Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

другий (магістерський)
(назва освітнього ступеня)

на тему: Розробка системи зберігання резервних копій даних в хмарних сховищах на базі C# та Asp.Net Core

Виконав: студент 6 курсу, групи СПм-61
спеціальності 121- Інженерія програмного забезпечення
(шифр і назва спеціальності)

(підпис) Гарасівка А. В.
(прізвище та ініціали)

Керівник (підпис) д.т.н., проф. Пастух О. А.
(прізвище та ініціали)

Нормоконтроль (підпис) к.ф.-м.н, доц. Бойко І. В.
(прізвище та ініціали)

Завідувач кафедри (підпис) д.ф.-м.н., проф. Петрик М. Р.
(прізвище та ініціали)

Рецензент (підпис) д.т.н., проф. Лупенко С. А.
(прізвище та ініціали)

Тернопіль
2020

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ
УНІВЕРСИТЕТ ІМЕНІ ІВАНА ПУЛЮЯ

Факультет комп'ютерно-інформаційних систем та програмної інженерії

Кафедра Програмної інженерії

Рівень вищої освіти - другий (магістерський)

Спеціальність 121-Інженерія програмного забезпечення

(код і повна назва)

Тип програми освітньо-наукова програма.

Освітня програма Інженерія програмного забезпечення.

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові _____Гарасівці Андрію Володимировичу_____

(прізвище, ім'я, по батькові)

1. Тема роботи _ Розробка системи зберігання резервних копій даних в хмарних сховищах на базі C# та Asp.Net Core_____
затверджена наказом університету від "___" _____20 ___ р № _____

заповнюється вручну після отримання наказу

2. Термін подання студентом роботи до екзаменаційної комісії: _____2020 р.
3. Вихідні дані до роботи *алгоритми роботи програми, ілюстраційні матеріали, пояснювальна записка, схеми та діаграми, слайди презентації*_____
4. Перелік питань, що потрібно опрацювати в роботі *мета роботи, аналіз проблемної галузі і постановка задачі, огляд методів створення та зберігання резервних копій даних іншими реалізаціями, способи збереження локальних резервних копій іншими реалізаціями, методи авторизації в хмарні сервіси, методи роботи з хмарними сховищами.*_____

5 Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім’я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
4.1 Охорона праці	Осухівська Галина Михайлівна		
4.2 Безпека в надзвичайних ситуаціях	Клепчик Василь Михайлович		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка *
1.	Аналіз предметної галузі		
2.	Огляд існуючих методів		
3.	Основна частина		
4.	Підготовка пояснювальної записки		
5.	Спецчастина		
6.	Підготовка презентації та доповіді		
7.	Попередній захист		
8.	Нормоконтроль, рецензування		
9.	Занесення диплома в електронний архів		
10.	Допуск до захисту у зав. кафедри		
* заповнюється вручну після виконання чергового пункту			

Дата видачі завдання _____ _____2020 р.

Студент _____

(підпис)

Керівник роботи _____ _____ _____

(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Атестаційна робота магістра містить: 72 с., 23 рис.,6 табл., 22 джер.

БЕЗПЕКА ДАНИХ, НАСТІЛЬНИЙ ДОДАТОК, РЕЗЕРВНЕ КОПІЮВАННЯ, ХМАРНІ ТЕХНОЛОГІЇ, ASP.NET CORE, C#, CLOUD, .NET FRAMEWORK, KESTREL, REST, WINFORMS.

Метою роботи є дослідження та розробка рішення для автоматизованого резервного копіювання даних з використанням хмарних технологій.

Методи розробки базуються на інструментах розробки настільних-застосувань з використанням платформи .NET, протоколу передачі даних HTTP, локальним серверним програмним забезпеченням Kestrel, фреймворку ASP.NET Core для розробки веб-застосунків на платформі .NET Core.

В результаті роботи розглянуто методи резервного копіювання даних, алгоритми роботи існуючих рішень, та розроблено програмну реалізацію системи автоматичного резервного копіювання, яка представляє собою настільний-додаток на основі фреймворку .NET Framework WinForms, та локальний сервер для багато-платформного середовища .NET Core для можливості завантаження даних на локальний сервер.

ASP.NET CORE, BACKUP, CLOUD, CLOUD TECHNOLOGIES, C#, DESKTOP APP, DATA SECURITY, .NET FRAMEWORK, KESTREL, WINFORMS, REST.

The aim of the work is to research and develop a solution for automated data backup using cloud technologies.

Development methods are based on desktop application development tools using the .NET platform, HTTP data transfer protocol, local Kestrel server software, ASP.NET Core framework for developing web applications on the .NET Core platform.

As a result, the methods of data backup, algorithms for existing solutions, and software implementation of automatic backup system, which is a desktop application based on the .NET Framework WinForms framework, and a local server for multi-platform .NET Core environment for the possibility. upload data to the local server.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface (програмний інтерфейс)

Back-end – серверна (прихована) частина програмної системи

Backup, бекап – Резервна копія даних

CLI – Common Language Infrastructure (платформа, на якій виконуються програми .Net)

DI - Dependency injection (механізм впровадження залежностей).

GUI – Graphical User Interface (інтерфейс користувача).

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту).

JSON – JavaScript Object Notation (нотація об’єктів JavaScript).

ORM – Object–Relational Mapping (об’єктно–реляційне відображення).

REST – Representational State Transfer (передача репрезентативного стану).

SQL – Structured Query Language (мова структурованих запитів).

UML – Unified Modeling Language (уніфікована мова моделювання)

URL – Uniform Resource Locator (уніфікований локатор ресурсу).

БД – База даних.

ОС – Операційна система.

ПЗ – Програмне забезпечення.

СУБД – Система управління базами даних.

ЗМІСТ

ВСТУП

Створення резервних копій – обов’язкова частина будь-якої надійної системи. Резервне копіювання – це процес створення копії даних, призначення якої – відновлення даних в результаті пошкодження, видалення чи втрати доступу. Наявність резервних частин системи, необхідна умова для систем, які повинні виконувати роботу навіть після аварійного завершення роботи чи в разі втрати чи пошкодження критично важливих даних.

Дані - це результати роботи користувачів чи комп’ютера, в сучасному світі це одна з речей які найлегше втратити через коротке замикання чи втрату пристрою чи ненавмисне видалення. Це можуть бути Ваші персональні фото, важливі документи чи звіти менеджерів, електронні таблиці бухгалтерії, схеми конструкторського бюро з 50 річною історією чи стан бази даних адміністраторів компанії з мільйонами користувачів їх систем. На сьогоднішній день, персональні чи робочі дані людей - одна з найбільших цінностей. Адже у випадку їх видалення, пошкодження, втрати чи викрадення можуть привести до сумних наслідків: безсонні ночі переписування документів, провалені терміни виконання робіт чи завдань, завалений диплом, втрата звітності чи унікальних креслень, позиви в суд з багатомільйонними штрафами через безвідповідальне ставлення до системи безпеки даних.

На сьогоднішній день актуальною залишається проблема резервного копіювання даних користувачів без достатнього досвіду роботи за персональним комп’ютером. Часто користувачі залишають документи в випадкових папках на робочому столі, а при пошкодженні накопичувача чи персонального комп’ютера ці дані, які були в одному екземплярі стають недоступні. Вирішенням проблеми стане автоматизоване створення резервних даних користувачів, яке можна налаштувати.

Автоматизоване створення резервних копій даних буде важливим елементом комп’ютерних систем і мереж користувачів, команд чи організацій, які відповідально ставляться до захисту даних. Резервна копія даних дає можливість виконати відновлення інформації в разі втрати чи пошкодження. Для забезпечення

збереження даних на достатньому рівні резервні копії необхідно зберігати на кількох незалежних накопичувачах, а також робити перевірки (тести), чи правильно дані відновлюються після виконання резервного копіювання.

Одним з найкращих місць для довготривалого зберігання резервних копій даних є хмарне сховище. На поточному етапі розвитку широкосмугового і безлімітного доступу до інтернету хмарні технології дозволяють завантажувати і зберігати велетенські обсяги інформації. Провайдери хмарних сховищ надають готову, надійну, перевірену інфраструктуру для завантаження, збереження і обробки ваших файлів на їхніх серверах. Користувачу не потрібно переживати за резервне живлення для його сервера, виділену IP-адресу чи налаштування маршрутизатора для коректного доступу до своїх серверів чи сервісів резервного копіювання. Сучасні хмарні рішення надійно дозволяють відслідковувати авторизацію та отримання доступу до ваших даних іншими людьми чи додатками, тому ви завжди будете знати коли і хто змінив той чи інший файл.

Метою дослідження є – порівняльний аналіз існуючих програмних рішень для виконання резервного копіювання та визначення їхніх проблем та недоліків.

Об’єктом дослідження виступає процес архівування та збереження резервної копії на інший накопичувач або хмарний сервіс.

Предметом дослідження є – проектування та розробка системи зберігання резервних копій даних в хмарних сховищах на базі C# та Asp.Net Core.

Інноваційність результатів дослідження дозволить поєднати автоматизоване створення резервних копій даних, а використати хмарні технології для заощадження дискового місця на комп’ютері користувача. Таким чином тільки користувач отримує доступ до різних версій даних в себе в авторизованому хмарному сховищі і йому не прийдеться використовувати безліч розрізнених програм для постійної синхронізації даних з хмарою.

Актуальність теми резервного копіювання також розглянуто в публікації «НЕОБХІДНІСТЬ РЕЗЕРВНОГО КОПІЮВАННЯ ДАНИХ В ПОВСЯКДЕННОМУ ЖИТТІ».

1 АНАЛІТИЧНА ЧАСТИНА

1.1 Опис предметної області

З кожним днем кількість інформації збереженої і обробленої інформації зростає і займає все більше місця на наших комп’ютерах, серверах чи інших накопичувачах. А тому зростає і розмір цінних для нас даних, які потребують захисту – важливі документи, річна звітність, бази даних з мільйонами записів чи архів фотографій за останні кілька років, віртуальні машини з тисячами улюблених сайтів чи баланс вашого банківського рахунку.

Якщо вірити статистиці [1] основними причинами втрати даних є збої апаратного забезпечення (43%), помилки користувача (30%), помилка ПЗ (12%), комп’ютерні віруси (7%), крадіжка (5%), інші причини(3%). Як видно помилки, які призводять до втрати конфіденційних даних можуть виникати, як через апаратну (фізичну) так і з програмною складовою комп’ютера.

Попередити чи запобігти всім можливим загрозам для ваших даних неможливо, але мінімізувати негативний вплив від втрати даних може добре налаштоване, автоматичне, резервне копіювання даних. Воно дозволить, як відновити стан втрачених документів, так і «відкотити» зміни при внесенні помилок в дані.

Проблема втрати повного обсягу або цілісності даних може виникнути в будь-який момент, в будь-якого користувача чи комп’ютера. Зазвичай це відбувається доволі несподівано і ніхто до цього не буває готовий. Але вже після першого інциденту втрати даних і безсонних ночей спроб їх відновити або оплати чеку сервісу по відновленню інформації, люди нарешті згадують або починають створювати резервні копії даних.

Резервне копіювання або бекап (англ. backup) — процес створення копії даних з носія (жорсткого диска, дискети тощо), призначений для відновлення цих даних у разі їх пошкодження або видалення [2].

Виконання резервного копіювання даних дає можливість виконати відновлення інформації при втраті або пошкодженні оригіналу, для якого було виконано резервне копіювання. Втрата може включати: навмисне чи не навмисне видалення, шифрування інформації вірусом-вимагачем, пошкодження файлів вірусом, редагування даних без можливості подальшого використання, втрата прав доступу до даних, відмова технічного обладнання, помилка драйвера пристрою, помилки запису або читання файлів операційною системою.

Основним і найдешевшим накопичувачем (на гігабайт інформації) досі залишається магнітна стрічка (використовується рідко) і жорсткий диск. Ніякий виробник не зможе гарантувати роботу своїх пристроїв роками, особливо в різних середовищах. Наприклад, центри обробки даних (дата-центри) кожного кварталу публікують статистику по пристроях накопичення інформації, де можна оцінити жорсткі диски якого виробника є найнадійнішими чи довговічнішими [3].

Надійність резервного копіювання можна багаторазово збільшити за допомогою правильного розкладу та дублювання інформації на кілька незалежних пристроїв, в тому числі з використанням хмарного сховища, як місце довготривалого зберігання даних.

Хмарне сховище (англ. cloud storage) - це модель зберігання даних у комп'ютері, в якій цифрові дані зберігаються в логічні пули, а фізичне зберігання охоплює кілька серверів (зазвичай у кількох місцях). Фізичне середовище, як правило, належить хостинговим компаніям, вони ж і керують цим середовищем. Ці постачальники хмарних систем зберігання даних відповідають за зберігання наявної інформації та доступ до неї, а також за роботу фізичного середовища. Користувачі можуть купувати у постачальників послуг хмарного сховища можливість зберігати там дані [3].

Для доступу до послуг хмарного сховища можна використовувати веб-інтерфейс або фірмові додатки, які використовують API. В окремих випадках провайдер надає доступ до API і будь-який розробник може інтегрувати доступ до сховища в свій додаток.

1.2 Аналіз стану розв'язання проблеми і існуючих аналогів

Для вирішення проблеми безпеки даних зазвичай досить ПЗ, яке наглядно і прозоро буде копіювати ваші дані в надійне сховище. Навіть найпростіше дублювання інформації з системного диска (не розділу!) на інший диск вирішує може допомогти в майбутньому.

На сьогодні на ринку ПЗ для резервного копіювання представлено достатньо багато рішень, в тому числі від великих компаній. Такі рішення можуть підтримувати велику кількість функцій, пристроїв, сценаріїв роботи, проте при цьому доволі складні.

Також існують випадки, коли користувачу чи команді користувачів необхідно інтегрувати рішення для резервного копіювання даних в свій програмний продукт – зазвичай це неможливо, через закритість API.

1.2.1 Acronis True Image

Acronis True Image – комп’ютерне ПЗ для резервного копіювання, відновлення, а також перенесення даних для ОС Microsoft Windows, Apple Mac OS. Надає можливість створення завантажувальних носіїв (флеш, диск), які можуть запустити програму перед завантаженням основної ОС. Також надає такі інструменти: клонування диска, універсальний відновник, паралельний доступ [5].

Клонування диска це інструмент копіювання розділів одного диска на інший, може використовуватися з метою перенесення ОС та/або файлів користувача на інший диск.

Створення «рятівного» носія це корисний інструмент для створення завантажувального оптичного диска або флеш-накопичувача. Така флешка чи диск дозволяє запустити до завантаження ОС огляд, копіювання і відновлення розділів.

Універсальний відновник це інструмент створення повної резервної копії комп’ютера на зовнішній носій для подальшого швидкого відновлення дієздатності

комп'ютера в разі настання несправності операційної системи чи втрати певних файлів.

Паралельний доступ це функція отримання доступу до основного комп'ютера з іншого пристрою. Інтерфейс програми зображено на рисунку 1.1.

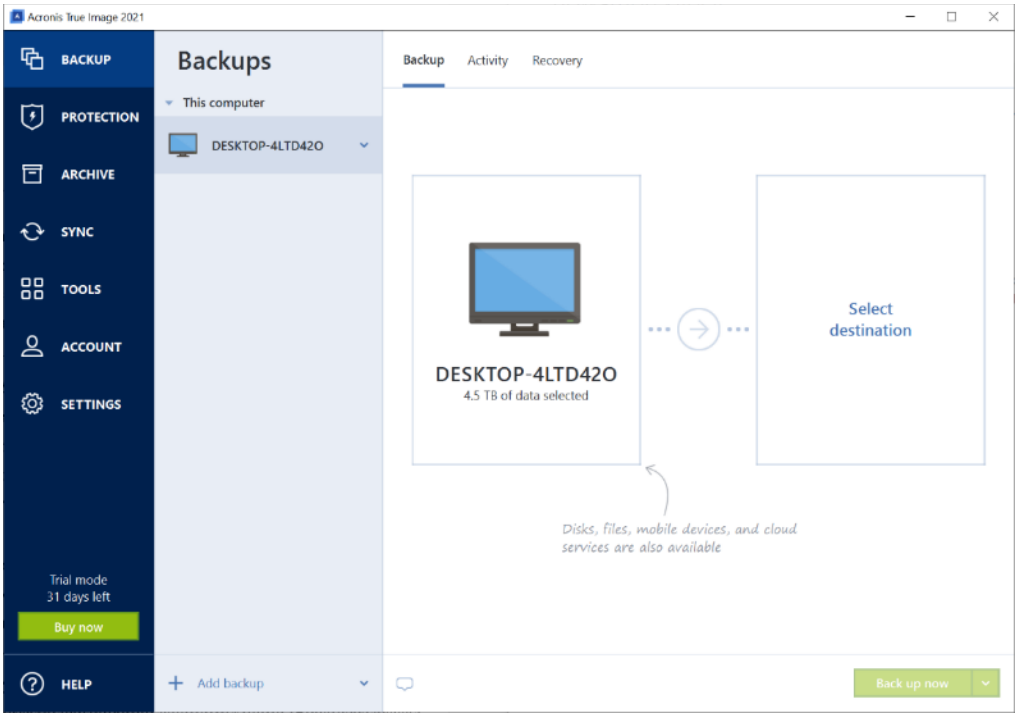

Рисунок 1.1 – головне вікно програми Acronis True Image

Загалом Acronis приємно дивує комплексністю підходів а також можливістю завантаження повної резервної копії даних у власне хмарне сховище – Acronis Cloud. Проте мінусом буде суттєва ціна у варіантах з великим розміром сховища даних 100$ в рік за 1ТБ сховища.

1.2.2 Veritas NetBackup

Veritas NetBackup (раніше Symantec NetBackup - до того, як Symantec продала Veritas) - це пакет ПЗ для резервного копіювання та відновлення для великих підприємств. Він забезпечує міжплатформену функцію резервного копіювання для великої кількості операційних систем Windows, UNIX та Linux.

NetBackup має центральний головний сервер, який управляє як медіасерверами (що містять резервні носії), так і клієнтами. Основні серверні

платформи включають Solaris, HP-UX, AIX, Tru64, Linux та Windows [6]. Інтерфейс оновленого головного вікна зображено на рисунку 1.2.

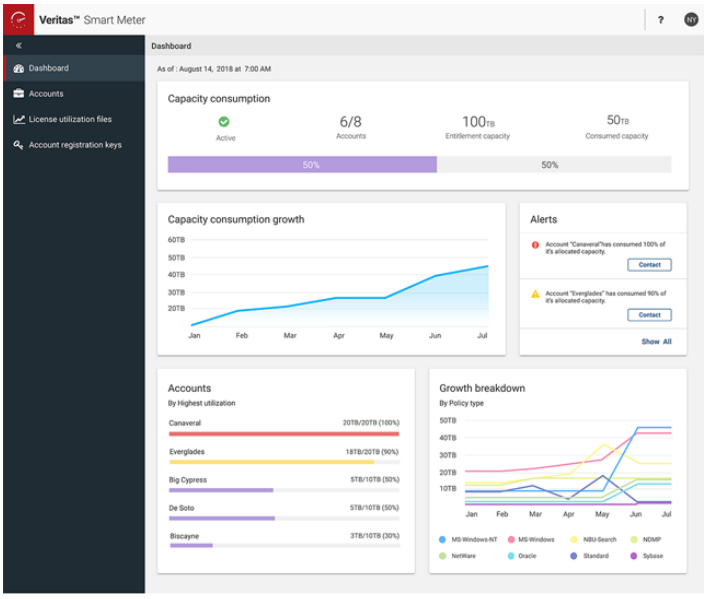

Рисунок 1.2 – Інтерфейс головного вікна Veritas NetBackup Smart Meter

1.2.3 Bacula

Bacula - це комп'ютерне ПЗ з відкритим вихідним кодом для резервного копіювання для комп'ютерних мереж на рівні підприємства. Він призначений для автоматизації завдань резервного копіювання, які часто вимагали втручання системного адміністратора або оператора комп'ютера [7]. Клієнти є для багатьох ОС. Вигляд головного вікна програми для Linux зображено на рисунку 1.3.

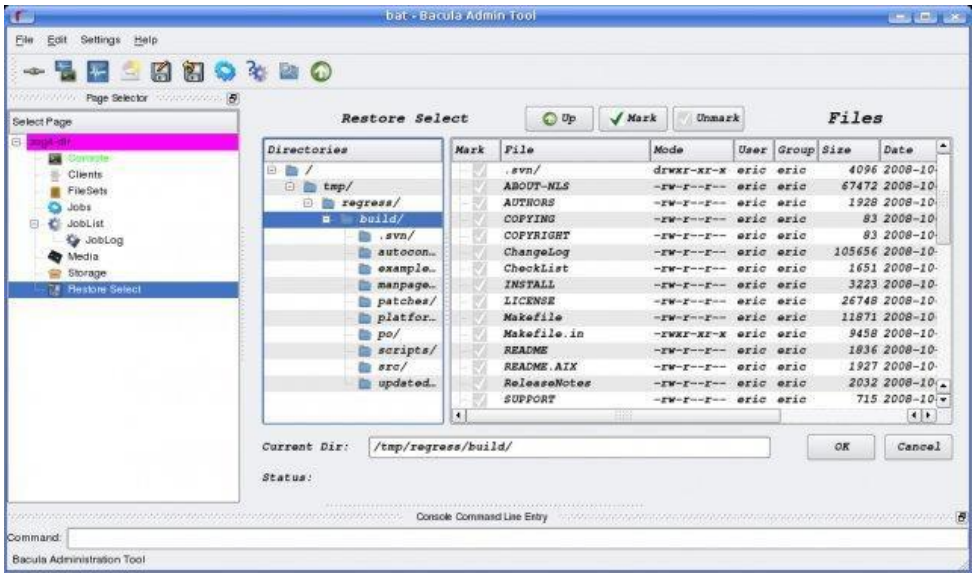

Рисунок 1.3 – Вигляд головного вікна Bacula Admin Tool

Bacula підтримує клієнти резервного копіювання Linux, UNIX, Windows та macOS, а також ряд професійних пристроїв резервного копіювання, включаючи накопичувачі на магнітній стрічці. Адміністратори та оператори можуть налаштувати систему через консоль командного рядка, графічний інтерфейс або веб-інтерфейс. його back-end - це каталог даних, що зберігається в MySQL, PostgreSQL або SQLite [7].

Через відкритість коду кожен може перевірити цю систему на відповідність необхідним вимогам, проте це все одно не дає гарантій, як було складено це ПЗ.

Отже, було виконано аналіз стану вирішення проблеми щодо збереження резервних копій даних для визначення рівня вирішеності та обґрунтування мети роботи. На сьогоднішній день лише один представник з досліджених має наближений до ідеального, функціонал для збереження резервних копій даних в хмарному сховищі (Acronis Cloud). Далі можна охарактеризувати вимоги до об’єкту дослідження.

1.3 Аналіз вимог. Визначення вимог

Основною вимогою до класу ПЗ для резервного копіювання можна визначити надійне збереження даних не тільки на одному пристрої. Звичайно, можна просто зберегти копію на інший накопичувач чи комп’ютер і заспокоїтися, проте не все так просто. ПЗ повинно самостійно виконувати взаємодію з файлами користувача в фоновому режимі, з використанням хмарного сховища користувача, а тому додаткові вимоги до безпеки неминучі. Вимоги, які ставить бізнес до ПЗ резервного копіювання даних відомі давно [8], їх необхідно лише описати.

Вимоги до програмної системи часто класифікуються як функціональні, нефункціональні й вимоги предметної області.

Функціональні вимоги:

1. Система повинна надавати можливість авторизуватися в провайдера хмарного сховища (авторизація і автентифікація).

2. Система повинна завантажувати резервну копію даних в хмарне сховище.
3. Система повинна вміти відновлювати дані з резервної копії.
4. Система повинна вміти створювати резервні копії даних на вимогу користувача та за визначеним розкладом (період чи певний час дня).
5. Система повинна вміти архівувати дані для забезпечення їх цілісності.
6. Серверний додаток повинен завантажувати та зберігати дані на сервері.

Не функціональні вимоги:

1. Взаємодія між клієнтом і хмарним сховищем відбувається тільки після успішної авторизації і автентифікації за допомогою токенів авторизації.
2. Система повинна зберігати дані авторизації локально (на комп'ютері користувача) і максимально прозоро для користувача.
3. Клієнтський додаток повинен відповідати максимально швидко (1-5 секунд), при виконанні складних чи довготривалих запитів необхідно ділити результати на сторінки з рівною кількістю результатів (пагінація).
4. Система повинна вести журнал подій, який повинен відображатися користувачу та записуватися в файли на локальному комп'ютері.
5. Резервна копія містить всі необхідні об'єкти, зі збереженням структури файлів і папок.
6. Сервер додатків повинен дозволяти доступ тільки авторизованим IP-адрес або користувачам, для усіх інших IP-адрес заборонити доступ.

Клієнтський додаток повинен мати свою систему авторизації для запобіганню втручанню третіх осіб. Також повинен надавати відкрите API для інших додатків.

Система повинна ігнорувати ідентифікаційні дані і використовувати тільки токени авторизації, для запобігання несанкціонованого доступу.

Для взаємодії з зовнішніми API хмарних провайдерів, необхідно використовувати наявні офіційні SDK від самих провайдерів хмарних сховищ (наприклад Drive.Api для Google Drive чи Dropbox.Api для Dropbox). Клієнтський

додаток повинен мати іконку в системному треї операційної системи (MS Windows) та відображати поточний стан резервного копіювання.

Всі активності в системі (зміна чи додавання хмарного провайдера даних, зміна стану сценарію резервної копії) повинні записуватися та відображатися на логах системи, демонструватися користувачу при необхідності.

Резервна копія повинна відображати реальний стан об’єкта резервної копії (папка чи файл), містити повну структуру об’єктів, з вкладеними папками та файлами та забезпечувати 100% відтворення стану файлів у разі відновлення інформації.

1.4 Аналіз ризиків проекту

Аналіз ризиків – це важливий етап проектування будь-якої системи чи проекту. Спрощено ризик можна розуміти як імовірність прояву яких-небудь несприятливих обставин, що негативно впливають на реалізацію проекту. Можна виділити три типи ризиків:

1. *Ризики для проекту,* які впливають на графік робіт або ресурси, необхідні для виконання проекту. Відображені в таблиці 1.1.

2. *Ризики для розроблювального продукту,* що впливають на якість або продуктивність розроблювального програмного продукту. Відображені в таблиці 1.2.

3. *Бізнес-Ризики,* що відносяться до організації-розроблювача або постачальника.

Таблиця 1.1. Можливі ризики програмних проектів

Ризик	Тип ризику	Опис ризику
Викрадення чи використання інформації	Ризик для проекту, ризик для продукту	Викрадення інформації для авторизації клієнта в службі хмарного сховища
Зміни в умовах використання хмарних провайдерів	Ризик для проекту, ризик для продукту	Зміна умов використання може заборонити використання API хмарних провайдерів
Блокування хмарних сховищ	Ризик для проекту, ризик для продукту	Повне чи часткове блокування хмарних сховищ може позбавити користувача його резервних копій
Відсутність апаратних засобів	Ризик для проекту	Апаратні засоби, які необхідні для серверного додатку, не готові до експлуатації
Зміна вимог	Ризик для проекту	Поява великої кількості непередбачених змін у вимогах, пропонованих до розроблювального ПЗ, призводить до змін пропорційних поточному прогресу над проектом
Затримка в розробці специфікації	Ризик для продукту	Специфікації основних підсистем не надійшли до розробників відповідно до графіка робіт або надійшли в невірному вигляді
Недооцінка розміру розроблювальної системи	Ризик для проекту	Термін виконання проекту перевищив приблизні попередні оцінки
Поява конкурентів	Бізнес-ризик	Поява на ринку конкуруючого рішення

Таблиця 1.2. Список ризиків після проведення їхнього аналізу

Ризик	Імовірність	Ступінь збитку
Викрадення чи використання інформації	Середня	Катастрофічна
Зміни в умовах використання хмарних провайдерів	Висока	Серйозна
Блокування хмарних сховищ	Середня	Серйозна
Відсутність апаратних засобів	Середня	Терпима
Зміна вимог	Низька	Терпима
Затримка в розробці специфікації	Низька	Терпима
Недооцінка розміру розроблювальної системи	Низька	Терпима
Поява конкурентів	Середня	Катастрофічна

Отже, в наслідок аналізу потенційних ризиків при розробці проекту було виявлено найбільш пріоритетні загрози, і завдяки цьому можна виділити найбільш критичні етапи розробки ПЗ, де потрібно звернути увагу на безпеку резервних копій даних користувача та авторизацію в хмарних сховищах.

1.5 Аналіз результатів дослідження

Виконане дослідження підтвердило сумніви щодо можливості використання існуючих програмних засобів для створення резервних копій даних в купі з хмарним сховищем користувача. Наявне ПЗ вміє використовувати хмарне сховище власної розробки (Acronis) проте ніхто не може гарантувати надійність збереження даних користувача та обженення доступу до них. Також це ПЗ володіє доволі високою ціною і пропонує часом надлишок функцій, які ускладнюють інтерфейс і взаємодію користувача з системою.

Вирішенням проблеми буде спроба розробити власний програмний додаток (клієнт і сервер) для створення резервних копій користувача та зберігання їх в авторизованих хмарних сховищах.

Отже, в результаті аналізу предметної області, виявлення потенційних небезпек для даних користувача, порівняння існуючих рішень на ринку ПЗ для резервного копіювання даних було визначено вимоги до комп’ютерної системи, яка б при реалізації мінімізувала негативний вплив від втрати даних, була б безпечною, дешевшою і більш практичною для кінцевого користувача чи невеличких команд користувачів, яким необхідно рішення яке можна було б інтегрувати в власне ПЗ. Для цього необхідно надавати власне «доступне» API, яким ніякий інший «софт» не може похвалитися.

2 ТЕОРЕТИЧНА ЧАСТИНА

2.1 Вибір архітектури

Вибір архітектури ПЗ — дуже важливий етап для подальшого проектування. Архітектура це спосіб структурування програмної системи, вона дозволяє відокремити окремі рівні абстракції елементів системи на ранніх фазах розробки проеку. Система зазвичай складається з кількох рівнів абстракції, для коректного розподілу роботи над кожним з них. Обраний архітектурний шаблон стане фундаментом для всіх наступних кроків проектування програмного продукту.

Для виконання всіх завдань і функцій програмного продукту буде достатньо одного клієнтського додатку встановленого на комп’ютері користувача та додаткового сервера, якщо такий знадобиться, який буде додатковим провайдером сховища, для якого дані будуть зберігатися не в хмарному сховищі, а на локальному сервері користувача.

Оптимальним вибором архітектури для проектованої системи буде клієнт-серверна архітектура [9]. Вона дозволить поділити логіку програмної системи на дві незалежні частини – клієнтський додаток та серверний. Концептуальна схема взаємодії зображена на рисунку 2.1.

Клієнт-серверна архітектура це один із архітектурних шаблонів ПЗ та є домінуючою концепцією у сфері створення застосунків з можливістю мережевої взаємодії і обміну даними між ними. Вона передбачає такі основні компоненти:

- набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;
- набір клієнтів, які використовують сервіси, що надаються серверами;
- мережа, яка забезпечує взаємодію між клієнтами та серверами.

Для проектуваннння клієнт-серверного додатку нам підійде будь-який настільний додаток з графічним інтерфейсом користувача для ОС Microsoft

Windows, який буде приємний та знайомий користувачу, а також серверний додаток, який може бути встановлений на будь-якій серверній ОС.

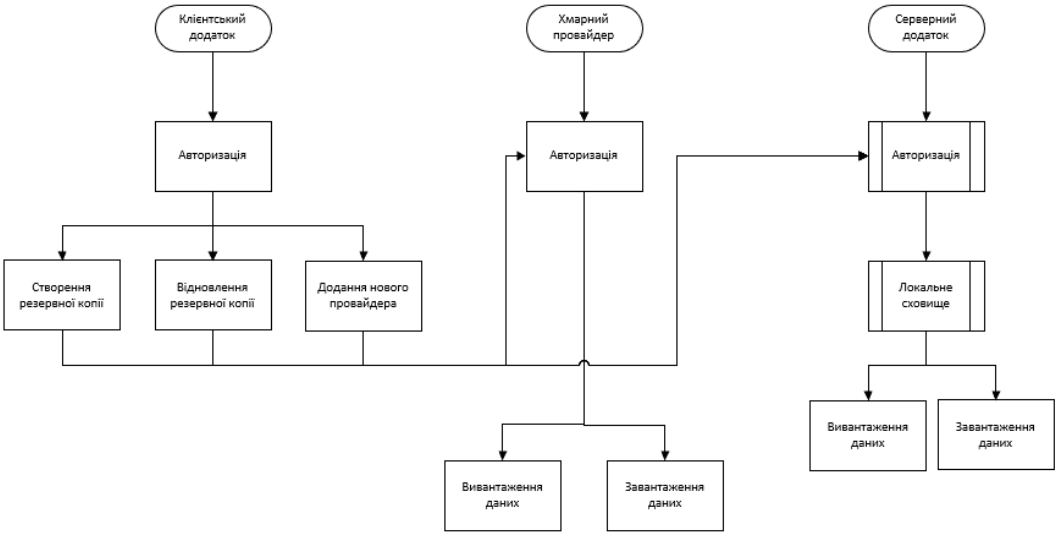

Рисунок 2.1 – концепція архітектури майбутньої системи

Отже, проектування необхідної системи буде включати клієнтський додаток з графічним інтерфейсом, а також серверний додаток для зберігання резервних копій в локальній мережі.

2.2 Проектування прецедентів системи

Визначення відношень між акторами та прецедентами в системі є важливим етапом до узагальнення та опису майбутніх варіантів використання системи. Відношення відображають набір можливих дій чи операцій з системою, а також учасників цих операцій. Необхідність варіантів використання складно переоцінити, так як вона дозволяє концептуально оцінити масштаби системи, набір функцій, які повинні виконуватися та пов’язати їх з відповідальними акторами.

Діаграма є графом, що складається з набору прецедентів (овалів), акторів (чоловічки) та зв’язків між ними.

До акторів системи можна віднести:

- Кінцевий користувач – основний клієнт програмного продукту, який взаємодіє зі всіма функціями клієнтського додатку, такими як

створення резервних копій даних, відновлення даних, додавання та менеджмент підключеними провайдерами даних.

- Провайдер хмарного сховища – сервіс який надає послуги зберігання даних на своїх віддалених серверах через мережу інтернет. Надає ресурси (сервер) та відповідну інфраструктуру для керування хмарним сховищем для користувача.
- Адміністратор сервера – фактично це може бути і кінцевий користувач, за наявності сервера – цей актор надає доступ до свого сервера, де можна встановити другу частину програмного продукту – серверну частину.

Варіанти використання інформаційної системи представлені на наступній діаграмі (рис. 2.2).

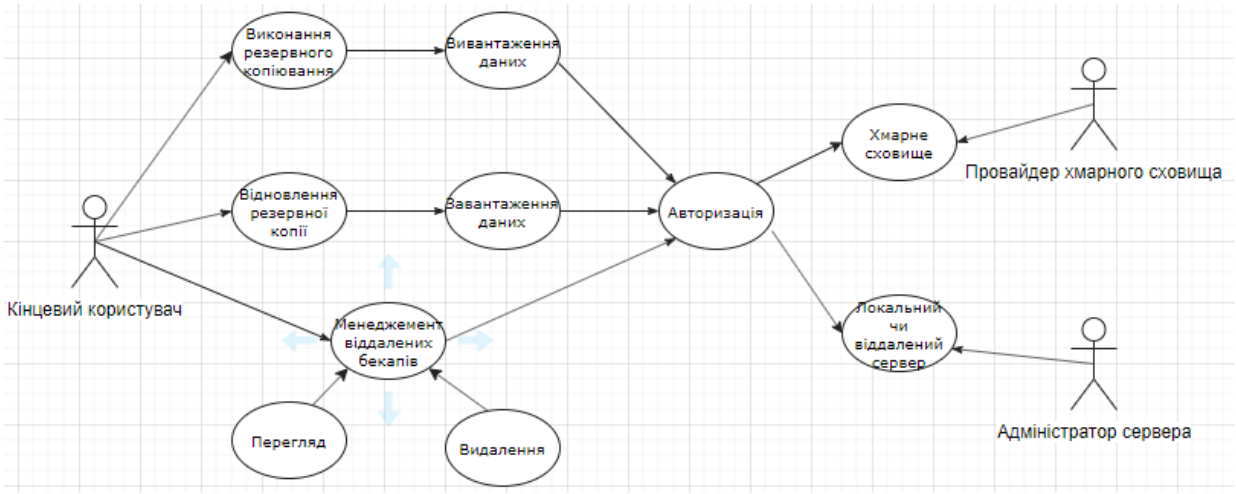

Рисунок 2.2 – Діаграма варіантів використання

Отже, на діаграмі зображено кілька робочих операцій.

Перш за все це потік виконання резервного копіювання, для цього необхідно, щоб користувач створив сценарій резервного копіювання та вибрав шлях до необхідних даних. Після цього програма стисне дані в архів та спробує авторизуватися автоматично до хмарного сховища користувача для подальшого завантаження резервних копій даних.

Наступний потік виконання це відновлення даних. Тут все доволі схоже: необхідно, щоб користувач створив сценарій для відновлення резервної копії та

вибрав шлях до необхідних даних. Після цього програма спробує авторизуватися автоматично до хмарного сховища користувача для подальшого завантаження резервних копій даних в тимчасову директорію та подальшого відновлення даних.

Останній потік виконання передбачає менеджмент віддалених резервних копій даних, наприклад видалення з хмарного сховища. Це можна виконувати для того щоб тримати на віддаленому сервері лише кілька останніх резервних копій даних або щоб обмежити використання місця в хмарному сховищі.

2.3 Проектування діаграми послідовностей

Діаграма послідовності це певний різновид діаграми в UML, вона відображає взаємодії об’єктів системи в хронологічному порядку. Також діаграма послідовності відображає послідовність відправлених повідомлень та пов’язані об’єкти, як внутрішні, так зовнішні.

Діаграма послідовності створення резервної копії даних та завантаження їх на сервер (рисунок 2.3).

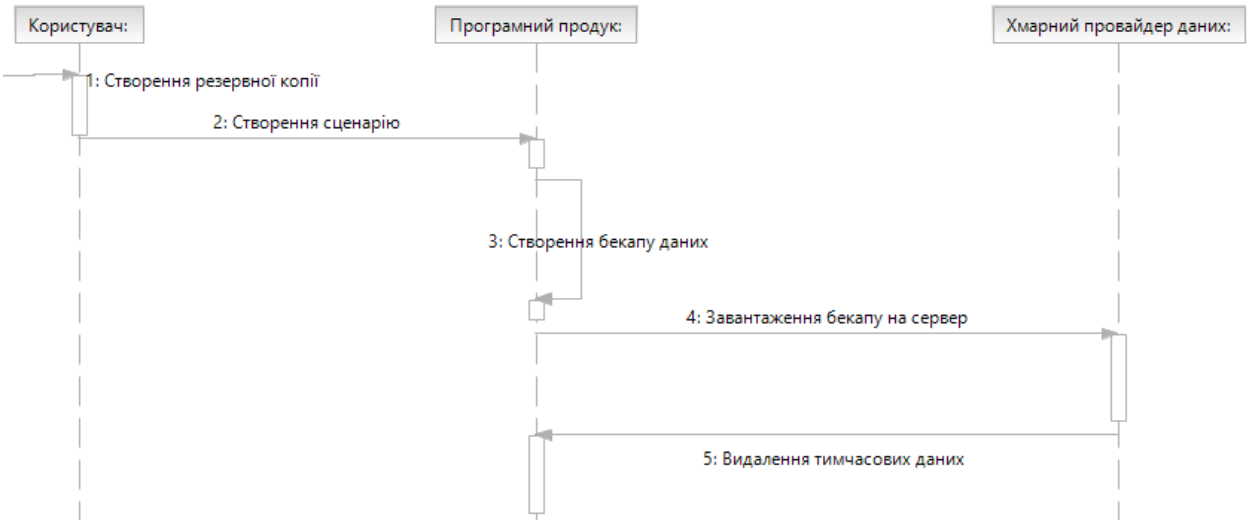

Рисунок 2.3 – Діаграма послідовності створення бекапу

Послідовність говорить сама за себе: для створення резервної копії користувач створює сценарій, після цього система створює резервну копію та

стискає дані, після цього стиснений архів відправляється на сервер для подальшого зберігання.

Діаграма послідовностей етапів відновлення даних до стану резервної копії з сервера (рисунок 2.4) включає зворотній набір операцій системи з хмарним сховищем. Для цього необхідно для початку перевірити наявність резервних копій на віддаленому сховищі та завантажити їх список. Далі одну з цих копій необхідно завантажити та розпакувати в тимчасову директорію. Після відновлення даних тимчасову копію можна видалити.

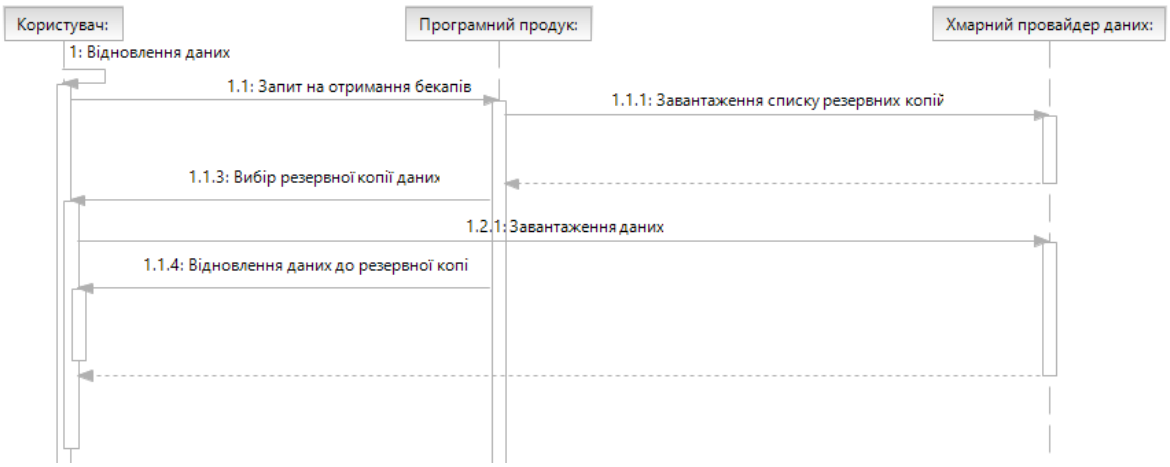

Рисунок 2.4 – Діаграма послідовності відновлення бекапу

2.4 Проектування діаграми класів

Діаграма класів в UML - це тип статичної структурної діаграми, що описує концептуальну структуру системи, показуючи класи системи, їх атрибути, функції (або методи) та взаємозв'язки між класами та об’єктами. Діаграма класів є дуже важливим елементом об’єктно-орієнтованого моделювання. Вона використовується для загального концептуального моделювання структури програми та для детального моделювання переведення моделей у код програми. Діаграми класів також можуть бути використані для моделювання даних.

Клас на діаграмі класів зазвичай позначається у вигляді прямокутника поділеного на 3 частини: заголовок класу чи інтерфейсу (назва класу), атрибути класу (це можуть бути поля, властивості, події – в залежності від обраної мови

програмування чи фреймворку) та секція методів (або функцій). Після двокрапки для атрибутів і методів вказується тип (інший клас, інтерфейс, перелічення або вбудований тип фреймворку), який повертається методом. Це позначення фактично накладає певні обмеження – оголошує контракт класу, і визначає набір необхідної функціональності для нього.

Також на діаграмі класів відображаються зв’язки, вони можуть бути різних типів.

Залежність - це однонаправлений зв’язок між залежними та незалежними елементами моделі. Існує між двома елементами, якщо зміни у визначенні одного елемента (сервера або цілі) можуть спричинити зміни для іншого (клієнта або джерела).

Асоціація представляє родину посилань. Асоціацію можна назвати, а кінці асоціації можна прикрасити іменами ролей, показниками власності, кратністю, видимістю та іншими властивостями.

Агрегація є варіантом взаємозв'язку "має", агрегація є більш конкретною, ніж асоціація.

Композиція відображає елементи єдиної системи. Елементи не можуть існувати як окремі частини, відокремлені від конкретної системи.

Спадкування вказує на те, що один із двох пов'язаних класів (підклас) вважається спеціалізованою формою іншого (супер тип), а суперклас - узагальненням підкласу. На практиці це означає, що будь-який екземпляр підтипу є також екземпляром суперкласу.

На відображеній схемі – рисунок 2.5 зображено основні елементи системи, такі як Ядро (Core), сценарій (BackupScript), крок (BackupStep), аккаунт (CloudStorageAccount), провадер для комунікації з хмарним сховищем (CloudStorageProviderBase), а також його реалізації DropboxCloudProvider і GoogleDriveCloudProvider.

Клас Ядро (Core) – це кореневий елемент, який має створюватися в одному екземплярі (паттерн Одинак) відразу після запуску програми, завантажувати всі

необхідні модулі (файли .dll), ініціалізувати плагіни (такі як CloudStorageManager чи SingleLogger), завантажувати налаштування користувача (клас UserSettings).

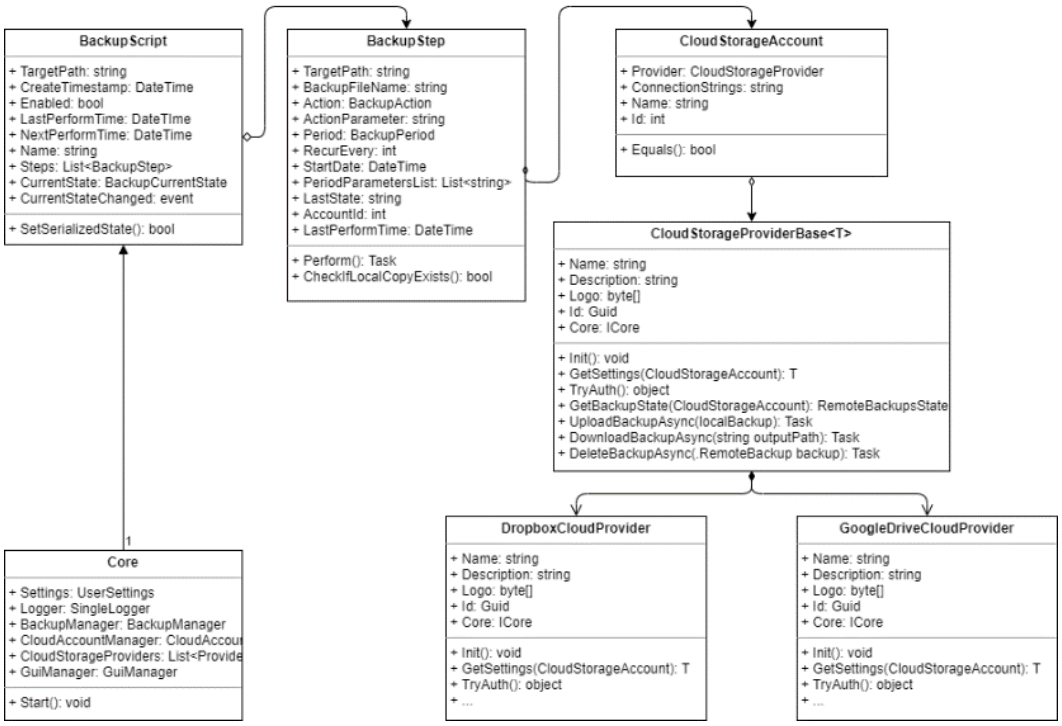

Рисунок 2.5 – діаграма класів майбутньої системи

Клас сценарій (BackupScript) – містить логіку і дані необхідні для виконання резервного копіювання. Саме об’єкти цього класу будуть використовуватися для візуального відображення рядків в таблиці на панелі «Скрипти». Сценарій містить початковий шлях для файлів, а також ім’я і набір кроків виконання, які містять необхідні операції користувача. Метод SetSerializedState() необхідний для оновлення закешованих даних, наприклад збережених кроків в JSON.

Клас крок (BackupStep) – містить дані необхідні для виконання поточної операції в процесі резервного копіювання. Цей клас містить необхідні параметри, як от вибраний аккаунт хмарного сховища для резервного копіювання, ім’я для вихідних файлів резервних копій, шлях до файлів чи папки (може відрізнятися) від шляху скрипта, налаштування періодичності створення резервної копії, рядок відображення останнього стану виконання кроку (порожній означає що все нормально). Також наявна дата початку виконання і дата останнього виконання. Метод Perform необхідний для початку виконання дій, CheckIfLocalCopyExists() –

для перевірки наявності локальної копії даних. При відсутності локальної копії програма повинна запропонувати відновити (при наявності віддаленої копії даних) дані в ту ж саму директорію. Ще є параметри для задання повтору операції, і варіанти для вибору дат (дні тижня чи місяці).

Клас аккаунт (CloudStorageAccount) – містить необхідні властивості (збережені в форматі JSON) для авторизації в API хмарного сховища. Серіалізація параметрів з'єднання дозволяє зробити аккаунт незалежним від форматів записів даних авторизації для різних видів API і легко їх зберігати в вигляді рядка (TEXT) в базі даних.

Базовий абстрактний клас провадер (CloudStorageProviderBase) визначає основні методи, які необхідно буде реалізувати класам-наступникам і базову реалізацію методу GetSettings(Account), який спростить використання всередині класів-наступників.

Класи DropboxCloudProvider і GoogleDriveCloudProvider класи наступники від CloudStorageProviderBase і саме вони несуть необхідні операції для взаємодії з API Dropbox.API і Google.Apis.Discovery.v1, Google.Apis.Drive.v3 відповідно. Для взаємодії з API рекомендується прочитати відповідну документацію.

2.5 Проектування діаграми активностей

Діаграми діяльності - це графічні зображення (в UML) робочих процесів поетапних діяльностей та дій з підтримкою вибору, ітерації та паралельності. В UML діаграми діяльності призначені для моделювання як обчислювальних, так і організаційних процесів (тобто робочих процесів), а також потоків даних, що перетинаються з відповідними діями.

Хоча діаграми діяльності в основному показують загальний потік контролю, вони також можуть включати елементи, що показують потік даних між діями через один або кілька сховищ даних (додаток Б, рисунок 1). На цій схемі зображено виконання програми з умовами в залежності від необхідної дії користувача.

Початок і кінець відображаються заокругленими прямокутниками або кругами. Виконання операції відображаються прямокутником, очікування на ввід/вивід користувача – ромб. За допомогою елементів вибору та циклів можна відобрази варіативність програми та можливі шляхи використання.

Діаграми діяльності можна розглядати як форму структурованої блок-схеми в поєднанні з традиційною діаграмою потоків даних. У типових техніках блок-схеми відсутні конструкції для вираження паралельності. Однак символи об’єднання та розділення на діаграмах діяльності вирішують це лише для простих випадків; значення моделі не зрозуміле коли їх довільно поєднують з рішеннями або циклами. В UML версії 1.0 діаграми діяльності були спеціалізованою формою діаграм стану, тому вони не дуже підходять для відображення комплексної картини стану програми. В UML версії 2.0 діаграми діяльності були реформовані на основі мережевої семантики Петрі, збільшивши обсяг ситуацій, які можна моделювати за допомогою діаграм діяльності.

2.5 Вибір технологій розробки

Для розв’язання визначених завдань необхідно обрати технологію розробки, а також підходящий фреймворк для роботи та створення ПЗ. Для побудови настільних додатків з графічним інтерфейсом підійдуть безліч фреймворків, від QT чи PySimpleGUI або Electron. Проте серед технологій необхідно вибрати ту, яка вже давно зарекомендувала себе на ринку ПЗ для ОС Microsoft Windows і яка надає зручні елементи проектування програмних додатків. Тому вибір пав на .NET Framework.

.NET Framework (читається дот-нет) — програмна технологія, запропонована фірмою Microsoft як платформа для створення як звичайних програм, так і веб-застосунків. Багато в чому є продовженням ідей та принципів, покладених в технологію Java. Однією з ідей .NET є сумісність служб, написаних різними

мовами. Хоча ця можливість рекламується Microsoft як перевага .NET, платформа Java має таку саму можливість [10].

Кожна бібліотека (збірка) в .NET має свідчення про свою версію, що дозволяє усунути можливі конфлікти між різними версіями збірок [10].

.NET поділяється на дві основні частини — середовище виконання (по суті віртуальна машина) та інструментарій розробки [10].

.NET Framework починався як закрите програмне забезпечення, хоча розробник працював над стандартизацією пакета програм майже відразу, навіть до його першого випуску. Незважаючи на ці зусилля, спрямовані на стандартизацію, розробники, переважно ті, що належать до спільнот вільного програмного забезпечення та програм із відкритим кодом, висловили своє занепокоєння обраними умовами та перспективами будь-якого впровадження вільного та відкритого коду, особливо щодо патентів на програмне забезпечення. З тих пір Microsoft змінила підхід до розробки .NET, щоб більш точно слідувати сучасній моделі проекту який розробляється спільнотою, включаючи випуск оновлень своїх патентів, обіцяючи вирішити проблеми.

Також, необхідно використовувати, як мінімум, ще одну фундаментальну технологію розробки баз даних – SQL.

SQL (англ. Structured query language — мова структурованих запитів) — декларативна мова програмування для взаємодії користувача з базами даних, що застосовується для формування запитів, оновлення і керування реляційними БД, створення схеми бази даних та її модифікації, системи контролю за доступом до бази даних. Сама по собі SQL не є ані системою керування базами даних, ані окремим програмним продуктом [11].

Отже, саме ці дві технології стануть фундаментом для подальших рішень. На їх основі можна буде приступати до практичної частини реалізації програмного проекту.

3 ПРАКТИЧНА ЧАСТИНА

3.1 Вибір інструментів розробки

3.1.1 C# та Microsoft Visual Studio

В якості інструментів розробки були вибрані такі засоби:

Мова програмування C# (вимовляється Сі-шарп) — об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET (належить Microsoft). Синтаксис C# близький до C++ і Java. Мова має строгу статичну типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML.

Використане середовище розробки Microsoft Visual Studio 2019 Community Edition (рисунок 3.1).

Microsoft Visual Studio - це інтегроване середовище розробки (IDE) від Microsoft. Застосовується для розробки комп’ютерних програм, а також веб-сайтів, веб-програм, веб-сервісів та мобільних додатків. Visual Studio використовує платформи розробки програмного забезпечення Microsoft, такі як Windows API, Windows Forms, Windows Presentation Foundation, Windows Store та ASP.NET, ASP.NET Core. Може створювати, компілювати, як і керований код (C#/VB), так і некерований код (C++).

Visual Studio включає редактор коду, що підтримує IntelliSense (компонент авто-завершення коду), а також рефакторинг коду. Інтегрований налагоджувач працює як налагоджувач вихідного коду, так і налагоджувач машинного коду. Інші вбудовані інструменти включають профайлер для детального дослідження процесів, дизайнер для побудови графічних інтерфейсів, веб-дизайнер, дизайнер класів та конструктор схем баз даних. Доступна установка плагінів, які розширюють функціональність майже на кожному рівні - включаючи додавання підтримки для систем керування версіями коду (таких як Git) та додавання нових

наборів інструментів, таких як редактори та візуальні дизайнери для інших мов, або наборів інструментів для інших аспектів розробки програмного забезпечення, життєвого циклу ПЗ (як клієнт Azure DevOps: Team Explorer для командної взаємодії).

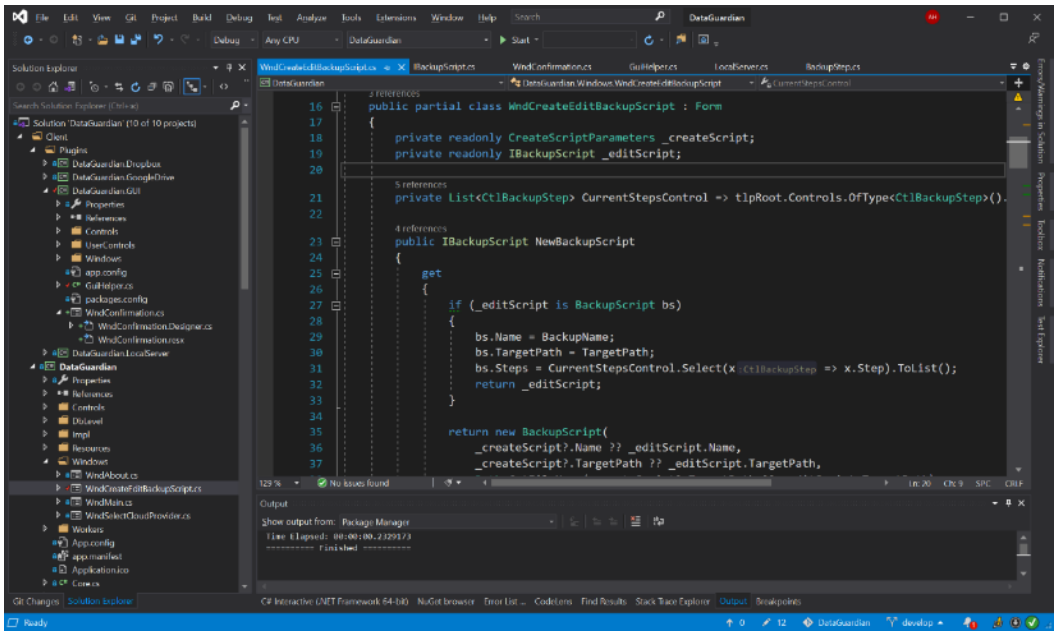

Рисунок 3.1 – Вікно з вихідним кодом в середовищі Microsoft Visual Studio

Visual Studio підтримує 36 різних мов програмування і дозволяє редактору коду та налагоджувачу підтримувати (різною мірою) майже будь-яку мову програмування, за умови існування сервісу, що стосується конкретної мови.

Для розробки програмного продукту використовується фреймворк .Net Framework а також ASP.NET Core, для побудови графічного інтерфейсу використовувався WinForms фреймворк.

3.1.2 Windows Forms

Windows Forms (WinForms) - це безкоштовна графічна (GUI) бібліотека класів із відкритим кодом (з 2018 року), що входить до складу Microsoft .NET Framework, забезпечує платформу для написання клієнтських програм з графічним інтерфейсом для настільних, портативних та планшетних ПК. Хоча вона розглядається як заміна попередньої та більш складної бібліотеки C++ класів Microsoft Foundation на базі C (MFC), вона не пропонує ніякого порівняння, а

виступає лише платформою для рівня користувальницького інтерфейсу для мов C# та VB.NET. На заході Microsoft Connect 2018 року Microsoft оголосила про випуск Windows Forms як проект з відкритим кодом на GitHub. Він випускається за ліцензією MIT. З цим випуском Windows Forms став доступним для проектів, націлених на платформу .NET Core. Однак фреймворк все ще доступний лише на платформі Windows, а Mono залишається неповна реалізація Windows Forms.

Усі візуальні елементи в бібліотеці класів Windows Forms наслідуються від класу Control. Він забезпечує мінімальну функціональність елемента графічного інтерфейсу, наприклад розмір, колір, шрифт, розташування, текст, а також загальні події, такі як клік чи перетягування. Майже всі елементи доступні в дизайнері (рисунок 3.2).

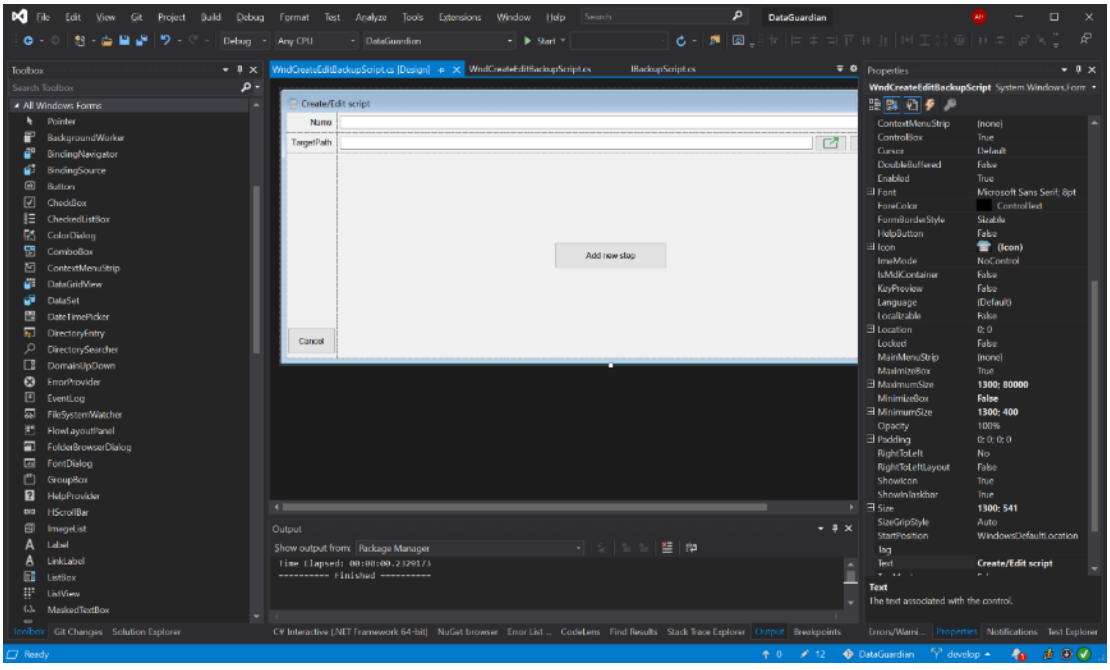

Рисунок 3.2 – дизайнер графічного інтерфейсу Windows Forms.

3.1.3 ASP.NET Core

ASP.NET Core - це безкоштовний веб-фреймворк з відкритим кодом і наступник ASP.NET, розроблений Microsoft. Це модульний фреймворк, який працює з .NET Framework (до версії 3.0), так з мульти-платформномим .NET Core. Цей фреймворк був повністю переписаний, для того щоб поєднати роз’єднані раніше ASP.NET MVC та ASP.NET Web API в єдину модель програмування. Попри

те, що це новий фреймворк, побудований на новому веб-стеку, він має високий ступінь сумісності концепцій з ASP.NET. Додатки ASP.NET Core підтримують паралельне керування версіями, в якому різні програми, що працюють на одній машині, можуть використовувати різні версії ASP.NET Core. Це було неможливо з попередніми версіями ASP.NET.

Він володіє наступними перевагами в порівнянні зі «старим» ASP.NET:

— Відсутність компіляції для розробників (тобто компіляція безперервна, так що розробнику не потрібно викликати команду компіляції)

— Модульний фреймворк, що поширюється як пакети NuGet (і плюс і мінус)

— Спрощена взаємодія з іншими сервісами через відкритий веб-інтерфейс для підтримки .NET (OWIN) - працює в IIS або автономно

— Уніфікований процес побудови веб-інтерфейсу та Web API

— Система конфігурації, що легко адаптується під хмарні рішення

— Легкий та модульний конвеєр запитів HTTP

— Підтримка створення та запуску крос-платформних додатків ASP.NET Core для Windows, Mac та Linux

— Відкритий код та орієнтований на спільноту

— Паралельна підтримка версій додатків при націлюванні на .NET Core

— Вбудована підтримка введення залежностей (DI)

3.1.4 Git

Для зберігання та версіонування вихідного коду були використані система контролю версій Git, а також SourceTree – графічний клієнт для роботи з Git репозиторієм.

Git це розподілена система керування версіями вихідного коду та спільної роботи в команді розробників. Git є незамінним засобом розробки, він дозволяє залишати тільки необхідні зміни в проекті, повертати зміни назад, вести розробку

кількох різних версій програми, синхронізувати (зливати) зміни інших людей в цьому репозиторії та багато чого іншого.

SourceTree - безкоштовний багатоплатформовий візуальний клієнт системи управління версіями Git і Mercurial, який працює на Windows, Mac OS X. SourceTree спрощує взаємодію з репозиторіями Git і Mercurial, щоб ви могли зосередитися на кодуванні. Візуалізація і управління репозиторіями через простий інтерфейс SourceTree.

3.1.5 SQLite

СУБД SQLite - вільна система керування реляційними базами даних.

Це система керування базами даних (SQLite) з відкритим кодом, вона була створена як альтернатива комерційним системам. Зараз SQLite — одна з найпоширеніших вбудованих систем баз даних.

SQLite - це система управління реляційними базами даних (СУБД), що міститься в бібліотеці C. На відміну від багатьох інших систем управління базами даних, SQLite не є механізмом баз даних клієнт-сервер. Зазвичай її вбудовано в кінцеву програму.

На відміну від систем управління базами даних клієнт-сервер, двигун SQLite не має автономних процесів, з якими програма програми взаємодіє. Натомість бібліотека SQLite пов’язана і, таким чином, стає невід’ємною частиною прикладної програми. Посилання може бути статичним або динамічним. Прикладна програма використовує функціональні можливості SQLite за допомогою простих викликів функцій, які зменшують затримку доступу до бази даних: виклики функцій у межах одного процесу ефективніші, ніж міжпроцесовий зв'язок.

SQLite зберігає всю базу даних (визначення, таблиці, індекси та самі дані) як єдиний крос-платформний файл на хост-машині. Він реалізує цю просту конструкцію, блокуючи весь файл бази даних під час запису. Операції читання SQLite можуть бути багатозадачними, хоча записи можуть виконуватися лише послідовно.

SQLite використовує незвичну систему типів для SQL-сумісної СУБД: замість присвоєння типу стовпцю, як у більшості систем баз даних SQL, типи присвоюються окремим значенням; в мовному відношенні він динамічно типізується. Більше того, він слабко типізується тими самими способами, що і Perl: можна вставити рядок у цілочисельний стовпець (хоча SQLite спробує перетворити рядок у ціле число, якщо бажаним типом стовпця є ціле число). Це додає гнучкості стовпцям, особливо якщо вони прив’язані до динамічно набраної мови сценаріїв. Однак методика не є портативною для інших продуктів SQL. Поширеною критикою є те, що в системі типів SQLite відсутній механізм цілісності даних, який забезпечується статично набраними стовпцями в інших продуктах. Веб-сайт SQLite описує режим "суворого спорідненості", але ця функція ще не додана.

Зазвичай таблиці містять прихований стовпець рядового індексу, що забезпечує швидший доступ. Якщо база даних містить стовпець Integer Primary Key, SQLite, як правило, оптимізує її, розглядаючи її як псевдонім для rowid, змушуючи вміст зберігатись як строго набране 64-розрядне ціле число зі знаком і змінюючи свою поведінку дещо схожим на стовпець із інкрементом. Майбутні версії SQLite можуть включати команду для самоаналізу, чи має стовпець поведінку, подібну до такої, як rowid, щоб відрізнити ці стовпці від слабо набраних, не автоматично збільшуваних цілих первинних ключів.

Кілька комп'ютерних процесів або потоків можуть одночасно отримувати доступ до однієї бази даних. Кілька доступів для читання можуть бути задоволені паралельно. Доступ до запису може бути задоволений лише в тому випадку, якщо в даний час не обслуговується жоден інший доступ. В іншому випадку доступ до запису не вдається з кодом помилки (або його можна повторити автоматично, поки не закінчиться настроюваний час очікування). Ця паралельна ситуація доступу зміниться при роботі з тимчасовими таблицями.

В SQLite версії 3.7.4 вперше було додано модуль FTS4 (full-text 4 search, повнотекстовий пошук), який має вдосконалення порівняно зі старим модулем FTS3. FTS4 дозволяє користувачам здійснювати повнотекстовий пошук у документах, подібних до того, як пошукові системи шукають веб-сторінки.

У 2015 році з розширенням json1 та новими інтерфейсами підтипів SQLite представив управління вмістом JSON.

Відповідно до останніх версій бібліотеки, максимально підтримуваний розмір БД становить 281 ТБ.

Підтримує крос-платформовість: з коробки використовується на Unix (Linux і Mac OS X), OS/2, Windows (Win32 і WinCE). Легко переноситься на інші системи.

Для взаємодії з файлом бази даних SQLite рекомендується використовувати клієнтський додаток - середовище DB Browser for SQLite.

DB Browser for SQLite — інструмент з можливістю візуального проектування баз даних та виконання запитів SQL, що об'єднує проектування, моделювання, створення й експлуатацію БД в єдине оточення для системи БД SQLite. Він дозволяє як створити базу даних, так додати чи змінити таблиці та дані в графічному режимі.

3.1.6 Використані бібліотеки

Для ведення журналу логів використовується NuGet пакет Serilog. Для взаємодії з такими хмарними провайдерами, як Google Drive та Dropbox необхідно використовувати їхнє API, можна завантажити як NuGet пакети: Google.Apis.Drive.v3 та Dropbox.Api відповідно. Для початку їх використання рекомендується почати ознайомлення з документаціїєю: https://developers.google.com/drive/api/v3/reference для Drive а також https://www.dropbox.com/developers/documentation/dotnet, для Dropbox. Також рекомендується використовувати пакет Newtonsoft.Json для Json серіалізації та десеріалізації JSON.

Для взаємодії БД також будуть використовуватися ORM EntityFramework та micro-ORM Dapper, які теж доступні через NuGet.

Для підключення простору імен коннектора SQLite в IDE MS Visual Studio необхідно використати вбудований менеджер пакетів NuGet та встановити пакет

System.Data.SQLite.Core, який можна завантажити безкоштовно, як для платформи .Net Framework, так і для .Net Core.

3.2 Взаємодія з СУБД

В результаті виділення сутностей предметної області була спроектована схема бази даних, яка дозволить зберігати достатню та необхідну інформацію про сценарії резервного копіювання, провайдерів даних та їх властивості.

Створення бази даних і необхідних таблиць буде виконувати сам програмний продукт, а саме при запуску необхідного контексту по роботі з базою даних [12]. Наступний клас виконує створення таблиці “Accounts” – лістинг 3.1.

```
    public class AccountsDbContext : DbContext
    {
        public AccountsDbContext(string dbConnection) :
            base(new SQLiteConnection(dbConnection), true)
        {
            Database.ExecuteSqlCommand(
                @"CREATE TABLE IF NOT EXISTS ""Accounts"" (
     ""Id""    INTEGER NOT NULL UNIQUE,
     ""Name""  TEXT NOT NULL UNIQUE,
     ""ConnectionSettings""   TEXT NOT NULL,
     ""CloudId""    TEXT NOT NULL,
     PRIMARY KEY(""Id"" AUTOINCREMENT)
);");
        }

        public DbSet<CloudProviderAccount> Accounts { get; set; }
    }
```

Далі відображено клас, який містить логіку роботи з «Backups» лістинг 3.2.

```
    internal class BackupsDbContext : DbContext
    {
        public BackupsDbContext(string dbConnection) :
            base(new SQLiteConnection(dbConnection), true)
        {
            Database.ExecuteSqlCommand(@"CREATE TABLE IF NOT EXISTS
""BackupScripts"" (
     ""Id""    INTEGER NOT NULL UNIQUE,
     ""CreateDate"" TEXT NOT NULL,
     ""Enabled""    INTEGER,
     ""Name""  TEXT NOT NULL UNIQUE,
```

```
     ""TargetPath"" TEXT NOT NULL,
    ""Steps"" TEXT,
     PRIMARY KEY(""Id"" AUTOINCREMENT)
);");
        }

        public DbSet<BackupScript> BackupScripts { get; set; }
    }
```

3.3 Реалізація класів та методів

Клієнтську частину було вирішено поділити на набір класів, кожен з яких буде відповідати за свою роль.

Після створення головного вікна додатку починає завантажуватися ядро (Core) – спеціальний кореневий об’єкт, в єдиній подобі (паттерн Одинак), який буде утримувати майже всі об’єкти програмного продукту. Конструктор класу Core показано в лістингу 3.3. Він виконує ініціалізації внутрішніх властивостей, таких як Settings, Logger, BackupManager, CloudAccountsManager, GuiManager. Також ядро встановлює посилання на себе в статичному класі CoreStatic, щоб з будь-якого місця програми можна було «отримати» єдиний об’єкт ядра.

```
        public Core()
        {
            _handler = new UnhandledExceptionHandler();
            Settings = new UserSettings();
            Logger = new SingleLogger();
            BackupManager = new BackupManager();
            CloudAccountsManager = new CloudAccountsManager();
            GuiManager = new GuiManager();

            CoreStatic.SetCore(this);

            Start();
        }
```

Після створення ядра, воно ж пробує завантажити всі необхідні збірки .NET за допомогою механізму рефлексії, ініціалізувати всі необхідні об’єкти для роботи та завантажити збережені дані з бази даних SQLite – провайдери, скрипти, лістинг 3.4.

```
        private void Start()
        {
            _cloudProviders.Clear();

            var files = GetAssembliesToLoadPlugins();
            foreach (var file in files)
            {
                var assembly = Assembly.LoadFrom(file);

_cloudProviders.AddRange(GetObjectsFromAssembly<ICloudStorageProvide
r>(assembly));
            }

            var pluginsToInit = GetPluginsToInit();
            foreach (var plugin in pluginsToInit)
            {
                plugin.Init(this);
            }
        }
```

Розглянемо клас CloudAccountsManager, а саме завантаження і ініцілізація модуля, також він містить необхідні поля, властивості і події, наприклад подію зміни стану аккаунтів AccountsChanged, лістинг 3.5:

```
    public class CloudAccountsManager : PluginBase, ICloudAccountsManager
    {
        private readonly List<IAccount> _accounts = new List<IAccount>();
        public event EventHandler<AccountsChangedEventArgs>
AccountsChanged;
        private AccountsDbWorker _dbWorker;
        public IEnumerable<IAccount> Accounts => _accounts;

        public override void Init(ICore core)
        {
            base.Init(core);

            _dbWorker = new
AccountsDbWorker(Core.Settings.ConnectionString);
            _accounts.AddRange(_dbWorker.ReadAccounts());
        }
```

Звісно ж, клас CloudAccountsManager відповідає і за авторизацію нових, аккаунтів користувача, наприклад лістинг 3.6:

```
        public void AddAccount(object form = null)
        {
            try
            {
                using (var dlg = new WndSelectCloudProvider())
                {
                    DialogResult dialogResult;
```

```
                    if (form is Control frm)
                        dialogResult = dlg.ShowDialog(frm);
                    else
                        dialogResult = dlg.ShowDialog();
                    if (dialogResult == DialogResult.OK)
                    {
                        var selectedProvider = dlg.ActiveCloudProvider;
                        var name = GuiHelper.ReadLine("Please enter name
for new cloud account");
                        var connectionInfo = selectedProvider.TryAuth();
                        var newAccount = new CloudProviderAccount(name,
selectedProvider, connectionInfo.ToString());

                        _dbWorker.SaveProvider(newAccount);
                        _accounts.Add(newAccount);
                        FireAccountsChanged();
                    }
                }
            }
            catch (Exception ex)
            {
                Core?.Logger?.Log("AddAccount", ex);
            }
        }
```

Приклад реалізації одного з провайдерів хмарного сховища, DropboxCloudProvider, лістинг 3.7:

```
    public class DropboxCloudProvider :
CloudStorageProviderBase<DropboxSettings>
    {
        public override string Name => "Dropbox";

        public override byte[] Logo => Resources.Img_Dropbox;

        public override string Description =>
Resources.Str_DropboxCloudStorageProvider_Description;

        public override Guid Id => Guid.Parse("{D799FFF5-CACC-4E02-
ACFD-ED2275F3BE56}");
```

Це всього лише оголошення необхідних полів для роботи плагіну, для роботи з DropboxAPI використовуються класи і конектори з одноіменного простору імен: Dropbox.Api.DropboxClient, саме він використовується і для доступу до сховища, і для завантаження даних і для їхнього скачування. Наприклад, отримання списку файлів, лістинг 3.8:

```
        private async Task<RemoteBackupsState>
ListFolderAsync(DropboxClient client, string backupFileName, string
path)
        {
            var list = await client.Files.ListFolderAsync(path);
            var files = list?.Entries?.Where(i => i.IsFile &&
i.Name.Contains(backupFileName)).Select(x => x.AsFile).ToList() ??
new List<FileMetadata>();

            while (list != null && list.HasMore)
            {
                list = await
client.Files.ListFolderContinueAsync(list.Cursor);

                var filesMetadata = list.Entries.Where(m => m.IsFile
&& m.Name.Contains(backupFileName))
                    .Select(x => x.AsFile);

                files.AddRange(filesMetadata);
            }

            var filtered = files.Select(x => (x.AsFile.Id, x.Name,
x.AsFile.ClientModified));
            return new RemoteBackupsState(this, filtered);
        }
```

Для створення архівів для резервного копіювання необхідно було написати рекурсивні методи для обходу внутрішніх каталогів, лістинг 3.9:

```
        public static IEnumerable<FileInfo> GetFilesRecursive(string
path, List<FileInfo> currentData = null)
        {
            if (currentData == null)
                currentData = new List<FileInfo>();

            var directory = new DirectoryInfo(path);

            foreach (var file in directory.GetFiles())
                currentData.Add(file);

            foreach (var d in directory.GetDirectories())
GetFilesRecursive(d.FullName, currentData);
            return currentData;
        }
```

Метод контролера на сервері для завантаження файлів з вхідного запиту з можливість передачі в потоковому режимі, лістинг 3.10:

```
        [DisableFormValueModelBinding]
        [RequestSizeLimit(MaxFileSize)]
```

```
        [RequestFormLimits(MultipartBodyLengthLimit = MaxFileSize)]
        [HttpPost]
        public async Task<IActionResult> ReceiveFile()
        {
            if
(!MultipartRequestHelper.IsMultipartContentType(Request.ContentType))
                throw new Exception("Not a multipart request");

            var boundary = MultipartRequestHelper.GetBoundary(
                MediaTypeHeaderValue.Parse(Request.ContentType), int.MaxValue);
            var reader = new MultipartReader(boundary, Request.Body);

            // for a single file,
            var section = await reader.ReadNextSectionAsync();

            if (section == null)
                throw new Exception("No sections");

            if
(!ContentDispositionHeaderValue.TryParse(section.ContentDisposition, out var
contentDisposition))
                throw new Exception("No content disposition in multipart
defined");

            var fileName = contentDisposition.FileNameStar.ToString();
            if (string.IsNullOrEmpty(fileName))
                fileName = contentDisposition.FileName.ToString();

            if (string.IsNullOrEmpty(fileName))
                throw new Exception("No filename defined.");

            using var fileStream = section.Body;
            await _storageManager.SaveFileToFileSystem(fileName, fileStream);

            return await Task.FromResult(Ok("file uploaded"));
        }
```

Частина методу для створення zip – архіву зі стисненням, даний архів буде вміщати всі користувацькі файли зі повним збереженням структури каталогів, що є надзвичайно важливим параметром для програмного забезпечення цього класу. Даний метод використовує рекурсивний метод обходу каталогів і файлів, для занесення усього дерева папок в архів, лістинг 3.11:

```
        public static void CreateZip(string zipPath, string
destinationPath)
        {
            using (var archive = ZipFile.Open(zipPath,
ZipArchiveMode.Create))
            {
                if (FileSystem.IsDirectory(destinationPath))
                {
                    var fileList =
FileSystem.GetFilesRecursive(destinationPath);

                    foreach (var fileInfo in fileList)
                    {
```

```
                        CreateArchiveEntry(fileInfo, destinationPath,
archive);
                    }
                }
                else
                {
                    var fileInfo = new FileInfo(destinationPath);

                    CreateArchiveEntry(fileInfo,
Directory.GetParent(destinationPath).FullName, archive);
                }
            }
        }
```

3.4 Реалізація графічного інтерфейсу

Існує кілька підходів створення елементів графічного інтерфейсу:

1. Складніший - створення елементів з коду, інколи це необхідно, коли невідомо скільки і яких елементів потрібно відобразити на екрані, це доволі складне завдання, оскільки потрібно задавати вручну багато властивостей, від імені і розміру певного надпису чи кнопки (рисунок 3.3).

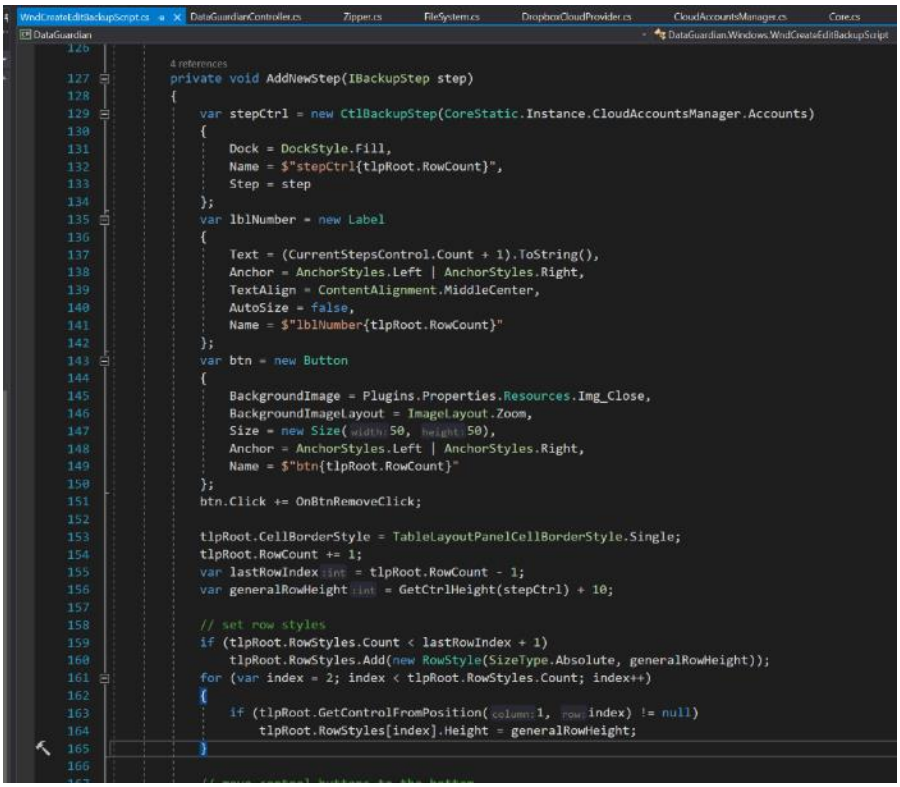

Рисунок 3.3 – ініціалізація елементів інтерфейсу в коді

Так само необхідно додавати і обробники подій, як от клік чи зміна розміру. Також ці елементи необхідно вручну позиціонувати на батьківському (Parent) елементі, самі вони собі місця не знайдуть.

2. Простіший підхід до побудови користувацького інтерфейсу – графічний дизайнер. Повністю автоматизований дизайнер, з підтримкою властивостей, подій, майже всіх елементів (є можливість відобразити свої елементи та елементи з інших бібліотек) зображено на рисунку 3.2. Дизайнер теж генерує код, який зберігається в тому ж класі (завдячуємо ключовому слову partial) але в окремому файлі з приставкою “Designer.cs”. Згенерований код і ресурси містять повне оголошення і ініціалізацію графічних елементів, які зберігаються на UserControl або Form - рисунок 3.4.

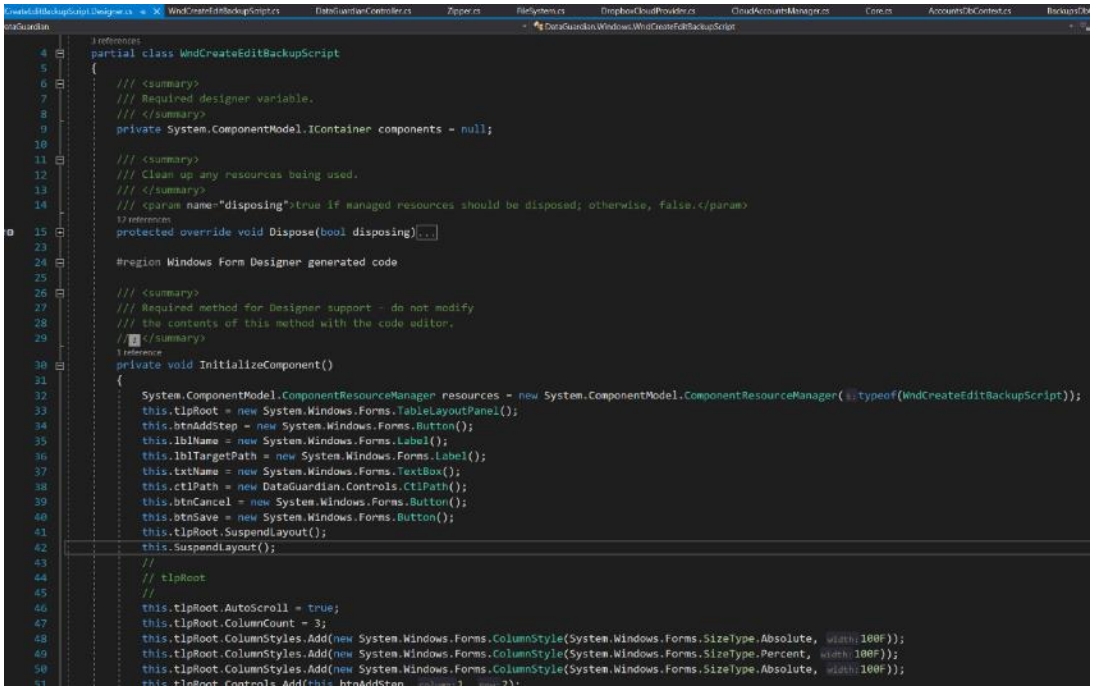

Рисунок 3.4 – Згенерований дизайнером код ініціалізації елементів графічного інтерфейсу

3.5 Технічні характеристики програмного продукту

З технічної точки зору клієнтська частина програмного продукту – це Windows Forms додаток, який може працювати в фоновому режимі, виконувати резервне копіювання заданих користувачем сценаріїв. Серверна частина це

ASPNET.Core сервіс для Windows IIS або Linux, який «слухає» активних клієнтів в локальній мережі та дозволяє зберігання резервних копій на сервері.

Розробка програмного продукту можлива в середовищі Microsoft Windows, оскільки вона підтримує усі необхідні можливості. Програмування систем буде вестися з використанням мови програмування C#, середовища розробки (IDE) Microsoft Visual Studio. Microsoft Середовище .Net дозволяє швидко і ефективно створювати додатки різного ступеня складності. Клієнтський додаток буде спілкувати з сервером за допомогою REST API, за допомогою викликів GET та POST.

Для збереження даних буде використовуватися СУБД DB Browser for SQLite для роботи з файлами бази даних SQLite [12].

Подальший запуск систем буде можливий на різних клієнтських системах, докладні системні вимоги клієнтського додатку описані в таблиці 3.1. Докладні системні вимоги серверного додатку описані в таблиці 3.2.

Таблиця 3.1 – рекомендовані системні вимоги до клієнтського додатку.

Операційна система	Microsoft Windows 7 чи новіше / Linux 4.5 чи новіше / Mac OS 10 чи новіше
CPU (центральний процесор)	Intel Pentium 1GHz чи краще / AMD Athlon 1GHz чи краще
RAM (оперативна пам’ять)	2 GB
SSD/HDD (сховище даних)	128 MB (додаток) + 3 GB (тимчасові резервні копії)
GPU (графічний адаптер)	DirectX 9.0 чи вище
Роздільна здатність	800x600 px
Software (додаткове програмне забезпечення)	Internet Explorer 9 або новіший /Google Chrome 60 або новіший /Opera 50 або новіший / Microsoft Edge 10 або
Додатково	Підключення до мережі інтернет

Таблиця 1.2 – рекомендовані системні вимоги до серверного додатку.

Операційна система	Microsoft Windows 7 чи новіше / Microsoft Windows Server 2012 чи новіше / Linux 4.5 чи новіше
CPU (центральний процесор)	Intel Pentium 1GHz чи краще / AMD Athlon 1GHz чи краще
RAM (оперативна пам’ять)	1 GB

SSD/HDD (сховище даних)	128 MB (додаток) + 3 GB (тимчасові резервні копії)
GPU (графічний адаптер)	-
Роздільна здатність	800x600 px
Software (додаткове програмне забезпечення)	Середовище виконання .Net Core 3.0
Додатково	-

3.6 Інтеграційне тестування

Тестування програмного забезпечення - це дослідження, що проводиться з метою надання зацікавленим сторонам інформації про якість програмного продукту чи послуги, що перевіряється. Тестування програмного забезпечення може також забезпечити об’єктивний, незалежний погляд на програмне забезпечення, щоб дозволити бізнесу оцінити та зрозуміти ризики впровадження програмного забезпечення. Тестові методи включають процес запуску програми чи програми з метою виявлення помилок програмного забезпечення (помилок чи інших дефектів) та перевірки придатності програмного продукту для використання.

Тестування буває кількох видів:

Юніт-тестування – перевірка функціонування і правильної роботи окремих функцій чи модулів програми. Зазвичай використовується рідко, адже не надає комплексної картини, як саме програмне забезпечення буде працювати в результаті. Юніт-тестування широко використовується при розробці ПЗ, адже з написанням і виконанням тестів для окремих класів чи методів справляються самі розробники.

Інтеграційне тестування - це будь-який тип тестування ПЗ, який має на меті перевірити контракти елементів і взаємодію між компонентами програми. Програмні компоненти можуть бути інтегровані ітеративним способом або всі разом. Зазвичай перше вважається кращою практикою, оскільки воно дозволяє швидше і виправляти проблеми з графічним інтерфейсом.

Системне тестування перевіряє повністю інтегровану і налаштовану систему, щоб переконатися, що система відповідає її вимогам. Наприклад, системне тестування може передбачати тестування інтерфейсу вхідних даних, потім створення та редагування запису, а також відправлення або друк результатів, після чого підсумкова обробка або видалення (або архівування) записів, а потім виходу з системи.

Приймальна перевірка виконується перед самим «релізом» ПЗ. Вона включає кілька типів перевірок: тестування прийняття користувача, випробування на приймання в експлуатацію, випробування на виконання контрактів та регулятивних вимог, альфа та бета тестування.

Важливу роль у процесі розробки програмного забезпечення грає саме інтеграційне відлагодження та тестування програмного забезпечення. За допомогою технології Windows Forms та мови програмування C# вдалося реалізувати графічний інтерфейс для програмного продукту – головне вікно зображено на рисунку 3.5. В головному вікні містяться три основні елементи:

Ліва панель – відображення доданих провайдерів хмарного сховища.

Права верхня панель відображає всі завдання по резервному копіюванню.

Нижня панель відображає журнал повідомлень про виконані події.

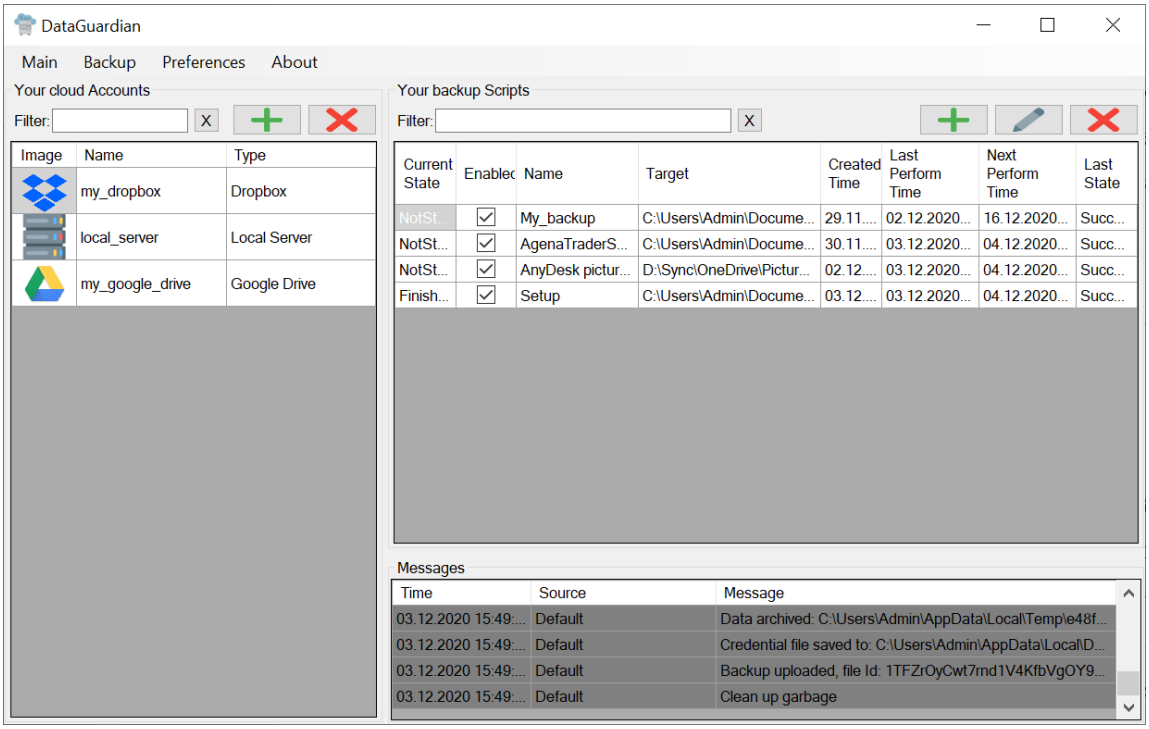

Рисунок 3.5 – головне вікно програмного продукту

Перш за все для користування програмою необхідно додати хмарне сховище даних. Для цього необхідно натиснути кнопку «+» в секції «Аккаунти», після цього послідовно з’являються кілька вікон.

Перше вікно відображає доступні в програмі сховища даних для резервного копіювання. Вони відображаються в таблиці з назвою і описом кожного з них (рисунок 3.6).

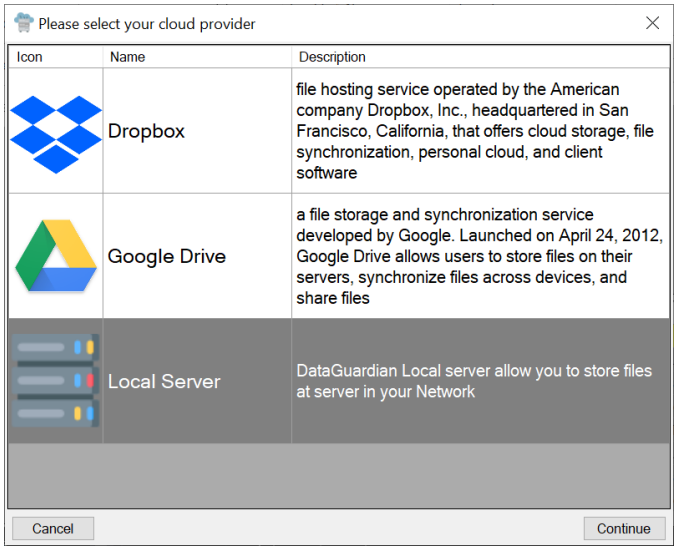

Рисунок 3.6 – вікно вибору хмарного провайдера сховища

Далі, якщо користувач підтвердив свій вибір, потрібно ввести ім’я для нового аккаунта, щоб зберегти дані авторизації (access token, refresh token, унікальні токени авторизації тощо). Ім’я повинно бути унікальним, для коректної роботи програми і бази даних, рисунок 3.7.

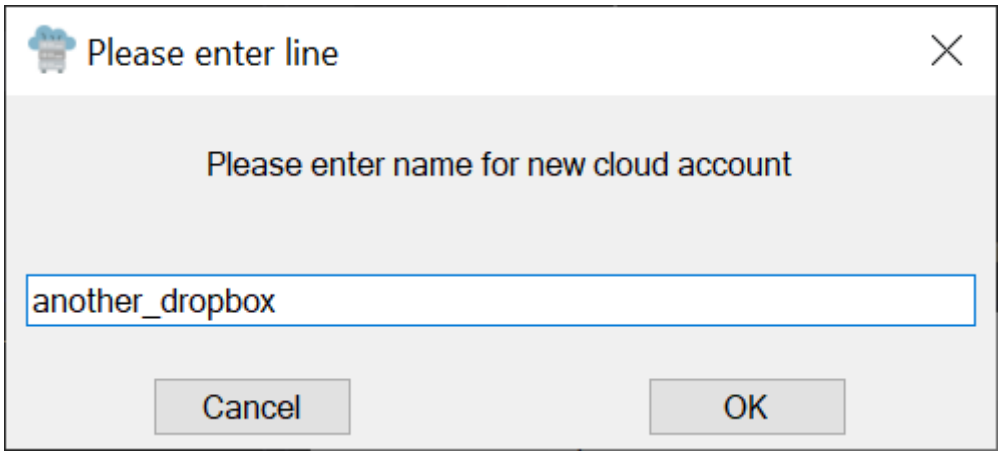

Рисунок 3.7 – вікно вводу імені для нового аккаунта

Далі додаток відкриває в зовнішньому “default” (за замовчуванням) браузері сторінку авторизації певного провайдера даних, він запитує певні права доступу до окремої папки в хмарному сховищі користувача. Авторизація відбувається за схемою OAuth2, тобто за всю авторизацію і автентифікацію відповідає сервер чи сервіс авторизації (в цьому випадку Dropbox). Така схема виграшна для кінцевих додатків, адже вона знімає відповідальність за коректну авторизацію (яка може включати багато-факторну авторизацію за допомогою зовнішнього пристрою чи телефону) та розподіл прав доступу. Права доступу можуть відрізнятися для додатків: одним потрібен доступ на читання, іншим на запис. Також у випадку Google Drive є ще інші права доступу на функцію поширення доступу до файлів чи коментування, наприклад. Отже, наш додаток DataGuardian запитує доступ на запис в сховищі користувача, рисунок 3.8.

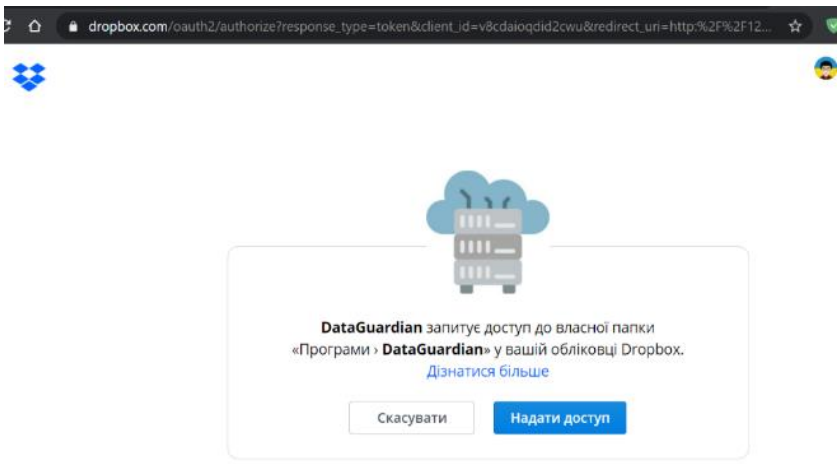

Рисунок 3.8 – запит програми доступу до хмарного сховища Dropbox користувача

Наступний крок підтвердження буде з’являтися поки розробник не опублікує додаток на сервісі підтверджених додатків Dropbox https://www.dropbox.com/apps . Провайдери даних активно і дуже ретельно перевіряють використання їхніх API і сервісів, щоб попередити і запобігти використанню в цілях спаму, взлому, чи будь-яких інших незаконних діяльностях. Всі додатки проходять суворий аудит, який має виявити, як і в яких цілях додаток збирається використовувати доступ до даних користувачів, а також сервісів компанії. Компанія Dropbox дуже цінує свій бренд, його не було помічено ні в одному інциденті «зливу» даних користувачів, а тому користувачі можу бути впевнені, що доступ до даних є тільки в них самих.

Наступне повідомлення (рисунок 3.9) демонструє попередження користувачу, що він повинен продовжити роботу тільки, якщо впевнений в своїх намірах і знає розробника цього додатку

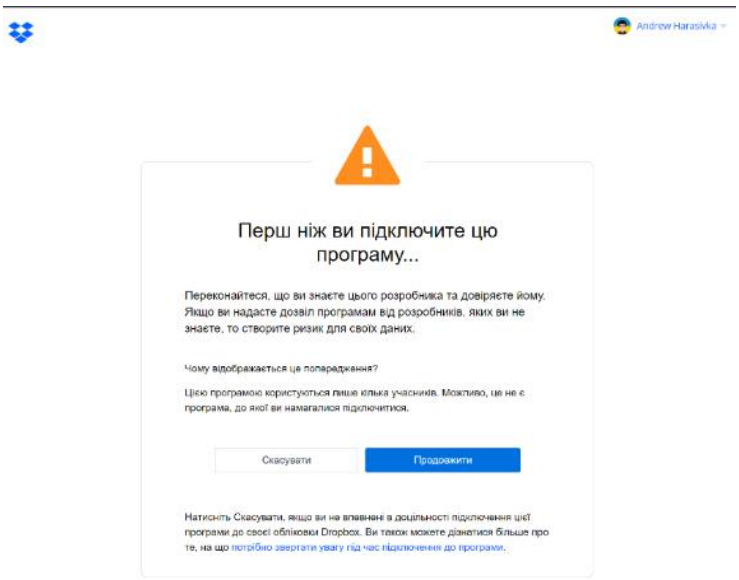

Рисунок 3.9 – демонстрація повідомлення хмарним сервісом про небезпеку для даних користувача

Після успішної авторизації додаток отримує необхідні токени і зберігає їх завдяки серіалізації в JSON та додає в базу даних SQLite. Запис виконує клас AccountsDbContext. Далі новий аккаунт відображається в панелі «Аккаунти» (рисунок 3.10).

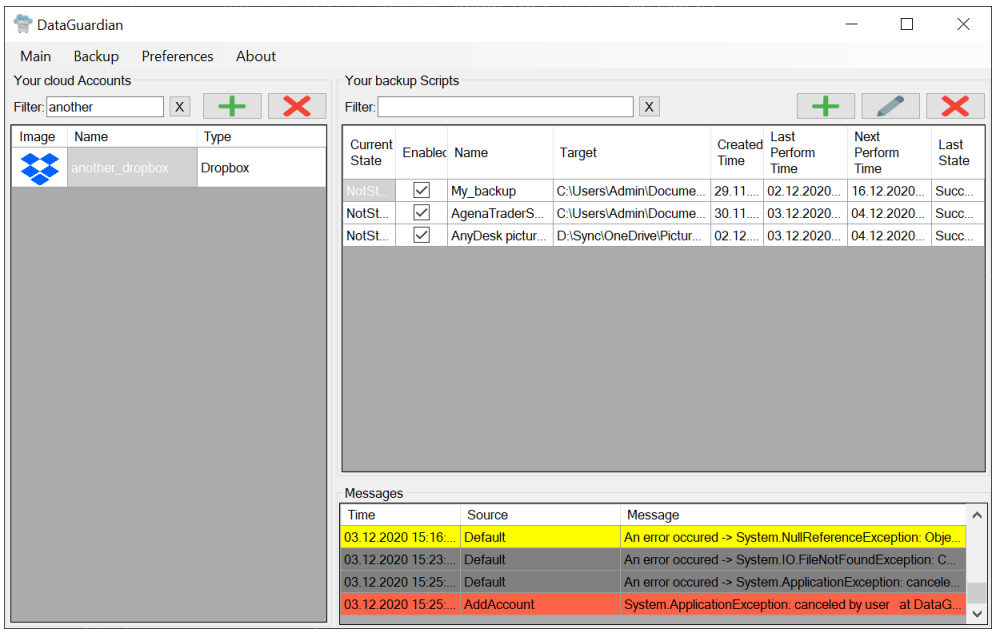

Рисунок 3.10 – відображення щойно доданого аккаунта в вікні

Для додавання нового сценарію резервного копіювання необхідно натиснути кнопку «+» на панелі скриптів, або виконати команду «Backup->Add new Script». З’явиться нове вікно, яке буде розширюватися по мірі додавання нових кроків

резервного копіювання. Користувач може за допомогою простих та зрозумілих елементів інтерфейсу створити новий сценарій для резервного копіювання даних (рисунок 3.11). Користувач може налаштувати директорію чи шлях до файлу, який необхідно копіювати, період копіювання (день, тиждень, місяць, одноразово), дату початку, а також задати назву для вихідних файлів в хмарному сховищі. Для полегшення сприйняття колір фону «Back color» елемента змінюється на випадковий.

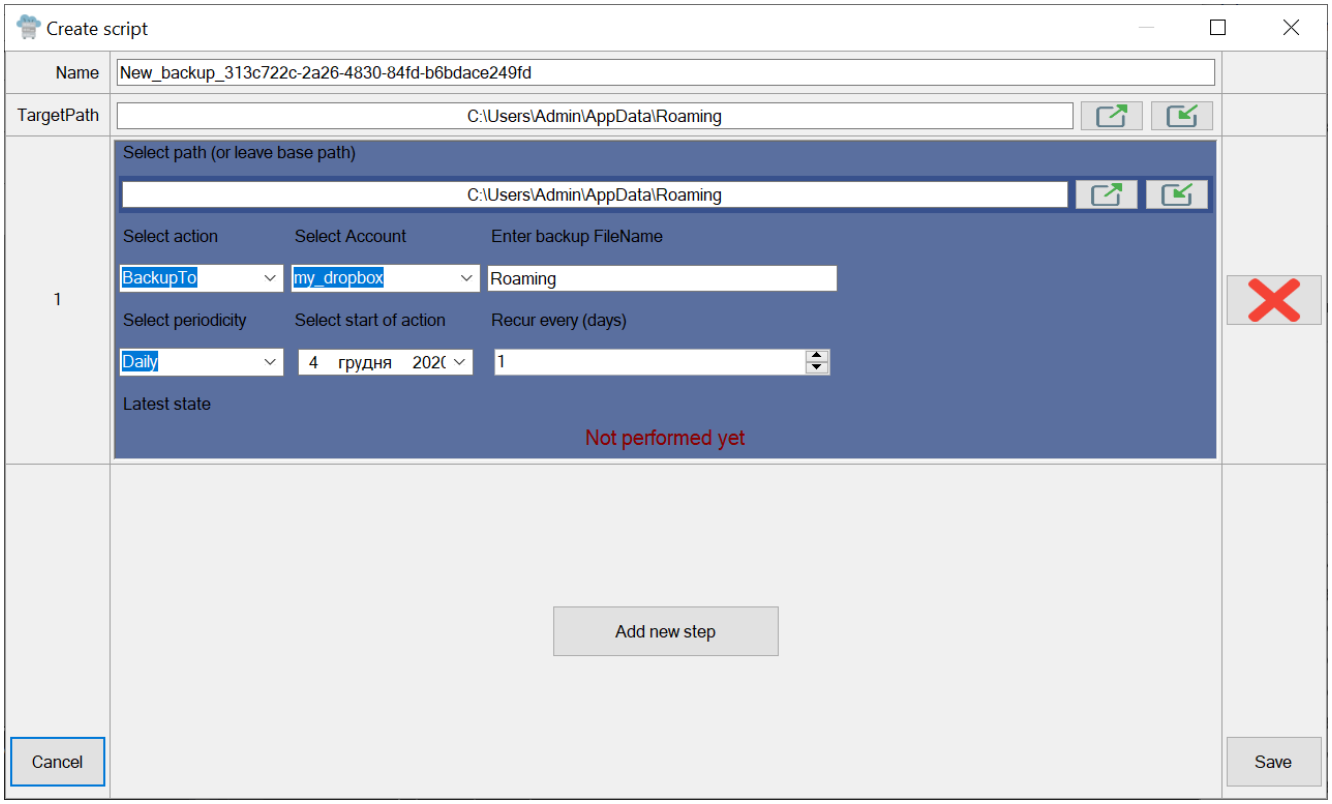

Рисунок 3.11 – інтерфейс програмного продукту

Після завершення створення скрипта для резервного копіювання програма відразу ж перевірить чи потрібно виконувати дії з цим скриптом, наприклад створення чи відновлення резервної копії даних. Перевірити стан резервних копій даних можна через веб-інтерфейс відповідного сервісу хмарного сховища, зробити це можна через контекстне меню.

Після створення архівованої резервної копії даних вона завантажується в хмарне сховище, де тільки користувач може ним керувати – рисунок 3.12 і рисунок 3.13.

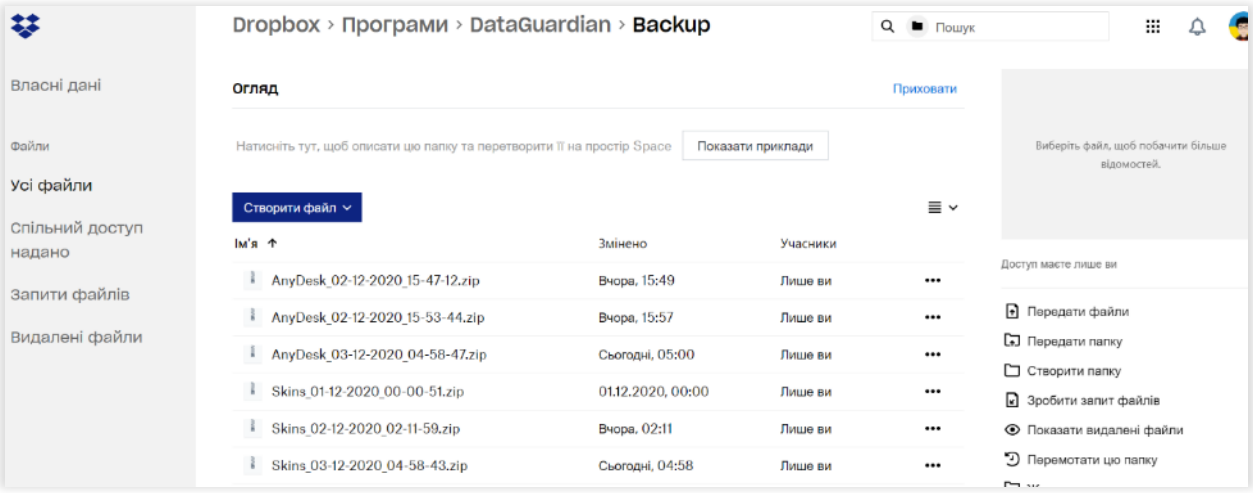

Рисунок 3.12 – резервна копія даних в веб-клієнті хмарного сховища Dropbox

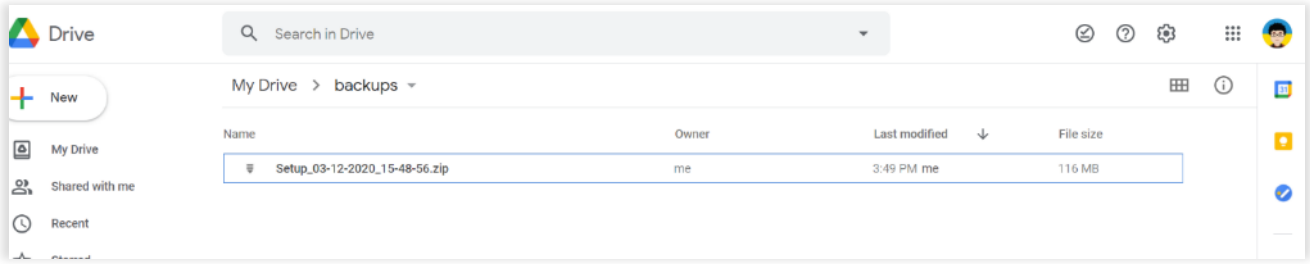

Рисунок 3.13 – резервна копія даних в веб-клієнті хмарного сховища Google Drive

3.7 Розгортання програмної системи та системні вимоги

Розгортання програмної системи планується за допомогою безкоштовного інсталлера Wix http://wixtoolset.org/ .

Набір інструментів XML для Windows Installer (WiX) - це безкоштовний набір інструментів ПЗ, який вміє створювати пакети Windows Installer з допомогою XML. Він складається з середовища скриптів для командного рядка, яке розробники можуть інтегрувати у свої процеси збірки для створення пакетів MSI та MSM. WiX був першим проектом Microsoft, який було випущено за ліцензією з відкритим кодом та Загальною публічною ліцензією.

Для створення інсталера для програми необхідно встановити сам фреймворк Wix, а також розширення для Visual Studio необхідної версії. Після цього необхідно створити новий проект (Setup Project for Wix 3) і написати конфігурацію в файлі

Product.wxs (додаток Б), для опису вихідних файлів інсталятора, місця розташування, додаткових кроків.

Якщо додаток володіє всіма залежностями, то можна додати тільки один файл в конфігурацію.

```
<File Source="$(var.DataGuardian.TargetPath)" />
```

Готовий інсталер зображений на рисунку 3.14

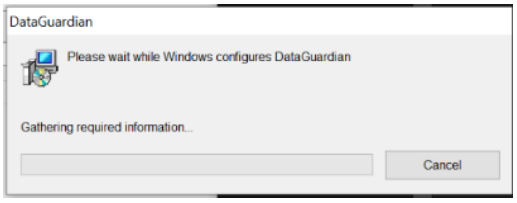

Рисунок 3.14 – Процес встановлення додатку

Розахівований додаток в директорії Program Files (x86)– зображений на рисунку 3.15.

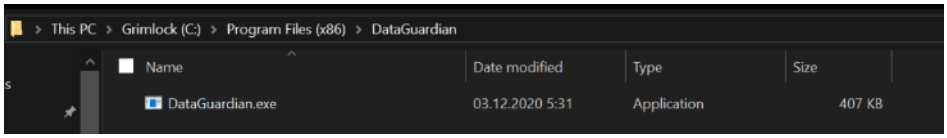

Рисунок 3.15 – додаток після інсталяції

За допомогою інсталлера можна просто виконати упакування програмних бібліотек та виконуваних файлів в інсталятор для Microsoft Windows в вигляді файлу .exe чи .msi, налаштувати зовнішній вигляд інсталера, вибрати дії які будуть виконуватися при установці чи оновленні.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці

Будь-яка сфера діяльності людини в тому чи іншому степені має певний вплив на працівника. В залежності від тривалості і інтенсивності негативні фактори вносять певні зміни в наше здоров'я. Для когось це може бути погіршення постави, для інших це може бути вади зору. Запобігти цьому повинне робоче місце і процес побудований у відповідності до вимог охорони праці.

Робочі місця повинні бути оснащені засобами праці, які необхідні при виконанні повного обсягу виробничих функцій. Наприклад: засоби механізації, автоматизації, інструменти та інші пристосування, які забезпечують безпеку праці при виконанні штатних обов'язків.

Правильний набір і комплектація технічного обладнання, інвентарю і пристосування надає найбільшу ефективність використання виробничих площ приміщення, уникнути зайвої тісноти, створює умови для безпечного виконання штатних обов'язків.

Збільшення ефективності та швидкості роботи, виконання раніше неможливих задач завдяки ЕОМ зробило надзвичайно актуальним питання охорони праці користувачів комп'ютерів. Стандарт під назвою "Державні санітарні норми і правила роботи з візуальними дисплейними терміналами (ВДТ) електронно-обчислювальних машин" є найбільш вичерпним нормативним документом в питанні забезпечення охорони праці користувачів ВДТ.

Виконання вищезгаданих вимог може суттєво зменшити наслідки негативного впливу на працівників шкідливих та небезпечних факторів, що супроводжують роботу з відео терміналами, включаючи можливість візуальних відхилень, нервово-емоційних або серцево-судинних захворювань. Для цього, роботодавцю необхідно гарантувати виконання та дотримання гігієнічних та ергономічних вимоги до організації робочих приміщень для роботи з ПК, робочого

середовища, робочих місць з ВДТ, способу роботи та перерв при роботі з ВДТ тощо, які викладені в правилах.

Згідно з встановленими гігієнічно-санітарними вимогами роботодавець повинен забезпечити на підприємстві в приміщеннях з ВДТ оптимальні параметри виробничого середовища[13].

Для того, щоб створити сприятливі умови для здорової роботи, які б запобігли виникненню професійних захворювань, швидкій втомлюваності очей, нещасних випадків і сприяли зростанню продуктивності праці та якості продукції, освітлення виробничого приміщення повинно відповідати наступним вимогам:

– на робочій поверхні повинна створюватися освітленість, яка б відповідала характеру зорової роботи і була б не нижчою за встановлені норми;

– освітлення повинно забезпечувати необхідну рівномірність та постійність рівня освітленості у виробничих приміщеннях, для того щоб уникнути частої переадаптації органів зору;

– не створювати засліплювальної дії, як від самих джерел освітлення, так і від предметів, що можуть знаходитися в полі зору;

– засоби освітлення повинні бути надійними і простими для використання, економічними та естетичними.

Дизайн робочого місця користувача комп'ютера повинен забезпечувати підтримку оптимальної робочої пози. Робочі станції ВДТ слід розташовувати відносно вікон так аби сонячне світило зліва направо.

Робочі місця з ВДТ потрібно розташовувати на відстані 1,5 м від вікон, а також, як мінімум 1 м від інших стін, відстань між окремими ВДТ повинна становити не менше ніж 1,5 м.

Швидке та точне сприймання інформації з екрану ВДТ в зоні найкращого огляду може забезпечити площина екрана монітора, яка має бути перпендикулярною нормальній лінії зору. Для цього необхідно передбачити можливість переміщення монітора навколо його вертикальної осі в межах ±30° (справа наліво) та нахилу вперед до 85° і назад до 105° з можливістю фіксації в зручному положенні.

Клавіатуру необхідно розміщувати так, щоб було зручно використовувати її обома руками. Розміщення клавіатури на необхідній глибині 100…300 мм від краю проводиться кожним оператором індивідуально. Також необхідно, щоб клавіатура мала можливість змінювати кут нахилу відносно столу межах від 5 до 15°, для підтримки зап'ястя на долонях рук в горизонтальному положенні.

Через тривалі статичні навантаження при роботі з ПК в деяких користувачів може виникнути так званий «синдром зап'ястного каналу». Для профілактики і лікування синдрому зап'ястного каналу необхідно правильно організувати робоче місце, а також графік перерывів і обов'язкове виконання комплексу вправ для рук. Перерви потрібно робити як мінімум раз в годину, тривалість може змінюватися, мінімально 3-5 хв, також рекомендовано піднімати кисті рук вгору і робити невелику зарядку для всього тіла.

Принтер потрібно розміщувати у зручному для положенні місці, не тільки так, щоб не викликати складностей отримання доступу до клавіш керування пристроєм в користувачів але й не створювати перешкод в переміщенні по приміщенню.

Робочий стіл повинен відповідати всім вимогам ергономіки та забезпечувати можливість нормального розміщення на робочій поверхні робочого обладнання, з урахуванням кількості одиниць та певних конструктивних особливостей (розмір клавіатури, монітора, принтера, ПК та ін.), і враховувати характер роботи, що виконується [14].

В період роботи з ПК для якомога більшого збереження здоров'я працівників, відвернення профзахворювання і підтримки працездатності необхідно підтримувати внутрішньозмінні регламентовані перерви для відпочинку.

Під час перерв рекомендується виконувати провітрювання приміщення за можливості, невелику прогулянку, комплекс розвантажувальних вправ для хребта, очей, рук, поліпшення мозкового кровообігу тощо.

Також важливі параметри щодо обладнання та організації робочого місця: площа, відведена на одне робоче місце працівника має становити не менше 6 кв. м., а об'єм – не менше 20 куб. м. Конструкція робочого місця повинна забезпечувати

оптимальну робочу позу (тобто така, яка дозволяє працівникові працювати з мінімальними фізичними навантаженнями та уникати втоми під час роботи та після робочого процесу). Правильна робоче положення має дуже важливе значення для збереження здоров’я працівника, через те що, тривале перебування його в незручній або напруженій позі може спричинити такі хвороби, як сколіоз (викривлення хребта), варикозне розширення вен, плоскостопість тощо. Було досліджено, що виконання роботи в зігнутому положенні збільшує витрати енергії на 20%, а при більш значному нахилі — на 45% в порівнянні з правильним положенням корпуса.

Отже, небезпечні фактори погіршують умови праці людини причиняючи шкідливу дію на її організм. Наприклад, ті хто працюють довгий час за екраном монітора можуть відчувати напруженість очей, затікання чи оніміння кінцівок, головний біль, запаморочення, погіршення пам'яті, підвищена стомлюваність, і т.д. Подібні зміни в роботі ряду органів і систем організму людини можуть спричинити негативні зміни в емоційному стані аж до стресових ситуацій.

4.2 Безпека в надзвичайних ситуаціях

Пожежа в приміщенні чи будівлі є найбільш поширеним видом надзвичайних ситуацій при експлуатації ЕОМ.

Усі споруди і приміщення, де експлуатуються відеотермінали та ЕОМ, повинні мати задокументовану категорію з вибухопожежної і пожежної безпеки відповідно до ОНТП 24-86 "Определение категорий помещений и зданий по взрывопожарной и пожарной опасности", затверджених МВС СРСР 27.02.86, та клас зони згідно з ПВЕ. Відповідні позначення необхідно нанести на вхідні двері приміщення [15].

Додатково, приміщення з підключеними ВДТ, ЕОМ оснащують релейними автоматичними вимикачами напруги на кожен екземпляр ЕОМ та вхідним

автоматичним вимикачем на ціле приміщення. Доступ до щитка необхідно обмежувати за допомогою механічних замків

Приміщення, обладнані комп'ютерами, повинні бути оснащені сучасною автоматичною системою пожежної сигналізації відповідно до вимог Переліку подібних об'єктів, які мають бути обладнані автоматичними системами пожежогасіння та пожежної сигналізації, з розрахунку детектори диму та портативні вогнегасники з вуглекислим газом 2 шт. на кожні 20 кв. м площі поверху з урахуванням гранично допустимих концентрацій вогнегасної рідини відповідно до вимог правил пожежної безпеки в Україні. Теплові пожежні сповіщувачі можна встановити в інших приміщеннях. . Необхідно забезпечити вільний прохід до засобів пожежогасіння [15].

Основними причинами пожеж у кабінеті інженера-програміста можуть бути:

- коротке замикання або займання електричного обладнання, що працює (монітор, принтер, клавіатура тощо);

- додаткові електрообігрівачі;

- система штучного освітлення;- загоряння паперу чи інших легко запалювальних матеріалів.

Розташування робочого місця в кабінеті повинно забезпечувати можливість безперешкодного руху працівника до виходу із приміщення в разі пожежі. Підходи до аварійних виходів повинні бути вільними та не містити перешкод.

ВИСНОВКИ

В результаті виконаного дослідження було завершено аналіз предметної області та стану розв’язання проблеми надійності зберігання резервних копій даних. На основі існуючих рішень на ринку ПЗ було сформовано ряд вимог до майбутньої системи. В ході теоретичного дослідження були охарактеризовані актори, функції додатку та його архітектура, були окреслені вхідні та вихідні дані та визначені вимоги до системи. Для забезпечення найбільшої ефективності було обрано клієнт-серверну архітектуру.

Також було визначено спосіб вирішення задачі щодо ефективного зберігання резервних копій даних в хмарному сховищі, з використанням різних API, як Google Api та Dropbox Api. Авторизацію в хмарних сховищах було реалізовано через механізм OAuth, а тому всі конфіденційні дані залишаються тільки в користувача і хмарного сервісу. Таким чином користувач отримує змогу користуватися багато-факторною авторизацією і бути впевненим в обмеженому доступі до своїх даних.

Було запроектовано програмний додаток, який може легко розширюватись за допомогою модульної системи. Також було розроблено серверний додаток, для копіювання резервних копій на локальний сервер. Дана програму можна використовувати і в фоновому режимі, яка буде запускатися при старті комп’ютера.

Програмний додаток має кілька інноваційних ідей, які дозволять вийти на ринок кінцевих користувачів та завоювати їх довіру. Простий і надійний сервер дозволять розділити та анонімізувати папки та резервні копії користувачів. Підвищений контроль і прозорість роботи над даними повинні задовільнити користувачів, для яких збереження даних є основним пріоритетом. Такі ж вимоги ставлять малі команди розробників, яким потрібне зручне і надійне рішення для резервного копіювання даних їхніх додатків.

В результаті виконання дипломного проекту було виконано основне завдання – реалізовано клієнтський програмний додаток для створення, архівування та завантаження резервних копій даних в хмарне сховище:

– даний додаток реалізовано засобами .Net Framework, ASP.NET Core та мовою програмування C#;

– використано бібліотеку Windows Forms для графічного інтерфейсу;

– спроектовано та реалізовано користувацький інтерфейс;

– проведено тестування системи з реальними даними.

Проект буде корисним в будь-яких сферах роботи з комп’ютером, адже ніхто не застрахований від перепадів напруги, неполадки в операційній системі, виходу з ладу технічного пристрою чи операційної системи. Також актуальним він буде через активність вірусів, шифрувальників та помилки користувачів, які часто спричиняють втрату прав доступу до файлів або їх часткове чи повне пошкодження. Завжди краще відновити дані ніж починати їх створення заново.

Дану тему дослідження буде додатково описано VIII Науково-технічній конференції 2020, в Тернопільському Національному технічному університеті Імені Івана Пулюя, в тезі «Необхідність резервного копіювання даних в повсякденному житті».

СПИСОК ЛІТЕРАТУРИ

1. Які найбільш поширені причини втрати файлів?. Блог про відновлення даних. URL: https://www.r-explorer.com/uk/blog/general-issues/data-loss-reasons.php (дата звернення: 15.11.2020).

2. Резервне копіювання – Вікіпедія. Дата оновлення: 20 липня 2020. URL: https://uk.wikipedia.org/wiki/Резервне_копіювання (дата звернення: 20.11.2020).

3. Хмарні сховища – Вікіпедія. Дата оновлення: 28 жовтня 2020. URL: https://uk.wikipedia.org/wiki/Хмарні_сховища (дата звернення: 02.12.2020).

4. Backblaze Hard Drive Stats Q2 2020. A loot at the Current 2020 Hard Drive Failure Rates. URL: https://www.backblaze.com/blog/backblaze-hard-drive-stats-q2-2020/ (дата звернення: 02.12.2020).

5. Acronis True Image– Вікіпедія. Дата оновлення: 21 лютого 2020. URL: https://uk.wikipedia.org/wiki/Acronis_True_Image (дата звернення: 02.12.2020).

6. NetBackup – Wikipedia. Дата оновлення: 17 жовтня 2020. URL: https://en.wikipedia.org/wiki/NetBackup (дата звернення: 02.12.2020).

7. Bacula – Wikipedia. Дата оновлення: 28 жовтня 2020. URL: https://en.wikipedia.org/wiki/Bacula (дата звернення: 02.12.2020).

8. 10 Backup Statistics Every Business Needs To Know. Веб-сайт: justgilbey.co.uk, 2020р. – URL: https://www.justgilbey.co.uk/blog/10-backup-statistics-every-business-needs-to-know/ (дата звернення: 01.12.2020).

9. Клієнт-серверна архітектура – Вікіпедія. 26 вересня 2020. URL: https://uk.wikipedia.org/wiki/Клієнт-серверна_архітектура (дата звернення 08.12.2020).

10. .NET Framework – Вікіпедія. Дата оновлення: 19 серпня 2020 https://en.wikipedia.org/wiki/.NET_Framework (дата звернення: 25.11.2020).

11. SQL- Вікіпедія. Дата оновлення: 12 травня 2020. https://uk.wikipedia.org/wiki/SQL (дата звернення: 23.11.2020).

12. Проектування та впровадження системи баз даних SQLite / Sibsankar Haldar – O'Reilly, 2015. – 256 с.

13. Охорона праці користувачів пк - Основи охорони праці. URL: https://pidru4niki.com/1321011738196/bzhd/ohorona_pratsi_koristuvachiv (дата звернення: 08.12.2020).

14. Вимоги безпеки щодо організації робочих місць. URL: https://buklib.net/books/31185/ (дата звернення: 08.12.2020).

15. Про затвердження Правил охорони праці під час експлуатації електронно-обчислювальних машин (НПАОП 0.00-1.31-99). URL: https://zakon.rada.gov.ua/laws/show/z0382-99#Text (дата звернення: 08.12.2020).

16. Рихтер Дж. CLR via C#. Программирование на платформе Microsoft.NET Framework 4.5 на языке C# / Дж. Рихтер – СПб: Пітер, 2017. – 896 с.

17. Кренке Д. Теорія і практика побудови бази даних / Д. Кренке – СПб.: Питер, 2005. – 800 с.

18. Mark Masse. REST API Design Rulebook - _220 – O`Reilly, 2012 p. – 114 c.

19. Проектування та впровадження системи баз даних SQLite / Sibsankar Haldar – O'Reilly, 2015. – 256 с.

20. В. І. Кравчук, М. В. Грайворонський. РОЗПОДІЛЕНА СИСТЕМА РЕЗЕРВНОГО КОПІЮВАННЯ Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Фізико-технічний інститут. URL: https://ela.kpi.ua/bitstream/123456789/20803/1/7.Кравчук.с.143-146.pdf (дата звернення 30.12.2020).

21. МЕТОДИЧНІ ВКАЗІВКИ до виконання атестаційної роботи магістра за спеціальністю 121 – ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ. Веб-сайт dl.tntu.edu.ua, 2020 р. – URL: https://dl.tntu.edu.ua/ (дата звернення: 01.12.2020).

22. Арсеновськи Д. Рефакторинг в C# і ASP.NET для професіоналів / Д. Арсеновськи – Вільямс, 2010. – 528 с.

ДОДАТКИ

Додаток А

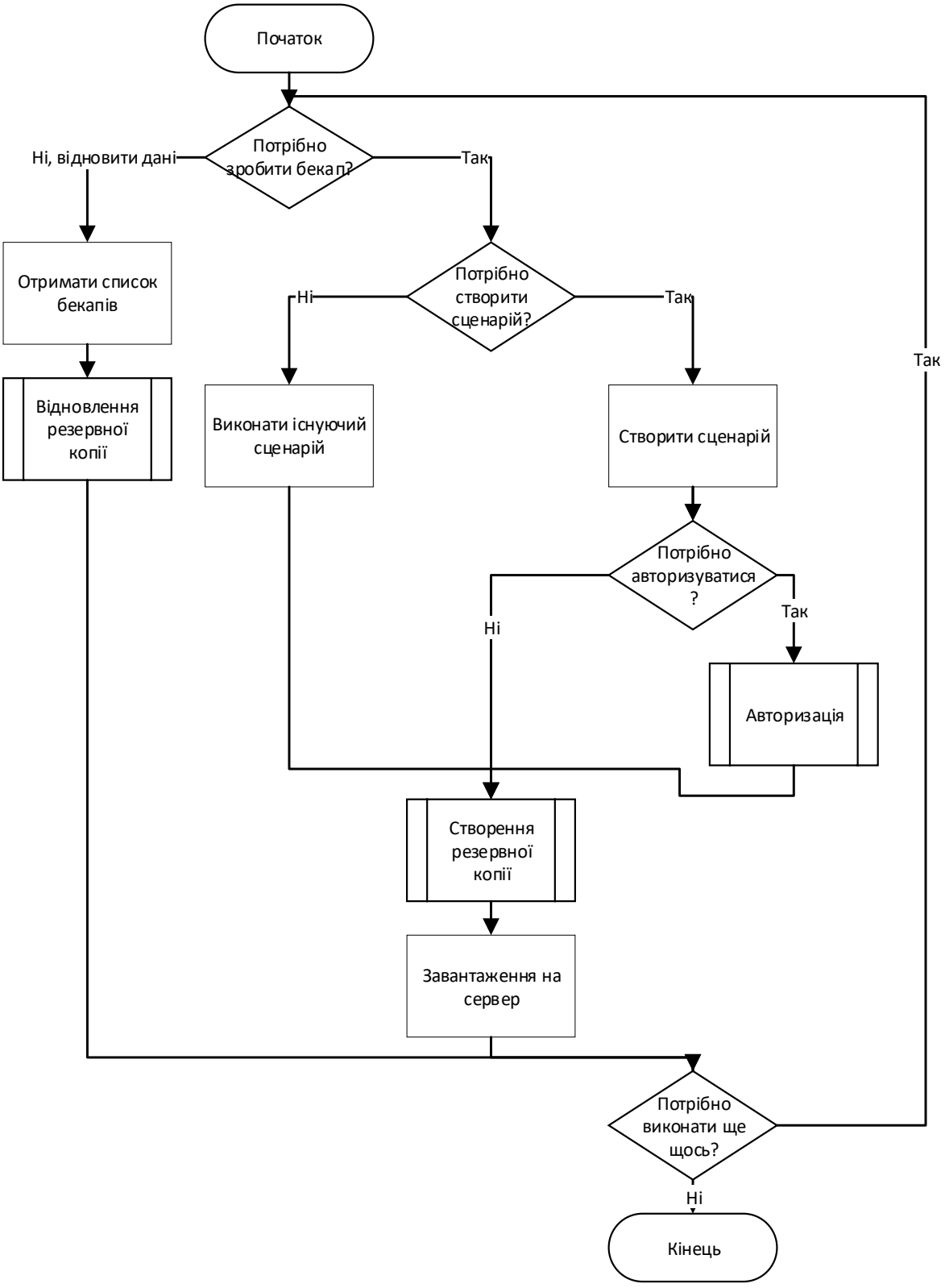

Рисунок А.1 – Діаграма активностей програмного продукту

Додаток Б

Додаток В

УДК 004.63

д.т.н., проф. О. А. Пастух, А. Гарасівка

Тернопільський національний технічний університет імені Івана Пулюя

НЕОБХІДНІСТЬ РЕЗЕРВНОГО КОПІЮВАННЯ ДАНИХ В ПОВСЯКДЕННОМУ ЖИТТІ

Ph.D., prof. O. A. Pastukh, A. Harasivka

Ivan Pulyuy Ternopil National Technical University

THE NEED TO BACK UP DATA IN EVERYDAY LIFE

Інформаційні технології здавна зайняли велику і важливу нішу обробки інформації. Адже комп’ютер може виконувати дуже складні задачі, за доволі короткий проміжок часу. В результаті будь-якої роботи, в будь-якій сфері необхідно зберігати якісь результати. Результат роботи комп’ютера – дані - обчислені, завантажені чи просто збережені зазвичай становлять велику цінність для їх власників. На сьогодні, персональні чи робочі дані людей - одна з найбільших цінностей роботи за комп’ютером. Адже в результаті їх видалення, втрати чи викрадення можуть привести до сумних наслідків: безсонні ночі переписування документів, провалені терміни виконання робіт чи завдань, завалений диплом, втрата бухгалтерської звітності чи важливих, унікальних креслень.

Запобігти сумним наслідкам втрати даних з комп’ютера можуть резервні копії даних – бекапи (англ. backup). Дані – результат роботи користувача чи комп’ютера, які обчислені, завантажені чи збережені на накопичувач комп’ютера.

Створення і підтримка резервних копій або повне дублювання дозволяють також проводити необхідне технічне обслуговування комп’ютерів та серверів, без ризику втратити чи пошкодити користувацькі дані.

Є кілька типів резервного копіювання:

Повне – коли копіюються важливі файли.

Диференціальне – копіюється тільки частина даних яка була змінена з останнього часу.

Набагато ефективнішим можна назвати диференціальне копіювання, адже продуктивніше, оскільки часу на копіювання при ньому йде куди менше, та й місця економиться в сховищі неабияк багато, проте в нього є певні обмеження і воно не завжди можливе. Наприклад робота з великими (>10 ГБ) файлами буде обмежена

дуже повільним механізмом обчислення хеш-сум для порівняння стану файлів. Та й відновити файли буде складніше, якщо щось сталося: для відновлення потрібен не один архів, а всі можливі редакції змінених файлів.

Також важливим питанням є періодичність з якою виконувати резервне копіювання, а також скільки робити копій і на що дублювати дані. Відповідь досить очевидна: чим важливіше інформація, тим частіше і тим на більшу кількість носіїв копіювати. Так само пропорційно повинна зростати і надійність сховищ даних.

Якщо інформація надзвичайно важлива, то робити резервні копії потрібно щодня і на два-три пристрої. В ідеальному випадку ці пристрої будуть зберігатися в різних місцях. І в такому випадку в якості сховищ краще всього використовувати SSD-диски (твердотільні накопичувачі) і USB-накопичувачі з великим об'ємом пам'яті.

Вартість твердотільного накопичувача досі бажає кращого – на разі середня вартість накопичувача на MLC пам’яті за гігабайт становить біля 7 грн. Мінімально коштують накопичувачі з QLC пам’яттю – біля 3 грн за гігабайт. Зате, в порівнянні з жорстким диском даним не будуть страшні ні магнітні поля, ні прямий фізичний вплив (молотком, звичайно, бити не варто, але падіння на підлогу вони цілком здатні витримати) У випадку з флешкою, однак, слід пам'ятати, що якщо інформацію потрібно часто перезаписувати, то такий носій незабаром прийде в непридатність.

Досі найпопулярнішим пристроєм для резервного копіювання є жорсткий диск. Вони використовують перевірені роками технології, мають доволі великий об’єм і є відносно бюджетними – досі їхня вартість тримається біля 1 грн за гігабайт.

Також, за наявності хорошого з’єднання можна використовувати хмарні сховища. Зберігання резервних копій даних в хмарному сховищі буде важливим компонентом інформаційних систем, в яких надзвичайну чи високу цінність мають саме користувацькі чи робочі дані. Адже хмарні провайдери потурбуються за вас і за сервер, і за його живлення, і за цілодобову доступність а також ще це все продублює – на випадок відімкнення цілого дата-центру.

Якщо інформація не настільки важлива, або ж фінансові можливості не дозволяють придбати зазначені вище носії, то можна використовувати диски (CD, DVD або Blu-ray (останні – найбільш об'ємні)). Це у багато разів дешевше, але також і у багато разів менш безпечно: крихкість таких дисків усім відома.

Сучасне здешевлення виробництва комп’ютерних комплектуючих дозволяють придбати накопичувачі по мінімальній ціні. Тому не потрібно лінуватися або чекати біди, а зразу - після збереження нових даних потрібно задумуватися про механізми їх резервного копіювання.