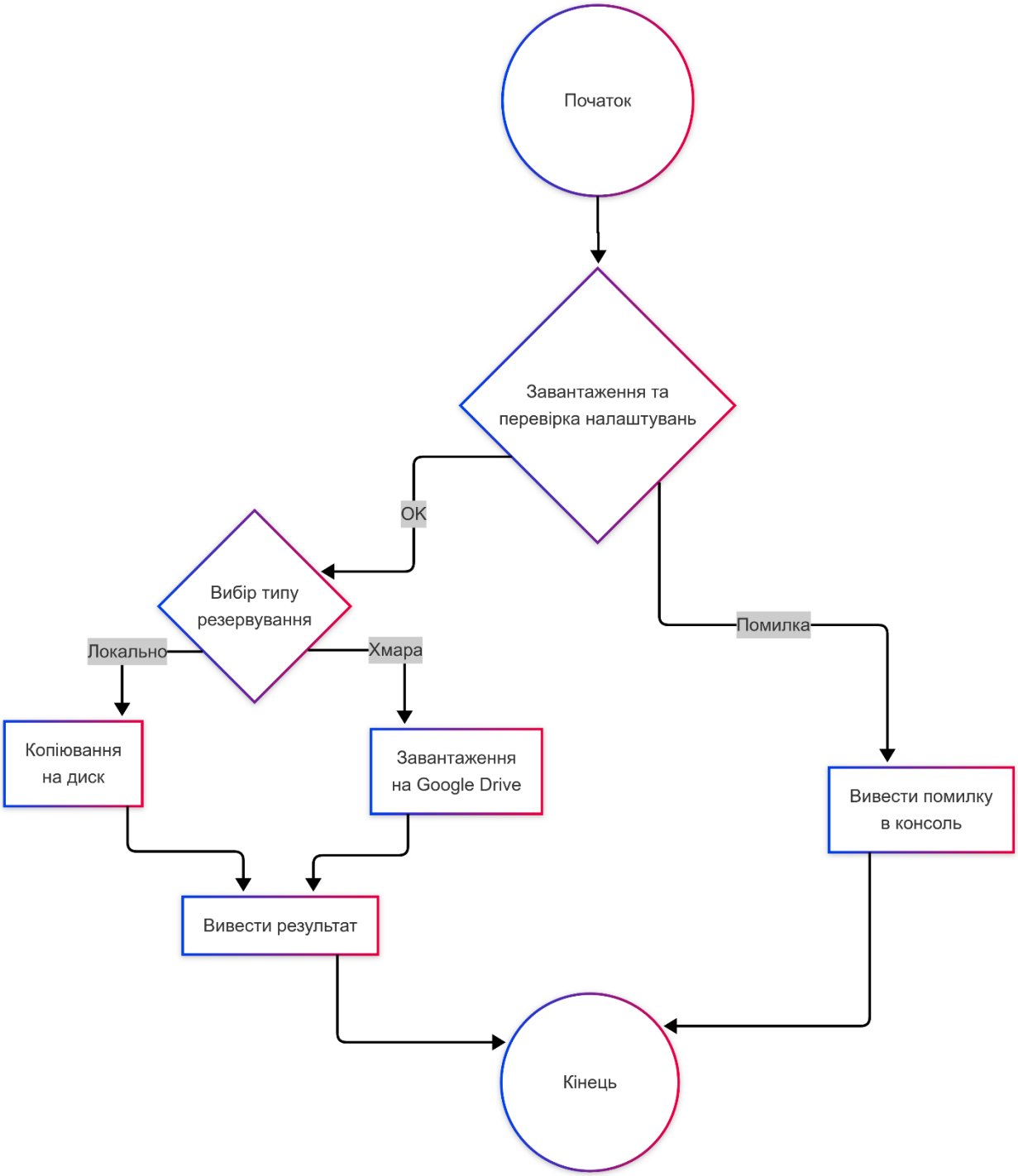

Рисунок 3.4 – Блок-схема логіки роботи модуля runner.py

3.3 Тестування розроблених сценаріїв у різних ОС

Тестування програмного засобу проводилося для перевірки його функціональності, стабільності та коректності роботи в основному цільовому середовищі – ОС Windows 10/11.

Мета тестування полягала в комплексній перевірці всіх аспектів роботи системи: від коректності функціонування графічного інтерфейсу та обробки дій користувача до правильності виконання обох сценаріїв резервного копіювання та адекватної реакції на помилкові й нештатні ситуації. Тестування проводилося у наступному середовищі: ОС Windows 11 Pro з встановленим інтерпретатором Python версії 3.10 та ключовими бібліотеками pydrive2 і tkinter.

Хоча основне тестування проводилось на Windows, завдяки кросплатформеності Python та використаних бібліотек, очікується, що скрипти будуть коректно працювати і в середовищах Linux та macOS за умови встановлення необхідних залежностей та наявності відповідних шляхів до файлів.

Всі основні тестові кейси було пройдено успішно (див. табл 3.1). Програмний засіб продемонстрував стабільну роботу та коректну обробку як штатних, так і помилкових сценаріїв, що підтверджує правильність обраної архітектури та реалізації.

На основі проведеної розробки та тестування можна зробити висновки щодо ключових характеристик отриманого програмного засобу. З точки зору ефективності, він повністю виконує поставлену задачу, дозволяючи користувачеві один раз налаштувати параметри і надалі не брати участі в процесі, що значно економить час та знижує ризик пропустити важливе копіювання. Обрана двокомпонентна архітектура з розділенням конфігуратора та виконавця виявилася виправданою, оскільки забезпечує гнучкість використання та спрощує автоматизацію. Програма продемонструвала високу стабільність завдяки використанню блоків обробки винятків (try...except), що дозволяє їй коректно завершувати роботу або продовжувати її, навіть при

виникненні проблем з окремими файлами, наприклад, відсутність доступу чи неіснуючий файл.

Таблиця 3.1 – Тестові кейси та результати

ID	Опис кейсу	Очікуваний результат	Фактичний результат
TC-01	Локальне копіювання 5 файлів у існуючу папку.	Усі 5 файлів скопійовано. У консолі повідомлення "Успішно скопійовано 5 файлів.".	Успішно
TC-02	Локальне копіювання 3 файлів у неіснуючу папку.	Папку створено. Усі 3 файли скопійовано.	Успішно
TC-03	Хмарне копіювання 2 файлів (перший запуск).	Відкривається браузер для OAuth 2.0. Після авторизації файли завантажено.	Успішно
TC-04	Спроба зберегти налаштування без вибору файлів.	GUI видає повідомлення про помилку "Не вибрано файлів.".	Успішно
TC-05	Спроба локального збереження без вибору папки.	GUI видає повідомлення про помилку "Вкажіть папку призначення...".	Успішно
TC-06	Спроба хмарного збереження без введення ID папки.	GUI видає повідомлення про помилку "Вкажіть ID папки Google Drive.".	Успішно
TC-07	Запуск runner.py з налаштуваннями на копіювання неіснуючого файлу.	У консолі виводиться попередження про пропуск файлу. Інші файли копіюються.	Успішно

Щодо продуктивності, вона є абсолютно прийнятною для задач резервного копіювання персональних даних або даних малого офісу. Оскільки Python є інтерпретованою мовою, а процес копіювання обмежується швидкістю дискової підсистеми або інтернет-з'єднання, швидкість виконання самого коду не є вузьким місцем. Нарешті, архітектура є достатньо масштабованою: модульна структура дозволяє в майбутньому легко додавати підтримку інших хмарних сервісів (наприклад, Dropbox, OneDrive) шляхом написання нових функцій, аналогічних upload_to_drive(), або розширювати функціонал GUI новими налаштуваннями, такими як створення розкладу чи шифрування даних.

ВИСНОВКИ

У даній кваліфікаційній роботі було успішно вирішено актуальну задачу розробки програмного засобу для автоматизації процесу резервного копіювання даних, що є критично важливим аспектом забезпечення інформаційної безпеки в сучасних умовах.

На першому етапі роботи було проведено теоретичний аналіз основ системного адміністрування та ролі автоматизації в підвищенні його ефективності. Було обґрунтовано, що автоматизація рутинних завдань, зокрема резервного копіювання, дозволяє мінімізувати ризик людської помилки, оптимізувати часові витрати та підвищити загальну надійність IT-інфраструктури.

На другому етапі було здійснено порівняльний аналіз основних засобів створення сценаріїв: Bash, PowerShell та Python. На основі аналізу їхніх переваг, недоліків та сфер застосування було зроблено обґрунтований висновок про доцільність вибору мови Python для реалізації даного проєкту. Ключовими факторами стали її кросплатформеність, простота синтаксису, а також наявність великої екосистеми бібліотек, зокрема Tkinter для створення графічного інтерфейсу та PyDrive для легкої інтеграції з хмарними сервісами, такими як Google Drive.

На третьому, практичному етапі роботи, було спроєктовано та реалізовано програмний засіб, що відповідає всім поставленим вимогам. Розроблена двокомпонентна архітектура, яка складається з модуля конфігурації (config_gui.py) та окремого виконавчого модуля (runner.py), виявилася ефективною. Вона дозволила чітко розділити логіку взаємодії з користувачем та логіку автономного виконання завдань.

Модуль config_gui.py надає користувачеві простий та інтуїтивно зрозумілий графічний інтерфейс для вибору файлів, типу збереження (локальний диск або Google Drive) та вказання всіх необхідних параметрів.

Виконавчий модуль runner.py зчитує збережену у файлі settings.json конфігурацію та виконує резервне копіювання в автоматичному режимі, підтримуючи автентифікацію OAuth 2.0 для роботи з Google Drive API.

Тестування розробленого програмного засобу підтвердило його стабільність, функціональність та коректну обробку як штатних, так і помилкових сценаріїв.

Таким чином, мета кваліфікаційної роботи була повністю досягнута. Створено готовий до використання, гнучкий програмний засіб, що надає простий та доступний інструмент для автоматизації резервного копіювання. Практичне значення роботи полягає в тому, що розроблене рішення може бути використане широким колом користувачів для захисту важливих даних. Крім того, архітектура програми є масштабованою та дозволяє в майбутньому легко додавати новий функціонал, наприклад, підтримку інших хмарних сервісів, шифрування даних чи вбудований планувальник завдань.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Гуртовцев А. Л. Управление безопасностью мультисервисных сетей. Минск: БГУИР, 2018. 248 с.
2. Tanenbaum A. S., Wetherall D. J. Computer Networks. 5th ed. Pearson Education, 2011. 960 p.
3. Nemeth E., Snyder G., Hein T. R., Whaley B. UNIX and Linux System Administration Handbook. 5th ed. Addison-Wesley Professional, 2017. 1232 p.
4. Shotts W. The Linux Command Line: A Complete Introduction. 2nd ed. No Starch Press, 2019. 528 p.
5. Jones D., Hicks J. Learn PowerShell Scripting in a Month of Lunches. 3rd ed. Manning Publications, 2018. 480 p.
6. Lutz M. Learning Python. 5th ed. O'Reilly Media, 2013. 1648 p.
7. Al Sweigart. Automate the Boring Stuff with Python: Practical Programming for Total Beginners. 2nd ed. No Starch Press, 2019. 592 p.
8. Офіційна документація Python. URL: https://docs.python.org (дата звернення: 15.05.2024).
9. Google Drive API Documentation. URL: https://developers.google.com/drive (дата звернення: 20.05.2024).
10. PyDrive2: The simplified Google Drive API wrapper library. URL: https://pydrive2.github.io/pydrive2/ (дата звернення: 21.05.2024).