**РОЗДІЛ 2. ПРОЄКТУВАННЯ І РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАСОБУ**

**2.1 Постановка задачі: вимоги до функціоналу**

У сучасних умовах інформаційної безпеки збереження даних користувача є одним з головних аспектів роботи з комп'ютерами. Основна мета розробленого програмного забезпечення — автоматизоване створення резервної копії важливих файлів користувача з можливістю зберігання як на локальному носії, так і в хмарному сховищі Google Drive.

Програма-скрипт повинна забезпечувати зручний та інтуїтивно зрозумілий інтерфейс користувача, а також мати набір необхідних функцій для виконання резервного копіювання важливих даних. Нижче подано детальний опис кожної функціональної можливості:

**Вибір користувачем файлів або папок для резервного копіювання.**

Програма повинна надавати користувачеві можливість самостійно обрати, які саме файли або папки будуть скопійовані. Це реалізується через графічний інтерфейс, де за допомогою стандартного діалогу вибору файлів можна додати необхідні об’єкти до списку для подальшого копіювання. Таким чином, користувач має повний контроль над тим, які саме дані підлягають резервному збереженню.

**Можливість вибору типу збереження: локально або в хмару.**

Програма повинна дозволяти користувачу обирати, куди саме зберігатимуться резервні копії — на локальний носій (наприклад, інший диск, USB-накопичувач тощо) або ж до хмарного сервісу Google Drive. Це забезпечується за допомогою перемикачів у графічному інтерфейсі. Такий підхід дає змогу забезпечити гнучкість використання та задовольнити потреби як звичайних, так і більш досвідчених користувачів.

**Збереження вибору користувача у конфігураційному файлі.**

Після того, як користувач обирає файли, тип резервного копіювання та шлях до папки призначення (у випадку локального збереження), усі ці параметри мають бути збережені у спеціальному конфігураційному файлі, зазвичай у форматі JSON. Це дає змогу зчитувати ці налаштування при наступних запусках програми без необхідності повторного введення, що значно підвищує зручність користування.

**Авторизація в Google Drive через обліковий запис користувача.**

Для збереження файлів у хмарі передбачено механізм авторизації через обліковий запис Google. Програма використовує протокол OAuth 2.0, що забезпечує безпечний доступ до Google Drive користувача. Після одноразової авторизації програма отримує дозвіл на створення та завантаження файлів до обраної папки Google Drive. Цей процес реалізується з використанням бібліотеки PyDrive.

**Реалізація процесу копіювання в обраний користувачем спосіб.**

В залежності від обраного методу збереження, програма виконує відповідний процес копіювання. Якщо користувач обрав локальне копіювання, файли копіюються до вказаної директорії з використанням стандартних засобів мови Python (наприклад, модуля shutil). Якщо ж обрано хмарне збереження, то кожен файл завантажується до папки в Google Drive через API. Програма повідомляє користувача про успішне завершення кожного етапу копіювання або виводить повідомлення про помилки у разі виникнення проблем.

**Можливість запуску процесу автоматично при старті системи.**

Однією з ключових особливостей є автоматичний запуск процесу резервного копіювання під час завантаження операційної системи. Це дозволяє користувачу не турбуватись про необхідність запуску програми вручну. Такий функціонал реалізується шляхом створення виконуваного файлу з основним сценарієм копіювання та додаванням ярлика до нього у папку автозавантаження Windows. У результаті цього резервне копіювання відбуватиметься автоматично, кожного разу при ввімкненні комп’ютера, забезпечуючи постійний захист важливих даних.

**2.2 Вибір інструментів реалізації**

Для реалізації програмного забезпечення обрано мову програмування Python. Основними причинами вибору Python є: простота синтаксису, що дає змогу швидко розробляти і тестувати функціональність;

Велика кількість бібліотек для взаємодії з файлами, веб-сервісами, базами даних, а також підтримка роботи з GUI;

Кросплатформеність, що забезпечує можливість запуску розробленої програми як у середовищі Windows, так і на інших операційних системах (Linux, MacOS);

Активна спільнота розробників, велика кількість документації, прикладів та інструкцій в Інтернеті, що сприяє розв'язанню технічних труднощів під час розробки.

Для створення графічного інтерфейсу користувача (GUI) було обрано стандартну бібліотеку Tkinter, яка входить до складу мови Python. Tkinter забезпечує зручні інструменти для створення форм, кнопок, полів вводу, списків і повідомлень, що дозволяє реалізувати простий і зрозумілий інтерфейс. Оскільки метою є надання користувачу змоги налаштовувати резервне копіювання за допомогою діалогових вікон, вибір Tkinter виявився доцільним і ефективним.

Для реалізації збереження даних у хмарі було обрано Google Drive як популярний, надійний та безкоштовний хмарний сервіс. Для інтеграції з Google Drive було використано бібліотеку PyDrive — це високорівнева обгортка над Google Drive API. Вона спрощує аутентифікацію користувача через OAuth 2.0, дозволяє отримувати доступ до файлів на хмарному диску, завантажувати, створювати та оновлювати їх без необхідності писати складний HTTP-запит вручну. PyDrive забезпечує просту авторизацію з відкриттям браузера, де користувач підтверджує доступ, після чого програма може працювати з Google Drive автоматично.

Для компіляції програми в виконуваний файл .exe обрано бібліотеку PyInstaller. Цей інструмент дозволяє створити один автономний файл, який містить усі залежності, необхідні для запуску програми на будь-якому комп’ютері з Windows, навіть якщо на ньому не встановлено Python. Створення .exe-файлу важливе для зручного поширення програми серед користувачів, які не мають технічних навичок чи досвіду встановлення середовища розробки.

Також для автоматичного запуску програми після ввімкнення комп’ютера використано можливості операційної системи Windows, зокрема директорію автозавантаження (%APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup). У цю папку додається ярлик до виконуваного файлу програми, що забезпечує її запуск одразу після старту системи.

Отже, вибір мови програмування та бібліотек є обґрунтованим, оскільки всі інструменти взаємодіють між собою, забезпечують необхідний функціонал, мають належний рівень підтримки та документації, а також відповідають вимогам до функціональності, зручності й безпеки розробленого програмного забезпечення.

**2.3 Архітектура програми та сценарії використання**

Програма складається з двох основних компонентів:

1. GUI-застосунок, що виконує роль інтерфейсу налаштування, де користувач вказує, які файли копіювати та куди.
2. Скрипт резервного копіювання, який зчитує збережені параметри конфігурації та автоматично виконує копіювання без взаємодії з користувачем.

Такий розподіл дозволяє розділити функціональність на два логічні рівні: рівень взаємодії з користувачем (інтерфейс налаштування) та рівень автоматизованого виконання (резервне копіювання), що позитивно впливає як на зручність використання, так і на підтримку програми.

Компонент графічного інтерфейсу реалізовано за допомогою бібліотеки Tkinter. Всі введені користувачем параметри зберігаються у вигляді конфігураційного файлу, що дозволяє зберігати персональні налаштування для подальшого використання.

Другий компонент — це автономний скрипт резервного копіювання, який не потребує відкриття графічного інтерфейсу. Його завдання — автоматичне виконання копіювання файлів у відповідності до налаштувань, збережених у конфігураційному файлі. Цей компонент може бути доданий до автозавантаження Windows, що дозволяє здійснювати резервне копіювання при кожному увімкненні комп’ютера або у фоновому режимі. Завдяки цьому користувач більше не мусить турбуватися про запуск програми вручну чи про забуте копіювання важливих файлів.

Архітектура системи побудована таким чином, щоб забезпечити модульність, масштабованість та простоту у використанні. Умовно, архітектуру можна подати як двошарову модель:

1. Шар налаштувань — забезпечує отримання даних від користувача, їх перевірку, збереження у структурованому форматі JSON.
2. Шар виконання — використовує збережену конфігурацію для ініціювання копіювання, з використанням або функцій операційної системи, або зовнішніх API (наприклад, Google Drive API через бібліотеку PyDrive).

Сценарії використання програми можна описати наступним чином:

Сценарій 1 — початкове налаштування:

* Користувач запускає GUI-програму;
* Вибирає файли для резервного копіювання;
* Обирає тип копіювання — локальне або хмарне;
* Вказує директорію призначення або ідентифікатор папки Google Drive;
* Зберігає налаштування та закриває програму.

Сценарій 2 — автоматичне резервне копіювання:

* Користувач вмикає комп’ютер;
* Сценарій резервного копіювання автоматично запускається (через автозапуск або планувальник завдань);
* Скрипт зчитує дані з конфігураційного файлу;
* Якщо тип копіювання — локальне, файли копіюються у вказану папку;
* Якщо тип копіювання — хмарне, ініціюється з’єднання з Google Drive та виконується завантаження;
* У разі успіху/помилки скрипт може зберігати лог у текстовому файлі або виводити повідомлення у консоль (або в майбутньому — у вигляді системного повідомлення).

Завдяки такій архітектурі програмне забезпечення має низький поріг входу для користувача, є достатньо гнучким для адаптації під різні сценарії та потенційно може бути доповнене новими функціями.

Обрана архітектура дозволяє досягти основної мети — створення простої, надійної та автоматизованої системи резервного копіювання, яка працює за принципом «налаштуй і забудь». Такий підхід є особливо актуальним в умовах, коли користувач може не мати спеціальних знань у галузі інформаційних технологій, але потребує захисту важливої інформації.

**2.4 Побудова графічного інтерфейсу користувача**

Графічний інтерфейс користувача (GUI) є важливою складовою будь-якого прикладного програмного забезпечення, оскільки саме через нього здійснюється основна взаємодія між користувачем і програмою. У розробленому програмному забезпеченні для побудови інтерфейсу було використано стандартну бібліотеку Tkinter, яка входить до складу мови програмування Python.

Tkinter — це зручний інструмент для створення віконних додатків, що дозволяє швидко створювати функціональні елементи керування, зокрема кнопки, поля введення, мітки, перемикачі тощо. Його основною перевагою є простота у використанні, що дозволяє створювати повноцінні віконні інтерфейси з мінімальним обсягом коду.

Інтерфейс програми реалізований у вигляді одного основного вікна, в якому послідовно розташовано всі елементи керування. Він складається з таких ключових компонентів:

1. Кнопка для вибору файлів:

Цей елемент дозволяє користувачу вибрати один або декілька файлів, які потрібно включити до списку резервного копіювання. Після натискання кнопки відкривається стандартне вікно вибору файлів, де користувач може обрати потрібні об’єкти. Обрані файли зберігаються у внутрішньому списку для подальшої обробки. Одразу після вибору на екрані виводиться кількість обраних файлів як підтвердження успішної дії.

2. Перемикач типу резервного копіювання:

У цьому блоці користувач має змогу обрати, куди саме будуть копіюватися файли — локально (тобто на диск, флешку, зовнішній накопичувач) чи у хмарне сховище (Google Drive). Цей перемикач реалізований у вигляді двох радіокнопок, що дозволяє забезпечити простий вибір лише одного варіанту. Обраний варіант записується у змінну, яка впливає на подальшу логіку виконання резервного копіювання.

3. Кнопка для вибору папки призначення:

Якщо користувач обирає локальне копіювання, стає активною кнопка, яка дозволяє обрати цільову директорію. Це відкриває вікно вибору папки, де можна вказати шлях для збереження резервних копій. Якщо користувач не обрав папку, і спробує розпочати копіювання, програма повідомить про необхідність вказати місце призначення.

4. Кнопка «Копіювати файли»:

Ця кнопка є фінальною дією у вікні — вона запускає процес резервного копіювання згідно з обраними параметрами. Перед запуском здійснюється перевірка: чи обрано файли, чи обрано тип копіювання, і (у випадку локального резервування) чи вказано папку призначення. Якщо всі умови виконано, відбувається копіювання файлів у відповідне місце — або на диск, або в Google Drive.

5. Відображення статусу:

У нижній частині вікна розташована текстова мітка, яка виконує роль інформаційного поля. Вона використовується для відображення поточного статусу операції. Наприклад: «Обрано 5 файлів», «Папку призначення не обрано», «Файли скопійовано успішно», «Помилка при копіюванні» тощо. Це дозволяє користувачу легко орієнтуватися в тому, що відбувається у програмі, без необхідності заглядати в технічні логи.

Кожна дія в інтерфейсі супроводжується повідомленнями у вигляді діалогових вікон або текстових підказок. Наприклад, якщо копіювання пройшло успішно, з’являється вікно-повідомлення з підтвердженням. У випадку виникнення помилки (наприклад, проблеми з доступом до папки або відсутність підключення до Інтернету для Google Drive) програма показує відповідне повідомлення з описом проблеми.

Завдяки використанню Tkinter, вдалося створити інтерфейс, який є простим для розуміння навіть для недосвідчених користувачів. Інтерфейс не перевантажений зайвими елементами, а всі дії мають логічну послідовність, що відповідає природному процесу прийняття рішень користувачем.

У майбутньому можливе розширення інтерфейсу, зокрема додавання таких функцій, як:

* підтримка обрання цілих папок, а не лише окремих файлів;
* можливість встановити автоматичний графік запуску копіювання;
* ведення журналу резервних копій;
* налаштування параметрів хмарного з’єднання без необхідності повторної авторизації.

Таким чином, створений графічний інтерфейс виконує роль зручного інструменту налаштування, забезпечуючи доступ до основних функцій резервного копіювання у зрозумілій і доступній формі.

**2.5 Робота з Google Drive API**

Для роботи з Google Drive програма використовує PyDrive. Авторизація здійснюється через OAuth2, при цьому користувач має надати дозвіл на доступ до власного Google Drive. Після авторизації файли завантажуються в заздалегідь створену папку, ID якої зберігається в коді.

Функція upload\_to\_drive використовує методи:

* CreateFile для створення файлу-об'єкта;
* SetContentFile для прикріплення локального файлу;
* Upload для відправлення файлу в Google Drive.

**2.6 Збереження конфігурації користувача**

Після вибору файлів для копіювання, типу резервного копіювання (локальне чи хмарне), а також каталогу призначення, всі ці параметри зберігаються в окремому конфігураційному файлі у форматі JSON, наприклад у файлі settings.json. Формат JSON (JavaScript Object Notation) є простим для розуміння та читання як людиною, так і програмою. Це дозволяє ефективно серіалізувати дані, тобто зберегти об’єкти Python (списки, словники, рядки тощо) у вигляді текстового файлу, який можна легко завантажити при наступному запуску програми.

Під час першого запуску програми користувач взаємодіє з графічним інтерфейсом, вказуючи файли, які потрібно резервно копіювати, та місце призначення — локальну папку або хмарне сховище Google Drive. Після завершення налаштувань ці параметри автоматично записуються до конфігураційного файлу.

Завдяки цьому при наступному запуску резервного скрипту (без запуску графічного інтерфейсу) програма може одразу прочитати налаштування з файлу settings.json та автоматично виконати копіювання без необхідності повторного введення тих самих параметрів.

Цей підхід суттєво спрощує процес резервного копіювання для користувача, особливо в контексті автоматичного запуску програми під час старту операційної системи. Наявність попередньо збережених параметрів дозволяє уникнути повторної конфігурації та забезпечує автономність роботи сценарію. Це також знижує ризик помилки при ручному введенні даних і дає змогу централізовано змінювати конфігурацію, редагуючи лише один файл.

Окрім того, збереження налаштувань у JSON-файлі дозволяє реалізувати можливість резервного відновлення. Наприклад, якщо користувач змінить комп’ютер або середовище роботи, достатньо перенести лише саму програму разом із файлом налаштувань, і вона продовжить працювати з тими ж параметрами.

Важливою перевагою є й те, що файл конфігурації може зберігатися в прихованій папці користувача (наприклад, у %APPDATA%) або в директорії самої програми, що мінімізує ймовірність випадкового видалення або редагування з боку недосвідченого користувача.

Також при розробці було передбачено механізм обробки ситуацій, коли файл налаштувань відсутній або пошкоджений. У такому випадку програма ініціює графічний інтерфейс для створення нової конфігурації, що робить програму більш стабільною та зручною у використанні.

**2.7 Реалізація логіки резервного копіювання**

Сценарій резервного копіювання (наприклад, backup\_runner.py) зчитує файл конфігурації, перевіряє тип резервного копіювання та здійснює копіювання в локальну папку або завантаження в хмару. Таким чином, програма діє повністю автоматично, згідно з попередньо заданими параметрами.

**2.8 Створення виконуваного .exe файлу та додавання до** автозавантаження

Щоб користувач не запускав скрипт вручну, створено виконуваний файл із використанням PyInstaller:

pyinstaller --onefile backup\_runner.py

Файл .exe можна додати до автозавантаження Windows:

* Створити ярлик до .exe;
* Перемістити його до папки C:\Users<UserName>\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup.

Це гарантує запуск сценарію резервного копіювання щоразу при запуску комп'ютера.

Загалом, реалізований програмний засіб відповідає всім вимогам, є інтуїтивно зрозумілим для користувача, легко налаштовується та забезпечує збереження важливих даних як локально, так і в хмарі.