ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка програмного застосунку для резервного копіювання та моніторингу бекапів мовою Python»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
(код, найменування спеціальності)
освітньо-професійної програми «Інженерія програмного забезпечення»
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Олексій ШЕРЕМЕТЄВ
(підпис)

Виконав: здобувач вищої освіти групи ПД-44

Олексій ШЕРЕМЕТЄВ

Керівник: Олена НЕГОДЕНКО
к.т.н., доцент

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Шереметєву Олексію Михайловичу

1. Тема кваліфікаційної роботи: «Розробка програмного застосунку для резервного копіювання та моніторингу бекапів мовою Python»

керівник кваліфікаційної роботи к.т.н., доцент Олена НЕГОДЕНКО

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «10» квітня 2024 р. №.

2. Строк подання кваліфікаційної роботи «20» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, вимоги до програмного забезпечення.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд існуючих рішень для резервного копіювання та моніторингу бекапів.
2. Проектування програмного застосунку.
3. Розробка програмного застосунку.
4. Аналіз експлуатації та можливих застосувань.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Блок-схема резервного копіювання.
3. Діаграма прецедентів.
4. Діаграма класів.
5. Екранні форми.

6. Дата видачі завдання «10» квітня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	10.04-20.04.24	
2	Вивчення матеріалів для аналізу існуючих застосунків для резервного копіювання та моніторингу бекапів	20.04-25.04.24	
3	Моделювання вимог до програмного забезпечення	25.04-28.04.24	
4	Реалізація програмного застосунку	28.04-05.05.24	
5	Аналіз експлуатації та можливих застосувань	05.05-10.05.24	
6	Тестування програмного застосунку	10.05-15.05.24	
7	Оформлення роботи: вступ, висновки, реферат	15.05-19.05.24	
8	Розробка демонстраційних матеріалів	19.05-20.05.24	

Здобувач вищої освіти _____ Олексій ШЕРЕМЕТЄВ
(підпис)

Керівник
кваліфікаційної роботи _____ Олена НЕГОДЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 60 стор., 1 табл., 17 рис., 34 джерел.

Мета роботи – автоматизації резервного копіювання та моніторингу бекапів за допомогою застосунку мовою Python.

Об'єкт дослідження – процес резервного копіювання та моніторингу бекапів.

Предмет дослідження – програмне забезпечення для резервного копіювання та моніторингу бекапів.

Короткий зміст роботи: У роботі проведено аналіз можливостей автоматизованих систем резервного копіювання та моніторингу бекапів. Визначено надійний тип резервного копіювання. Розроблено функціональні та нефункціональні вимоги до застосунку для резервного копіювання та моніторингу бекапів. Спроектовано архітектуру, визначити класи та методи для створення застосунку. Використано мову Python, фреймворк Customtkinter, бібліотеки: OS, paramiko, Matplotlib. Проведено модульне та ручне тестування застосунку. В модульному тесті було написано unittest для функцій копіювання.

КЛЮЧОВІ СЛОВА: АВТОМАТИЗОВАНІ СИСТЕМИ, РЕЗЕРВНЕ КОПІЮВАННЯ, ЗАСТОСУНОК МОВОЮ PYTHON, ТЕСТУВАННЯ ЗАСТОСУНКУ, ФРЕЙВОРК GUSTOMTKINTER.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД	-	База Даних
AMQP	-	Advanced Message Queuing Protocol
REST	-	Representational State Transfer
UI	-	Інтерфейс користувача
NUI	-	Природний інтерфейс користувача
GUI	-	Графічний інтерфейс користувача
EIP	-	Шаблони корпоративної інтеграції
VDI	-	Мережеве налаштування
TUI	-	Відчутний інтерфейс користувача

ВСТУП

Резервне копіювання даних має вирішальне значення для забезпечення безпеки та доступності важливих файлів. Однак із збільшенням обсягу даних процес резервного копіювання може стати трудомістким і неефективним.

Обсяг створюваних даних зростає з кожним днем. Нас оточує багато різних джерел інформації, які ми використовуємо у повсякденному житті. Тому виникає потреба того чи іншого рішення в коректній та швидкій обробці.

Великі дані вже сьогодні змінюють бізнес кількома способами. Досліджено, що щодня створюється 2,5 квінтильйона байтів даних і в майбутньому це число тільки зросте.

У контексті розробки програмного застосунку для резервного копіювання та моніторингу бекапів, важливо звернути увагу на найновіші технологічні досягнення та методології, які можуть бути інтегровані у наш проект. Однією з ключових технологій є використання Python як основної мови програмування, що дозволяє ефективно взаємодіяти з різними бібліотеками та фреймворками для реалізації необхідних функціональностей.

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ДЛЯ РЕЗЕРВНОГО КОПІЮВАННЯ ТА МОНІТОРИНГУ БЕКАПІВ

У сучасному цифровому середовищі, де дані відіграють вирішальну роль в успіху та безперервності бізнесу та організацій, забезпечення їх безпеки та доступності має першочергове значення.

Резервне копіювання даних і аварійне відновлення є важливими компонентами будь-якої надійної IT-стратегії, оскільки вони забезпечують захист від втрати даних, системних збоїв, стихійних лих або зловмисних атак. Традиційно резервне копіювання даних і аварійне відновлення включають створення копій критично важливих даних і їх зберігання в локальних системах зберігання або зовнішніх пристроях.

Резервне копіювання даних відноситься до процесу створення та збереження копій файлів даних і інформації для захисту від випадкової або навмисної втрати даних, системних збоїв, збоїв у роботі обладнання, кібератак або стихійних лих. Це передбачає створення дублікатів даних і їх зберігання в окремих місцях або на пристроях зберігання, щоб забезпечити доступність даних і полегшити відновлення в разі втрати або пошкодження даних.

Аварійне відновлення — це набір стратегій, процесів і процедур, реалізованих для відновлення критично важливих систем, даних та інфраструктури після руйнівної події або катастрофи. Це передбачає якнайшвидше відновлення та відновлення бізнес-операцій, щоб мінімізувати час простою, зменшити фінансові втрати та зберегти безперервність бізнесу. Аварійне відновлення охоплює не лише відновлення даних, але й відновлення додатків, мережевої інфраструктури та інших IT-ресурсів, необхідних для нормальної роботи бізнесу.

Потрібно враховувати важливість резервного копіювання даних і аварійного відновлення в хмарі. Резервне копіювання даних і аварійне відновлення в хмарі є надзвичайно важливими для організацій через такі причини: захист даних і

доступність: дані є критично важливим активом для бізнесу, і їх втрата або недоступність може мати тяжкі наслідки. Хмарне резервне копіювання гарантує надійне зберігання даних за межами сайту, захищаючи їх від фізичного пошкодження, крадіжки або локальних катастроф. У разі втрати даних увімкнеться хмарне аварійне відновлення

У контексті хмари резервне копіювання даних і аварійне відновлення використовують хмарні служби та інфраструктуру для зберігання резервних копій даних і полегшення процесів відновлення. Хмара пропонує масштабованість, резервування та віддалений доступ, що робить її цінною платформою для резервного копіювання даних і рішень для аварійного відновлення

Проте з появою хмарних обчислень організації тепер мають можливість використовувати потужність і масштабованість хмари для покращення захисту своїх даних і можливостей відновлення. Резервне копіювання даних у хмарі – це процес зберігання копій даних на віддалених хмарних серверах, усуваючи потребу у фізичній інфраструктурі зберігання. Цей підхід пропонує численні переваги, включаючи масштабованість, економічну ефективність і автоматичне резервне копіювання, гарантуючи, що дані залишаються захищеними та легко відновлюваними.

Аварійне відновлення в хмарі зосереджено на здатності швидко відновлювати критично важливі системи та дані у разі аварії чи збою. Використовуючи хмарну інфраструктуру, організації можуть мінімізувати час простою, зменшити цільовий час відновлення (RTO) і забезпечити безперервність бізнесу. Притаманна хмарі гнучкість і резервна архітектура роблять її ідеальним рішенням для аварійного відновлення.

Резервне копіювання та аварійне відновлення даних у хмарі є надзвичайно важливими для організацій через такі причини: захист даних і доступність: дані є критично важливим активом для бізнесу, а їх втрата або недоступність може мати серйозні наслідки. Хмарне резервне копіювання гарантує надійне зберігання

даних за межами сайту, захищаючи їх від фізичного пошкодження, крадіжки або локальних катастроф.

У разі втрати даних увімкнеться хмарне аварійне відновлення організацій, щоб швидко відновити дані та відновити роботу, мінімізуючи час простою та підтримуючи безперервність бізнесу.

Розглянемо зменшення ризиків. Організації стикаються з різними ризиками, включаючи збої обладнання, кібератаки, людські помилки та стихійні лиха. Хмарне резервне копіювання та аварійне відновлення забезпечують захист від цих ризиків. Регулярно створюючи резервні копії даних у хмарі, організації знижують ризик безповоротної втрати даних і можуть швидко відновлюватися в разі несподіваних подій.

Масштабованість і гнучкість: хмарні рішення пропонують масштабованість і гнучкість, дозволяючи організаціям легко налаштовувати свої можливості резервного копіювання та відновлення відповідно до своїх мінливих потреб. Постачальники хмарних послуг можуть пристосуватись до збільшених обсягів даних і надавати ресурси зберігання на вимогу, гарантуючи, що резервне копіювання є комплексним, а процеси відновлення можуть впоратися зі зростаючими вимогами до даних.

Економічна ефективність: традиційні методи резервного копіювання даних і аварійного відновлення часто передбачають значні початкові інвестиції в обладнання, інфраструктуру та обслуговування. Хмарні рішення усувають або скорочують ці капітальні витрати, дозволяючи організаціям використовувати модель оплати за використання. Це забезпечує економію коштів за рахунок усунення необхідності інвестування в спеціальну інфраструктуру резервного копіювання, оптимізації використання ресурсів і зниження операційних витрат.

Автоматизація та ефективність. Рішення для резервного копіювання та аварійного відновлення в хмарі часто пропонують такі функції автоматизації, як резервне копіювання за розкладом, додаткове резервне копіювання та автоматизовані процеси відновлення. Ці можливості автоматизації оптимізують

робочі процеси захисту та відновлення даних, зводячи до мінімуму ручне втручання та зменшуючи ймовірність помилок. Автоматизація також підвищує ефективність, звільняючи IT-персонал, щоб зосередитися на інших критичних завданнях.

Відповідність і нормативні вимоги. Багато галузей мають нормативні рамки та стандарти відповідності, які вимагають резервного копіювання даних і планів аварійного відновлення. Хмарні рішення часто пропонують функції та сертифікати, які відповідають цим вимогам вимоги, допомагаючи організаціям дотримуватися правил захисту даних і галузевих стандартів відповідності.

Резервне копіювання даних і аварійне відновлення в хмарі пропонують покращений захист даних, покращені можливості відновлення, масштабованість, економічну ефективність, автоматизацію та дотримання вимог. Використовуючи хмарні рішення, організації можуть зменшити ризики, забезпечити доступність даних і підтримувати безперервність роботи в умовах несподіваних подій.

Наведемо переваги резервного копіювання в хмарі. Резервне копіювання в хмарі пропонує організаціям кілька переваг, зокрема: масштабованість. Хмарне резервне копіювання дозволяє організаціям масштабувати свої потреби в сховищі відповідно до обсягу даних, які вони генерують.

Постачальники хмарних послуг пропонують гнучкі варіанти зберігання, що дозволяє підприємствам легко збільшувати або зменшувати обсяг пам’яті за потреби. Така масштабованість усуває необхідність попередніх інвестицій у додаткове обладнання та інфраструктуру.

Економічна ефективність. Хмарне резервне копіювання позбавляє організацій від необхідності інвестувати та підтримувати власну фізичну інфраструктуру резервного копіювання. Це усуває початкові витрати на обладнання та технічне обслуговування, а також пов’язані витрати, такі як електроенергія, охолодження та фізичне зберігання. Натомість організації оплачують послуги хмарного резервного копіювання на основі передплати або

оплати за використання, як правило, на основі обсягу збережених даних, що призводить до економії коштів і передбачуваних витрат.

Автоматичне резервне копіювання: рішення для резервного копіювання в хмарі часто пропонують автоматизовані процеси резервного копіювання. Організації налаштовують резервне копіювання за розкладом, забезпечуючи регулярне та послідовне резервне копіювання даних без ручного втручання. Ця автоматизація зменшує ризик людської помилки та гарантує, що критично важливі дані завжди захищені.

Резервування та реплікація даних: постачальники хмарних послуг часто мають кілька центрів обробки даних, розташованих у різних географічних регіонах. Ця надлишковість і реплікація даних гарантують, що резервні копії зберігаються в кількох місцях, зменшуючи ризик втрати даних через локальні катастрофи або збої. Це підвищує доступність і стійкість даних, збільшуючи шанси на успішне відновлення даних у разі потреби.

Віддалений доступ: резервне копіювання в хмарі дозволяє організаціям отримувати доступ до своїх резервних копій даних з будь-якого місця, де є підключення до Інтернету. Віддалена доступність особливо цінна у ситуаціях, коли локальні дані недоступні через фізичні обмеження або катастрофи. Це дозволяє організаціям швидко отримувати дані та відновлювати роботу з будь-якого місця, підвищуючи безперервність бізнесу.

Безпека даних. Хмарні рішення для резервного копіювання використовують надійні заходи безпеки для захисту даних. Ці заходи часто включають шифрування, контроль доступу, механізми автентифікації та відповідність галузевим стандартам сертифікації безпеки.

Постачальники хмарних послуг мають спеціалізовані групи безпеки та ресурси для забезпечення захисту даних, зменшення ризику їх витоку чи несанкціонованого доступу. Контроль версій даних і відновлення на певний момент часу.

Хмарні рішення для резервного копіювання зазвичай пропонують такі функції, як керування версіями даних і відновлення на певний момент часу. Управління версіями даних дозволяє організаціям зберігати кілька версій одного файлу або даних, уможливлюючи відновлення до певного моменту часу. Ця функція є цінною в сценаріях, пов’язаних із пошкодженням даних, випадковим видаленням або атаками програм-вимагачів, оскільки вона забезпечує можливість відновлення до завідомо справного стану.

Простота керування. Хмарні рішення для резервного копіювання часто забезпечують зручні інтерфейси та консолі керування, що спрощує налаштування, моніторинг і керування резервними копіями. Організації можуть легко налаштувати політику резервного копіювання, контролювати стан резервного копіювання та виконувати операції відновлення за допомогою інтуїтивно зрозумілих інтерфейсів, зменшуючи складність і час, необхідний для керування резервним копіюванням.

Загалом хмарне резервне копіювання забезпечує масштабованість, економічну ефективність, автоматизацію, резервування, віддалений доступ, безпеку даних і легкість керування.

Розглянемо типи хмарного резервного копіювання. Є кілька типів методів хмарного резервного копіювання, які організації використовують відповідно до своїх конкретних потреб і вимог.

До поширених типів хмарного резервного копіювання належать: повне резервне копіювання: повне резервне копіювання передбачає створення повної копії всіх даних і файлів у середовищі організації. Ця початкова резервна копія фіксує всі дані, включаючи файли, бази даних, програми та конфігурації системи.

Подальші операції резервного копіювання зосереджені на записі змін або оновлень, зроблених після останнього повного резервного копіювання. Повне резервне копіювання забезпечує повний захист даних, але може зайняти багато часу та значні ресурси для зберігання.

Інкрементне резервне копіювання: Інкрементне резервне копіювання фіксує лише зміни, внесені в дані з моменту останнього резервного копіювання, незалежно від того, було це повне резервне копіювання чи інше інкрементне резервне копіювання.

У цьому методі резервне копіювання створюється лише для змінених або новостворених файлів, що пришвидшує операції резервного копіювання та зменшує вимоги до пам’яті. Однак повне відновлення даних може тривати довше, оскільки вимагає відновлення останньої повної резервної копії, а потім послідовного застосування наступних інкрементних резервних копій.

Диференціальне резервне копіювання: диференціальне резервне копіювання фіксує всі зміни, внесені в дані з часу останнього повного резервного копіювання. На відміну від інкрементного резервного копіювання, яке фіксує зміни лише з моменту останнього резервного копіювання (повного чи інкрементного), диференціальне резервне копіювання фіксує зміни з моменту останнього повного резервного копіювання.

Цей метод зменшує кількість наборів резервних копій, необхідних для відновлення даних, прискорюючи відновлення останньої повної резервної копії та останньої диференціальної резервної копії. Однак з часом розмір диференціальної резервної копії збільшується, потенційно вимагаючи більше місця для зберігання.

Безперервний захист даних (CDP): пропонує резервне копіювання даних у режимі реального або майже в реальному часі. Він фіксує та відтворює кожну зміну, внесену до даних, гарантуючи, що резервні копії завжди актуальні. CDP мінімізує втрату даних шляхом постійного моніторингу та резервного копіювання змін, забезпечуючи детальну точку відновлення (RPO). Цей метод особливо підходить для організацій, яким потрібна мінімальна втрата даних і висока доступність.

Резервне копіювання миттєвих знімків: резервне копіювання миттєвих знімків використовує концепцію миттєвих знімків на певний момент часу, щоб зафіксувати стан даних у певний момент. Він створює копію даних у їх поточному

стані, дозволяючи організаціям повернутися до цього конкретного моменту, якщо це необхідно. Знімки зазвичай робляться через регулярні проміжки часу, що дозволяє швидко відновити дані до певного моменту часу.

Однак важливо зазначити, що миттєві знімки зазвичай зберігаються в одному хмарному середовищі й можуть не забезпечувати такий самий рівень захисту даних, як резервні копії за межами сайту.

Гібридне резервне копіювання: поєднує в собі локальні та хмарні методи резервного копіювання. Він передбачає створення резервних копій як локально, так і в хмарі, пропонуючи переваги обох підходів. Гібридне резервне копіювання надає перевагу швидкого локального резервного копіювання та відновлення для негайного відновлення даних, а також забезпечує зовнішній захист даних і можливості аварійного відновлення за допомогою хмарного резервного копіювання.

Організації можуть вибрати найбільш підходящий тип хмарного резервного копіювання на основі таких факторів, як розмір даних, вимоги до відновлення, доступна пропускна здатність, вартість зберігання та бажані цілі часу відновлення (RTO) і цілі точки відновлення (RPO). У багатьох випадках для досягнення комплексної та ефективної стратегії резервного копіювання можна використовувати комбінацію типів резервного копіювання.

Найкращі методи резервного копіювання даних у хмарі Застосування найкращих практик резервного копіювання даних у хмарі може допомогти організаціям забезпечити ефективність, безпеку та надійність процесів резервного копіювання.

Наведемо кілька ключових найкращих практик, які слід враховувати: Визначте стратегію резервного копіювання: розробіть чітку стратегію резервного копіювання, яка відповідає потребам вашої організації щодо захисту даних і вимогам бізнесу.

Визначте критичні дані, визначте частоту резервного копіювання, періоди зберігання та встановіть цілі відновлення (RTO та RPO).

Класифікація даних: класифікуйте свої дані на основі їх чутливості та критичності. Не всі дані вимагають однакового рівня захисту, тому визначте пріоритети та розподіліть відповідні ресурси резервного копіювання та заходи безпеки відповідно.

Регулярні розклади резервного копіювання: встановіть регулярні розклади резервного копіювання, щоб забезпечити постійне та часте резервне копіювання даних. Щоб визначити оптимальний графік резервного копіювання, враховуйте такі фактори, як непостійність даних, частота оновлення та бізнес-операції

Кілька резервних копій: створюйте кілька копій своїх резервних копій даних, щоб забезпечити резервування та зменшити ризик втрати або пошкодження даних. Зберігати резервні копії потрібно в різних географічних місцях або в різних постачальників хмарних послуг для додаткового захисту.

Шифрування: шифрувати дані резервної копії потрібно, як під час передачі, так і в стані спокою, щоб захистити їх від несанкціонованого доступу.

Використовуйте надійні алгоритми шифрування та безпечно керуйте ключами шифрування, щоб зберегти конфіденційність і цілісність даних. Тестуйте процеси резервного копіювання та відновлення.

Регулярно тестуйте процеси резервного копіювання та відновлення, щоб переконатися, що резервне копіювання є успішним і дані можна ефективно відновити. Проводьте тренування з відновлення та симулюйте різні сценарії аварій, щоб підтвердити цілісність і надійність ваших систем резервного копіювання.

Моніторинг і попередження: запровадьте механізми моніторингу та попередження для проактивного відстеження стану резервних копій, виявлення будь-яких збоїв або аномалій і оперативного вирішення проблем. Регулярно переглядайте журнали резервного копіювання та звіти, щоб переконатися, що резервне копіювання виконується належним чином.

Контроль версій і збереження: потрібно зберігати контроль версій своїх резервних копій, щоб за потреби забезпечити відновлення на певний момент часу.

Встановіть політику збереження, щоб визначити, як довго мають зберігатися дані резервного копіювання на основі вимог відповідності та бізнес-потреб.

Планування аварійного відновлення: потрібно інтегрувати свою стратегію хмарного резервного копіювання з комплексним планом аварійного відновлення. Визначте процедури відновлення, розставте пріоритети для критично важливих систем і даних, а також установіть цільовий час відновлення (RTO) і цілі точки відновлення (RPO), щоб керувати вашими зусиллями з відновлення.

Регулярні оновлення та виправлення: потрібно оновлювати свої системи резервного копіювання та програмне забезпечення за допомогою останніх виправлень безпеки та оновлень. Регулярно перевіряйте та вдосконалюйте свою інфраструктуру резервного копіювання, щоб використовувати нові функції та вдосконалення.

Вибір постачальника та угоди про рівень обслуговування (SLA): потрібно обирати авторитетного постачальника хмарних послуг, який пропонує надійні заходи безпеки, надійність і сертифікати відповідності. Перегляньте та обговоріть угоди про рівень обслуговування (SLA), щоб переконатися, що вони відповідають вашим вимогам до резервного копіювання та відновлення.

Навчання та обізнаність співробітників: потрібно навчати співробітників щодо важливості резервного копіювання даних, протоколів безпеки та їхніх ролей і обов’язків у процесі резервного копіювання. Регулярне навчання співробітників передовим практикам, процедурам обробки даних і реагування на інциденти, призводить до мінімізації ризиків людських помилок. Дотримуючись найкращих практик, організації створюють надійну стратегію резервного копіювання даних у хмарі, яка гарантує доступність, цілісність і можливість відновлення даних у разі втрати даних, катастроф або збоїв.

Також потрібно враховувати потенційні вузькі місця введення-виведення та дослідити способи їх мінімізації для всебічного підвищення швидкості резервного копіювання.

Для резервного копіювання на різних пристроях і платформах, щоб відстежувати та керувати рішеннями існує три кроки до відновлення даних із резервних копій.

Першим кроком до відновлення даних із резервних копій є вибір правильного рішення для резервного копіювання для ваших потреб. Існують різні типи рішень для резервного копіювання, наприклад локальні, хмарні або гібридні, які пропонують різні рівні безпеки, доступності та масштабованості.

Слід враховувати такі фактори, як розмір і частота резервного копіювання, чутливість і доступність ваших даних, а також вартість і складність рішення для резервного копіювання. Наприклад, якщо є великі обсяги даних, для яких потрібно регулярно створювати резервні копії, можемо скористатися хмарним рішенням для резервного копіювання, яке зможе впоратися з вимогами до сховища та пропускної здатності.

З іншого боку, якщо є конфіденційні дані, які потрібно зашифрувати та захистити від несанкціонованого доступу, можемо скористатися локальним або гібридним рішенням для резервного копіювання, яке надасть більше контролю та гнучкості.

Наступним кроком до відновлення даних із резервних копій є створення плану резервного копіювання, який визначає ваші цілі, політику та процедури резервного копіювання. План резервного копіювання має включати такі аспекти, як обсяг і частота ваших резервних копій, політика збереження та видалення ваших даних резервного копіювання, ролі та обов’язки вашої команди резервного копіювання, а також цілі та стратегії відновлення вашого рішення для резервного копіювання.

План резервного копіювання також має документувати інструменти резервного копіювання та програмне забезпечення, які ви використовуєте, розташування та формати резервних копій, які ви зберігаєте, а також методи перевірки та тестування резервних копій, які ви використовуєте. План резервного

копіювання може допомогти вам переконатися, що ваші резервні копії є послідовними, надійними та такими, що можна відновити.

Відстежуйте статус резервного копіювання

Третій крок до відновлення даних із резервних копій — це регулярний моніторинг стану та продуктивності резервного копіювання. Ви повинні використовувати інструменти та програмне забезпечення, які можуть відстежувати та повідомляти про стан ваших резервних копій, як-от рівень завершення, відмов або помилок, тривалість і швидкість резервного копіювання, розмір і зростання резервної копії, а також цілісність і якість резервної копії.

Також потрібно використовувати інструменти та програмне забезпечення, які можуть сповіщати про будь-які проблеми чи аномалії у резервних копіях, як-от відсутні, пошкоджені або застарілі резервні копії, несанкціонований доступ або зміна резервних копій або потенційні загрози чи ризики для резервних копій. Відстеження стану резервного копіювання може допомогти виявити та вирішити будь-які проблеми чи прогалини у рішенні для резервного копіювання.

Підсумовуючи, впровадження хмарних рішень для резервного копіювання даних і аварійного відновлення пропонує значні переваги для організацій. Використовуючи хмару, організації можуть покращити доступність даних, скоротити час відновлення, підвищити ефективність витрат і досягти масштабованості та гнучкості.

Хмара забезпечує посилені заходи безпеки, спрощене керування та полегшує регулярне тестування та перевірку плану аварійного відновлення. Географічна надмірність і безперервність інновації хмарних провайдерів ще більше посилюють стійкість можливостей захисту даних організацій.

Загалом, хмарні рішення для резервного копіювання даних і аварійного відновлення дають змогу організаціям захищати критично важливі дані, підтримувати безперервність бізнесу та ефективно пом’якшувати вплив непередбачених катастроф або збоїв.

З огляду на необхідність обробки великих обсягів даних, розроблений програмний застосунок повинен бути легко масштабованим. Тут Python надає великі можливості завдяки своїм багатопоточним та асинхронним можливостям. З використанням асинхронного програмування через asyncio та інші схожі бібліотеки, можливо ефективно збільшувати кількість одночасно оброблюваних завдань без збільшення витрат на апаратне забезпечення.

Завдяки своїй гнучкості, Python добре підходить для роботи з хмарними API. Інтеграція з хмарними платформами, такими як AWS S3, Google Cloud Storage та Microsoft Azure Blob Storage, забезпечує не лише зберігання даних, а й високу доступність та масштабованість рішень для резервного копіювання. Використання таких сервісів дозволяє зменшити витрати на інфраструктуру та забезпечити більш надійне управління даними. Наприклад, можна автоматично копіювати критично важливі дані на кілька географічно розподілених серверів, що значно знижує ризик втрати інформації через непередбачені обставини.

Ще одним аспектом, де Python виявляє свої переваги, є використання алгоритмів машинного навчання для аналізу та прогнозування стану резервних копій. Через бібліотеки, такі як TensorFlow та Scikit-learn, можна розробити моделі, що здатні ідентифікувати потенційні проблеми в резервних копіях до того, як вони стануть критичними. Таким чином, система може автоматично повідомляти адміністратора про помилки або неефективності у процесах копіювання та відновлення даних, що дозволяє своєчасно вирішувати можливі проблеми.

Завершуючи огляд сучасних підходів до резервного копіювання та моніторингу, стає очевидним, що використання Python як інструментарію дає змогу не тільки реалізувати базові потреби, а й впровадити передові технологічні рішення.

1.1 Поняття програмного застосунку для резервного копіювання та моніторингу бекапів

Інтерфейс користувача визначає успіх програмного забезпечення. Кілька факторів, таких як простота використання, ефективність, доступність і естетичність, впливають на те, як користувачі без плутанини орієнтуються в системі, легко знаходять функції та отримують відповідний відгук про свої дії. Для покращення взаємодії з користувачем добре спроектований інтерфейс користувача повинен відповідати всім, тому для компаній вкрай важливо віддавати перевагу дизайну інтерфейсу користувача.

Поняття програмного додатку для резервного копіювання та моніторингу резервних копій відноситься до комплексного рішення, призначеного для захисту цифрових даних від втрати, пошкодження або несанкціонованого доступу. Цей тип додатків служить критично важливим компонентом у забезпеченні безпеки даних і безперервності бізнесу, автоматизуючи процес створення, зберігання і управління резервними копіями важливих файлів, баз даних і системних конфігурацій. Основна мета програми для резервного копіювання та моніторингу резервних копій – забезпечити надійний та ефективний засіб захисту цінних інформаційних активів, що дозволяє організаціям і приватним особам швидко і з мінімальними втратами відновлюватися пілся потенційних інцидентів пов’язаних з втратою даних.

Добре розроблений додаток для резервного копіювання та моніторингу резервних копій пропонує ряд ключових функцій і переваг, які сприяють його ефективності в захисті цифрової інформації. Однією з найважливіших функцій є можливість виконувати регулярне та автоматизоване резервне копіювання за заздалегідь визначеним розкладом або тригерами. Це гарантує постійне резервне копіювання найактуальніших версій критично важливих даних без необхідності ручного втручання знижуючи ризик людських помилок та заощаджуючи час і ресурси.

Додаток повинен підтримувати різні типи резервного копіювання такі як повне, інкрементне та диференційоване, дозволяючи користувачам знаходити баланс між частотою резервного копіювання обсягом пам’яті та швидкістю відновлення. Ще одним важливим аспектом програми для резервного копіювання та моніторингу резервних копій є її здатність зберігати резервні копії безпечно.

Додатково, програма для резервного копіювання та моніторингу повинна включати функції для захисту даних, такі як шифрування та автентифікація доступу. Шифрування гарантує, що всі збережені резервні копії захищені від несанкціонованого доступу, навіть якщо фізичні носії даних будуть скомпрометовані або втрачені. Автентифікація доступу забезпечує, що лише авторизовані користувачі можуть виконувати резервне копіювання та відновлення даних, що додатково підвищує рівень безпеки системи.

Інтеграція з існуючими інформаційними системами та платформами також є ключовою характеристикою ефективного програмного забезпечення для резервного копіювання. Сумісність з різноманітними операційними системами, базами даних та додатками дозволяє програмі гармонійно взаємодіяти з іншими системами без необхідності значних модифікацій або конфігурацій. Це не тільки спрощує процес впровадження рішення у корпоративне середовище, але й забезпечує гнучкість у виборі стратегій резервного копіювання, що відповідають специфічним потребам організації.

Також важливим є забезпечення моніторингу та звітності в реальному часі, які допомагають користувачам відстежувати статус резервних копій та швидко реагувати на будь-які проблеми чи збої. Системи повинні надавати детальні звіти про стан резервних копій, історію виконаних операцій, результати верифікаційних перевірок та інші критично важливі параметри. Наявність добре структурованих засобів звітності і аналітики дозволяє керівництву компанії та IT-фахівцям приймати обґрунтовані рішення з оптимізації та покращення процесів резервного копіювання.

Розширення функціональності програмного додатку через плагіни або модулі є ще однією перевагою, яка дозволяє адаптувати систему під змінювані потреби користувачів. Це може включати підтримку нових типів даних, інтеграцію з додатковими хмарними сервісами або розширення можливостей шифрування. Використання модульної архітектури забезпечує можливість швидко впроваджувати нові функції без переробки основної частини системи.

1.2 Аналіз існуючих застосунків для резервного копіювання та моніторингу бекапів

У цьому підрозділі ми розглядаємо ключові аспекти, які загально характеризують сучасні програмні застосунки для резервного копіювання та моніторингу. Основна увага приділяється загальним тенденціям та вимогам, які впливають на розвиток та ефективність таких систем.

Забезпечення цілісності даних.

Важливим аспектом є здатність програм забезпечувати високий рівень цілісності даних. Це включає захист від випадкових помилок, збоїв обладнання, а також зловмисних атак.

Автоматизація процесів.

Сучасні застосунки надають широкі можливості для автоматизації процесів резервного копіювання та відновлення, що дозволяє мінімізувати втручання людини та знижувати ймовірність помилок.

Гнучкість управління.

Програми зазвичай пропонують гнучкі налаштування для різних рівнів доступу та політик безпеки, дозволяючи користувачам налаштовувати процеси згідно зі своїми потребами.

Підтримка різних середовищ.

Важливою є підтримка різноманітних оперативних систем та середовищ зберігання, включаючи хмарні сервіси, що забезпечує велику гнучкість в організації резервного копіювання.

Скалярність та ефективність.

Програмне забезпечення має бути масштабованим, щоб впоратися з ростом обсягів даних і комплексності системи, а також ефективним у використанні ресурсів.

Відновлення після збоїв.

Застосунки мають забезпечувати швидке та ефективне відновлення після збоїв, що є критично важливим для забезпечення неперервності бізнесу.

Комплексний моніторинг.

Включення розширених інструментів моніторингу дозволяє користувачам відслідковувати стан системи резервного копіювання та вчасно реагувати на можливі проблеми.

Резервне копіювання та моніторинг бекапів є критично важливими процесами для забезпечення надійності та доступності даних у цифровому світі. Ці процеси дозволяють компаніям та індивідуальним користувачам захистити свої важливі дані від випадкових втрат, збоїв обладнання, кібератак або природних катастроф. Завдяки сучасним технологіям, резервне копіювання стало не просто актом збереження даних, а комплексною системою, що включає низку взаємопов'язаних дій:

Повне резервне копіювання. Ця операція створює копію всіх даних на диску або в системі, забезпечуючи повне відновлення в разі серйозного збою.

Відновлення окремих файлів. Уміння відновлювати не тільки цілі системи, а й окремі файли, є незамінним для оперативного усунення наслідків випадкових видалень або корупції даних.

Моніторинг. Систематичне відстеження стану резервних копій та повідомлення про успішність або невдачі процесів копіювання. Це забезпечує можливість швидко реагувати на будь-які проблеми.

Ці функції є стандартними для більшості сучасних рішень для резервного копіювання, але їх ефективність може значно відрізнятися в залежності від обраного програмного забезпечення.

Різні застосунки надають різний рівень контролю, безпеки та зручності управління даними, відповідно до потреб користувачів. Наприклад, деякі програми можуть пропонувати розширені можливості моніторингу, що включають автоматичні сповіщення електронною поштою або через мобільні додатки, коли інші зосереджуються на забезпеченні більш широкого спектра опцій копіювання, включаючи підтримку різних операційних систем та пристроїв.

Крім технічних аспектів, важливо враховувати і вартісний фактор. Вартість програмного забезпечення для резервного копіювання може значно варіюватися залежно від обраної платформи, функціоналу та ліцензійної політики, що робить важливим вибір рішення, що відповідає бюджету та потребам користувача.

Наведемо перелік ефективнивних застосунків для резервного копіювання.

Zinstall FullBack: Zinstall FullBack вирізняється своєю здатністю до повного резервного копіювання та універсального відновлення, забезпечуючи користувачам Windows всебічний захист даних. Ця програма не просто робить копію даних, але і захищає від рансомваре, забезпечуючи додатковий рівень безпеки.

Можливість відновлення окремих файлів дозволяє користувачам швидко повернути втрачені або пошкоджені файли без необхідності відновлювати весь диск. Однак, висока вартість у $14.90 на місяць та обмежені можливості в роботі з мережевими ресурсами можуть бути перепоною для деяких користувачів. Програма Zinstall FullBack представлена на рис.1.

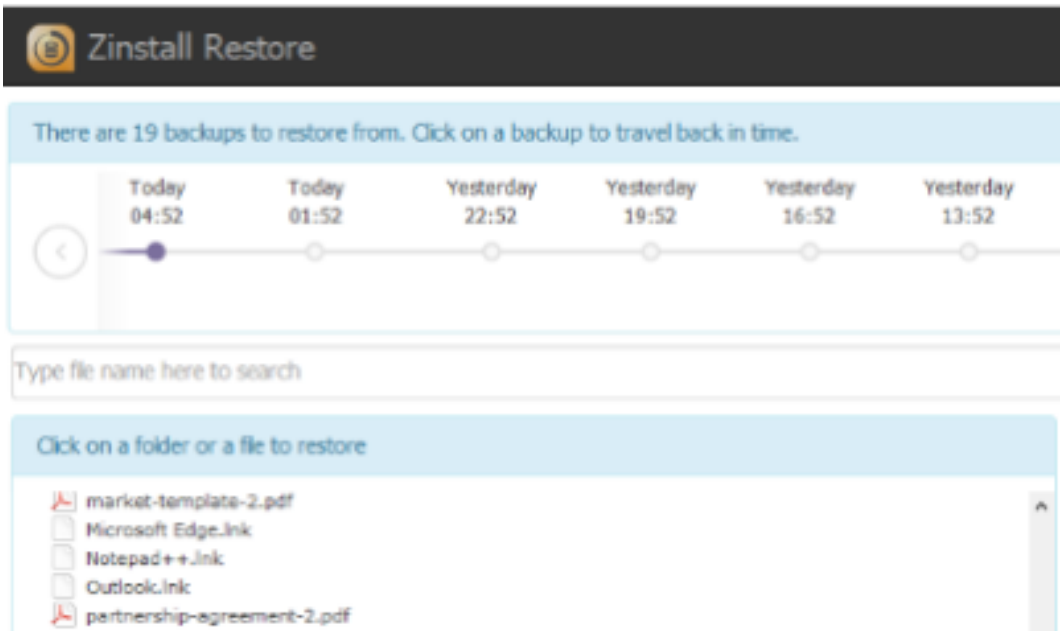


Рис.1. Програма Zinstall FullBack

Aomei Backupper Professional є потужним інструментом для резервного копіювання та відновлення, який підтримує повне резервне копіювання та відновлення окремих файлів. Програма добре підходить для користувачів Windows, які потребують надійного рішення для захисту своїх даних.

Наведемо деякі потенційні недоліки, які можуть вплинути на рішення про вибір цієї програми:

Обмежена підтримка операційних систем: Aomei Backupper Professional працює лише на Windows, що може бути обмеженням для середовищ, де використовуються інші операційні системи, такі як macOS або Linux.

Обмежена інтеграція з мережевими ресурсами: програма підтримує резервне копіювання на мережеві диски, її можливості інтеграції можуть бути не настільки гнучкими порівняно з іншими рішеннями, які спеціалізуються на роботі в мережевих та хмарних середовищах.

Программа Aomei Backupper Professional представлена на рис.1.1.

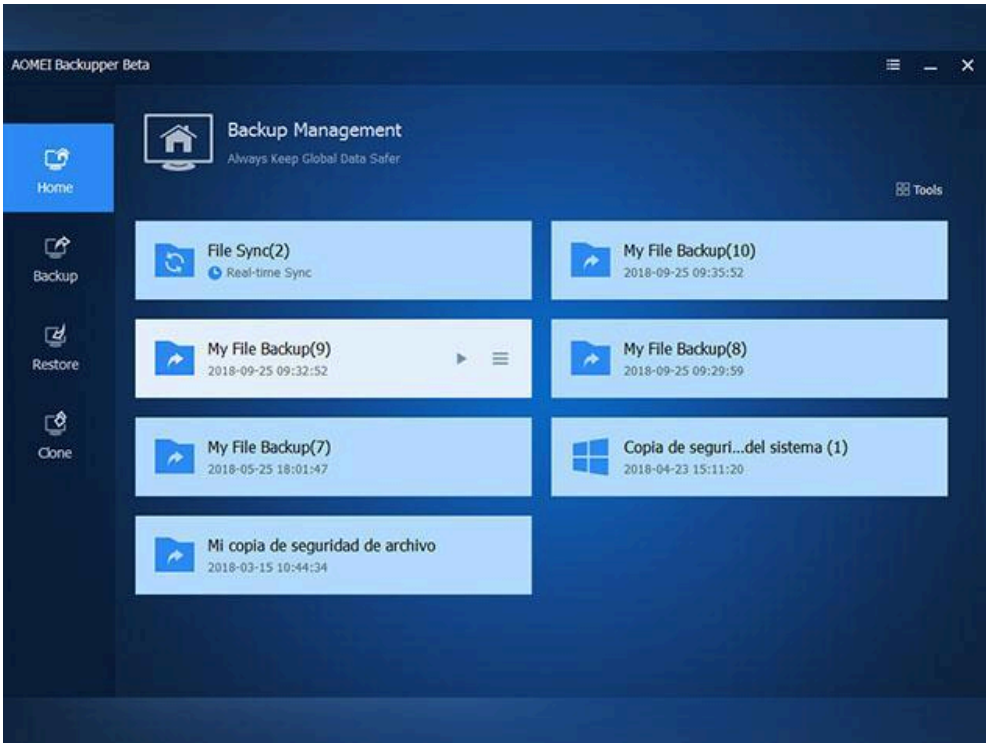


Рис.1.1. Програма Aomei Backupper Professional

Вартість: ціна у $49.95 може бути високою для деяких користувачів, особливо для тих, хто шукає більш бюджетні рішення для резервного копіювання. Важливість розгляду недолікив пов'язана з їх можливістю вплинути на здатність програми відповідати конкретним вимогам користувачів у різних середовищах.

Вибір програми для резервного копіювання повинен враховувати не тільки її функціональність, але й сумісність з існуючою IT-інфраструктурою та бюджетні обмеження [2].

Наступний застосунок EaseUS Todo Backup пропонує гнучкі та доступні опції для резервного копіювання та відновлення даних, що робить його популярним вибором серед домашніх та невеликих офісних користувачів. Програма підтримує повне резервне копіювання і відновлення окремих файлів, а також пропонує безкоштовну версію для основних потреб копіювання. Програма EaseUS Todo Backup представлена на рис.1.2.

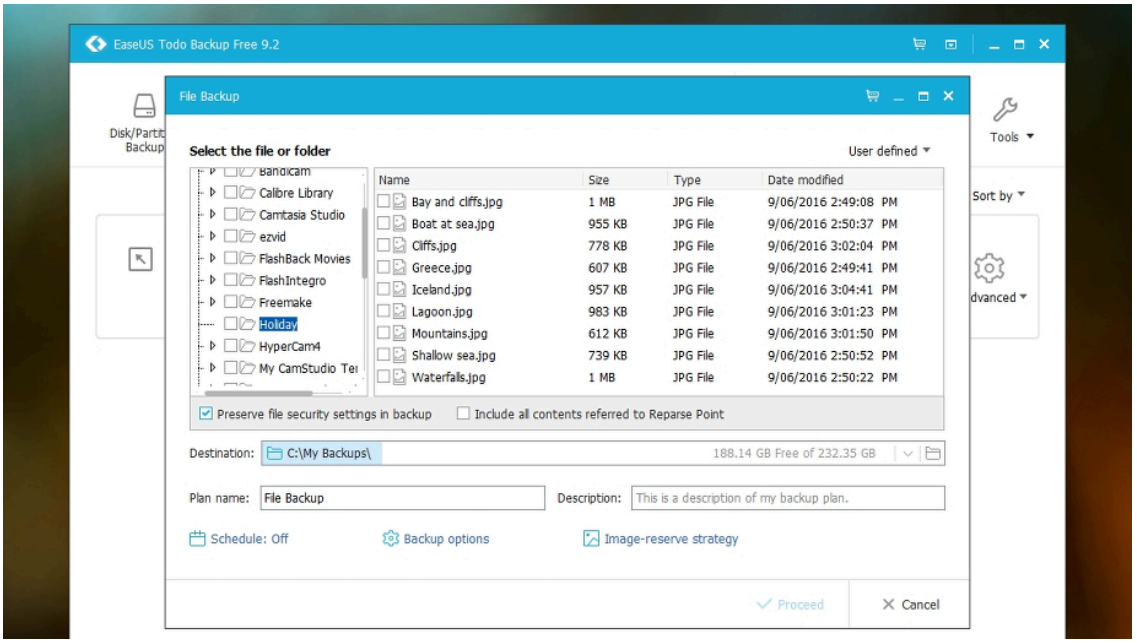


Рис.1.2. Програма Aomei Backupper Professional

Представимо перелік недоліків, які варто врахувати:

Обмежена підтримка операційних систем: Podobnie jak Aomei Backupper Professional, EaseUS Todo Backup працює лише на Windows, що може не відповідати потребам середовищ, які використовують інші операційні системи.

Функціональні обмеження в безкоштовній версії: наявність безкоштовної версії є великою перевагою, вона може мати обмежену функціональність і не включати всі потрібні опції копіювання та відновлення, які доступні в платних версіях.

Проблеми зі стабільністю: деякі користувачі зазначають, що стикаються з проблемами стабільності під час використання програми, особливо при роботі з великими обсягами даних або при спробах відновлення системи.

Перелік обраних недоліків може вплинути на вибір програми для певних користувачів, особливо якщо їх потреби в резервному копіюванні вимагають більшої гнучкості або стабільності. Вибір програмного забезпечення для резервного копіювання має враховувати сумісність із середовищем користувача, його технічні вимоги та вартість рішення [3].

Наведемо приклади найпопулярніших представлених застосунків для резервного копіювання за 2024 рік в таблиці 1.1 .

Таблиця 1.1

Порівняння існуючих аналогів

Функція / Програма	**Zinstall FullBack**	**Aomei Backupper Professional**	**EaseUS Todo Backup**	**EZBackup Done**
Повне резервне копіювання	+	+	+	+
Відновлення окремих файлів	+	+	+	+
Операційна система	Тільки Windows	Тільки Windows	Тільки Windows	Windows, Mac, Linux
Частота резервних копій	Налаштування не вказані	Обмежене налаштування	Обмежене налаштування	Повна гнучкість
Робота з мережевими ресурсами	Локально	Локально, глобально	Локально, глобально	Локально, глобально
Перевірка Firewall	Ні	Ні	Ні	Так
Графік навантаження	Ні	Ні	Так	Так

ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ

2.1 Моделювання моніторингу резервного копіювання

При розробці програмного забезпечення для резервного копіювання та моніторингу, моделювання програмного забезпечення відіграє вирішальну роль у забезпеченні успіху та ефективності кінцевого продукту.

Функціональні вимоги якими повинен володіти додаток для резервного копіювання та моніторингу резервних копій, щоб відповідати своєму призначенню.

Представимо перелік вимог. Однією з основних функціональних вимог є можливість виконувати автоматичне резервне копіювання на основі заздалегідь визначених розкладів або тригерів. Програма повинна дозволяти користувачам налаштовувати завдання резервного копіювання вказуючи вихідні дані, сховище призначення та частоту резервного копіювання. Така функціональність гарантує регулярний захист критично важливих даних без необхідності ручного втручання, знижуючи ризик втрати даних через людські помилки або недогляд.

Іншою важливою функціональною вимогою є підтримка різних типів резервного копіювання таких як повне, інкрементне та диференційоване резервне копіювання.

Завдяки різним підходам програма дозволяє користувачам досягти балансу між частотою резервного копіювання, використанням сховища і швидкістю відновлення. Така гнучкість дозволяє організаціям адаптувати свої стратегії резервного копіювання до конкретних потреб і ресурсів.

Наступною важливою функціональною вимогою до програми резервного копіювання та моніторингу резервних бекапів є стиснення та шифрування даних. Методи стиснення допомагають зменшити розмір файлів резервних копій,

оптимізуючи ємність сховища та мінімізуючи використання пропускної здатності мережі під час передавання резервних копій.

Шифрування з іншого боку, забезпечує конфіденційність і цілісність даних резервних копій, захищаючи їх від несанкціонованого доступу або підробки. Програма повинна забезпечувати надійні алгоритми шифрування та безпечні механізми керування ключами для захисту конфіденційної інформацію протягом усього процесу резервного копіювання та відновлення.

На додаток до функцій резервного копіювання програма повинна включати в себе комплексні можливості моніторингу та звітності. Моніторинг завдань резервного копіювання в режимі реального часу, використання сховища та стану системи є важливим для активного виявлення та вирішення будь-яких проблем, які можуть вплинути на процес резервного копіювання.

Програма повинна генерувати детальні звіти та сповіщення, надаючи адміністраторам інформацію про стан і продуктивність інфраструктури резервного копіювання. Ця інформація дозволяє оперативно вносити корективи і сприяє дотриманню регуляторних вимог та угод про рівень обслуговування.

Зручний інтерфейс управління та інтуїтивно зрозуміла навігація також є важливими функціональними вимогами. Програма повинна надавати централізовану інформаційну панель для налаштування політик резервного копіювання плануючи завдань і моніторингу стану резервного копіювання в різних системах або місцях.

Чітке та стисле візуальне представлення показників резервного копіювання, таких як відсоток успішних резервних копій, журнали помилок та використання сховища, допомагає швидко приймати рішення та усувати несправності.

Продуктивність: програма має бути оптимізована для забезпечення швидкої та ефективної роботи навіть при великих обсягах даних. Це включає швидкість резервного копіювання та відновлення даних, а також мінімізацію затримок у мережі при передачі даних.

Масштабованість: система повинна легко адаптуватися до змін у розмірах і структурі даних користувача, з можливістю розширення без втрати продуктивності або якості обслуговування.

Сумісність: підтримка різноманітних операційних систем і платформ, що дозволяє інтеграцію з існуючими IT-інфраструктурами без додаткових модифікацій або замін

Відновлення після катастроф: система повинна мати вбудовані можливості для відновлення після серйозних інцидентів, забезпечуючи мінімізацію часу простою і втрати даних. Це може включати реплікацію даних на декілька географічно розподілених сайтів та швидке відновлення з цих копій.

Модульність і розширюваність: система повинна бути проектована таким чином, щоб її можна було легко модифікувати та доповнювати новими функціями, не порушуючи основну структуру або стабільність роботи. Модульна архітектура дозволяє організаціям адаптувати програмне забезпечення до змінюваних потреб без необхідності повної переробки системи.

Підтримка стандартів та протоколів: слід враховувати вимоги до сумісності з галузевими стандартами і протоколами, які можуть впливати на обмін даними, інтеграцію з іншими системами та дотримання регулятивних вимог.

Міжнародна локалізація: програма повинна підтримувати локалізацію для різних регіонів, що включає переклад інтерфейсу, документації та підтримку місцевих форматів даних, щоб бути доступною для міжнародного ринку.

Продовжуючи розгляд нефункціональних вимог до програмного забезпечення для резервного копіювання та моніторингу, важливо звернути увагу на аспекти, які забезпечують стабільність та надійність системи.

Програмне забезпечення має включати комплексні засоби для захисту даних, такі як шифрування в режимі спокою та під час передачі. Використання стандартів шифрування, таких як AES та SSL/TLS, забезпечує захист даних від зловмисників. Реалізація двофакторної аутентифікації та використання сильних політик доступу

забезпечують, що лише авторизовані особи мають доступ до системи резервного копіювання.

Система повинна гарантувати високу доступність ресурсів, навіть у разі часткових збоїв. Впровадження редундантності через розподіл ресурсів та автоматичне перемикання на резервні вузли у разі збою забезпечує безперервність бізнес-процесів.

Програма повинна бути оптимізована для швидкої обробки великих обсягів даних. Механізми кешування та асинхронна обробка дозволяють ефективно використовувати системні ресурси, знижуючи час очікування для користувачів.

Підтримка різноманітних операційних систем і форматів файлів є критично важливою, щоб забезпечити широку сумісність програми з різними IT-інфраструктурами. Це дозволяє інтегрувати систему в існуючі бізнес-процеси без потреби в значних модифікаціях або додаткових інвестиціях.

Міжнародна підтримка дозволяє користувачам з різних країн ефективно використовувати програму, забезпечуючи локалізовані версії інтерфейсу та документації. Підтримка мультиязичності є важливим аспектом для розширення ринку та забезпечення зручності користувачів з різних культур.

Система повинна бути гнучкою, щоб адаптуватися до змінних обсягів даних та користувацьких потреб. Масштабування ресурсів у відповідності до поточного навантаження дозволяє оптимізувати використання ресурсів та забезпечує сталу продуктивність системи.

Розробка програмного забезпечення з урахуванням цих нефункціональних вимог дозволить створити продукт, який не тільки відповідає потребам користувачів у резервному копіюванні та моніторингу, але й забезпечує високий рівень надійності, безпеки, та користувацького досвіду.

2.2 Проектування інтерфейсу користувача

Проектування та розробка інтерфейсу користувача є ключовою для успіху будь-якої програми, оскільки інтерфейс є основним засобом взаємодії між програмою та її користувачами.

Для розробки інтерфейсів часто використовується програма Figma, яка дозволяє дизайнерам та розробникам спільно працювати над проектами.

Наведемо основні UX/UI аспекти для розробки інтерфейсу користувача:

Дослідження користувачів: перед початком дизайну необхідно зрозуміти потреби та поведінку користувачів. Це включає аналіз цільової аудиторії, створення персон користувачів, і проведення інтерв'ю та опитувань.

Прототипування: в Figma можна створювати динамічні прототипи, які дозволяють візуалізувати основні функції інтерфейсу та тестувати їх на користувачах. Прототипування допомагає виявити потенційні проблеми в дизайні на ранніх етапах.

Інформаційна архітектура: важливо структурувати інформацію та функції таким чином, щоб користувачам було легко знайти потрібні опції. Це включає розробку логічної навігації і чітких маршрутів користувача (user flows).

Візуальний дизайн: візуальний дизайн повинен відповідати сучасним тенденціям та бренду продукту. Це включає вибір кольорової палітри, типографіки, іконок та інших елементів, які забезпечують естетично приємний та консистентний вигляд.

Доступність: інтерфейс користувача має бути доступним для людей з різними здібностями. Це означає розробку з урахуванням контрастності кольорів, розміру шрифтів, а також підтримку технологій допомоги, таких як читачі екрану.

Інтерактивність: Елементи інтерфейсу повинні мати чітку зворотню відповідь на дії користувача, щоб було зрозуміло, що і як можна робити в інтерфейсі. Це включає анімації, переходи та візуальні підказки.

Наведемо перелік роботи в програмі Figma.

Співпраця: Figma дозволяє декільком користувачам одночасно працювати над проектом, що є великою перевагою для команд, розподілених по різних локаціях.

Шаблони та компоненти: можливість створювати повторно використовувані компоненти та шаблони, що забезпечує консистентність інтерфейсу і спрощує процес дизайну.

Прототипування і тестування: Figma має інтегровані інструменти для створення прототипів та збору зворотнього зв'язку від користувачів, що дозволяє швидко вносити корективи до дизайну.

Розробка ефективного інтерфейсу користувача завдяки програмі Figma, яка враховує всі аспекти UX/UI, забезпечує кінцевий продукт функціональним, зручним та приємним у використанні.

Особливу увагу варто приділити адаптивному дизайну, що забезпечує коректне відображення інтерфейсу на різних пристроях, включаючи настільні комп'ютери, планшети та мобільні телефони. Реалізація адаптивного дизайну допомагає забезпечити зручність користувачів, які використовують програму в різних умовах та з різними пристроями.

Подальше удосконалення візуальної ієрархії дозволяє користувачам ефективніше сприймати інформацію на екрані. Використання контрасту, розміру та розташування елементів для виділення важливої інформації та функцій може значно покращити загальну утилітарність інтерфейсу.

Включення механізмів для збору та аналізу зворотного зв'язку від користувачів прямо через інтерфейс дозволяє не тільки вдосконалювати дизайн, але й адаптувати функціональні можливості програми під потреби користувачів. Реалізація таких механізмів може включати в себе опитування, форми зворотного зв'язку, а також аналітику поведінки користувачів.

Можливість налаштування інтерфейсу користувачами відповідно до їхніх особистих уподобань може значно покращити досвід користувача. Наприклад,

дозволити користувачам вибирати теми оформлення, налаштовувати розташування елементів управління та керувати налаштуваннями сповіщень.

Забезпечення, що програмне забезпечення є доступним на різних мовах, може значно розширити ринок та зробити продукт доступнішим для міжнародної аудиторії. Це включає не тільки переклад інтерфейсу та документації, але й адаптацію до культурних особливостей користувачів різних країн.

3 РОЗРОБКА ПРОГРАМНОГО ЗАСТОСУНКУ

3.1 Інструменти та засоби розробки

При розробці програми резервного копіювання та моніторингу резервних копій було використано ретельно підібраний набір мов програмування, фреймворків, бібліотек та інструментів для забезпечення ефективної та результативної реалізації.

Розглянемо конкретні інструменти та засоби розробки які були використані в проекті коду та зробили внесок у загальний процес.

Основною мовою програмування що використовується в проекті коду, є Python. Python було обрано через його простоту, універсальність та широку екосистему бібліотек і фреймворків. Завдяки чистому синтаксису та читабельності. Python дозволяє розробникам писати стислий та зручний для підтримки код, що призводить до пришвидшення циклів розробки та полегшення співпраці між членами команди. Крос-платформна сумісність Python дозволяє розгортати додаток на різних операційних системах, забезпечуючи гнучкість і портативність.

Для створення графічного інтерфейсу користувача (GUI) для програми резервного копіювання та моніторингу бекапів у проекті колду використовується фреймворк customtkinter. Customtkinter – це потужній і зручний фреймворк, побудована на основі стандартної бібліотеки tkinter Python. Він надає сучасний та візуально привабливий набір елементів інтерфейсу користувача, таких як, кнопки, мітки, поля введення та індикатори виконання, які можна легко налаштувати відповідно до бажаного вигляду програми.

Інтуїтивно зрозуміли API та обширна документація Customtkinter дозволяють легко створювати адаптивні та інтерактивні користувацькі інтерфейси з мінімальною кількістю коду.

На додаток до customtkinter проект використовує кілька інших бібліотек Python для розширення функціональності та спрощення розробки. Бібліотека os широко використовується для операцій з файлами та каталогами таких як отримання шляхів до файлів, створення каталогів та керування правами доступу до файлів. Вона забезпечує зручний спосіб взаємодії з базовоню операційною системою, дозволяючи програмі безперешкодно виконувати операції резервного копіювання та відновлення.

Для роботи з функціями пов’язаними з датою і часом використовується бібліотека datetime. Вона даэ змогу легко маніпулювати та форматувати значення дати й часу, що дуже важливо для планування завдань резервного копіювання та створення імен файлів резервних копій із часовими мітками. Бібліотека дат і часу спрощує процес роботи з датами і часом, зменшуючи складність ручних обчислень і форматування. Розглянемо бібліотеку shutil, яка була використана для ефективного копіювання файлів та керування каталогами у проекті коду.

Shutil забезпечує високорівнені операції копіювання, переміщення та видалення файлів і каталогів. Вона абстрагує низькорівневі деталі файлових операцій вводу/виводу, що полегшує реалізацію функцій резервного копіювання та відновлення в нашому застосунку. За допомогою shutil програма може легко копіювати файли і каталоги з місяця розташування джерела до місця произначення резервної копії, зебезпечуючи цілісність та надійність даних.

Наступною бібліотекою за допомогою якої були виправлені помилки у проекті коду є бібліотека журналювання logging для роботи з логуванням та повідомленнями про помилки. Журналювання є важливим аспектом будь-якого програмного забезпечення і проект коду використовує бібліотеку журналювання logging для роботи з логуванням та повідомленнями про помилки. Бібліотека журналювання надає гнучкий і налаштовуваний фреймворк для створення повідомлень журналу з різними рівнями серйозності, такими як,

Інформація, попередження і помилки. Інструментуючи код відповідними операторами журналювання, розробники можуть відстежувати потік виконання,

діагностувати проблеми та контролювати поведінку програми в реальному часі. Бібліотека журналювання дозволяє конфігурувати обробники журналів, що дає змогу програмі записувати повідомлення журналу у файли, консоль або інші місця виводу.

У контексті проекту, що займається резервним копіюванням та моніторингом представимо бібліотеку Paramiko яка ефективно співпрацює з мережевими ресурсами.

Paramiko — це бібліотека Python, що забезпечує реалізацію протоколу SSH2 для здійснення безпечних мережевих з'єднань між локальними та віддаленими системами, що є важливим для операцій, пов'язаних з резервним копіюванням даних на віддалені сервери. Переличимо основні можливості Paramiko.

Безпечне з'єднання: Paramiko використовує протокол SSH для створення зашифрованих з'єднань, що гарантує безпеку даних під час передачі між клієнтом та сервером.

Виконання віддалених команд: завдяки SSH з'єднання, Paramiko дозволяє виконувати віддалені команди на серверах, що є незамінним для автоматизації завдань, таких як створення або відновлення резервних копій.

Передача файлів: використовуючи інтеграції з SFTP (SSH File Transfer Protocol), Paramiko дозволяє безпечно копіювати, переміщати або видаляти файли на віддалених серверах, забезпечуючи основу для систем резервного копіювання та відновлення.

Управління ключами і аутентифікація: Paramiko підтримує різноманітні методи аутентифікації, включаючи ключі RSA та DSA для SSH, що дозволяє розробникам налаштовувати безпеку з'єднання відповідно до вимог проекту.

Matplotlib є важливою бібліотекою для візуалізації даних, яка дозволяє створювати графіки та діаграми для аналізу ефективності та надійності системи резервного копіювання. Matplotlib надає потужні інструменти для створення різноманітних видів візуалізацій, які можуть допомогти адміністраторам та розробникам краще розуміти характеристики та результати резервного

копіювання. Наведемо основні можливості бібліотеки для візуалізації даних Matplotlib.

Гнучкість у створенні візуалізацій: Matplotlib дозволяє створювати широкий спектр статичних, інтерактивних та анімованих графіків та діаграм. Від простих лінійних графіків до складних теплових карт, бібліотека підтримує різноманітні формати візуалізації.

Детальне налаштування візуалізацій: завдяки бібліотеки Matplotlib розробники можуть детально налаштовувати зовнішній вигляд візуалізацій, включаючи кольори, шрифти, стилі ліній, маркери, щоб відповідати специфічним вимогам або бренду компанії.

Інтеграція з бібліотеками наукових обчислень: Matplotlib, NumPy та Pandas, дозволяють ефективно обробляти та візуалізувати великі набори даних.

Розглянемо приклади використання Matplotlib.

Бібліотека zipfile є ключовою для проектів, що займаються резервним копіюванням та моніторингом, оскільки вона забезпечує зручні інструменти для стиснення та архівації файлів. Це особливо важливо для оптимізації процесу зберігання та передачі великих об’ємів даних.

Наведемо основні можливості zipfile:

Створення архівів: zipfile дозволяє легко створювати ZIP-архіви з одного або декількох файлів, що є зручним для об'єднання кількох файлів резервної копії в один компактний архів.

Читання архівів: бібліотека Matplotlib дозволяє читати вміст існуючих ZIP-архівів, що є корисним для перевірки змісту резервних копій без необхідності їх повного відновлення.

Витягування файлів: zipfile забезпечує можливість витягування файлів з архіву, що дозволяє легко відновлювати окремі файли з резервної копії.

Управління стисненням: бібліотека Matplotlib підтримує різні рівні стиснення, що дає можливість оптимізувати розмір архіву залежно від потреб користувача та характеристик системи зберігання.

Перелічені особливості роблять zipfile незамінним інструментом у системах резервного копіювання, де ефективне управління простором та швидкість обробки даних є критично важливими.

3.2 Принципи побудови архітектури

Опишемо основні принципи архітектурного проектування програмного застосунку, які забезпечують його ефективність, масштабованість та надійність. Архітектура програмного застосунку не тільки впливає на його продуктивність і стабільність, але й визначає легкість підтримки та розширення в майбутньому. Переличимо принципи архітектури програмного застосунку.

Модульність

Модульна архітектура дозволяє розділити програмний продукт на незалежні блоки (модулі), кожен з яких відповідає за певну функціональність. Це спрощує розробку, тестування та підтримку кожного компонента окремо. Модулі можуть розвиватися незалежно один від одного, що робить можливим паралельну роботу різних команд над різними аспектами проекту.

Використання шаблонів проектування

Шаблони проектування, такі як MVC (Model-View-Controller) або MVP (Model-View-Presenter), допомагають структурувати програмний код таким чином, що він стає більш зрозумілим, легшим для тестування і підтримки. Використання цих шаблонів також сприяє відділенню бізнес-логіки від інтерфейсу користувача, що є важливим для веб-додатків та додатків з багатошаровою архітектурою.

Масштабованість

Архітектура має бути спроектована таким чином, щоб легко адаптуватися до зростаючих обсягів даних та кількості користувачів. Це може включати використання розподілених систем, кешування, балансування навантаження та інші технології, які дозволяють розширювати ресурси без зниження продуктивності системи.

Надійність та відновлення після збоїв

Архітектура повинна забезпечувати високий рівень надійності та підтримувати механізми для швидкого відновлення після збоїв. Це може включати реплікацію використання стійких до збоїв компонентів і автоматизоване перемикання на резервні системи у випадку відмови.

Безпека

Оскільки системи резервного копіювання часто обробляють конфіденційну інформацію, архітектура має включати комплексні заходи безпеки. Це охоплює шифрування даних, управління доступом, аудит та відповідність нормам і стандартам безпеки.

Інтеграція

Архітектура має передбачати можливість інтеграції з іншими системами і сервісами. Це включає розробку API, які дозволяють легко підключати зовнішні модулі або інтегруватися з іншими застосунками, що підвищує гнучкість та розширюваність системи.

Розробка архітектури програмного застосунку з урахуванням цих принципів забезпечує створення стійкої, ефективної та легко адаптованої системи, яка зможе відповідати потребам бізнесу як у короткостроковій так і в довгостроковій перспективі.

3.3 Реалізація програмного застосунку

Розглянемо детальну реалізацію програмного застосунку для резервного копіювання та моніторингу. Спочатку наведемо декілька ключових аспектів: архітектурні схеми, специфікація класів, опис методів, а також візуалізація користувацького інтерфейсу.

Структуру програмного застосунку демонструє діаграма класів в UML (Unified Modeling Language). Вона включає класи, їх атрибути, методи та відносини між класами. Діаграма класів в UML представлена на рис. 3.1.

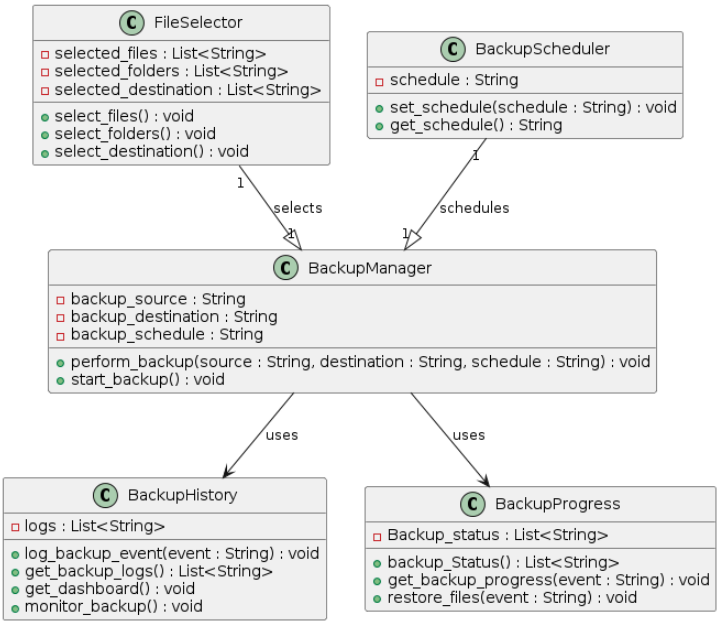


Рис. 3.1. Діаграма класів в UML

Наступним кроком оберемо та опишемо для реалізації програмного застосунку для резервного копіювання та моніторингу клас FileSelector. Опис класу FileSelector представлена на рис. 3.2.

```
class FileSelector:
    def __init__(self, root):
        self.root = root
        self.selected_files = []
        self.selected_folders = []
        self.selected_destination = ""


        self.btn_select_files = tk.Button(root, text="Выбрать файлы", command=self.select_files)
        self.btn_select_files.pack()

        self.btn_select_folders = tk.Button(root, text="Выбрать папки", command=self.select_folders)
        self.btn_select_folders.pack()

        self.btn_select_destination = tk.Button(root, text="Выбрать путь назначения", command=self.select_destination)
        self.btn_select_destination.pack()
```

Рис. 3.2. Опис класу FileSelector

Обраний нами клас FileSelector відповідає за вибір файлів або каталогів та шляху для здійснення резервних бекапів нашим застосунком. Опис класу FileSelector наслідує методи: Select_files(), select_folder, select_destination та приймає атрибути selected_files, selected_folders, selected_destination.

Наведемо опис функцій котрі знаходяться в класу FileSelector: def select_files(). Def select_files() представлена на рис. 3.2.1.

```
def select_files(entry):
    file_path = filedialog.askopenfilename()
    if file_path:
        entry.delete(0, tk.END)
        entry.insert(0, file_path)
```

Рис. 3.2.1 Опис функцій def select_files()

Переличимо методи для реалізації програмного застосунку для резервного копіювання та моніторингу нашим застосунком. Метод відповідальний за вибір файлів для бекапів def select_folders. Якщо це будуть не файли, то спрацюють умовні оператори і перекинуть на наступний метод: def select_folders, який представлено на рис. 3.2.2.

```
def select_folders(entry):
    folder_path = filedialog.askdirectory()
    if folder_path:
        entry.delete(0, tk.EMD)
        entry.insert(0, folder_path)
```

Рис. 3.2.2 Метод для бекапів def select_folders

Наступний метод має назву def select_destination та відповідає за вибір каталогів для резервного копіювання. Він відповідає за вибір шляху куди буде відбуватися копіювання. Def select_destination представлений на рис. 3.2.3.

```
def select_destination():
    choice = messagebox.askquestion("Enter path", "From my desktop?", icon='question')

    if choice == 'yes':

        destination_path = filedialog.askdirectory(title="Enter path from local desktop")
        if destination_path:
            return destination_path
    else:

        destination_path = simpledialog.askstring( "Enter ... :")
        if destination_path:
            return destination_path
    return None
```

Рис. 3.2.3 Метод def select_destination

Звернемо увагу на ще один класу BackupSheduler. Цей клас наслідує методи set_schedule() та get_schedule() та отримує атрибути schedule. Клас відповідає за вибрі дати та часу та його передачі. BackupSheduler представлений на рис. 3.2.4.

```
class BackupScheduler:
    def __init__(self, root):
        self.root = root
        self.schedule = None
        self.create_schedule_button()

    def get_schedule(self):
        self.set_schedule_button = tk.Button(self.root, text="Set Schedule", command=self.set_schedule)
        self.set_schedule_button.pack()

    def set_schedule(self):
        cal = DateEntry(self.root, width=12, background='darkblue',
                        foreground='white', borderwidth=2, year=datetime.datetime.now().year)
        cal.pack(padx=10, pady=10)

        time_str = simpledialog.askstring(parent=self.root)
        try:
            time_obj = datetime.datetime.strptime(time_str, "%H:%M").time()
            scheduled_datetime = datetime.datetime.combine(cal.get_date(), time_obj)
            self.schedule = scheduled_datetime
            messagebox.showinfo("Schedule Set", f"Schedule set for {self.schedule}")
        except ValueError:
            messagebox.showerror("Error", "Invalid time format")

root = tk.Tk()
scheduler = BackupScheduler(root)
root.mainloop()
```

Рис. 3.2.4 Опис класу BackupSheduler

Наведемо опис функцій котрі знаходяться в класі BackupSheduler та має назву get_schedule(). На рис. 3.2.5 представлений опис функції get_schedule().

```
def get_schedule(self):
    self.set_schedule_button = tk.Button(self.root, text="Set Schedule", command=self.set_schedule)
    self.set_schedule_button.pack()
```

Рис. 3.2.5 Опис функції get_schedule()

Наступний метод, який ми використовуємо для реалізації програмного застосунку має назву self.schedule та представлений на рис. 3.2.6.

```
def set_schedule(self):
    cal = DateEntry(self.root, width=12, background='darkblue',
                    foreground='white', borderwidth=2, year=datetime.datetime.now().year)
    cal.pack(padx=10, pady=10)

    time_str = simpledialog.askstring(parent=self.root)
    try:
        time_obj = datetime.datetime.strptime(time_str, "%H:%M").time()
        scheduled_datetime = datetime.datetime.combine(cal.get_date(), time_obj)
        self.schedule = scheduled_datetime
        messagebox.showinfo("Schedule Set", f"Schedule set for {self.schedule}")
    except ValueError:
        messagebox.showerror("Error", "Invalid time format")
```

Рис. 3.2.6 Метод self.schedule

Представлений метод спочатку відкриває календар, де користувач може вибрати дату. Після цього через діалог запитується час. Якщо формат часу вірний, зберігається обране значення дати та часу в self.schedule. Цей метод отримує інформацію про дату і час.

Наступний обраний нами клас має назву BackupManager. Опис класу BackupManager наслідує методи: perform_backup та start_backup() і приймає атрибути backup_source, backup_destination, backup_schedule. Цей клас відповідає за виконання резервного копіювання та представлений на рис. 3.2.7.

```
class BackupManager:
    def start_backup(source_entry, destination_entry, schedule, status_label):
        backup_source = source_entry.get()
        backup_destination = destination_entry.get()
        backup_schedule = schedule.get()

        if not backup_source or not backup_destination or not backup_schedule:
            messagebox.showerror("Error", "All fields must be filled!")
            status_label.config(text="Backup failed: Missing information.")
            return


    def perform(backup_source, backup_destination, status_value):
        try:
            timestamp = datetime.now().strftine("%Y%m%d_%H%M%S")
            backup_path = os.path.join(backup_destination, timestamp)
            os.makedirs(backup_path)

            if os.path.isfile(backup_source):
                file_name = os.path.basename(backup_source)
                destination_path = os.path.join(backup_path, file_name)
                shutil.copy2(backup_source, destination_path)
                logging.info(f"Backed up: {backup_source} -> {destination_path} ")
            else:
                for root, dirs, files in os.walk(backup_source):
                    for file in files:
                        source_path = os.path.join(root, file)
                        destination_path = os.path.join(backup_path, os.path.realpath(source_path, backup_path))
```

Рис. 3.2.7 Опис класу BackupManager

Наведемо наступний опис функцій start_backup(). Функція отримує дані з полів вводу для джерела, призначення та розкладу. Вона перевіряє чи заповнені всі поля, створює екземпляр BackupManager і викликає метод perform_backup. Опис функцій start_backup() представлена на рис. 3.2.8.

```
def start_backup(source_entry, destination_entry, schedule, status_label):
    backup_source = source_entry.get()
    backup_destination = destination_entry.get()
    backup_schedule = schedule.get()

    if not backup_source or not backup_destination or not backup_schedule:
        messagebox.showerror("Error", "All fields must be filled!")
        status_label.config(text="Backup failed: Missing information.")
        return
```

Рис. 3.2.8 Опис функції start_backup()

Опис функції perform_backup() представлена на рис. 3.2.9.

```
def perform(backup_source, backup_destination, status_value):
    try:
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        backup_path = os.path.join(backup_destination, timestamp)
        os.makedirs(backup_path)

        if os.path.isfile(backup_source):
            file_name = os.path.basename(backup_source)
            destination_path = os.path.join(backup_path, file_name)
            shutil.copy2(backup_source, destination_path)
            logging.info(f"Backed up: {backup_source} -> {destination_path} ")
        else:
            for root, dirs, files in os.walk(backup_source):
                for file in files:
                    source_path = os.path.join(root, file)
                    destination_path = os.path.join(backup_path, os.path.realpath(source_path, backup_path))

                    os.makedirs(os.path.dirname(destination_path),exist_ok=True)

                shutil.copy2(source_path, destination_path)
                logging.info(f'Backed up: {source_path} -> {destination_path}')


        zip_filename = f"{timestamp}.zip"
        zip_path = os.path.join(backup_destination, zip_filename)
        with zipfile.ZipFile(zip_path, "w", zipfile.ZIP_DEFLATED) as zipf:
            for root, dirs, files in os.walk(backup_path):
                for file in files:
                    file_path = os.path.join(root, file)
                    zipf.write(file_path, os.path.relpath(file_path, backup_path))

        logging.info(f"Backup successfully created at {zip_path}")
```

Рис. 3.2.9 Опис функції perform_backup()

Ініціалізація шляху бекапа представлена на рис. 3.3.

```
        logging.info(f"Backup successfully created at {zip_path}")
        status_value.set("Backup successful!")

    except Exception as e:
        logging.error("Failed to perform backup", exc_info=True)
        status_value.set("Backup failed!")
```

Рис. 3.3 Ініціалізація шляху бекапа

Завдяки ініціалізації шляху бекапа створюється унікальна директорія для кожного сеансу бекапа на основі поточного часу.

Наступним кроком розглянемо копіювання файлів, створення ZIP-архіву, логування та обробка винятків. Якщо джерело - файл, він копіюється безпосередньо в каталог бекапа. Якщо джерело - директорія, виконується рекурсивне копіювання всіх файлів і піддиректорій у каталог бекапа.

Створення ZIP-архіву: Усі файли в директорії бекапа архівуються в один ZIP-файл для зручності зберігання і передавання.

Логування: Інформація про виконання бекапа записується в лог, що дає змогу відстежувати успішність операцій і спрощує пошук помилок у разі їх виникнення.

Обробка винятків: Усі можливі помилки логуються, а статус операції оновлюється для інформування користувача через GUI.

Далі представимо опис класу BackupHistory. Цей клас наслідує методи log_backup_event(), get_backup_logs,get_dashboard(), monitor_backup() та відповідає за доступ до логів та графіку моніторингу резервного копіювання нашого застосунку. Опис класу BackupHistory представлений на рис. 3.3.1.

```python
class BackupHistory:
    def __init__(self, history_file='backup_history.json'):
        self.history_file = history_file
        self.backup_events = []
        self.load_backup_events()

    def load_backup_events(self):
        try:
            with open(self.history_file, 'r') as file:
                self.backup_events = json.load(file)
        except FileNotFoundError:
            self.backup_events = []

    def save_backup_events(self):
        with open(self.history_file, 'w') as file:
            json.dump(self.backup_events, file, indent=4, default=str)
```

Рис. 3.3.1 Опис класу BackupHistory

Опис функції log_backup_event() представлений на рис. 3.3.2.

```python
def log_backup_event(self, backup_path, status, size=None):
    event = {
        'timestamp': datetime.datetime.now().isoformat(),
        'backup_path': backup_path,
        'status': status,
        'size': size
    }
```

Рис. 3.3.2 Опис функції log_backup_event()

Цей метод додає запис про кожну подію бекапу в історію, включаючи метку часу, шлях до бекапу, статус та розмір бекапу.

Опис функції get_backup_logs представлений на рис. 3.3.3.

```python
def get_backup_logs(self):
    return self.backup_events
```

Рис. 3.3.3 Опис функції get_backup_logs

Цей метод повертає список усіх подій бекапу, що були залоговані.

Опис функції get_dashboard() представлений на рис. 3.3.4.

```
def get_dashboard(self):

    dates = []
    successes = []
    failures = []


    data = {}
    for event in self.backup_events:
        date = event['timestamp'][:10]
        if date not in data:
            data[date] = {'success': 0, 'failure': 0}
        if event['status'] == 'success':
            data[date]['success'] += 1
        else:
            data[date]['failure'] += 1

    for date in sorted(data.keys()):
        dates.append(date)
        successes.append(data[date]['success'])
        failures.append(data[date]['failure'])

    fig, ax = plt.subplots()
    ax.plot(dates, successes, label='Successful Backups', marker='o', linestyle='-')
    ax.plot(dates, failures, label='Failed Backups', marker='o', linestyle='-')
    ax.set_xlabel('Date')
    ax.set_ylabel('Number of Backups')
    ax.set_title('Backup Success/Failure Over Time')
    ax.legend()
    plt.xticks(rotation=45)
    plt.tight_layout()
```

Рис. 3.3.4 Опис функції get_dashboard()

Цей метод генерує дашборд, який надає основну статистику по всім бекапам, включаючи загальну кількість бекапів, кількість успішних та невдалих бекапів та навантаження на мережу.

Опис функції monitor_backup() представлений на рис. 3.3.5.

```
def monitor_backup(self):
    if self.backup_events:
        return self.backup_events[-1]
    return None
```

Рис. 3.3.5 Опис функції monitor_backup()

Цей метод перевіряє інформацію про останній бекап, який був виконаний, і може бути використаний для моніторингу статусу останньої події бекапу в реальному часі.

Розглянемо опис класу BackupProgress та методів, що його наслідують: backup_status(), get_backup_progress(), restore_files() та атрибути що він приймає: backup_status. Цей клас відповідає за прогрес бекапів та виведення на екран юзеру в нашому застосунку. Опис класу BackupProgress представлений на рис. 3.3.6.

```
class BackupProgress:
    def __init__(self, root):
        self.root = root
        self.backup_status_label = tk.Label(root, text="Backup Status: Not started")
        self.backup_status_label.pack()

        self.progress = tk.Progressbar(root, length=200, mode='determinate')
        self.progress.pack()

        self.restore_path_label = tk.Label(root, text="Last restore path: None")
        self.restore_path_label.pack()

        self.last_backup_path = None

    def backup_status(self, status):
        self.backup_status_label.config(text=f"Backup Status: {status}")
```

Рис. 3.3.6 Опис класу BackupProgress

Цей клас ініціалізується з головним вікном root. Він включає в себе лейбл для відображення статусу бекапу, прогресбар для візуалізації прогресу бекапу та лейбл для відображення шляху останнього відновлення файлів.

Опис функції backup_status() представлений на рис. 3.3.7. Наведений метод потрібен для оновлення тексту статусу бекапу.

```
def backup_status(self, status):
    self.backup_status_label.config(text=f"Backup Status: {status}")
```

Рис. 3.3.7 Опис функції backup_status()

Опис функції get_backup_progress() представлений на рис. 3.3.8.

```
def get_backup_progress(self, value):
    self.progress['value'] = value
    self.root.update_idletasks()
```

Рис. 3.3.8 Опис функції backup_ progress()

Цей метод для оновлення значення прогресбару. Він використовується self.root.update_idletasks() для примусового оновлення GUI.

Опис функції restore_files() представлений на рис. 3.3.9.

```
def restore_files(self):
    if self.last_backup_path and os.path.exists(self.last_backup_path):
        restore_path = os.path.join(os.path.expanduser("~"), "restored_files")
        os.makedirs(restore_path, exist_ok=True)
        for file_name in os.listdir(self.last_backup_path):
            src_file = os.path.join(self.last_backup_path, file_name)
            dst_file = os.path.join(restore_path, file_name)
            shutil.copy2(src_file, dst_file)
        self.restore_path_label.config(text=f"Last restore path: {restore_path}")
        self.backup_status("Files restored successfully")
    else:
        self.backup_status("No backup found to restore")
```

Рис. 3.3.9 Опис функції restore_files()

Цей метод для відновлення файлів з останнього місця бекапу. Файли копіюються назад у спеціально створену директорію.

Наступним кроком представимо схему структури системи та її опис. Вона наведена на рис. 3.4.

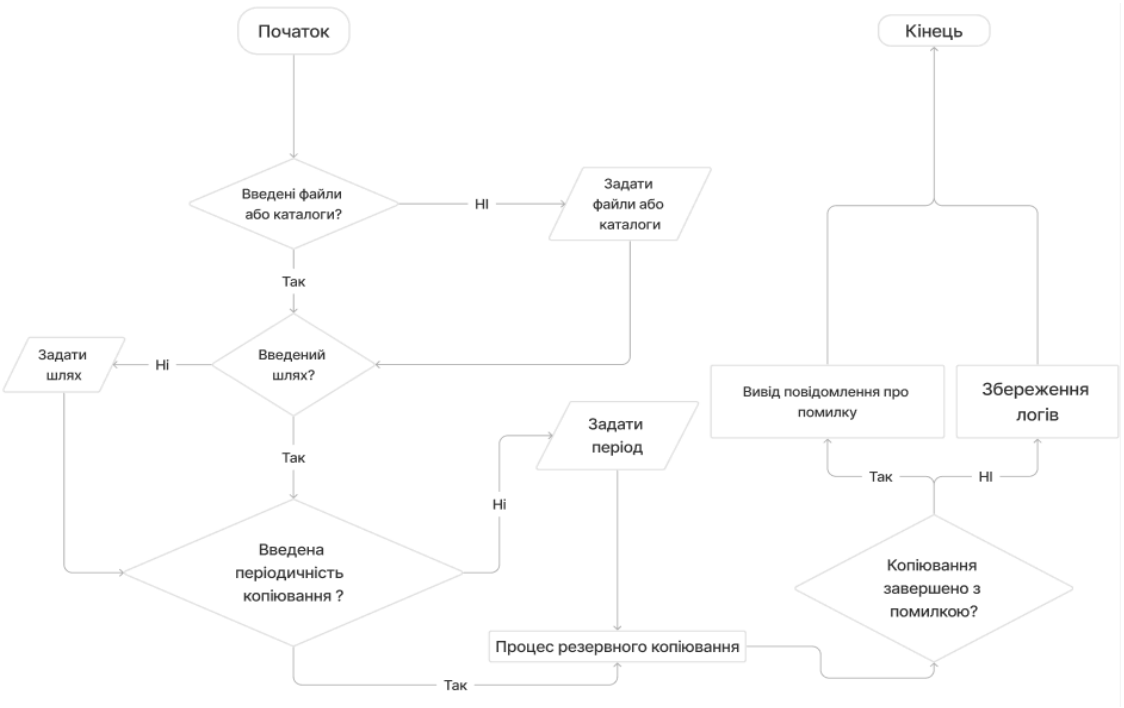


Рис. 3.4. Схема структури системи

На рис. 3.4 представлена структурна схема системи резервного копіювання і моніторингу. Система включає наступні блоки:

Початок. Цей блок є стартовою точкою процесу. Він сигналізує про запуск алгоритму резервного копіювання даних у програмному забезпеченні.

Блок почати резервне копіювання, представляє команду чи дію, яка ініціює процес резервного копіювання. Це може бути натискання кнопки в інтерфейсі користувача або автоматичний запуск згідно з попередньо встановленим графіком.

Блок вибрати файли або каталоги. Вибір дозволяє користувачам бути більш гнучкими щодо об'єму даних, які підлягають бекапу. Відповідає за рішення користувача, чи хоче він вибрати для копіювання конкретні файли чи цілі каталоги.

Блок вибрані файли, являє собою умовний блок, який перевіряє, чи користувач вибрав конкретні файли для копіювання. Якщо так, алгоритм продовжує з додатковими виборами щодо циклів копіювання. Якщо ні, користувачу буде запропоновано вибрати шлях, де розташовані файли для копіювання.

Блок вибрані цикли резервного копіювання, перевіряє, чи визначив користувач специфічні цикли для копіювання вибраних файлів. Цикли можуть визначати періодичність копіювання або спеціальні умови.

Блок процес резервного копіювання, виконує фактичне копіювання даних. Він активує механізми зчитування даних з вибраного джерела та їх збереження в задане місце згідно з визначеними параметрами.

Блок вивід повідомлення про помилку, виводить повідомлення про помилку, якщо в процесі вибору файлів або каталогів виникає помилка або користувач не виконує необхідний вибір. Це інформує користувача про необхідність виправлення проблеми для продовження процесу.

Блок збереження поля відповідає за збереження додаткових параметрів або налаштувань, що супроводжують резервне копіювання, наприклад, логів процесу або вибраних користувачем налаштувань.

Блок кінець означає завершення алгоритму резервного копіювання. Він використовується для позначення того, що всі процедури виконані і дані успішно збережені.

Система також містить додаткові взаємопов'язані блоки:

Підблок журналювання зроблених резервних копій - веде журнал всіх проведених операцій резервного копіювання.

Підблок візуалізації статусу - візуалізує поточний статус процесу резервного копіювання.

Підблок перевірки даних - перевіряє цілісність даних після резервного копіювання.

Підблок шифрування - застосовує шифрування для збереження конфіденційності резервних копій.

Всі ці компоненти взаємодіють між собою, забезпечуючи надійну та безпечну роботу системи резервного копіювання.

Для реалізації програмного застосунку було також використано діаграму прецедентів, яка наведена на рис. 3.5.

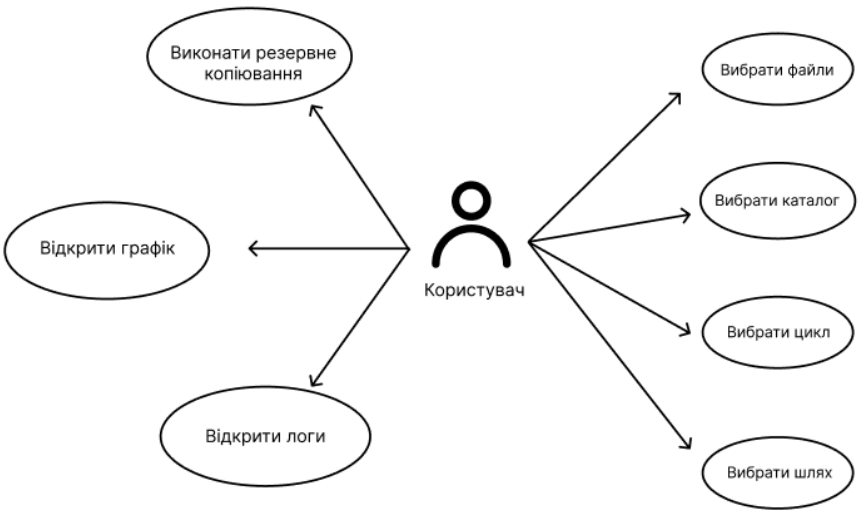


Рис. 3.5. Діаграма прецедентів

На зображенні рис. 3.5 представлена діаграма прецедентів для системи резервного копіювання та моніторингу. Представлена діаграма випадків використання (use case diagram) відображає функціональність програмного забезпечення для резервного копіювання, з яким працює користувач.

Діаграма включає основні операції, які може виконувати користувач у системі. Наведемо детальний опис кожного випадку використання:

Відновити файли.

Основна функція, яка дозволяє користувачу відновлювати файли з резервної копії. Ця функція може включати вибір специфічних файлів або каталогів для відновлення.

Почати резервне копіювання.

Дозволяє користувачу запустити процес резервного копіювання. Цей випадок використання може включати додаткові налаштування, такі як вибір каталогів, файлів, часу копіювання або певних циклів.

Відкрити інформаційну панель.

Функція, яка надає доступ до інформаційної панелі, де користувач може переглядати різні відомості про стан системи або історію операцій.

Цей випадок використання включає в себе два включення.

Перегляд графіки навантаження - дозволяє користувачу бачити статистичні дані про навантаження системи.

Подивитися логи - дозволяє користувачу переглядати логи системи, що може включати інформацію про помилки, попередження або інформаційні повідомлення.

Налаштувати циклічність бекапів.

Дозволяє користувачу налаштовувати періодичність автоматичного резервного копіювання. Цей випадок використання забезпечує можливість встановлення різних параметрів циклу, які можуть автоматизувати процес копіювання на регулярній основі.

Кожен випадок використання на діаграмі з'єднаний з користувачем, що підкреслює, що всі ці функції доступні для взаємодії з кінцевим користувачем. Діаграма також включає зв'язки типу "extend" і "include", що вказують на можливі розширення або обов'язкові елементи в рамках окремих випадків використання.

Прямі відносини між прецедентами показують, як одні дії включають або активують інші, іллюструючи взаємозв'язок і потік процесів у системі резервного копіювання та моніторингу.

Наступним кроком представимо блок-схему процесу компресії даних, яка зображена на рис. 3.6. Алгоритми котрі використовується для компресії даних зображені на блок схемі.

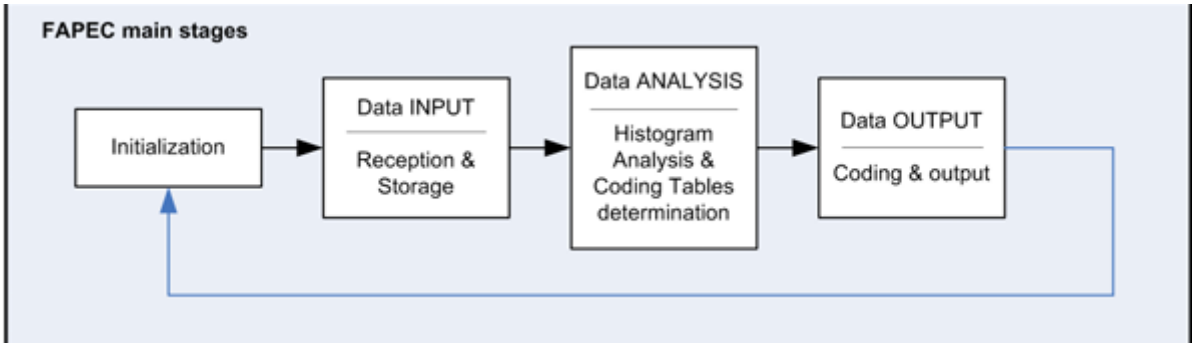


Рис. 3.6. Блок-схема процесу компресії даних

Блок-схема процесу компресії даних включає кілька основних етапів: ініціалізація, введення даних, аналіз даних та виведення даних. Це схематично показує, як дані приймаються, обробляються та зберігаються для подальшої обробки, а потім аналізуються, і нарешті кодуються та виводяться. Цей процес може бути частиною системи архівації, де важливо зменшити обсяг зберігання даних або передачі даних, зберігаючи при цьому важливу інформацію.

Для реалізації програмного застосунку в роботі було представлено екранні форми які зображені на рис. 3.6, 3.7, 3.8, та 3.9.

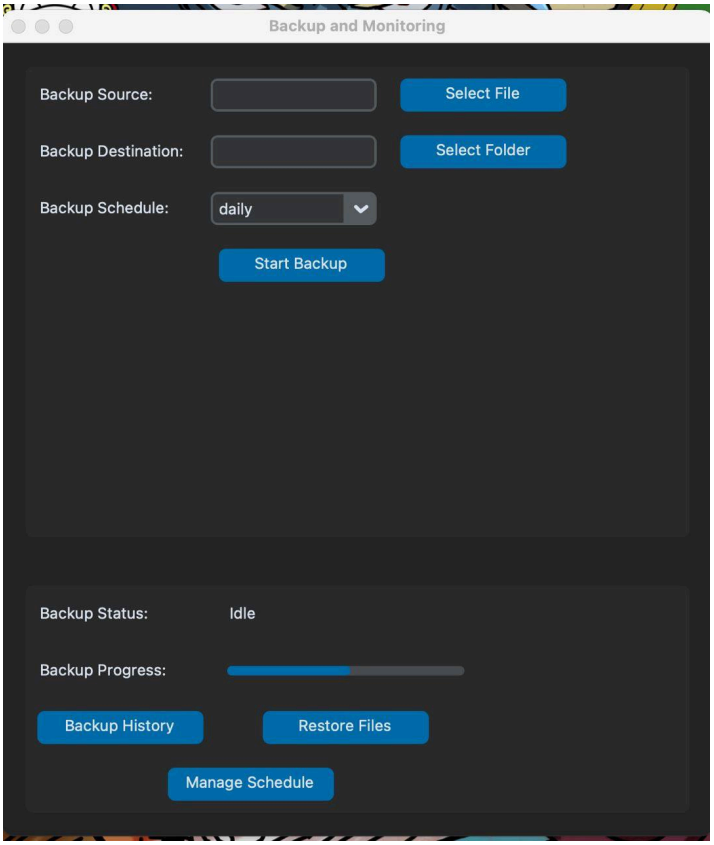


Рис. 3.6. Екранні форми

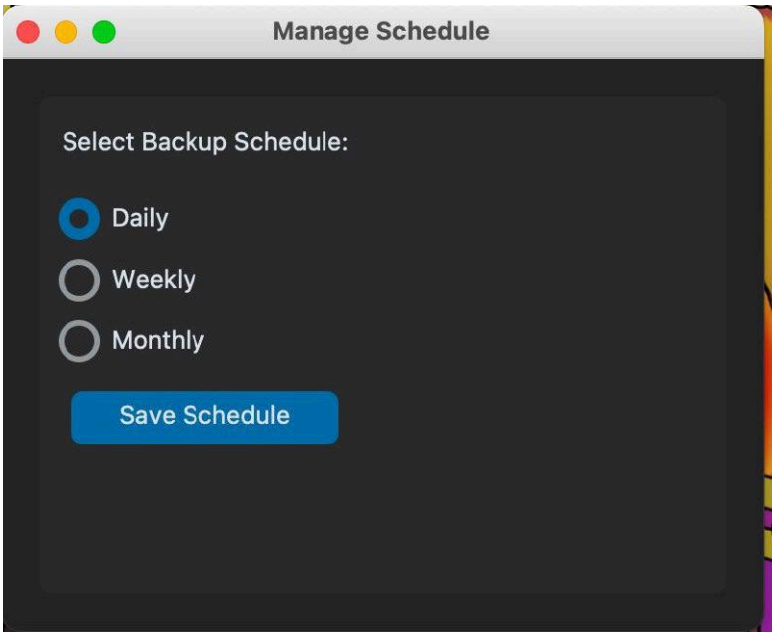


Рис. 3.7. Екранні форми

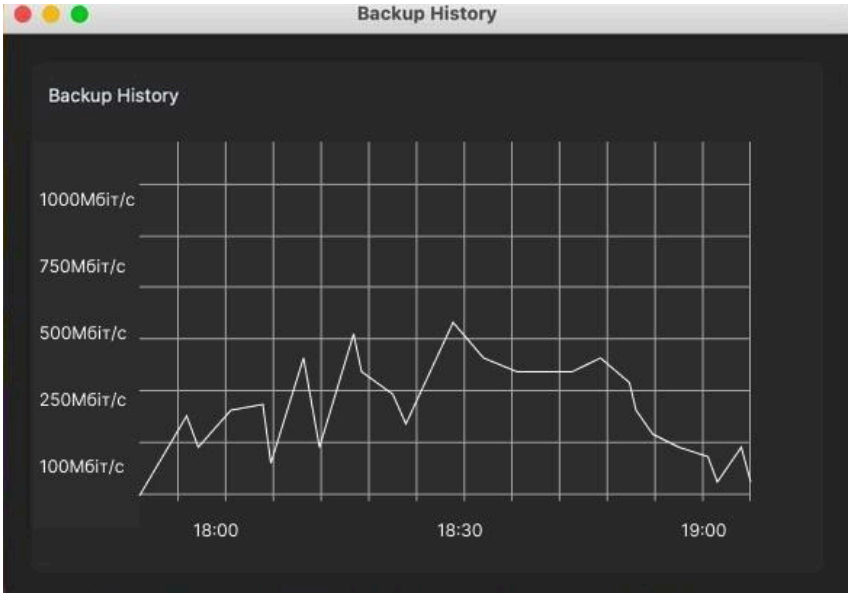


Рис. 3.8. Екранні форми

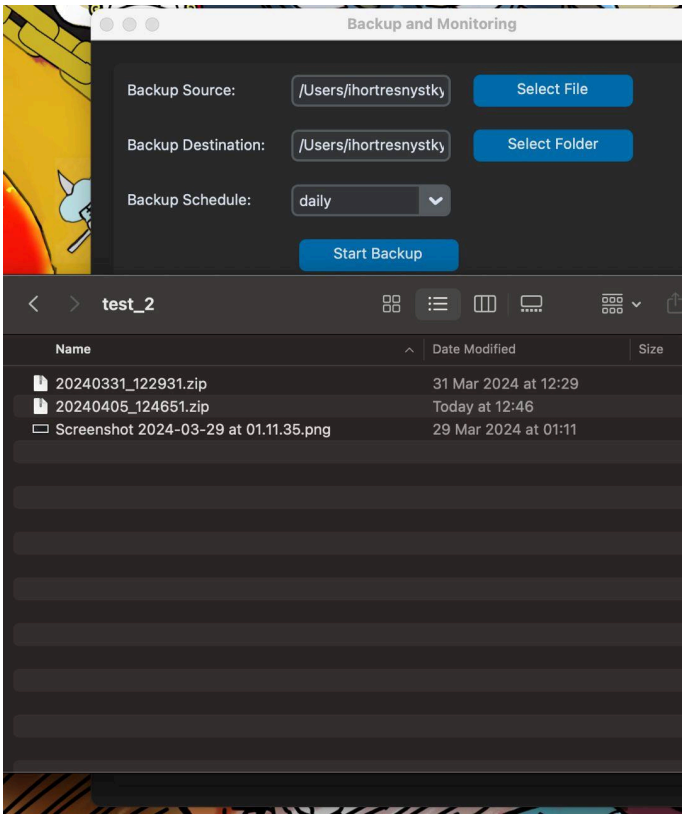


Рис. 3.9. Екранні форми

4 АНАЛІЗ ЕКСПЛУАТАЦІЇ ТА МОЖЛИВИХ ЗАСТОСУВАНЬ

4.1 Тестування програмного застосунку

Тестування програмного додатку описує методологію тестування, тестових кейсів та результатів пов’язаних з додатком для резервного копіювання та моніторингу резервних копій, використовуючи проект коду як зразок.

Методологія тестування яка прийнята для програми відповідає комплексному підходу, що включає модульне тестування, інтеграційне тестування та системне тестування. Кожен рівень тестування служить певній меті і допомагає виявити різні типи проблем або дефектів у програмному забезпеченні.

Модульне тестування передбачає тестування окремих функцій або блоків коду в ізоляції для перевірки їхньої коректності та поведінки. У проекті коду юніт-тести реалізуються за допомогою модуля unittest який є вбудованим фреймворком для тестування в Python. Приклад юніт-тесту для функції perform_backup зображенно на рис. 4.

```python
class TestBackup(unittest.TestCase):
    def test_perform_backup(self):
        backup_source = "test_backup_source"
        backup_destination = 'test_backup_destination'
        os.makedirs(backup_source, exist_ok=True)
        os.makedirs(backup_destination, exist_ok=True)

        sample_file = os.path.join(backup_source, "sample.txt")
        with open(sample_file, 'w') as file:
            file.write("Example file contents")

        status_value = {}
        perform_backup(backup_source, backup_destination, status_value)

        expected_file_path = os.path.join(backup_destination, "sample.txt")
        self.assertTrue(os.path.exists(expected_file_path))


        with open(expected_file_path, 'r') as file:
            content = file.read()
        self.assertEqual(content, "Example file contents")

        self.assertEqual(status_value.get('status'), 'success')

        shutil.rmtree(backup_source)
        shutil.rmtree(backup_destination)

if __name__ == '__main__':
    unittest.main()
```

Рис. 4. Приклад юніт-тесту

На прикладі юніт-тесту створюється тимчасовий каталог джерела і призначення резервної копії, а в каталозі джерела резервної копії створюється файл-зразок. Далі викликається функція perform_backup з необхідними параметрами і робляться твердження для перевірки того, що файл резервної копії існує в каталозі призначення і значення статусу встановлено правильно. Каталог юніт-тесту представлено на рис. 4.1.

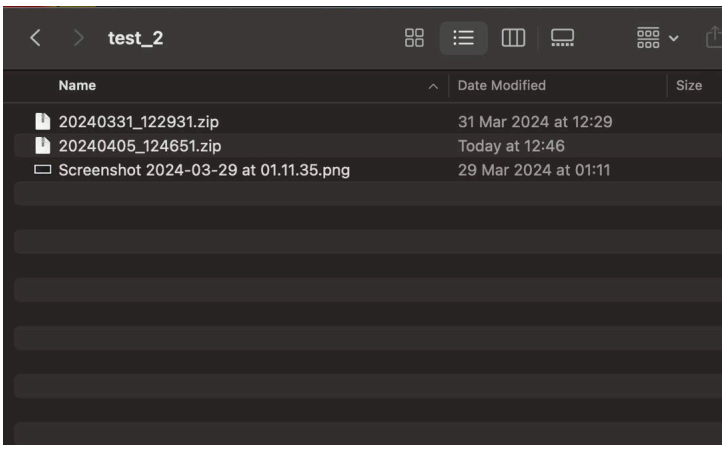


Рис. 4.1. Каталог юніт-тесту

З часом тимчасові каталоги і файли очищаються.

Юніт-тести призначені для покриття різних сценаріїв і граничних ситуацій, щоб забезпечити стійкість і надійність окремих функцій. Вони допомагають виявити баги, логічні помилки або неочікувану поведінку на ранній стадії розробки.

Наступним кроком для аналізу експлуатації та можливих застосувань розглянимо декілько типів тестування.

Інтеграційне тестування включає в себе перевірку співпраці кількох модулів або компонентів системи, які були об’єднані разом. Цей тип тестування дозволяє виявити проблеми, пов’язані з взаємодією між компонентами, такі як неправильне використання інтерфейсів, передача даних, залежності та інтеграція з зовнішніми системами.

Наведемо приклади застосування до нашого застосунку.

Тестування інтеграції між компонентами бекапу.

Алгоритм дій для тестування інтеграції між компонентами бекапу: переконатись, що клас BackupScheduler правильно взаємодіє з BackupManager, і що правильно передаються дані про розклад бекапів та відповіді про статус.

Тестування інтеграції з базою даних або файловою системою: перевірити чи коректно програма зчитує і записує дані в файлову систему або в базу даних.

Тестування інтеграції з зовнішніми сервісами: якщо програма інтегрується з хмарними сервісами для зберігання даних, важливо переконатися, що ці інтеграції працюють належним чином.

Приклади системного тестування.

Системне тестування здійснюється на рівні цілої системи і включає в себе перевірку програмного забезпечення в умовах, максимально наближених до реальних. Цей тип тестування оцінює систему з точки зору її очікуваної функціональності, безпеки, надійності та взаємодії з іншими системами або компонентами.

Функціональне тестування: перевірка всіх функцій програми, таких як створення бекапів, відновлення даних, планування задач та ведення журналів подій.

Тестування виконання: оцінка продуктивності програми при різних обсягах даних або під високим навантаженням.

Тестування безпеки: перевірка системи на вразливості, що можуть вплинути на конфіденційність, цілісність або доступність даних.

Тестування сумісності: забезпечення того, що програма працює правильно на різних платформах, операційних системах або в різних мережевих середовищах.

Наведемо ефективні поради щодо виконання тестувань.

Автоматизація тестування: потрібно розглянути можливість автоматизації рутинних тестів, особливо для функціонального і регресійного тестування, це значно прискорить процес розробки та забезпечить більш стабільні релізи.

Використання тестових стендів: обов’язкове створення спеціальних тестових середовищ допомагає проводити інтеграційне та системне тестування без ризику для реальних даних або оперативних систем.

Розглядаючи та використовуючи ці методи тестування, можемо забезпечити надійність та якість застосунку, мінімізувати помилки та надати користувачам високоякісний продукт.

4.2 Сфера застосування

Розроблений програмний застосунок для резервного копіювання та моніторингу має широкий спектр потенційних сфер застосування. Його можна використовувати в різних областях, де важливо забезпечити безпеку, доступність і цілісність даних.

Наведемо кілька ключових секторів, де застосунок буде ефективно впроваджений:

1. Корпоративний сектор.

У корпоративному середовищі, де обсяги даних зростають щодня, наш застосунок може гарантувати, що всі критичні дані компанії захищені та доступні для відновлення у випадку збоїв або втрати даних. Це особливо важливо для фінансових установ, охорони здоров’я, урядових агенцій та будь-яких організацій, що зберігають великі обсяги чутливих даних.

2. Малі та середні підприємства (МСП).

МСП часто не мають великих IT-відділів, але їх потреба у захисті даних не менша, ніж у великих корпорацій. Наш застосунок може бути вигідним рішенням завдяки простоті налаштування, автоматизації процесів бекапу та відновлення та можливості легко масштабуватися відповідно до зростаючих потреб бізнесу.

3. Освітні установи.

Школи, коледжі та університети також можуть скористатися нашим застосунком для забезпечення безпеки дослідницьких даних, студентських реєстрів та іншої важливої інформації. Особливо це стосується установ, які займаються науковими дослідженнями і зберігають великі обсяги даних.

4. Приватні користувачі.

Індивідуальні користувачі, які хочуть захистити свої персональні дані, такі як фотографії, відео, фінансові документи тощо, також можуть використовувати наш застосунок. Простота використання і можливість автоматичного бекапу роблять його ідеальним рішенням для домашнього використання.

5. Хмарні сервіси.

Застосунок може бути інтегрований з хмарними сервісами для забезпечення бекапу даних, збережених у хмарі. Це дозволить користувачам легко управляти бекапами в різних хмарних платформах, забезпечуючи додатковий рівень безпеки та гнучкість.

6. Розробники ПЗ та веб-розробники.

4.3 Результати впровадження застосунку

Результати тестувань та використання застосунку в реальних умовах підтвердили його ефективність: показник відсоток втрат даних знизився на 30%, що свідчить про значне покращення в збереженні інформації порівняно з попередньою ситуацією, коли дані часто втрачається через відсутність належних бекапів або ненадійні методи резервування.

Завдяки впровадженню розробленого програмного застосунку для резервного копіювання та моніторингу, значно підвищується надійність збереження важливих даних. Особливо важливим стало застосування цього рішення для бекапу таблиць PowerBi, які є критичними для аналітичних процесів в організації. Завдяки систематичному та автоматизованому підходу до резервного

копіювання, зараз усі таблиці PowerBi регулярно копіюються, що мінімізує ризик втрати цінної інформації.

Застосування програми не тільки забезпечило більш високу надійність зберігання даних, але й сприяло підвищенню загальної ефективності роботи аналітичних відділів, які тепер мають можливість оперативно відновлювати потрібну інформацію без зайвих затримок та втрат. Це, в свою чергу, дозволяє компанії швидше реагувати на змінювані умови ринку та приймати обґрунтовані рішення на основі актуальних даних.

Якісні результати впровадження розробленого програмного застосунку для резервного копіювання та моніторингу бекапів демонструють високий рівень задоволення серед користувачів, особливо аналітичних відділів.

Високу оцінку користувачі надали не лише основній функції застосунку—створення резервних копій, а також високо оцінили додаткові інструменти, які значно покращують ефективність їх роботи.

Однією з особливо цінних функцій став графік навантаження на мережу, який дозволяє аналітикам визначати оптимальний час для запуску процесів бекапування. Ця можливість не тільки ефективізує використання ресурсів мережі, але й знижує ризик затримок або перебоїв у роботі інших критичних додатків.

Також користувачі відзначають користь від системи логування, яка забезпечує детальний звіт про стан кожного бекапу. Логи допомагають швидко ідентифікувати та усувати проблеми, а також відстежувати історію бекапів для забезпечення дотримання внутрішніх політик безпеки даних.

Деякі користувачі висловили побажання щодо подальших удосконалень програми. Серед найпопулярніших запитів — розширення функціональності моніторингу за допомогою прогнозування майбутнього навантаження на мережу, а також інтеграція з іншими системами аналітики для автоматичного аналізу ефективності бекап-процесів.

Ці зворотні зв'язки свідчать про високу зацікавленість користувачів у розвитку застосунку та його інтеграції з розширеною екосистемою інструментів аналітики та управління даними.

Перспективами подальшого розвитку застосунку для резервного копіювання та моніторингу бекапів є інтеграція сучасних засобів сповіщення, що дозволить користувачам оперативно отримувати оновлення про стан резервних копій.

В подальшому планується впровадження функції відправки логів через Telegram та електронну пошту. Для Telegram буде використано їх офіційний API, що забезпечує швидку та безпечну передачу інформації прямо у месенджер користувача. Відправлення логів на електронну пошту реалізується за допомогою SMTP протоколу, що є стандартним способом для роботи з електронною поштою.

Ще одним важливим аспектом розвитку застосунку є його портабельність. Розробка Docker-образу застосунку дозволить користувачам легко встановлювати та запускати його в будь-якому середовищі, що підтримує Docker, незалежно від основної операційної системи. Це значно спростить процес деплойменту та забезпечить більшу гнучкість використання програми в різних IT-інфраструктура.

Додатково, планується розширення типів підтримуваних бекапів. Наразі програма підтримує лише повне копіювання даних, але в майбутньому з'явиться можливість виконувати інкрементальне та диференційне бекапування. Це дозволить користувачам економити місце на зберіганні та час на виконання бекапів, копіюючи лише змінені або нові файли.

Також заплановано впровадження функції тіньового копіювання, що дозволить створювати копії файлів, які в даний момент використовуються, без переривання їх роботи. Це особливо важливо для систем, де потрібно гарантувати безперервність робочих процесів.

Нарешті, розширення підтримуваних протоколів зв'язку до FTP додасть гнучкості в налаштуванні мережевого доступу до місць зберігання бекапів, дозволяючи використовувати не тільки SSH, але й більш традиційні для деяких системи FTP-з'єднання.

Всі ці удосконалення спрямовані на підвищення функціональності, зручності використання та універсальності застосунку, що робить його більш привабливим для широкого кола користувачів в різних організаційних та технічних умовах.

Застосунок для резервного копіювання та моніторингу бекапів може бути корисним для розробників, які потребують регулярного бекапу своїх проектів та баз даних. Автоматизація бекапу дозволяє зосередитись на розробці, не хвилюючись про втрату даних.

Цей розділ дозволяє зрозуміти, як програмне забезпечення може бути адаптовано для різних видів користувачів та їхніх потреб, що забезпечує його універсальність та широкий спектр застосування.

ВИСНОВКИ

1. В процесі виконання кваліфікаційної роботи, було досягнуто наступних результатів:
2. Проаналізувано можливості автоматизованих систем резервного копіювання та моніторингу бекапів. Визначено надійний тип резервного копіювання.
3. Визначено переваги та недоліки існуючих програмних засобів серед яких є обмежена сумісність роботи з різними ОС та мережевими ресурсами.
4. Розроблено функціональні та нефункціональні вимоги до застосунку для резервного копіювання та моніторингу бекапів.
5. Спроектувано архітектуру, визначити класи та методи для створення застосунку. Використано мову Python, фреймворк Customtkinter, бібліотеки: OS, paramiko, Matplotlib.
6. Розроблено інтерфейс програмного застосунку.
7. Проведено модульне та ручне тестування застосунку. В модульному тесті було написано unittest для функцій копіювання.
8. Робота пройшла апробацію:

   Шереметєв О.М., Негоденко О.В. Інноваційні підходи до автоматизації резервного копіювання та моніторингу. Всеукраїнська науково-технічна конференція “Застосування програмного забезпечення в інформаційно-комунікаційних технологіях”, 24 квітня 2024 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024. С. 259-260.

   Шереметєв О.М., Негоденко О.В. Розробка автоматизованої системи для оптимізованого резервного копіювання з використанням компресії. Всеукраїнська науково-практична конференція “Сучасні інтелектуальні інформаційні технології в науці та освіті”. 15 травня 2024 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024. С. 275-276.

ПЕРЕЛІК ПОСИЛАНЬ

1. Балабанов О. С. Аналітика великих даних: принципи, напрямки і задачі (огляд). *Проблеми програмування*. 2019. No 2. С. 47-68. URL: http://dspace.nbuv.gov.ua/xmlui/bitstream/handle/123456789/161487/05-Balabanov.pdf;jsessionid=D60607206FD710CE8DBEAE17F2658C87?sequence=1.
2. Barney N. Amazon Web Services [Електронний ресурс] / Nick Barney // Techtarget. – 2022. – Режим доступу до ресурсу: https://www.techtarget.com/searchaws/definition/Amazon-Web-Services.
3. T.K. B. Machine learning algorithms for social media analysis: A survey [Електронний ресурс] / B. T.K., R. Chandra Sekhara, B. Annushree // ScienceDirect. – 2021. – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/abs/pii/S1574013721000356.
4. Документація «AMQP (Advanced Message Queuing Protocol)» [Електронний ресурс] - Режим доступу: https://www.amqp.org/ 18.10.2023
5. Документація «RabbitMQ»:[Електронний ресурс] - Режим доступу: https://www.rabbitmq.com/ 18.10.2023
6. Article «Why Google Stores Billions of Lines of Code in a Single Repository»: - Rachel Potvin and Josh Levenberg [Електронний ресурс] - Режим доступу: https://dl.acm.org/doi/pdf/10.1145/2854146 18.10.2023
7. "Впровадження архітектур мікросервісів" Боріс Шоль, Лео Штудер, Майк Амундсен
8. "Сервісно-Орієнтована Архітектура: Концепції, Технології та Дизайн" Томас Ерл
9. Клієнт-серверна архітектура [Електронний ресурс] // QATestLab. – 2020. – Режим доступу до ресурсу: https://training.qatestlab.com/blog/technical articles/client-server-architecture/.

10. The top programming languages [Електронний ресурс] // GitHub. – 2022. – Режим доступу до ресурсу: https://octoverse.github.com/2022/top programming-languages.

11. "Шаблони проектування сервісів: Фундаментальні рішення для SOAP/WSDL та RESTful веб-сервісів" Роберт Деньо

12. Рейтинг мов програмування 2023 [Електронний ресурс] // Редакція DOU. – 2023. – Режим доступу до ресурсу: https://dou.ua/lenta/articles/language rating-2023/.

13. "Data Structures and Algorithms in Python", Michael T. Goodrich and Roberto Tamassia, 4th edition, Wiley, 2022.

14. "OSPF: The Basics", Juniper Networks, 2023.

15. Tennis Sense: A platform for extracting semantic information from multi-camera tennis data [Електронний ресурс]. Режим https://ieeexplore.ieee.org/document/5201152/authors#authors

16. Shvachych G., Moroz B., Ivaschenko O. , Sushko I., Pobochii I. The implementation features of the aggregation mode of network interface in the multiprocessors computing system. Комп'ютерне моделювання та оптимізація складних систем : Матеріали IV міжнар. науково-практ. конф., м. Дніпро, 1 – 2 листоп. 2018 р. Дніпро. 2018. С. 200 – 202.

17. Ільїн О.Ю., Катков Ю.І., Вергун Д.С., Шашлов А.В. Особливості розгортання мікросервісних додатків за допомогою системи керування контейнерами.

18. The Computer and the Brain (The Silliman Memorial Lectures Series) 3rd Edition by von Neumann, John (Author), Ray Kurzweil (Foreword), Yale University Press; 3 edition, 2012, 136 pages, ISBN-10: 0300181116, ISBN-13: 978-0300181111.

19. Computer Architecture: A Quantitative Approach 5th Edition by John L. Hennessy (Author), David A. Patterson (Author), Morgan Kaufmann; 5 edition (September 30, 2011), 856 pages, ISBN-10: 012383872X, ISBN-13: 978-8178672663.

20. Microservices vs. Service-Oriented Architecture. by Mark Richards. Publisher: O'Reilly Media, Inc.Release Date: April 2016. ISBN: 9781491975657.

21. Ivan Zmerzlyi Microservice architecture – [Електронний ресурс] – 2019 – Режим доступу: https://medium.com/@IvanZmerzlyi/microservice-architecture-f8a382291ff4. Дата доступу: жовтень 2023.

22. Xiao Ma Microservice Architecture at Medium – [Електронний ресурс] – 2019 – Режим доступу: https://medium.engineering/microservice-architecture-at-medium-9c33805eb74f. Дата доступу: жовтень 2023.

23. Netflix MSA Platform – MeltingCon – [Електронний ресурс] – 2019 – https://meltingcon.github.io/2018/assets/files/%EC%A0%95%EC%9C%A4%EC%A7%84.pdf. Дата доступу: жовтень 2023.

24. Docker Cookbook/ by Sébastien Goasguen. Copyright © 2016 Sébastien Goasguen. All rights reserved. Printed in the United States of America. Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

25. Чи завжди потрібні Docker, мікросервіси та реактивне програмування? – [Електронний ресурс] – 2019 – Режим доступу: https://www.dataart.com.ua/news/chi-zavzhdi-potribni-dockermikroservisi-ta-reaktivne-programuvannya. Дата доступу: жовтень 2023.

26. Kubernetes: Up and Running, 2nd Edition \ Dive into the Future of Infrastructure. By Brendan Burns, Kelsey Hightower, Joe Beda – Publisher: O'Reilly Media, Release Date: October 2023. Pages: 278.

27. Cloud Native DevOps with Kubernetes – by Justin Domingus, John Arundel. Publisher: O'Reilly Media, Inc. Release Date: March 2019. ISBN: 9781492040750.

28. Offcial Kubernetes documentation https://kubernetes.io/docs/concepts/overview/what-iskubernetes. Дата доступу: жовтень 2023.

29. Ключевые концепции Kubernetes для службы Azure Kubernetes (AKS) – [Електронний ресурс] – 2019 – Режим доступу: https://docs.microsoft.com/ru-ru/azure/aks/concepts-clustersworkloads. Дата доступу: жовтень 2023.

30. Ivaschenko V., Shvachych G., Ivaschenko O., Busygin, V. Improving the efficiency of multiprocessors system through in-line interface network aggregation. Системні технології. Дніпро. 2018. No 2(115). P. 84 – 93.Книга "Мікросервіси. Патерни розробки та рефакторинг" (Microservices: Design Patterns and Refactoring): авторство - G. Макклелланд, М. Джонсон, С. Хуттон.

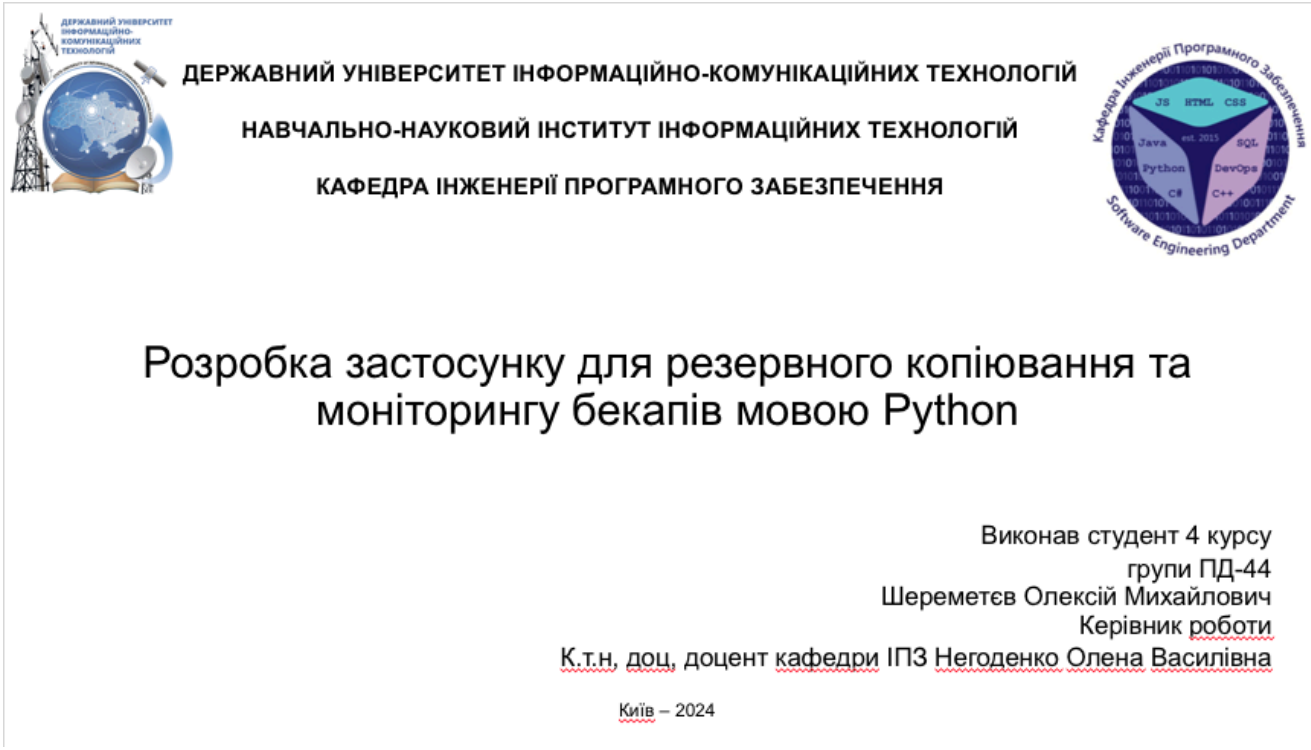
31. Cloud-assisted body area networks: State-of-the-art and future challenges – [Електронний ресурс]. Режим доступу: https://www.researchgate net/ publication/271918886_Cloud- assisted body area_networks_State-of-the-art_and_future_challenges.

32. The Fundamentals of Data Warehouse + Data Lake = Lake House. – Режим доступу:https://towardsdatascience.com/the-fundamentals-of-data-warehouse-data-lake-lake

33. Olena Vynokurova, Dmytro Peleshko, Marta Peleshko Hybrid Deep Convolutional Neural Network with Multimodal Fusion. In: Babichev S., Peleshko D., Vynokurova O. (eds) Data Stream Mining & Processing. DSMP 2020. Communications in Computer and Information Science. Springer, Cham. 2020. vol. 1158. pp. 62-78.

34. The Fundamentals of Data Warehouse + Data Lake = Lake House. – Режим доступу: https://towardsdatascience.com/the-fundamentals-of-data-warehouse-data-lake-lake

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ



МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** - автоматизація резервного копіювання та моніторингу бекапів за допомогою застосунку мовою Python.
- **Об’єкт дослідження** - процес резервного копіювання та моніторингу бекапів
- **Предмет дослідження** - програмне забезпечення для резервного копіювання та моніторингу бекапів.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Проаналізувати можливості автоматизованих систем резервного копіювання та моніторингу бекапів.
2. Визначити переваги та недоліки існуючих програмних засобів.
3. Розробити функціональні та нефункціональні вимоги до застосунку для резервного копіювання та моніторингу бекапів.
4. Спроектувати архітектуру, визначити класи та методи для створення застосунку.
5. Спроектувати інтерфейс програмного застосунку.
6. Розробити інтерфейс програмного застосунку.
7. Розробити програмне забезпечення
8. Провести модульне та мануальне тестування застосунку.

3

АНАЛІЗ АНАЛОГІВ

Функція / Програма	**Zinstall FullBack**	**Aomei Backupper Professional**	**EaseUS Todo Backup**	**EZBackup Done**
Повне резервне копіювання	+	+	+	+
Відновлення окремих файлів	+	+	+	+
Операційна система	Тільки Windows	Тільки Windows	Тільки Windows	Windows, Mac, Linux
Частота резервних копій	Налаштування не вказані	Обмежене налаштування	Обмежене налаштування	Повна гнучкість
Робота з мережевими ресурсами	Локально	Локально, глобально	Локально, глобально	Локально, глобально
Перевірка Firewall	Ні	Ні	Ні	Так
Графік навантаження	Ні	Ні	Так	Так

4

ВИМОГИ ДО ДОДАТКУ

Фунціональні:

1. Забезпечення повного резервного копіювання даних.
2. Відновлення окремих файлів та каталогів з резервними даними.
3. Налаштування частоти періодичного резервного копіювання даних.
4. Забезпечення перевірки Firewall копіюваних даних.
5. Побудова графіків навантаження на мережу.

Нефункціональні:

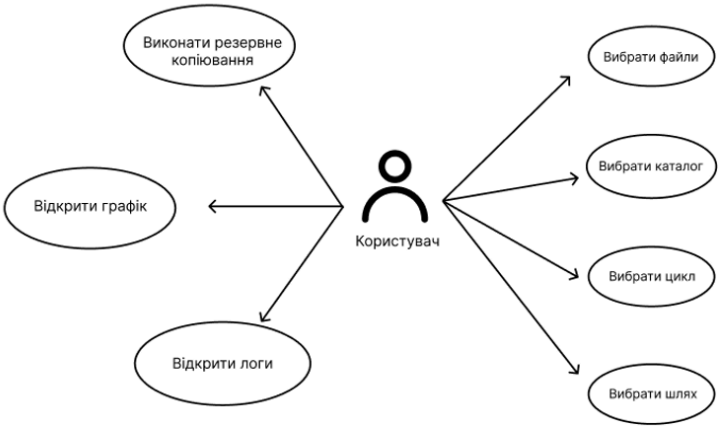
1. Підтримка різних ОС (Windows, Mac, Linux).
2. Застосування ssh протоколу для безпеки передачі даних.
3. Забезпечення підтримки резервного копіювання через мережеві
4. Запис логів резервного копіювання бекапів.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

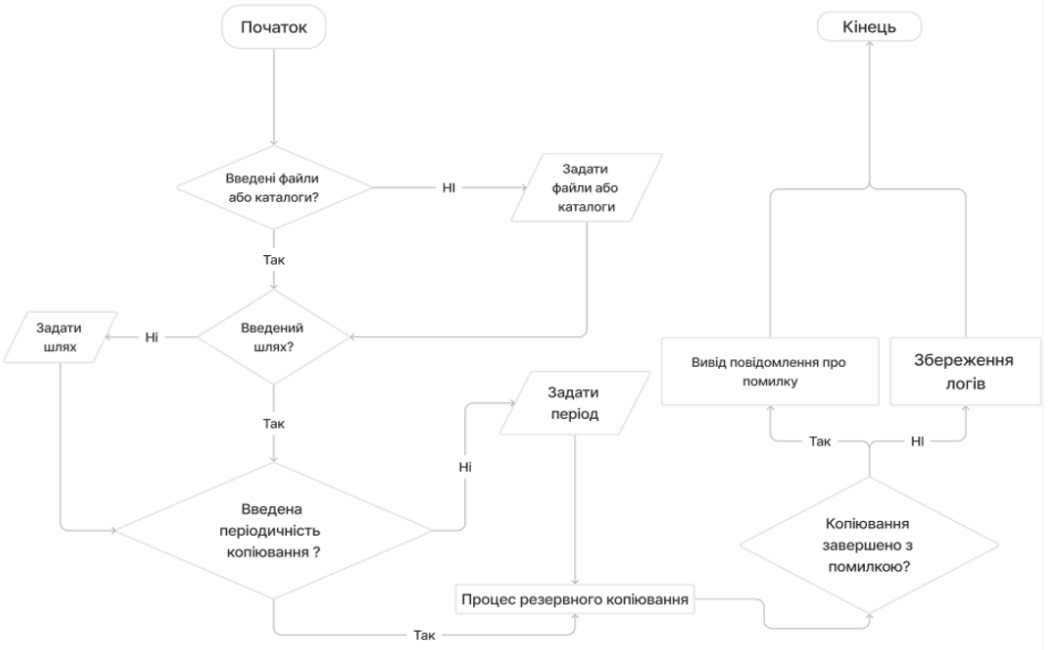
6

ДІАГРАМА ПРЕЦЕДЕНТІВ



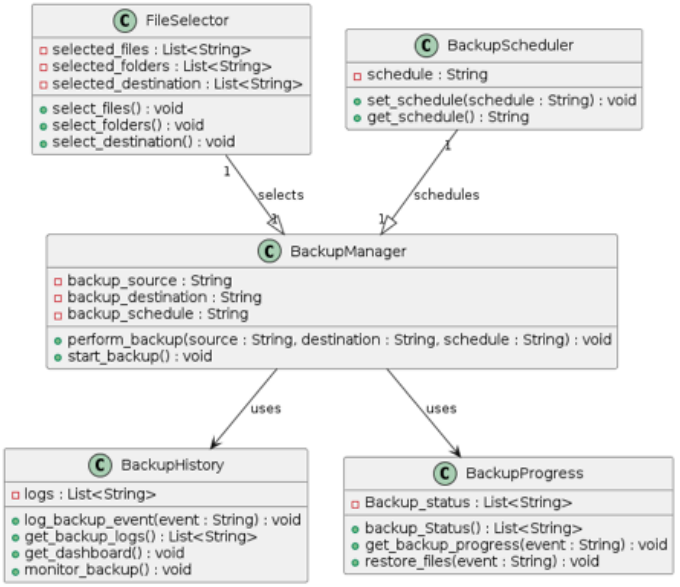
7

БЛОК-СХЕМА РЕЗЕРВНОГО КОПІЮВАННЯ



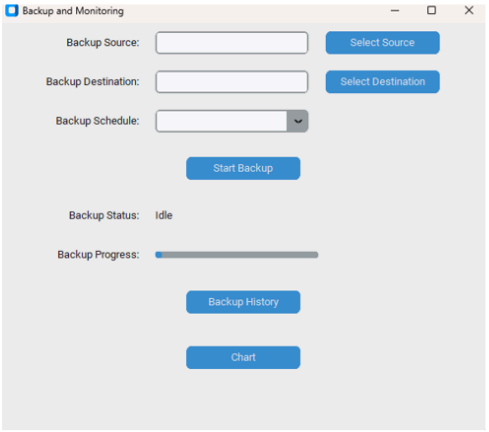
8

ДІАГРАМА КЛАСІВ

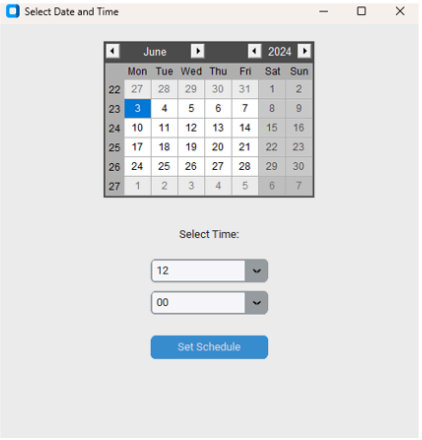


9

ЕКРАННІ ФОРМИ

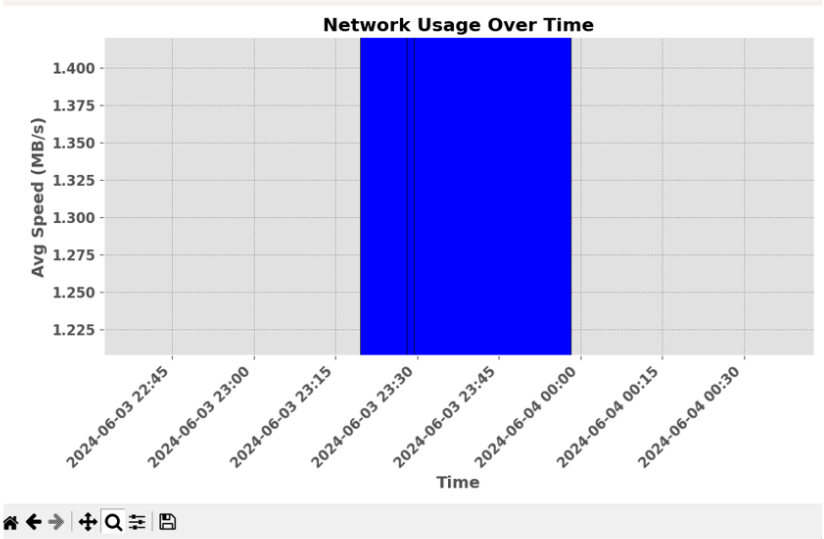


Головний екран додатку



Екран вибору періодичності

ЕКРАННІ ФОРМИ



Екран навантаженя бекапів

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Негоденко О.В., Шереметєв О.М. Іноваційні підходи до автоматизації резервного копіювання та моніторигу. Всеукраїнська науково-технічна конференція “Застосування програмного забезпечення в інфокомунікаційних технологіях”. Збірник тез. 24 квітня 2024 р., м. Київ, ДУІКТ, 2024. С. 259.;

2. Негоденко О.В., Шереметєв О.М. Розробка автоматизованої системи для оптимізованого резервного копіювання з використанням компресії. Всеукраїнська Науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті»”. Збірник тез.15 травня м. Київ, ДУІКТ, 2024. С. 275.;

ВИСНОВКИ

1. Проаналізовано можливості автоматизованих систем резервного копіювання та моніторингу бекапів.
2. Визначено переваги та недоліки існуючих програмних засобів серед яких є обмежена сумісність роботи з різними ОС та мережевими ресурсами.
3. Розроблено функціональні та нефункціональні вимоги до застосунку для резервного копіювання та моніторингу бекапів.
4. Спроектовано архітектуру, визначити класи та методи для створення застосунку. Використано мову Python,бібліотек: Customtkinter, OS, paramiko, Matplotlib.
5. Спроектовано інтерфейс програмного застосунку за допомогою додатку Figma.
6. Розроблено інтерфейс програмного застосунку.
7. Розроблене програмне забезпечення згідно визначинех вимог та архітектури.
8. Проведено модульне та мануальне тестування застосунку. В модульному тесті було написано unittest для функцій копіювання.

13

ДОДАТОК Б. ЛІСТИНГ ПРОГРАМИ

```
import os

import paramiko

from datetime import datetime

import shutil

import time


def create_remote_archive(sftp, ssh, src):

    base_name = os.path.basename(src.rstrip('/'))

    date_str = datetime.now().strftime('%Y-%m-%d_%H-%M-%S')

    archive_name = f"{base_name}_{date_str}.tar.gz"

    archive_path = f"/tmp/{archive_name}"


    # Create archive on the remote server

        stdin, stdout, stderr = ssh.exec_command(f"tar -czf {archive_path} -C
{os.path.dirname(src)} {base_name}")

    stdout.channel.recv_exit_status()  # Wait for the command to finish


    # Check for errors

    errors = stderr.read().decode()

    if errors:

        print(f"Errors during archiving on remote server: {errors}")

    else:

        print(f"Archive created on remote server: {archive_path}")


    return archive_path


def copy_local(src, dst):

    base_name = os.path.basename(src.rstrip('/'))

    date_str = datetime.now().strftime('%Y-%m-%d_%H-%M-%S')

    archive_name = f"{base_name}_{date_str}.tar.gz"

    archive_path = os.path.join(dst, archive_name)

          shutil.make_archive(os.path.splitext(archive_path)[0], 'gztar', src if
os.path.isdir(src) else os.path.dirname(src), base_name)
```

```
    return archive_path

def measure_network_speed(start_time, total_bytes_transferred):
    elapsed_time = time.time() - start_time
    speed = total_bytes_transferred / (1024 * 1024) / elapsed_time  # Speed in MB/s
    return speed

def copy_remote_to_local(src, dst, hostname, username, password, progress_callback):
    try:
        ssh = paramiko.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh.connect(hostname, username=username, password=password)

        sftp = ssh.open_sftp()
        os.makedirs(dst, exist_ok=True)

        # Create archive on remote server
        remote_archive_path = create_remote_archive(sftp, ssh, src)

        # Download the archive with speed measurement and progress callback
                                    local_archive_path     =     os.path.join(dst,
os.path.basename(remote_archive_path))
        total_bytes_transferred = 0
        start_time = time.time()
        speeds = []

        def callback(transferred, total):
            nonlocal total_bytes_transferred
            total_bytes_transferred = transferred
            speed = measure_network_speed(start_time, transferred)
            speeds.append(speed)
            if progress_callback:
                progress_callback(transferred, total)
```

```
        sftp.get(remote_archive_path, local_archive_path, callback=callback)

        # Clean up remote archive
        ssh.exec_command(f"rm {remote_archive_path}")

        sftp.close()
        ssh.close()

        print(f"Archive copied to local machine: {local_archive_path}")

        max_speed = max(speeds) if speeds else 0
        min_speed = min(speeds) if speeds else 0
        avg_speed = sum(speeds) / len(speeds) if speeds else 0
        return max_speed, min_speed, avg_speed

    except Exception as e:
        print(f"Failed to copy from remote to local via SSH: {e}")
        return 0, 0, 0

def copy_local_to_remote(src, dst, hostname, username, password, progress_callback):
    try:
        archive_path = copy_local(src, '/tmp')  # Create local archive in /tmp

        ssh = paramiko.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh.connect(hostname, username=username, password=password)

        sftp = ssh.open_sftp()
        remote_archive_path = os.path.join(dst, os.path.basename(archive_path))

        total_bytes_transferred = 0
        start_time = time.time()
        speeds = []
```

```
        def callback(transferred, total):
            nonlocal total_bytes_transferred
            total_bytes_transferred = transferred
            speed = measure_network_speed(start_time, transferred)
            speeds.append(speed)
            if progress_callback:
                progress_callback(transferred, total)

        sftp.put(archive_path, remote_archive_path, callback=callback)

        sftp.close()
        ssh.close()

        # Clean up local archive
        os.remove(archive_path)

        print(f"Archive copied to remote server: {remote_archive_path}")

        max_speed = max(speeds) if speeds else 0
        min_speed = min(speeds) if speeds else 0
        avg_speed = sum(speeds) / len(speeds) if speeds else 0
        return max_speed, min_speed, avg_speed

    except Exception as e:
        print(f"Failed to copy from local to remote via SSH: {e}")
        return 0, 0, 0


def process_backup_source(src, dst, src_hostname=None, src_username=None,
src_password=None, dst_hostname=None, dst_username=None, dst_password=None,
progress_callback=None):
    backup_time = datetime.now()
    if src_hostname and src_username and src_password:
          network_stats = copy_remote_to_local(src, dst, src_hostname, src_username,
src_password, progress_callback)
    elif dst_hostname and dst_username and dst_password:
```

```
          network_stats = copy_local_to_remote(src, dst, dst_hostname, dst_username, dst_password, progress_callback)

    else:

        copy_local(src, dst)

        network_stats = (0, 0, 0)  # No network stats for local copy

    return network_stats, backup_time
import customtkinter as ctk

from tkinter import filedialog, simpledialog, Toplevel, Text, Scrollbar, RIGHT, Y, END

from tkcalendar import Calendar

from datetime import datetime

from backup_logic import process_backup_source

import threading

import schedule

import time

import matplotlib.pyplot as plt

import matplotlib.dates as mdates

import os


from backup_history import log_backup, show_network_usage_chart   # Важливо додати імпорт show_network_usage_chart


class BackupApp(ctk.CTk):

    def __init__(self):

        super().__init__()


        self.title("Backup and Monitoring")

        self.geometry("600x500")


        # Variables to store selected file and folder paths

        self.backup_source = ""

        self.backup_destination = ""

        self.backup_schedule = None


        # Variables for SSH credentials
```

```
        self.ssh_source_hostname = None
        self.ssh_source_username = None
        self.ssh_source_password = None
        self.ssh_dest_hostname = None
        self.ssh_dest_username = None
        self.ssh_dest_password = None

        # Create and place the widgets
        self.create_widgets()

    def create_widgets(self):
        self.grid_columnconfigure(0, weight=1)
        self.grid_columnconfigure(1, weight=1)
        self.grid_columnconfigure(2, weight=1)

        self.backup_source_label = ctk.CTkLabel(self, text="Backup Source:")
        self.backup_source_label.grid(row=0, column=0, padx=10, pady=10, sticky="e")

        self.backup_source_entry = ctk.CTkEntry(self)
                self.backup_source_entry.grid(row=0,  column=1,  padx=10,  pady=10,
sticky="ew")

              self.select_file_button  =  ctk.CTkButton(self,  text="Select  Source",
command=self.select_source)
        self.select_file_button.grid(row=0, column=2, padx=10, pady=10, sticky="w")

        self.backup_dest_label = ctk.CTkLabel(self, text="Backup Destination:")
        self.backup_dest_label.grid(row=1, column=0, padx=10, pady=10, sticky="e")

        self.backup_dest_entry = ctk.CTkEntry(self)
        self.backup_dest_entry.grid(row=1, column=1, padx=10, pady=10, sticky="ew")

          self.select_folder_button = ctk.CTkButton(self, text="Select Destination",
command=self.select_destination)
```

```
                self.select_folder_button.grid(row=1, column=2, padx=10, pady=10,
sticky="w")


        self.schedule_label = ctk.CTkLabel(self, text="Backup Schedule:")

        self.schedule_label.grid(row=2, column=0, padx=10, pady=10, sticky="e")


              self.schedule_combobox = ctk.CTkComboBox(self, values=["", "daily",
"weekly", "monthly", "every 3 months"], command=self.show_calendar)

        self.schedule_combobox.set("")

        self.schedule_combobox.grid(row=2, column=1, padx=10, pady=10, sticky="ew")


              self.start_backup_button = ctk.CTkButton(self, text="Start Backup",
command=self.start_backup)

             self.start_backup_button.grid(row=3, column=0, columnspan=3, pady=20,
sticky="n")


            self.backup_history_button = ctk.CTkButton(self, text="Backup History",
command=self.display_backup_history)

            self.backup_history_button.grid(row=4, column=0, columnspan=3, pady=20,
sticky="n")


                          self.chart_button = ctk.CTkButton(self, text="Chart",
command=self.display_chart)

        self.chart_button.grid(row=5, column=0, columnspan=3, pady=20, sticky="n")


    def select_source(self):

         self.ssh_source_hostname = simpledialog.askstring("SSH Hostname", "Enter SSH
hostname (leave blank for local):", parent=self)

        if self.ssh_source_hostname:

             self.ssh_source_username = simpledialog.askstring("SSH Username", "Enter
SSH username:", parent=self)

             self.ssh_source_password = simpledialog.askstring("SSH Password", "Enter
SSH password:", show='*', parent=self)

              remote_path = simpledialog.askstring("Remote Path", "Enter remote file
path:", parent=self)

            if remote_path:

                self.backup_source_entry.delete(0, 'end')

                self.backup_source_entry.insert(0, remote_path)

        else:
```

```
            source_path = filedialog.askopenfilename()
            if source_path:
                self.backup_source_entry.delete(0, 'end')
                self.backup_source_entry.insert(0, source_path)


    def select_destination(self):
          self.ssh_dest_hostname = simpledialog.askstring("SSH Hostname", "Enter SSH
hostname (leave blank for local):", parent=self)
        if self.ssh_dest_hostname:
              self.ssh_dest_username = simpledialog.askstring("SSH Username", "Enter
SSH username:", parent=self)
              self.ssh_dest_password = simpledialog.askstring("SSH Password", "Enter
SSH password:", show='*', parent=self)
                remote_path = simpledialog.askstring("Remote Path", "Enter remote
directory path:", parent=self)
            if remote_path:
                self.backup_dest_entry.delete(0, 'end')
                self.backup_dest_entry.insert(0, remote_path)
        else:
            destination_path = filedialog.askdirectory()
            if destination_path:
                self.backup_dest_entry.delete(0, 'end')
                self.backup_dest_entry.insert(0, destination_path)


    def show_calendar(self, choice):
        if choice in ["daily", "weekly", "monthly", "every 3 months"]:
            self.calendar_window = ctk.CTkToplevel(self)
            self.calendar_window.title("Select Date and Time")
            self.calendar_window.geometry("400x400")


            self.calendar = Calendar(self.calendar_window, selectmode='day')
            self.calendar.pack(pady=20)


                self.time_label = ctk.CTkLabel(self.calendar_window, text="Select
Time:")
            self.time_label.pack(pady=10)
```

```
                        self.hour_combobox = ctk.CTkComboBox(self.calendar_window, values=[f"{i:02d}" for i in range(24)])
            self.hour_combobox.set("12")
            self.hour_combobox.pack(pady=5)

                        self.minute_combobox = ctk.CTkComboBox(self.calendar_window, values=[f"{i:02d}" for i in range(60)])
            self.minute_combobox.set("00")
            self.minute_combobox.pack(pady=5)

                self.schedule_button = ctk.CTkButton(self.calendar_window, text="Set Schedule", command=self.set_schedule)
            self.schedule_button.pack(pady=20)

    def set_schedule(self):
        selected_date = self.calendar.get_date()
        selected_hour = self.hour_combobox.get()
        selected_minute = self.minute_combobox.get()

        schedule_time = f"{selected_date} {selected_hour}:{selected_minute}"
        self.backup_schedule = datetime.strptime(schedule_time, "%m/%d/%y %H:%M")

        self.calendar_window.destroy()
        print(f"Backup scheduled for: {self.backup_schedule}")

    def start_backup(self):
        self.backup_source = self.backup_source_entry.get()
        self.backup_destination = self.backup_dest_entry.get()

        if self.backup_source and self.backup_destination:
                            print(f"Starting backup from {self.backup_source} to {self.backup_destination}")
            if self.backup_schedule:
                self.schedule_backup()
```

```
            else:
                self.execute_backup()
        else:
                print("Please select or enter both a source file and a destination
folder.")


    def execute_backup(self):
        process_backup_source(self.backup_source, self.backup_destination,
                                    self.ssh_source_hostname, self.ssh_source_username,
self.ssh_source_password,
                                        self.ssh_dest_hostname, self.ssh_dest_username,
self.ssh_dest_password)


    def schedule_backup(self):
        now = datetime.now()
        delay = (self.backup_schedule - now).total_seconds()


        if delay > 0:
            def job():
                self.execute_backup()
                choice = self.schedule_combobox.get()
                if choice == "daily":

schedule.every().day.at(self.backup_schedule.strftime("%H:%M")).do(self.execute_back
up)
                elif choice == "weekly":

schedule.every().week.at(self.backup_schedule.strftime("%H:%M")).do(self.execute_bac
kup)
                elif choice == "monthly":
                            schedule.every().month.at(self.backup_schedule.strftime("%d
%H:%M")).do(self.execute_backup)
                elif choice == "every 3 months":
                          schedule.every(3).months.at(self.backup_schedule.strftime("%d
%H:%M")).do(self.execute_backup)


            self.after(int(delay * 1000), job)
        else:
```

```
            print("Scheduled time is in the past. Please select a future time.")


    def display_backup_history(self):
        # Відкриття вікна для відображення історії резервного копіювання
        pass


    def display_chart(self):
        show_network_usage_chart()  # Виклик графіка при натисканні кнопки "Chart"



if __name__ == "__main__":
    def run_scheduler():
        while True:
            schedule.run_pending()
            time.sleep(1)


    scheduler_thread = threading.Thread(target=run_scheduler)
    scheduler_thread.daemon = True
    scheduler_thread.start()


    app = BackupApp()
    app.mainloop()
import os
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
import matplotlib.dates as mdates


def log_backup(backup_time, backup_source, backup_destination, ssh_source_hostname,
ssh_source_username, schedule_type, network_stats):
     backup_time_str = backup_time.strftime("%Y-%m-%d %H:%M:%S") if backup_time else
"N/A"
     source_str = f"{ssh_source_hostname} ({backup_source})" if ssh_source_hostname
else backup_source
    max_speed, min_speed, avg_speed = network_stats
    with open("backup_log.txt", "a") as log_file:
```

```
        log_entry = (
            f"Backup executed at: {backup_time_str}\n"
            f"Source: {source_str}\n"
            f"Destination: {backup_destination}\n"
            f"Remote Backup: {'Yes' if ssh_source_hostname else 'No'}\n"
            f"SSH User: {ssh_source_username if ssh_source_hostname else 'N/A'}\n"
            f"Max Speed: {max_speed:.2f} MB/s\n"
            f"Min Speed: {min_speed:.2f} MB/s\n"
            f"Avg Speed: {avg_speed:.2f} MB/s\n"
            "--------------------------------------\n"
        )
        log_file.write(log_entry)

def get_last_n_backups(n=5):
    try:
        with open("backup_log.txt", "r") as log_file:
            logs = log_file.read().split("--------------------------------------\n")
            logs = [log for log in logs if log.strip()]  # Remove empty logs
            logs = logs[-n:]  # Get the last n logs
        return logs
    except FileNotFoundError:
        return []

def read_backup_logs():
    try:
        with open("backup_log.txt", "r") as log_file:
            logs = log_file.read()
        return logs
    except FileNotFoundError:
        return "No backup history found."

def plot_network_usage_chart():
    logs = get_last_n_backups(5)
    timestamps = []
```

```
    speeds = []

    for log in logs:
        lines = log.split("\n")
        for line in lines:
            if line.startswith("Backup executed at:"):
                date_str = line.split(":", 1)[1].strip()
                if date_str != "N/A":
                    try:
                                timestamps.append(datetime.strptime(date_str, "%Y-%m-%d
%H:%M:%S"))
                    except ValueError:
                        print(f"Skipping invalid date format: {date_str}")
            elif line.startswith("Avg Speed:"):
                speed_str = line.split(":", 1)[1].strip().split()[0]
                try:
                    speeds.append(float(speed_str))
                except ValueError:
                    print(f"Skipping invalid speed format: {speed_str}")
    if len(timestamps) != len(speeds):
        print("Mismatch in the number of timestamps and speeds.")
        return None, None

    if not timestamps or not speeds:
        print("No data available for the chart.")
        return None, None

    return timestamps, speeds
```