НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут прикладного системного аналізу

Кафедра системного проектування

«На правах рукопису»
УДК 004.414.2

До захисту допущено:
Завідувач кафедри
_____ Вадим МУХІН
«___»_____2021 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою

“Інтелектуальні сервіс-орієнтовані розподілені обчислювання”

зі спеціальності 122 "Комп'ютерні науки"

на тему: «Аналіз безпеки використання хмарного сховища Google Drive при розробці прикладного програмного забезпечення»

Виконав:
студент (-ка) ІІ курсу, групи ДА-з01мп
Семенюк Максим Віталійович _____

Науковий керівник:
Доцент, кандидат наук
Капшук Олег Олексійович _____

Рецензент:
Професор кафедри НН ІПСА, доктор технічних наук
Бідюк Петро Іванович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2021

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра системного проектування

Рівень вищої освіти – другий (магістерський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Інтелектуальні сервіс-орієнтовані розподілені обчислювання»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Вадим МУХІН
«___»_____2021 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Семенюк Максим Віталійович

1. Тема дисертації «Аналіз безпеки використання хмарного сховища Google Drive при розробці прикладного програмного забезпечення»
науковий керівник дисертації Капшук Олег Олексійович, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від «09» листопада 2021 р. No 3704 - е.
2. Строк подання студентом дисертації – 15.12.2021
3. Об'єкт дослідження – безпечність використання хмарного середовища Google Drive
4. Предмет дослідження – безпечність зберігання даних в Google Drive
5. Перелік завдань:
   1. Проаналізувати безпечність використання хмарного сховища Google Drive в прикладному програмному забезпечені.
   2. Проаналізувати стек технологій використання хмарного сховища Google Drive в прикладному програмному забезпечені.
   3. Дослідження шляхів вирішення проблем безпечності використання хмарного середовища Google Drive.
   4. Провести розробку сервісу в прикладній програмі для використання хмарного середовища Google Drive.
   5. Розробити інтерфейс взаємодії користувача з Google Drive на основі розробленого сервісу.
   6. Провести тестування розробленої прикладної програми, виконати оптимізацію та виправлення помилок в прикладному програмному забезпечені.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: презентація на тему «Аналіз безпеки використання хмарного сховища Google Drive при розробці прикладного програмного забезпечення»

7. Орієнтовний перелік публікацій

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Отримання завдання	25.10.2021	
2	Збір та ознайомлення з літературою	16.11.2021	
3	Дослідження предметної області	23.11.2021	
4	Розробка посібника по веб-доступності	30.11.2021	
5	Оцінка результатів та утворення висновків	4.12.2021	
6	Оформлення дипломної роботи	15.12.2021	

Студент: _____ Семенюк М. В.

(підпис) (ініціали, прізвище)

Науковий керівник: _____ Капшук О. О.

(підпис) (ініціали, прізвище)

РЕФЕРАТ

на магістерську дисертацію

виконану на тему: «Аналіз безпеки використання хмарного сховища Google Drive при розробці прикладного програмного забезпечення»
студентом: Семенюком Максимом Віталійовичом

Робота складається зі вступу та чотирьох розділів. Загальний обсяг роботи: 56 аркуші основного тексту, 14 ілюстрацій. При підготовці використовувалася література з 13 різних джерел.

Актуальність. На сьогоднішній день користувачів хмарного середовища майже рівняється кількості людей на планеті, але чи безпечно їх використовувати без витіку секретних даних компанії або треті особи які її використовують? Тому є потреба дослідити як працюють хмарні середовища і чи є шляхи покращення безпеки використання їх.

Мета дослідження. Метою даної магістерської роботи є аналіз безпечності використання хмарного середовища Google Drive. Після аналізу, спробувати покращити безпечність використання хмарного середовища Google Drive.

Об'єкт дослідження — безпечність використання хмарного середовища Google Drive в прикладному програмному забезпечені.

Предмет дослідження — безпечність використання хмарного середовища Google Drive

Методи дослідження. Аналіз, пошук та тестування.

Наукова новизна одержаних результатів роботи полягає у наступному:

- запропоновано шляхи для підвищення рівня безпечності використання хмарного середовища Google Drive;
- реалізовано безкоштовний веб-застосунок який можна буде використовувати в будь-яких проектах які потребують інтеграцію з хмарним середовищем Google Drive;

Особистий внесок здобувача. Магістерське дослідження є самостійно виконаною роботою, в якій відображено особистий авторський підхід та особисто отримані теоретичні та прикладні результати, що відносяться до вирішення задачі проблем безпечності хмарного середовище Google Drive.

Практична цінність. Надання аналіз проблем та шляхи рішення проблем безпечності використання хмарного середовища Google Drive. Було надано універсальну програму для взаємодії з хмарним середовищем яка використовуватись з будь-яким проектом який потребує інтеграцію з хмарним середовищем Google Drive при невеликій кастомізації.

Keywords: cloud environment, Google Drive security analysis, software, encryption.

ABSTRACT
for a master's thesis

performed on the topic: "Analysis of the security to use Google Drive cloud storage in the development of application software"

student: Semeniuk Maksym Vitalievich

The work consists of an introduction and four chapters. Total amount of work: 56 sheets of the main text, 14 illustrations. Literature from 13 different sources was used in the preparation.

Topicality. Today, the number of people on the cloud is almost equal to the number of people on the planet, but is it safe to use them without leaking classified information of the company or third parties who use it? Therefore, there is a need to investigate how cloud environments work and whether there are ways to improve the safety of their use.

The aim of the research. The purpose of this master's thesis is to analyze the security of using the Google Drive cloud environment. After analyzing, try to improve the security of the Google Drive cloud environment.

The object of research is the security of using the Google Drive cloud environment in the application software.

The subject of the research is the security of using the Google Drive cloud environment

Research methods. Analysis, search and testing.

The scientific novelty of the obtained results is as follows:

- suggested ways to increase the security of the Google Drive cloud environment;
- implemented a free web application that can be used in any project that requires integration with the Google Drive cloud environment;

Personal contribution of the applicant. The master's research is a self-performed work, which reflects the personal author's approach and personally obtained theoretical and applied results related to solving the problem of security of the cloud environment Google Drive.

Practical value. Provides problem analysis and solutions to Google Drive cloud security. A universal cloud interaction program has been provided for use with any project that requires integration with the Google Drive cloud environment with little customization.

Keywords: cloud environment, Google Drive, security, analysis, software.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

SOLID	— принципи єдиної відповідальності, відкритості / закритості, підстановки Барбари Лісков, поділу інтерфейсу і інверсії залежностей)
SaaS	— програмне забезпечення як послуга.
IaaS	— інфраструктура як послуга.
PaaS	— платформа як послуга.
CLI	— загальна мовна інфраструктура.
IIS	— інтернет-сервер інформації

ВСТУП

Хмарне середовище — це набір служб і сховищ даних, у яких дані передаються та зберігаються у віддалених системах зберігання, де вони обслуговуються, керуються, створюються резервні копії та доступні користувачам через мережу (зазвичай Інтернет). Зазвичай користувачі платять за хмарне сховище щомісяця. Хоча вартість гігабайт даних була різко знижена, постачальники хмарних сховищ додали операційні витрати, які можуть значно збільшити технологію. Безпека хмарних служб зберігання даних продовжує турбувати користувачів. Постачальники послуг намагаються подолати ці побоювання, підвищивши свої можливості безпеки шляхом включення шифрування даних, багатофакторної аутентифікації та покращення фізичної безпеки у свої послуги. Хмарне сховище можна використовувати для копіювання образу віртуальної машини з хмари в локальне розташування або для імпорту образу віртуальної машини з локального розташування до хмарної бібліотеки. Крім того, хмарне сховище можна використовувати для переміщення зображень віртуальних машин між обліковими записами користувачів або між центрами обробки даних.

Існує три основні хмарні моделі доступу до пам’яті: загальнодоступна, приватна та гібридна.

Модель хмарного сховища стану забезпечує середовище зберігання для багатьох орендарів, яке найкраще підходить для неструктурованих даних на основі підписки. Дані зберігаються в центрах обробки даних постачальників послуг із зберіганням даних, яке поширюється на кілька регіонів або континентів. Клієнти, як правило, платять за використання, подібно до моделі комунальних платежів, і в багатьох випадках також стягується комісія за транзакцію на основі частоти та кількості даних, до яких можна отримати доступ. У цьому секторі ринку переважають Amazon, Google Cloud Storage і Microsoft Azure.

Служба приватного хмарного сховища забезпечується внутрішніми ресурсами сховища, розгорнутими як спеціальне середовище, захищене між брандмауером організації. Внутрішнє хмарне приватне сховище імітує деякі функції комерційно доступних загальнодоступних хмарних сервісів, забезпечуючи простий доступ і розподіл ресурсів зберігання для бізнес-користувачів, а також протоколи зберігання об’єктів. Приватні хмари підходять для користувачів, яким потрібно налаштувати та контролювати свої дані або мають суворі вимоги щодо безпеки даних або нормативні вимоги.

Гібридна хмара — це комбінація приватного хмарного сховища та сторонніх публічних хмарних служб для швидкої інтеграції двох платформ. Ця модель пропонує гнучкість бізнесу та більше можливостей розгортання даних. Наприклад, організація може зберігати активно використовувані та структуровані дані у локальній хмарі, а неструктуровані й архівовані дані у відкритій хмарі. Гібридне середовище також полегшує керування сезонними або непередбаченими стрибками у створенні даних або хмарному доступі до зовнішнього сховища та уникає необхідності додавати внутрішні ресурси зберігання. Останніми роками спостерігається збільшення поширення моделі гібридної хмари. Незважаючи на свої переваги, гібридна хмара створює технічні, бізнесові та управлінські проблеми. Наприклад, приватні робочі навантаження повинні мати доступ і взаємодіяти з постачальниками загальнодоступних хмарних сховищ, тому сумісність і надійне, достатнє підключення до мережі є дуже важливими факторами. Хмарна система сховища корпоративного рівня повинна бути масштабованою до поточних потреб, доступною з будь-якого місця та, крім того, незалежною.[8]

Характеристики хмарного сховища

Хмарне сховище базується на віртуалізованій інфраструктурі з доступними інтерфейсами, майже миттєвою стійкістю, масштабованістю та дозованими ресурсами. Хмарні дані зберігаються в логічних пулах на різних серверах продавців,

розташованих на території або в центрі обробки даних, яким керує сторонній постачальник хмарних послуг. Використовуючи API RESTful, протокол зберігання об’єктів зберігає файл і пов’язані метадані як один об’єкт і призначає йому ідентифікаційний номер. Коли вміст потрібно отримати, користувач подає системі ідентифікатор, і вміст збирається з усіма його метаданими, аутентифікацією та безпекою.

В останні роки постачальники сховища додали функції та функції файлової системи до свого програмного та апаратного забезпечення для зберігання об’єктів переважно через те, що об’єктне сховище було прийнято недостатньо швидко. Наприклад, шлюз хмарного сховища може забезпечувати передню частину емуляції файлової системи для свого об’єктного сховища, таке розташування часто дозволяє програмам отримувати доступ до даних без фактичної підтримки протоколу зберігання об’єктів. Усі програми резервного копіювання використовують протокол зберігання об’єктів, що є однією з причин того, що резервне копіювання в хмарний сервіс було початковою успішною програмою.[8]

1 АНАЛІЗ ПРОБЛЕМ БЕЗПЕЧНОСТІ ХМАРНОГО СЕРЕДОВИЩА GOOGLE DRIVE

Google Drive дозволяє користувачам зберігати та обробляти свої дані на серверах, що належать Google. Це викликає серйозні занепокоєння щодо безпеки та конфіденційності. Тому дуже важливо переконатися, що наші дані можна довіряти відповідному постачальнику послуг. Коли справа доходить до хмарних служб зберігання даних і особливо Google – компанії, яка заробляє гроші на даних користувачів. У цьому розділі я детальніше розгляну проблеми безпеки Google Drive.

Одне з ключових питань, над якими варто задуматися при інтеграції хмарного сховища: чи можливо забезпечити кращу безпеку, ніж постачальник послуг?

1.1 Категорії віддаленої обробки

Щоб дослідити ризики використання хмарного середовища, необхідно знати категорії (етапи) зберігання даних у хмарному середовищі. Як правило, майже всі хмарні середовища містять три основні категорії:

- Програмне забезпечення як послуга (SaaS) – програмне забезпечення постачається та керується віддалено одним або кількома постачальниками. Для початку, Software-as-a-Service або SaaS є популярним способом доступу та оплати програмного забезпечення. Замість того, щоб встановлювати програмне забезпечення на власних серверах, SaaS дозволяє орендувати розміщене програмне забезпечення, зазвичай за щомісячну або річну передплату. Все більше інструментів, пов’язаних з управлінням CRM, маркетингом і фінансами, використовують бізнес-аналітику та технології SaaS, і навіть Adobe Creative Suite прийняв цю модель.
- Інфраструктура як послуга (IaaS) - обчислювальні ресурси, доповнені можливостями зберігання, розміщення постачальників та доступними клієнтам за запитом.
- Платформа як послуга (PaaS) - широка колекція прикладних інфраструктурних сервісів. Ці послуги включають платформу додатків, інтеграцію, управління

бізнес-процесами та сервіси баз даних.

Немає сумнівів, що хмарне середовище забезпечує різноманітні типи рішень для зберігання даних, кожне з яких має свої обмеження та переваги.

1.2 Загрози та проблеми зберігання хмарних даних

Можна стверджувати, що зберігання даних є найважливішим компонентом хмарних обчислень, і саме тому безпека даних над розподіленими обчисленнями завжди є великою проблемою. Є багато питань безпеки щодо зберігання хмарних даних; нижче наведені деякі найбільш поширені та серйозні:

1.2.1 Анонімність

У хмарних сховищах даних анонімність - це метод затемнення опублікованих даних та ключової інформації, що запобігає асоційованій особі власника даних. Однак анонімність даних має різні вразливості, такі як прихована ідентичність супротивних загроз, лазівки в процедурі повторної ідентифікації.

1.2.2 Викрадення трафіку облікових записів або служб

Як правило, це запроваджується за допомогою вкрадених облікових даних і залишається найбільшою загрозою. За допомогою вкрадених облікових даних зловмисники часто можуть отримати доступ до критичних областей розгорнутих служб хмарних обчислень, що дозволяє їм порушити конфіденційність, цілісність і доступність цих служб. Різні шкідливі методи, такі як використання вразливих програм, таких як викрадення паролів / облікових даних і атаки переповнення буфера, а також фішингові атаки, можуть призвести до втрати контролю над обліковими записами користувачів. Це може призвести до того, що хакер отримає контроль над обліковим записом користувача та підслуховує транзакції, маніпулює даними та/або перенаправляє клієнтів на веб-сайт конкурента чи на невідповідні сайти.

1.2.3 Втрата даних та витік даних

Порушення даних є результатом нав'язливої дії, і втрата даних може статися, коли диск накопичується, не створюючи резервної копії. Це втрата конфіденційності, довіри та має прямий вплив на політику угоди про обслуговування, яка є основною проблемою користувачів хмари.

1.2.4 Криптографія

Часто криптографічні механізми, здається, не працюють, коли застосовуються заходи безпеки. Для подолання лазів у зонах безпеки в хмарі використовуються криптографічні методи. Однак такі проблеми, як основна розкладення великих чисел на множники в RSA і дискретні логарифмічні проблеми в ECC і неправильне виконання, можуть призвести до грубої атаки. Погане керування ключами, ефективність обчислень і перевірені дані є іншими проблемами хмарної криптографії.

1.2.5 Проблеми цілісності та конфіденційності

Цілісність є найважливішим елементом інформаційної системи для захисту даних від несанкціонованої зміни, видалення або зміни. Проблема безпеки виникає, коли параметри безпеки визначено неправильно. Хмара багатьох орендарів може призвести до порушення цілісності та конфіденційності, збільшення кількості користувачів, що підвищує ризик безпеки. Деякі відомі атаки на цілісність включають атаки середнього діапазону, крадіжку сесії, атаку з порушенням даних, атаки паролів, перехоплення пакетів і атаки соціальної інженерії, які можуть серйозно вплинути на конфіденційність інформації.

1.2.6 Зловмисне програмне забезпечення

Це стосується нападу електронних злочинів для введення зловмисного програмного забезпечення у хмарне сховище під назвою "Botnets", перетворення їх на "зомбі", спрямоване на атаку більшості комп'ютерних мережевих серверів.[8]

1.3 Аналіз зберігання даних в хмарному середовищі Google Drive

Надійна глобальна інфраструктура Google, що ведуть в галузі створення безпечної хмарної інфраструктури та програм у великих маштабах, величезні інвестиції в безпеку даних, поряд із високою концентрацією спеціальних знань у галузі безпеки, дозволяють компанії пропонувати кращу безпеку, ніж самі споживачі. Для більшості користувачів комп'ютерів Google Диск надійніший, має автоматичне резервне копіювання, відносно безпечний від програм-вимагачів і майже напевно більш захищений від крадіжок. Загалом переваги багато в чому переважають ризики.

Коли йде завантажування файлів на Google Диск, вони зберігаються у захищених центрах обробки даних Google. Google Диск шифрує дані, що зберігаються на диску, а також дані, що передаються на диск і з нього.

Google використовує 128-бітові або 256-бітові ключі AES (залежно від типу пристрою зберігання) для шифрування даних, що зберігаються на Google Диску, що допомагає захистити конфіденційність даних, що зберігаються на Google Диску. Але важливо відзначити, що Google також має ключі шифрування і потенційно може розшифрувати файли за своїм бажанням. Коли Google зберігає дані, контент розбивається на дрібніші частини, кожна з яких зашифрована власним ключем безпеки. Це означає, що потенційному хакеру потрібно зламати кілька ключів для доступу до даних.

Для даних, що передаються, Google використовує стандарт TLS (Transport Layer Security) для захисту даних в русі і запобігання перехоплення або атак типу «зловмисник в середині». TLS захищає канал зв'язку (використовуючи протокол https), але файли можуть стати вразливими після того, як вони стануть спільними для зовнішнього доступу, і кожен додатковий загальний ресурс збільшує ризик. Ризики витоку даних найбільш високі, коли користувачі створюють загальнодоступні посилання з повними правами, які дозволяють будь-кому, хто має посилання на файл, читати, змінювати, копіювати, роздруковувати або завантажувати документ. На даний момент шифрування TLS не може запобігти несанкціонованому доступу до файлів.

Не зважаючи на такий рівень безпеки все ж таки Google зберігає ключі шифрування для всіх файлів на Google Диску. Ключі шифрування - це інструменти, які дозволяють Google (або тому, хто має ключі) розшифровувати файли, змінюючи всю їхню безпеку. Оскільки Google контролюють ці ключі шифрування, це може призвести до вразливості для користувачів.

І, як це часто буває з хмарними сервісами, найбільші ризики пов'язані не з зашифрованою інфраструктурою, а з користувачем, і Google Drive має ряд уразливостей, пов'язаних з користувачем.

Наприклад, Google Диск не має узгоджених організаційних дозволів. Нік Сантора, генеральний директор Curricula, сказав: «Те, як Dropbox використовує папки, дозволяє нам сегментувати дані відділів і надавати доступ до цих папок тільки співробітникам цього відділу. Google робить це надзвичайно важким. Все, що ви робите, - разове. Система дозволів є спеціальною, що призводить до помилок".

1.4 Висновки

Хмарне середовище Google Drive пропонують безліч переваг, за потребою, масштабованість, ресурси, без необхідності вкладати кошти в нову інфраструктуру. Незважаючи на ці особливості, безпека зберігання є критичним моментом, який стоїть перед цією технологією.

2 СТЕК ТЕХНОЛОГІЙ ВИКОРИСТАННЯ ХМАРНОГО СЕРЕДОВИЩА GOOGLE DRIVE В ПРИКЛАДНОМУ ПРОГРАМНОМУ ЗАБЕЗПЕЧЕНІ

Одним із важливіших завдань є вибір технологій для вирішення тих чи інших проблем при написані програмного продукту. Проект був розроблений за допомогою середовищу розробки програмних додатків Microsoft Visual Studio та платформою .NET.

Для розробки сервісу використовувалась об’єктно-орієнтована мова програмування C#.

Для доступу та редагуванню реєстру хмарного середовища, було використано сервіс від компанії Google – Google Drive API, в якого є базові методи для взаємодії з хмарним середовищем Google Disk. [8]

2.1 Платформа .NET Framework

Платформа .Net - це платформа розробки програмного забезпечення, розроблена Microsoft. Рамка призначена для створення додатків, які працюватимуть на платформі Windows. Перша версія фреймворку .Net була випущена в 2002 році.

Версія отримала назву .Net Framework 1.0. З цього часу фреймворк .Net пройшов довгий шлях, а поточна версія - 4.7.1.

Платформа .Net може використовуватися для створення як програмних, так і веб-програм. Веб-сервіси також можуть бути розроблені за допомогою .Net Framework.

Платформа також підтримує різні мови програмування, такі як Visual Basic і C #. Тож розробники можуть вибирати мову для розробки потрібної програми.

Основна архітектура .Net фреймворку наведена нижче (Рисунок 3.1.1)

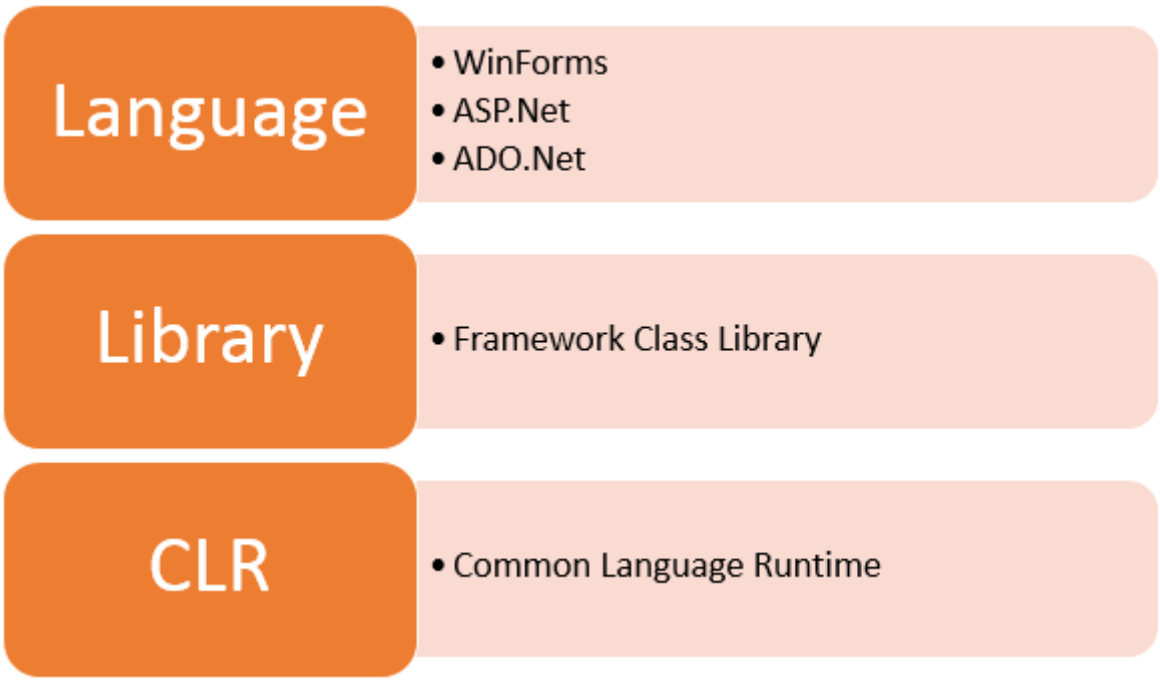


Рисунок 3.1.1 — Основна архітектура .Net фреймворку

Архітектура .Net Framework базується на наступних ключових компонентах;

2.1.1 Загальна мовна інфраструктура

"Загальна мовна інфраструктура" або CLI - це платформа, на якій виконуються програми .Net.

CLI має такі основні характеристики:

1. Обробка винятків - помилки, які виникають під час виконання програми.

Приклади винятків:

- Якщо програма намагається відкрити файл на локальній машині, але файл відсутній.
- Якщо програма намагається отримати деякі записи з бази даних, але підключення до бази даних недійсне.

2. Збір сміття - це процес видалення небажаних ресурсів, коли вони більше не потрібні.

Приклади збору сміття є:

- Ручка файлу, яка більше не потрібна. Якщо програма закінчила всі операції над файлом, обробка файлів більше не потрібна.
- Підключення до бази даних більше не потрібнf. Якщо програма закінчила

всі операції над базою даних, можливо, більше не потрібно буде з'єднання з базою даних.

3. Робота з різними мовами програмування -

Розробник може розробляти додаток на різних мовах програмування .Net.

- Мова - перший рівень - це сама мова програмування, найпоширеніші - VB.Net і C #.
- Компілятор - є компілятор, який буде окремим для кожної мови програмування. Отже, що лежить в основі мови VB.Net, буде окремий компілятор VB.Net. Так само і для C # у вас буде інший компілятор.
- Перекладач загальної мови - це останній рівень в .Net, який би використовувався для запуску програми .Net, розробленої на будь-якій мові програмування. Отже наступний компілятор відправить програму на рівень CLI для запуску програми .Net. [8]

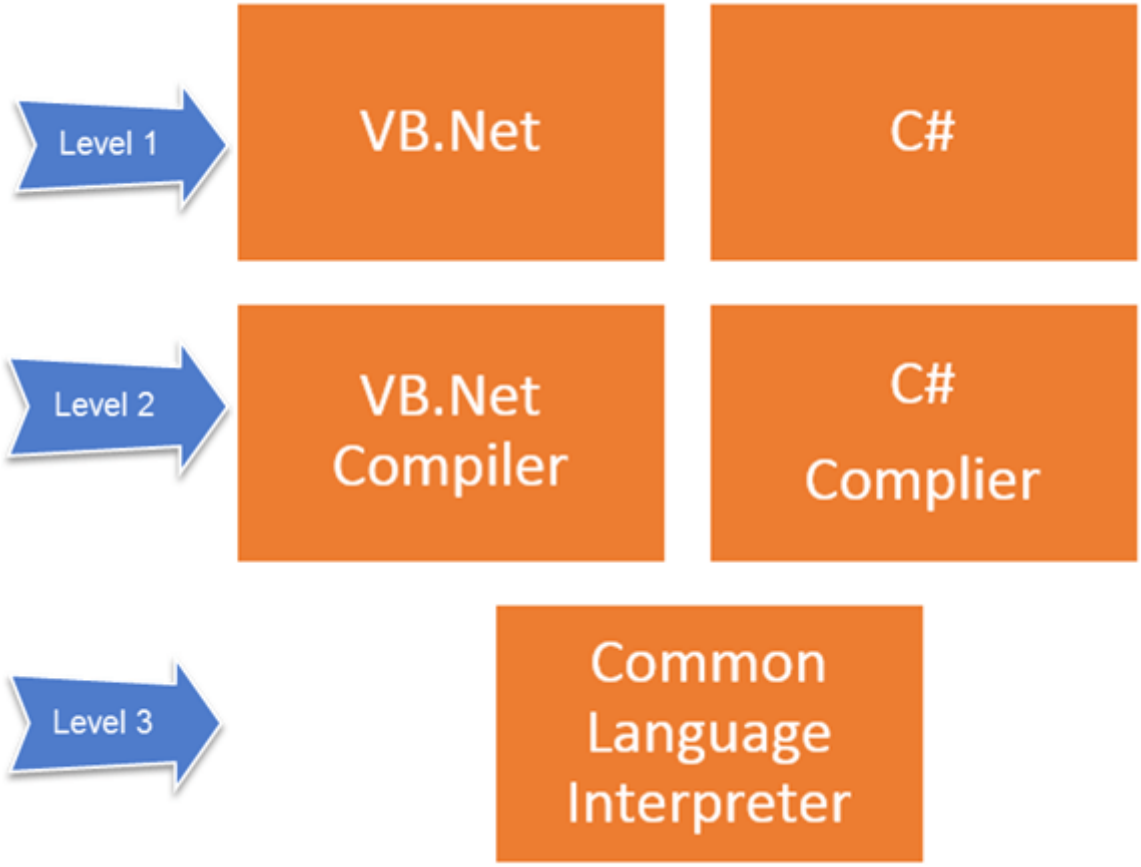


Рисунок 3.1.2 — Загальна мовна інфраструктура

2.1.2 Бібліотека класів

.NET Framework включає набір бібліотек стандартних класів.

Бібліотека класів - це сукупність методів та функцій, які можна використовувати для основної мети. Наприклад, є бібліотека класів з методами для обробки всіх операцій на рівні файлів. Отже, існує метод, який можна використовувати для читання тексту з файлу. Так само існує метод запису тексту у файл.

Більшість методів розділені на простори імен System. * Або Microsoft. *. (Зірочка * просто означає посилання на всі методи, що належать до простору імен системи або Майкрософт). Простір імен - це логічне розділення методів.

2.1.3 Мови

Типи додатків, які можна вбудувати в .Net Framework, класифікуються широко в наступні категорії:

- WinForms - використовується для розробки додатків на основі форм, які працюватимуть на машині кінцевого користувача. Блокнот - це приклад клієнтської програми.
- ASP.Net - використовується для розробки веб-додатків, які запускаються в будь-якому веб-переглядачі, наприклад Internet Explorer, Chrome або Firefox. Веб-додаток оброблятиметься на сервері, на якому встановлено Інтернет-інформаційні послуги. Internet Information Services або IIS - це компонент Microsoft, який використовується для виконання програми ASP.Net. Потім результат виконання надсилається на клієнтські машини, а вихід відображається в браузері.
- ADO.Net - технологія використовується для розробки програм які взаємодіють з базами даних, такими як Oracle або Microsoft SQL Server.

Microsoft завжди гарантує, що рамки .Net відповідають усім підтримуваним операційним системам Windows. [8]

2.1.4 Принципи дизайну .Net Framework

Наступні принципи проектування .Net Framework роблять його дуже актуальним для створення додатків .Net.

Сумісність – .Net Framework забезпечує чудову підтримку спини. Наприклад, якщо у вас є програма, побудована на старішій версії фреймворку .Net, скажімо, 2.0. І якщо ви спробували запустити ту ж програму на машині, яка мала вищу версію. Програма все одно працюватиме. Це тому, що з кожним випуском Microsoft гарантує, що старі версії фреймворку добре працюють з останньою версією.

Переносимість. Програми, засновані на .Net, можна змусити працювати на будь-якій платформі Windows. А нещодавно Microsoft також очікувала, що продукти Microsoft працюватимуть на інших платформах, таких як iOS та Linux.

Безпека – .NET Framework має хороший механізм безпеки. Вбудований механізм захисту допомагає як перевіряти, так і перевіряти програми. Кожна програма може чітко визначити свій власний механізм захисту. Кожен механізм безпеки використовується для надання користувачеві доступу до коду або до запущеної програми.

Управління пам'яттю - загальна мова виконання, яка виконує всю роботу або управління пам'яттю. Фреймворк .Net має всі можливості побачити ті ресурси, які не використовуються запущеною програмою. Тоді вони випустять ці ресурси відповідно. Це робиться за допомогою програми під назвою «Збір сміття», яка працює на .Net.

Збір сміття працює через регулярні проміжки часу і постійно перевіряє, які системні ресурси не використовуються, і утилізує їх відповідно.

Спрощене розгортання – .Net Framework також містить інструменти, які можна використовувати для упаковки програм, створених на .Net Framework. Потім ці пакети можна розповсюдити на клієнтські машини. Потім пакети автоматично встановлять програму. [8]

2.2 Середовище розробки Microsoft Visual Studio 2019

Спільнота Microsoft Visual Studio 2019 — це продукт від Microsoft, який включає інтегроване середовище розробки програмного забезпечення та ряд інших інструментів. Продукт дозволяє розробляти як консольні програми, так і програми з графічним інтерфейсом. Технологія ASP.NET використовується для вирішення проблем доступу та редагування хмарного реєстру.

Visual Studio дозволяє розробляти програми з використанням різних мов програмування: Visual C#, Visual Basic, Visual F#, Visual C++, Python і т. д. Також можна розробляти не тільки для Windows, але і для інших популярних платформ: Android, iOS.

Версія Visual Studio Community є абсолютно безкоштовною для студентів і розробників програмного забезпечення з відкритим кодом. Нові функції також доступні в новій версії Visual Studio 2019. [8]

2.2.1 Інтерфейс користувача

Коли ви відкриваєте Visual Studio, існує ряд вікон інструментів, які дозволяють взаємодіяти з вашим кодом: Вікна інструментів Visual Studio (Рисунок 3.2.1)

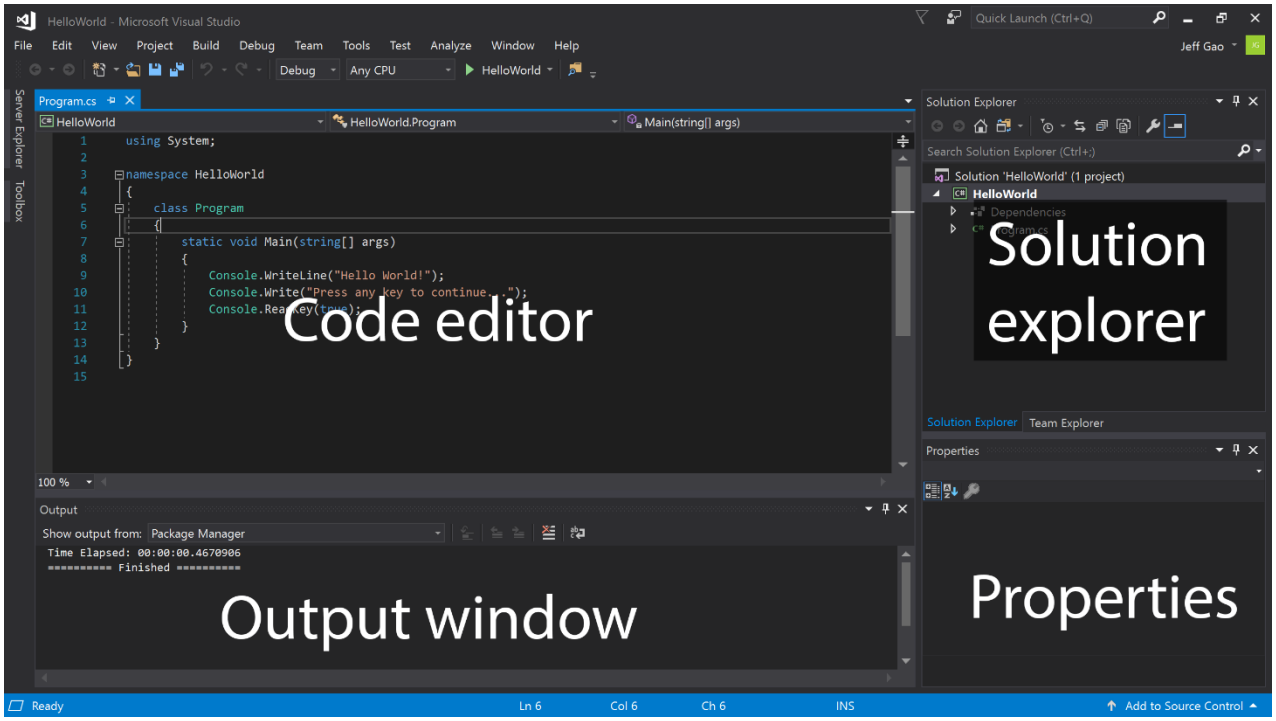


Рисунок 3.2.1 - Вікно інструментів

- Code editor - панель для написання коду.
- Solution explorer - показує файли, з якими ви працюєте.
- Properties - надає додаткову інформацію та контекст щодо вибраних частин вашого проекту.
- Output window - відображаються повідомлення про налагодження та помилки, попередження компілятора, повідомлення про стан та інші результати.

У верхній частині екрана - меню Visual Studio, яка використовується для запуску різних команд. Ось найважливіших з них (Рисунок 3.2.2).

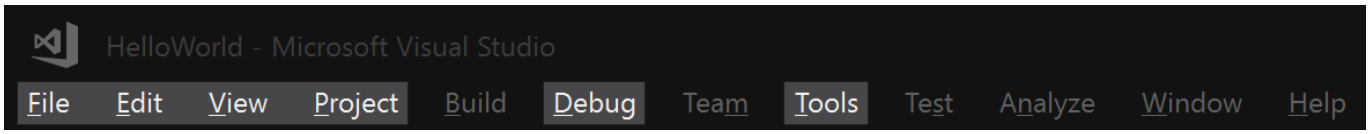


Рисунок 3.2.2 - Меню

- File - містить команди для створення, відкриття та збереження проектів.
- Edit - містить команди для пошуку, зміни та повторного змінення коду.
- View - ви відкриваєте додаткові вікна інструментів у Visual Studio.
- Project - дозволяє додавати файли та залежності в проект.
- Debug - містить команди для запуску коду та використання функцій налагодження.
- Tools - містить команди для зміни налаштувань, додавання функціональних можливостей Visual Studio через розширення та доступу до різних інструментів Visual Studio.

Панель інструментів Visual Studio, показана нижче меню, забезпечує швидкий доступ до найпоширеніших команд.

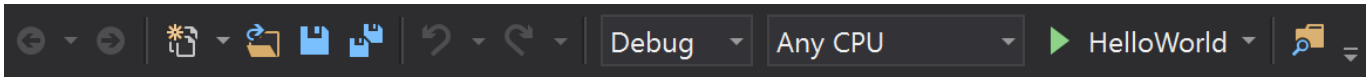


Рисунок 3.2.3 — Панель інструментів

2.2.2 Огляд нових можливостей

Розробка програмних рішень за допомогою Microsoft Visual Studio надає ряд переваг такі як:

- Розробка. Дозволяє зосередитись на одному, а також підвищує продуктивність завдяки оптимізації, можливості миттєвої очистки коду і більш точнішим результатом пошуку.
- Спільна робота. Можна скористатися можливостями спільної роботи в рамках робочого процесу Git-first, функціями редагування та налагодження, а також рецензування коду прямо в Visual Studio.
- Налагодження. Оптимізація використаної пам'яті і створення автоматичних моментальних знімків при виконанні програми.

2.3 Веб-інтерфейс

Веб-інтерфейс – це набір інструментів, які користувач використовує для взаємодії з веб-сайтом або веб-додатком через браузер. Веб-інтерфейси набули широкого поширення через зростання популярності всесвітньої павутини і, як наслідок, широкого використання веб-браузерів.

Однією з головних вимог до веб-інтерфейсів є їх однаковий зовнішній вигляд і однакова функціональність при роботі в різних браузерах.

Класичним і найпопулярнішим методом створення веб-інтерфейсів є використання HTML за допомогою CSS і JavaScript, зазвичай з використанням мов сценаріїв на стороні сервера. Однак різні реалізації HTML, CSS, DOM та інших специфікацій у браузерах викликають проблеми при розробці веб-додатків та їх подальшій підтримці. Крім того, здатність користувача налаштовувати багато параметрів браузера (наприклад, розмір шрифту, колір, вимкнути підтримку сценаріїв) може заважати належній роботі інтерфейсу.

З розвитком DHTML і JavaScript став популярним підхід до розробки інтерфейсної частини веб-додатків, який називається AJAX (асинхронний JavaScript). В основі технології лежить здатність веб-сторінки ініціювати запит на веб-сервер і

отримати необхідні дані, так що інтерфейс не перезавантажує сторінку повністю, а лише завантажує необхідні дані та змінює необхідні частини. сторінку, що робить їх більш інтерактивними та продуктивними.

Веб-інтерфейси зручні тим, що дозволяють співробітникам, які не знаходяться в одному офісі, працювати разом (наприклад, веб-інтерфейси часто використовуються для заповнення різних баз даних або публікації матеріалів в онлайн-ЗМІ).

Хорошим прикладом використання та переваг веб-інтерфейсу є Вікіпедія: майже весь вміст вільної енциклопедії світу створюється та додається на сторінки сайту за допомогою веб-інтерфейсу.

Веб-інтерфейс забезпечує універсальний віддалений доступ до сервісів і пристроїв, альтернатив цій технології практично немає. Але в той же час, оскільки такий інтерфейс доступний кожному, виникають серйозні проблеми безпеки, зокрема аутентифікація та авторизація користувачів, шифрування переданих даних від сторонніх очей тощо. [8]

2.4 Платформа ASP.NET MVC Framework

ASP.NET MVC Framework - платформа для створення веб-додатків, який реалізує шаблон Model-view-controller (про MVC в наступній главі).

Основні компоненти ASP.NET MVC:

Платформа ASP.NET MVC базується на взаємодії трьох компонентів: контролера, моделі та представлення. Контролер приймає запити, обробляє користувача введення, взаємодіє з моделлю і представленням і повертає користувачеві результат обробки запиту.

Модель представляє шар, що описує логіку організації даних в додатку. Подання отримує дані з контролера і генерує елементи призначеного для користувача інтерфейсу для відображення інформації. Для взаємодії з користувачем було додано такі технології двигун уявлень та маршрутизацію:

Двигун уявлень:

Для управління розміткою і вставками коду в поданні використовується движок уявлень. До версії MVC 5 використовувалися два двигуна: Web Forms і Razor.

Починаючи з MVC 5 єдиним двигуном, вбудованим за замовчуванням, є Razor. Движок WebForms використовує файли .aspx, а Razor - файли .cshtml і .vbhtml для зберігання коду уявлень. Основою синтаксису Razor є знак @, після якого здійснюється перехід до коду на мовах C # / VB.NET [26]. Також можливо і використання сторонніх движків. Файли уявлень не є стандартними статичними сторінками з кодом html, а в процесі генерації контролером відповіді з використанням уявлень компілюються в класи, з яких потім генерується сторінка html.

Маршрутизація:

При обробці запитів фреймворк ASP.NET MVC спирається на систему маршрутизації, яка зіставляє всі вхідні запити з певними в системі маршрутами, які вказують який контролер і метод повинен обробити такий запит. Вбудований маршрут за умовчанням передбачає триланкову структуру: контролер / дію / параметр. [8]

2.5 Висновки

Для розробки було встановлено середовище Microsoft Visual Studio 2017 з інтегрованою платформою розробки .NET Framework 4.6 і мовою програмування C# 8.0. Для використання інструментів для побудови розширень до середовища розробки було встановлено комплект розробки програмного забезпечення із усіма відповідними бібліотеками і шаблонами.

3 ВИКОРИСТАННЯ ХМАРНОГО СЕРЕДОВИЩА GOOGLE DRIVE В ПРИКЛАДНОМУ ПРОГРАМНОМУ ЗАБЕЗПЕЧЕНІ

В даному розділі міститься пояснення архітектурного рішення при розробці програмного засобу використання хмарного середовища

3.1 Паттерн MVC

Модель–представлення–контролер — архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення.

Шаблон передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль керування.

Моделі:

Містять або представляють дані, з якими працюють користувачі. Вони можуть бути простими моделями уявлень, які тільки представляють дані, що передаються між уявленнями і контролерами; або ж вони можуть бути моделями предметної області, які містять бізнес-дані, а також операції, перетворення і правила для маніпулювання цими даними.

Представлення:

Застосовуються для візуалізації деякої частини моделі у вигляді призначеного для користувача інтерфейсу.

Контролери:

Обробляють запити, виконують операції з моделлю і вибирають уявлення для візуалізації користувачеві.

Основна мета застосування цієї концепції полягає в розмежуванні бізнес-логіки (моделі) від її візуалізації (уявлення, виду). За рахунок такого поділу підвищується можливість повторного використання коду. Найбільш корисне застосування даної концепції в тих випадках, коли користувач повинен бачити ті ж самі дані одночасно в різних контекстах і / або з різних точок зору. Зокрема, виконуються наступні завдання:

- До однієї моделі можна приєднати кілька видів, при цьому не зачіпаючи реалізацію моделі. Наприклад, деякі дані можуть бути одночасно представлені у вигляді електронної таблиці, гістограми і кругової діаграми;
- Не торкаючись реалізацію видів, можна змінити реакції на дії користувача (натискання мишею на кнопці, введення даних) - для цього досить використовувати інший контролер;
- Ряд розробників спеціалізується тільки в одній з областей: або розробляють графічний інтерфейс, або розробляють бізнес-логіку. Тому можливо добитися того, що програмісти, які займаються розробкою бізнес-логіки (моделі), взагалі не будуть обізнані про те, яке уявлення буде використовуватися. [8]

3.2 Паттерн SOLID

SOLID - це абревіатура п'яти основних принципів проектування в об'єктно-орієнтованому програмуванні - Single responsibility, Open-closed, Liskov substitution, Interface segregation і Dependency inversion. У перекладі на українську: принципи єдиної відповідальності, відкритості / закритості, підстановки Барбари Лісков, поділу інтерфейсу і інверсії залежностей.

Абревіатура SOLID була запропонована Робертом Мартіном, автором кількох книг, широко відомих в співтоваристві розробників. Ці принципи дозволяють будувати на базі ООП масштабовані і супроводжувані програмні продукти зі зрозумілою бізнес-логікою:

- Single responsibility - принцип єдиної відповідальності
- Open-closed - принцип відкритості / закритості
- Liskov substitution - принцип підстановки Барбари Лісков
- Interface segregation - принцип поділу інтерфейсу
- Dependency inversion - принцип інверсії залежностей

Принцип єдиною обов'язки / відповідальності (single responsibility principle / SRP) позначає, що кожен об'єкт повинен мати один обов'язок і цей обов'язок повинен бути повністю інкапсульований в клас. Всі його сервіси повинні бути спрямовані виключно на забезпечення цього обов'язку.

Принцип відкритості / закритості (open-closed principle / OCP) декларує, що програмні сутності (класи, модулі, функції і т. п.) Повинні бути відкриті для розширення, але закриті для зміни. Це означає, що ці сутності можуть міняти свою поведінку без зміни їх вихідного коду.

Принцип підстановки Лісков (Liskov substitution principle / LSP) в формулюванні Роберта Мартіна: «функції, які використовують базовий тип, повинні мати можливість використовувати підтипи базового типу не знаючи про це».

Принцип поділу інтерфейсу (interface segregation principle / ISP) в формулюванні Роберта Мартіна: «клієнти не повинні залежати від методів, які вони не використовують». Принцип поділу інтерфейсів говорить про те, що занадто «товсті» інтерфейси необхідно розділяти на більш дрібні і специфічні, щоб клієнти маленьких інтерфейсів знали тільки про методи, які необхідні їм у роботі. У підсумку, при зміні методу інтерфейсу не повинні змінюватися клієнти, які цей метод не використовують.

Принцип інверсії залежностей (dependency inversion principle / DIP) - модулі верхніх рівнів не повинні залежати від модулів нижніх рівнів, а обидва типи модулів повинні залежати від абстракцій; самі абстракції не повинні залежати від деталей, а ось деталі повинні залежати від абстракцій. [8]

3.3 Файлове середовище Google Drive

Google Drive — це сховище даних, що належить Google Inc., яке дозволяє користувачам зберігати свої дані на серверах у хмарі та ділитися ними з іншими користувачами в Інтернеті. Google Drive включає в себе документи Google, електронні таблиці та презентації, офісний пакет, який дозволяє редагувати документи, електронні таблиці, презентації, зображення, форми тощо. Станом на жовтень 2014 року він мав 240 мільйонів активних користувачів щомісяця.

Для синхронізації файлів між комп'ютером користувача та хмарним сховищем потрібне програмне забезпечення Google Drive («клієнт») на комп'ютері користувача. Клієнт взаємодіє з Google Диском, тому оновлення з однієї сторони поширюються на іншу, тому вони зазвичай містять однакові дані.

Клієнтське програмне забезпечення Google Drive доступне для таких пристроїв: ПК під керуванням Windows XP, Windows Vista, Windows 7 і Windows 8 з NTFS або Mac OS X 10.6 (Snow Leopard) або новішої версії; Смартфони та планшети Android з ОС Android 2.1 (Eclair) і вище; iPhone та iPad з мікропрограмою 5.0 або новішої.

Диск Google доступний у автономному режимі у браузері Google Chrome через програму Chrome із Веб-магазину Chrome. Документи, електронні таблиці, презентації та зображення можна переглядати та редагувати за допомогою окремих програм Chrome.

Особливості

- Спільне користування

Google Drive включає в себе систему обміну файлами, де творець файлу або теки за замовчуванням є його власником. Власник має можливість регулювати видимість файлу або теки для інших користувачів. Право власності підлягає передачі. Файли і теки можуть спільно використовуватись приватно конкретними користувачами, що мають обліковий запис Google, використовуючи свої @gmail.com адреси електронної пошти. Для спільного використання файлів з користувачами, які

не мають облікового запису Google потрібно зробити файли «доступними за посиланням». Це створює секретний URL для файлу, який може бути відкритий через електронну пошту, блоги і т. д. Файли та теки можна зробити «загальнодоступними в Інтернеті», що означає, що вони можуть бути проіндексовані пошуковими системами і, таким чином, можуть бути знайдені і доступні будь-кому. Власник може також встановити рівень доступу. Три запропоновані рівня доступу, це «може редагувати», «може коментувати» і «може переглядати». Користувачі, що мають доступ для редагування можуть запрошувати інших редагувати. [8]

- Сторонні додатки

Існує ряд зовнішніх веб-додатків («apps»), які працюють з Google Drive. Ці програми доступні у веб-магазині Chrome і сумісні з усіма підтримуваними браузерами. Щоб використовувати додаток, користувач повинен увійти в Chrome Web Store і встановити додаток. Деякі з цих додатків розроблені Google, наприклад, Google Docs, Sheets and Slides. Додатки Drive, які працюють з інтернет-файлами, використовуються для перегляду, редагування і створення файлів в різних форматах, редагування зображення та відео, факсу і підписування документів, управління проектами, створення блок-схем, додатків і т. д. Drive також може бути програмою за замовчуванням для обробки файлів у форматах, які ним підтримуються. Деякі з цих додатків також працюють в автономному режимі, але лише на Google Chrome і Chrome OS. Усі сторонні додатки можна встановити безкоштовно. Тим не менш, деякі з них стягують додаткову плату за продовження використання або доступ до додаткових функцій. Більшість додатків Drive мають дозвіл на доступ до файлів користувачів за межами Google Drive. Збереження даних з сторонніх додатків на Google Drive вимагає авторизації в перший раз. Google Drive SDK працює спільно з Google Drive UI і Chrome Web Store, щоб створити екосистему додатків, які можуть бути встановлені в Google Drive.

- Пошук

Результати пошуку можуть бути звужені за типом файлу, власником, видимістю і додатком для перегляду. Google Drive підтримує логічні оператори. За

допомогою Google Goggles і технології оптичного розпізнавання символів (OCR), користувачі можуть шукати зображення за описом їх вмісту. Наприклад, пошук «гори» покаже всі фотографії гір, а також будь-які текстові документи про гори. Пошук розпізнає текст в перших 100 сторінках текстових документів і текстових файлів PDF, і перші 10 сторінок PDF-файлів на основі малюнків. Текст у PDF-файлах і зображеннях видобувається за допомогою OCR.

- Доступність

25 червня 2014, Google анонсувала ряд оновлень в Google Drive, які мали на меті зробити сервіс більш доступним для користувачів з вадами зору. Це включає поліпшений доступ до клавіатури, підтримка масштабування, режим високої контрастності і кращої сумісності з зчитувачами екрана.

3.3 OAuth 2.0 протокол

Google API використовують протокол OAuth 2.0 для аутентифікації та авторизації. Google підтримує поширені сценарії OAuth 2.0, такі як веб-сервер, клієнтська установка та додатки пристроїв обмеженого введення.

Для початку потрібно отримати облікові дані клієнта OAuth 2.0 з консолі Google API. Потім клієнтська програма запитує маркер доступу з сервера авторизації Google, витягує маркер із відповіді та надсилає маркер API Google, до якого ви хочете отримати доступ.

Усі програми дотримуються основної схеми під час доступу до Google API Console за допомогою OAuth 2.0. На високому рівні виконується п'ять кроків:

1. Отримання облікових даних OAuth 2.0 з консолі Google API Console.

В Google API Console можна отримати облікові дані OAuth 2.0, такі як ідентифікатор клієнта та секрет клієнта, які відомі як Google, так і в програмі. Набір значень змінюється залежно від типу програми. Наприклад, програма JavaScript не вимагає секрету, але програма веб-сервера вимагає.

2. Отримання маркер доступу з сервера авторизації Google.

Перш ніж програма може отримати доступ до приватних даних за допомогою API Google, вона повинна отримати маркер доступу, який надає доступ до цього API. Один маркер доступу може надавати різний ступінь доступу до декількох API. Змінна параметр, який називається область, керує набором ресурсів та операцій, які дозволяє маркер доступу. Під час запиту маркера доступу ваша програма надсилає одне або більше значень у параметрі області.

Існує кілька способів зробити цей запит, і вони залежать від типу програми. Наприклад, програма JavaScript може запитати маркер доступу за допомогою перенаправлення браузера на Google, тоді як програма, встановлена на пристрої, у якого немає браузера, використовує запити веб-служб.

Деякі запити потребують кроку аутентифікації, коли користувач входить у свій обліковий запис Google. Після входу в систему користувача запитують, чи готові вони надати один чи більше дозволів, про які запитує ваша програма. Цей процес називається згодою користувача.

Якщо користувач надає щонайменше один дозвіл, сервер авторизації Google надсилає програмі маркер доступу (або код авторизації, який програма може використовувати для отримання маркера доступу) та перелік областей доступу, наданих цим маркером. Якщо користувач не надає дозволу, сервер повертає помилку.

3. Управління доступом.

Потрібно порівняти області, що містяться у відповіді маркера доступу, з масштабами, необхідними для доступу до функцій та функцій програми, залежно від доступу до відповідного API Google.

Обсяг, включений у запит, може не відповідати обсягу, включеному у відповідь, навіть якщо користувач надав усі запитувані області. Потрібно звернутись до документації для кожного API Google щодо обсягів, необхідних для доступу. API може зіставити декілька значень рядка діапазону в одну область доступу, повертаючи ту саму рядок області для всіх значень, які дозволені в запиті.

4. Надсилання маркеру доступу в API.

Після того, як програма отримує маркер доступу, вона надсилає маркер API Google у заголовку запиту авторизації HTTP. Крім того, хороша практика REST уникати створення зайвих імен параметрів URI.

5. Оновлення маркеру доступу, якщо необхідно.

Токени доступу мають обмежений термін служби. Якщо програмі потрібен доступ до API Google за період дії одного маркера доступу, він може отримати маркер оновлення. Токен оновлення дозволяє вашій програмі отримувати нові маркери доступу. [8]

Далі наведено ілюстрація взаємодії користувача з Google Servers (Рисунок 4.4.1)

Рисунок 4.4.1 - Взаємодії користувача з Google Servers

4 БЕЗПЕКА ВИКОРИСТАННЯ ХМАРНОГО СЕРЕДОВИЩА GOOGLE DRIVE В ПРИКЛАДНОМУ ПРОГРАМНОМУ ЗАБЕЗПЕЧЕНІ

4.1 Шляхи вирішення проблем безпеки

Один із способів уникнути проблем із загальнодоступними посиланнями на файли Google Диска — дозволити ділитися лише певним людям. Однак, якщо ми хочемо обмежити те, що користувач може робити з нашими даними після їх отримання, ми повинні мати можливість контролювати дозволи на вміст.

Крім того, залежно від рівня підписки, Google Диск дозволяє деякий основний контроль над цифровими правами, пов'язаними з файлами та папками, наприклад змінювати, коментувати, лише переглядати, завантажувати тощо.

У компанії адміністратор також може обмежити доступ до певних ролей або відділів, вимкнувши спільний доступ або друк. Google Drive дозволяє працювати з Google Groups. Можна створити групу та додати людей, з якими ми хочемо поділитися файлом чи папкою. Видалення користувачів із групи позбавляє їх дозволів на доступ до файлів, якими з ними надано спільний доступ.

Ці функції дозволяють нам відповідати різним вимогам корпоративної безпеки, але не забезпечують ефективної захисту даних у певних сценаріях співпраці з третіми сторонами. Захист даних не пов'язаний із документами, тому як тільки дані залишають Google Диск, користувач і компанія втрачають контроль над наданими даними.

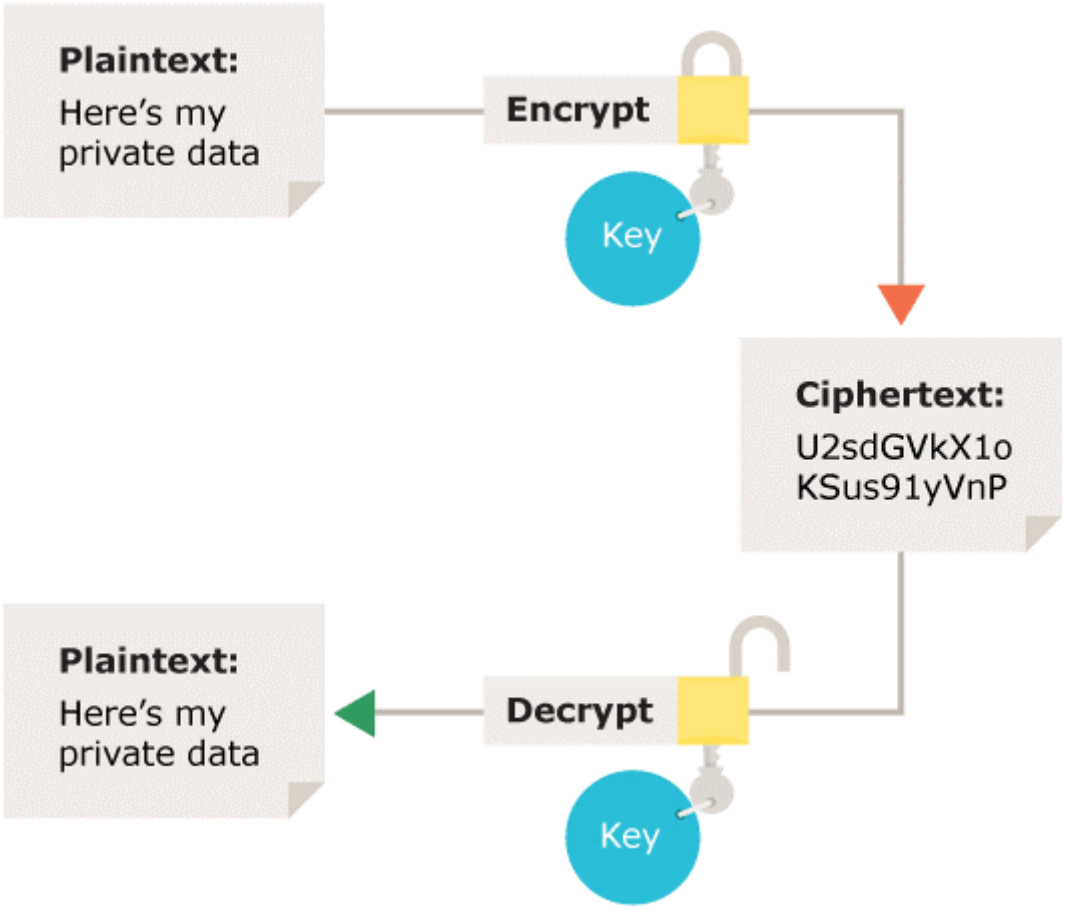
Тому в даній дипломній роботі було додано декілька рішень які дозволяють покращити безпеку на всіх рівнях

4.2 Шифрування даних перед відправкою на Google Disk

Щоб впевнетись що дані з Google диску не будуть використанні третіми особами, ми можемо, перед тим як завантажувати на диск, шифрувати їх, розглянемо коротко види шифрування та виберемо найоптимальніший із них.

Тип шифрування №1: Симетричне шифрування

Метод симетричного шифрування, як і випливає з назви, використовує один криптографічний ключ для шифрування та дешифрування даних. Використання одного ключа для обох операцій робить процес простим.



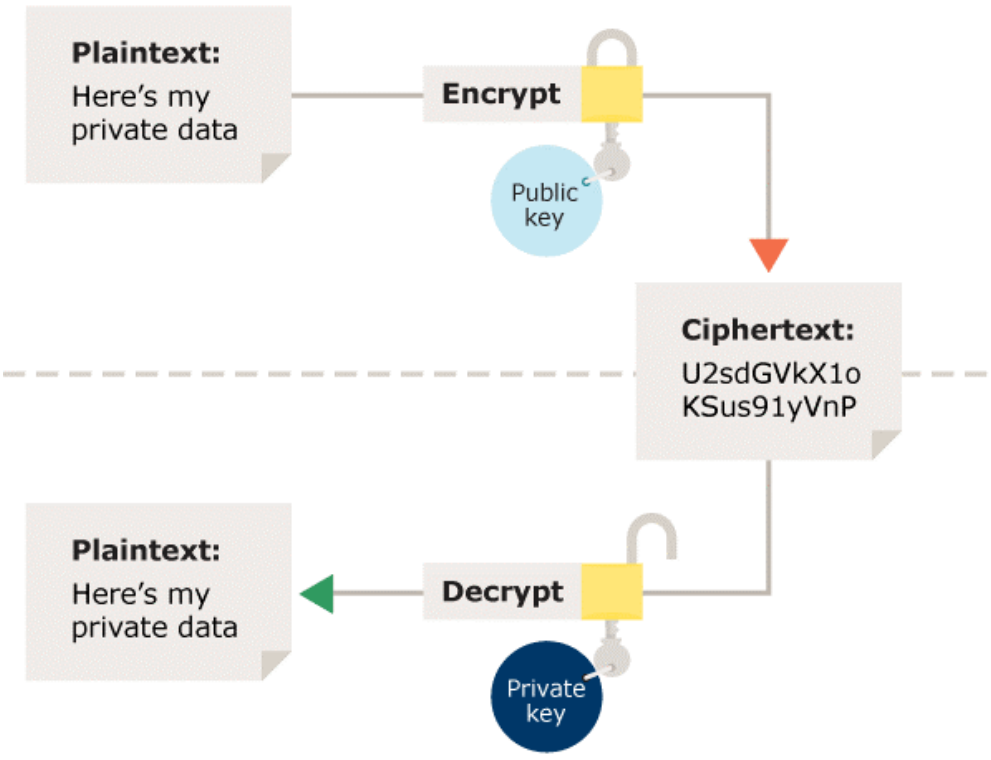
Найбільш видатною особливістю симетричного шифрування є простота процесу, оскільки використається один ключ як для шифрування, так і для дешифрування. Там, де необхідно зашифрувати великий шматок даних, симетричне

шифрування виявляється відмінним варіантом. В результаті алгоритми симетричного шифрування:

- Значно швидше, ніж їх аналоги асиметричного шифрування (про що ми скоро поговоримо);
- Потребують менше обчислювальної потужності;
- Не знижується швидкість інтернету.

Тип шифрування №2: асиметричне шифрування

Асиметричне шифрування, на відміну від симетричного, включає кілька ключів для шифрування і дешифрування даних, які математично пов'язані один з одним. Один із цих ключів відомий як «відкритий ключ», а інший – як «закритий ключ». Асиметричний метод шифрування також відомий як криптографія з відкритим ключем.



Що робить асиметричне шифрування відмінною технікою

Перше (і найбільш очевидне) перевага цього типу шифрування – безпека, яку він забезпечує. У цьому методі відкритий ключ – який є загальнодоступним –

використовується для шифрування даних, у той час як розшифрування даних виконується з використанням закритого ключа, який необхідно надійно зберігати. Це гарантує, що дані залишаються захищеними від атак «людина посередині» (MiTM). Для веб-серверів і серверів електронної пошти, які постійно підключаються до сотень тисяч клієнтів, потрібно керувати тільки одним ключем і захищати його. Інший ключовий момент полягає в тому, що криптографія з відкритим ключем дозволяє створювати зашифроване з'єднання без необхідності зустрічатися в автономному режимі, щоб спочатку обмінятися ключами.

Друга важлива особливість, яку пропонує асиметричне шифрування, – це аутентифікація. Як ми бачили, дані, зашифровані за допомогою відкритого ключа, можуть бути розшифровані тільки за допомогою закритого ключа, пов'язаного з ним. Отже, він гарантує, що дані бачить і дешифрує тільки той об'єкт, який повинен їх отримати. Простіше кажучи, це підтверджує, що ви розмовляєте чи обмінюєтеся інформацією з реальним людиною чи організацією.

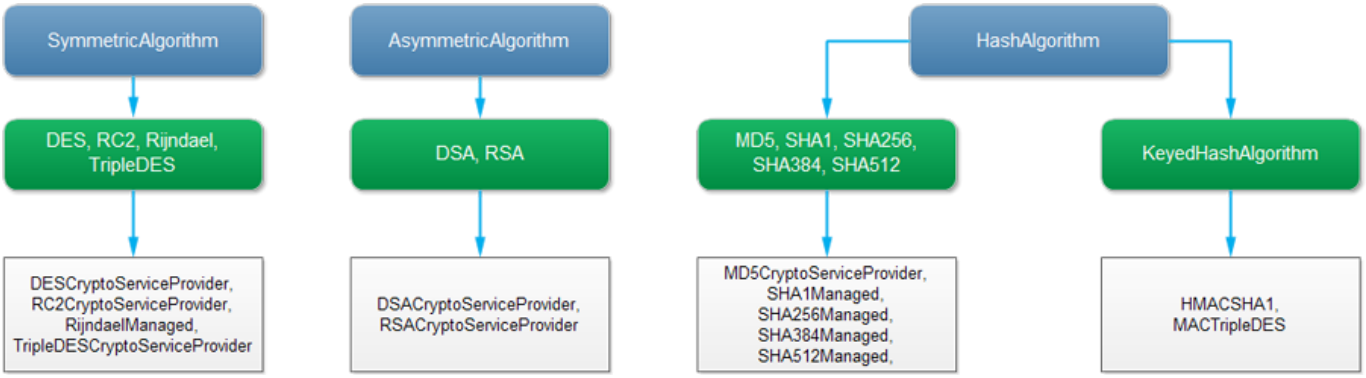
Отже, порівнявши два види шифрування, для вирішення нашої проблеми найкращим рішенням буде шифрувати дані симетрично, є ряд переваг:

1. Оскільки шифрувати і дешифрувати буде тільки сервер який взаємодіє з Google Drive Api то доречно використовувати один ключ.
2. Метод симетричного шифрування є швидшим та потребує меншої обчислювальної потужності, що дуже потрібно при роботі з файловою системою та доможе не перезавантажувати сервер додатковими операціями.

4.3 Вибір алгоритму шифрування

Платформа .Net вже надає встроєні класи для шифрування даних CryptoAPI.

На малюнку нижче показані класи із простору імен System.Security.Cryptography. Ця трирівнева організація забезпечує практично необмежену розширюваність. Створювати нові реалізації для існуючих криптографічних класів можна за рахунок успадкування існуючих класів алгоритмів:



Як бачимо на схемі є чотири алгоритма симетрично шифрування, давайте розберемось в чому між ними різниця. Надійність шифрування пропорційна довжині ключа, чим більша довжина ключа, тим менша ймовірність успішного зламування методом "грубої сили", оскільки потрібно буде перевіряти набагато більше можливих значень ключа.

Звичайно, збільшення довжини симетричного ключа також збільшує довжину зашифрованих повідомлень та час шифрування. Для більшості випадків хорошим вибором є алгоритм Rijndael. Він забезпечує високу продуктивність та підтримує ключі великого розміру.

Алгоритми DES, TripleDES і RC2 реалізовані в CryptoAPI, і тому потребують пакету High Encryption Pack. Довжина ключа для алгоритмів DES та TripleDES включає біти контролю парності, які не впливають на надійність шифрування. TripleDES з 192-бітним ключем використовує лише 168 біт, у той час як 128-бітний ключ – лише 112 біт.

У DES 64-бітний ключ використовує лише 56 біт. Тому він вважається відносно ненадійним, і краще замість нього застосовувати інші алгоритми.

Зважаючи на те що ключі з великою довжиною будуть певною мірою уповільнювати проект, а з маленькою легко взломати, було прийнято оптимальне рішення використовувати алгоритм Rijndael. Розглянемо детально цей алгоритм

4.4 Механіка алгоритму шифрування Рейндалю

Rijndael побудований як блоковий шифр. Він підтримує ключі розміром 128, 192 та 256 біт, при цьому обробка даних відбувається у 128-бітних блоках.

Виконання раундів

Шифрування під Rijndael досягається за допомогою серії матричних перетворень. Кожне перетворення відоме як раунд, і Rijndael використовує їх змінну кількість залежно від використовуваного розміру ключа або блоку. Використовується 9 раундів, якщо розмір ключа або блоку становить 128 біт. 11 раундів перетворення розгортаються, якщо розмір ключа або блоку становить 192 біти. Задіяно 13 раундів, якщо розмір ключа або блоку становить 256 біт.

Rijndael — це шифр підстановки, який використовує комбінацію трьох дискретних та зворотних шарів або однорідні матричні перетворення:

- Лінійне перетворення суміші
- Нелінійне перетворення
- Перетворення ключового додавання

Перед першим перетворенням або раундом виконується простий шар додавання ключа, що додає загальну безпеку процесу.

Потім виконуються раунди Nr-1, де Nr – загальна кількість раундів, які необхідно провести. Це число залежить від довжини блоку даних, що шифрується, і довжини ключа шифрування, що використовується.

В останньому раунді виконується крок Mix Column, де виконується множення матриці для кожного стовпця в масиві, отриманого в результаті попередніх перетворень, помножених на матрицю шифру.

Розгляд ключів

Розклад ключів шифрування для Rijndael просто вимагає, щоб розмір ключа був кратним 32 біт. Таким чином, можна використовувати ключі довжиною 160 або 224 біти. Також підтримуються розміри блоків 160 або 224 біти.

Ця гнучкість передбачена у переглянутій специфікації для Rijndael, яка вимагає 10 регулярних раундів перетворення (всього 11 раундів) для 160 біт та 12 регулярних раундів (всього 13 раундів) для 224 біт.

Зміна квадрата

Блоковий шифр Square, був уразливим до низки атак, відомих як атака Square. Стійкість до цього була досягнута шляхом заміни перетворення Shift Row на транспонування квадратної матриці байтів, що дозволило розповсюдити весь блок даних за допомогою змінних перетворень Mix Column та Mix Row.

Стабільність під час нападу

Незважаючи на те, що всі кандидати, які розглядалися для AES, були захищені від різних форм нападу, Rijndael був обраний через низькі вимоги до пам'яті та загальну ефективність.

Криптоаналітики взагалі погоджуються, що Rijndael виявиться безпечним для всіх його реальних застосувань – і цей процес можна посилити за рахунок додавання додаткових раундів трансформації. Обмежена кількість атак на алгоритм увінчалася успіхом, але вони були проведені в лабораторних умовах і переважно представляють теоретичні ситуації, які навряд чи трапляються у бізнес-контексті.

4.5 Додавання провайдеру ролей

Тепер коли дані які зберігаються на Google Disk вже є захищеними, потрібно також обмежити доступ на CRUD операції зі сторони клієнту який буде користуватися контролиреми для читання, редагування, додавання та видалення даних на сервері.

ASP.NET Identity є основною платформою, яка використовується для додавання можливостей аутентифікації та авторизації на сайті ASP.NET. Після інтеграції ASP.NET Identity з проектом ASP.NET створюється кілька таблиць бази даних, де можна зберегти відповідних користувачів. Давайте розглянемо деякі з цих таблиці:

- AspNetUs – таблиця, яка містить інформацію про користувача
- AspNetRoles – тут ми зберігаємо ролі користувачів в системі
- AspNetUserRoles – основна таблиця зіставлення, де ми зберігаємо інформацію про ролі конкретного юзера в системі.

Що ми отримуємо з коробки, так це створює користувачів, створюючи деякі ролі та призначаючи ролі користувачів. Це відповідає найкращій практиці, коли ми хочемо

в кінцевому підсумку призначити дозволи додаткам ролям, а не окремим користувачам.

Звісно не достатньо мати просто ролі юзера, наприклад адмін може виконувати CRUD операції, але що якщо конкретноу адміну не дозволено додавати або видаляти дані? Для цього AspNet Identity надає вже готовий функціонал дозволів.

Дозволи (ідентифікатор, опис) – у цій таблиці зберігається список усіх дозволів, які ми підтримуємо в програмі.

AspNetRolePermissions (RoleId, PermissionId) – таблиця зіставлення, де ми пов’язуємо роль з дозволами. Якщо в цій таблиці існує рядок, це означає що роль має конкретний дозвіл, інше ця роль не має дозволу.

Завдяки цій структурі таблиці ми маємо достатню інформацію, чи має користувач або роль певний дозвіл. Ми знаємо поточного користувача або роль (ми отримуємо це з ASP.NET Identity). Ми знаємо дозвіл, який ми перевіряємо – він передається нашому спеціальному атрибуту. Якщо ми знайдемо запис у AspNetRolePermissions для цієї комбінації/дозвіл, ми дозволяємо операцію продовжити.

5 ОПИС ІНТЕРФЕЙСУ ВЗАЄМОДІЇ КОРИСТУВАЧА З GOOGLE DRIVE В ПРИКЛАДНОМУ ПРОГРАМНОМУ ЗАБЕЗПЕЧЕНІ

В цьому розділі приведені системні вимоги для роботи з додатком та сценарії роботи користувача з ним.

5.1 Системні вимоги

Для забезпечення коректної та безвідмовної роботи програмного засобу доступу до хмарних середовищ потрібні мінімальні показники для роботи браузерів:

Процесор - intel Pentium 4 / Athlon 64 або пізнішої версії з підтримкою SSE2 .

Вільне місце на диску – 350 Мб.

Оперативна пам’ять – 512 Мб.

5.2 Робота користувача з програмним продуктом

Головне меню системи доступу до хмарного середовища має наступний вигляд, тут відображена коренева папка та основний функціонал системи (Рисунок 5.2.1).

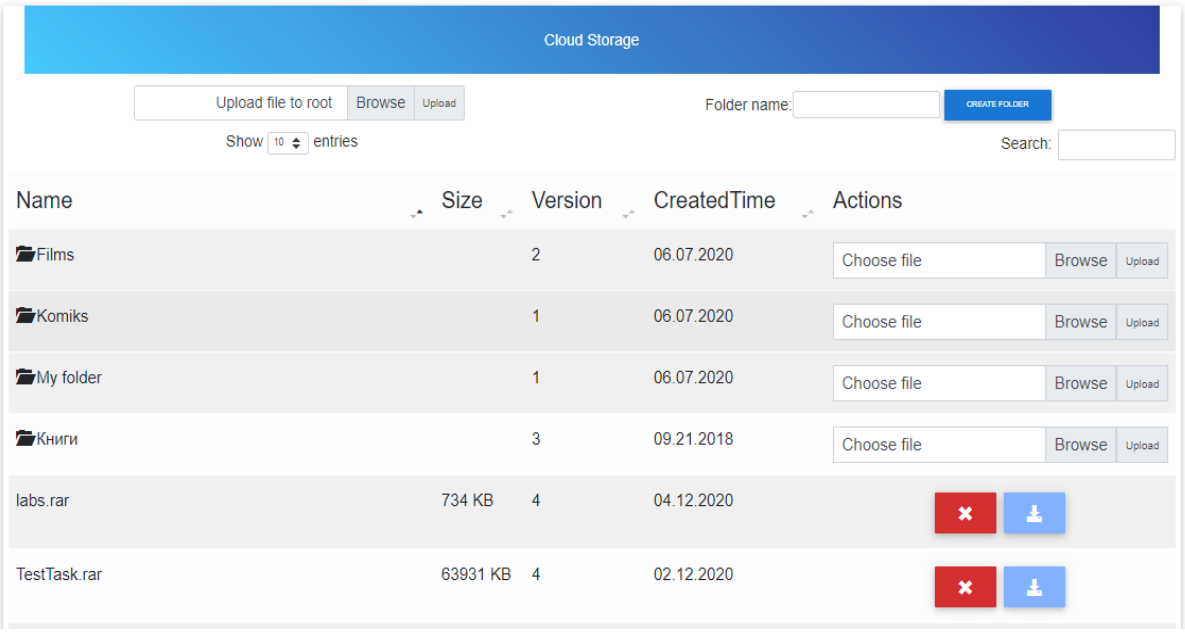


Рисунок 5.2.1 - Коренева папка хмарного сховища

Тут користувач має такі можливості:

- Перегляд директорій, файлів, папок і т. д які існують на Google Disk.
- Пошук по ключовому рядку (пошук ведеться по всім колонкам).
- Сторінкова навігація, з можливістю вибрати кількість об'єктів на сторінці .
- Додавання файлів в будь-яку папку.
- Додавання папок в кореневу або інші.
- Видалення та завантаження файлів.

Меню усіх папок, окрім кореневої має такий вигляд (Рисунок 5.2.2):

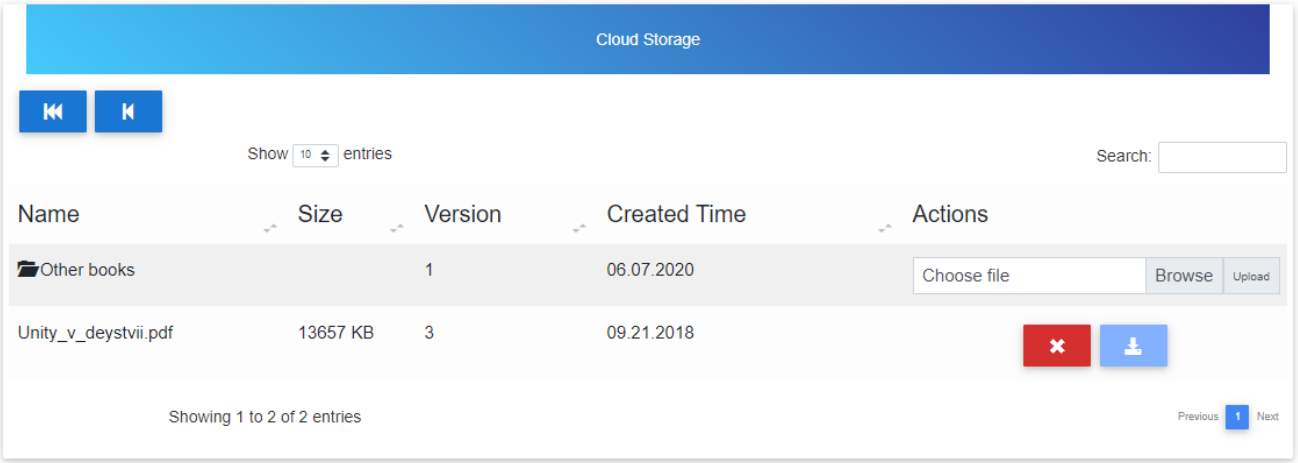


Рисунок 5.2.2 - Папка хмарного сховища

Окрім зазначених вище можливостей в папці окрім кореневої є можливість повернутись до «батьківської» папки або відразу до кореневої.

5.3 Висновки

Розроблена система має функціонал для роботи з файловою системою. Сервіс для обробки запитів працює досить швидко, тому відгук на запит виконується миттєво. Інтерфейс користувача збудований за допомогою стилів Bootstrap 4 через це, він є адаптивним до екранів будь-яких розмірів за винятком 400px та менше.

6 СТАРТАП-ПРОЕКТ «GOOGLE DRIVE SECURE»

Стартап – це тимчасова структура, яка спрямована на пошук і реалізацію масштабованої бізнес-ідеї.

В чому полягая суть проекту? На даний момент кількість користувачів Google Drive майже рівна кількості людей на планеті, тому виникає питання, чи безпечно використовувати Google Drive для бізнесу чи для зберігання важливих документів? Проект «Google Drive Secure» надає можливість використовувати хмарне середовище Google Drive безпечно та без витіку даних третім особам, а також дозволяє інтегрувати в будь-який існуючий проект який потребує інтеграцію з Google Drive при не великій кастомізації..

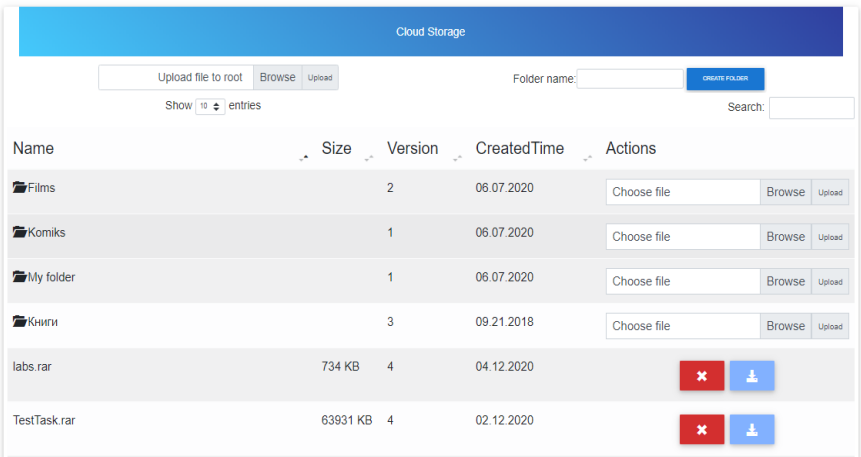
6.1 Прототип

Прототипом є прикалдне програмне забезпечення яке було описане в магістерскій дисертації. Поговоримо коротко про основні моменти.

Прикладне програмне забезпечення зроблене на базі новітніх технологій, це означає що технології проекту будуть актуальними як не меньше 5 років.

Додано рівні захисту витіку або втрати даних Google Drive за допомогою шифруванні відправлених даних на Google Drive та дадаванням права доступу на редагування певних ресурсів.

Веб інтерфейс вже створений та потребує лише не велику кастомізацію:



6.2 Монетизація

Так як проект може бути частиною (модулем) іншого проекту, компанії які мають потребу інтеграції з Google Drive будуть мати вже готовий варіант, потрібно лише внести перший внесок для доступу до програмного коду, надалі вони можуть користуватись ним у своїх цілях.

6.3 Маштабування

Користувачів Google Drive досить велика кількість, деякі з них будуть мати потребу в додатковому захисті їхніх даних. Для того щоб донести що існує такий додатковий захист великій аудиторії, потрібно на кошти інвесторів залучитись підтримкою маркетологів, які в свою чергу зможуть просунуть проект на перші сторінки пошуку.

7 ВИСНОВОК

В ході виконання даної роботи було проаналізована безпечність використання хмарного середовища Google Drive, надано шляхи покращення безпеки викристання хмарного середовища та розроблено прикладне програмне забезпечення для доступу та редагування реєстру хмарного середовища.

Програма була написана на мові програмування C# з використанням графічного веб-інтерфейсу.

Програму (модуль) можна використовувати як компонент до інших програм (проектів) без зміни коду.

В ході роботи було проведено огляд та зроблено аналіз засобів, що були використані для створення даного програмного забезпечення (середовища розробки Microsoft Visual Studio 2019, ASP.NET MVC framework та засобів створення веб-інтерфейсів: HTML5, CSS, JavaScript та Bootstrap 4).

Для роботи з даним програмним забезпеченням необхідний лише комп’ютер мінімальної потужності (для запуску браузера).

Користувач має змогу досить швидко додавати, редагувати, видаляти та завантажувати файли з та в хмарне середовище без використання сторонніх ресурсів.

СПИСОК ДЖЕРЕЛ

1. Платформа .NET [Електронний ресурс] — https://metanit.com/sharp/mvc5/

2. C# 6.0 Справочник, полное описание языка, Джозеф Албахари и Бен Албахари

3. Современный учебник CSS [Електронний ресурс] — Режим доступу: https://idg.net.ua/blog/uchebnik-css

4. Сайт Metanit all about C# [Електронний ресурс] — Режим доступу: https://metanit.com/sharp/general.php

5. Google Drive Api .NET Quickstart Complete the steps described in the rest of this page to create a simple .NET console application that makes requests to the Drive API. [Електронний ресурс] — Режим доступу: https://developers.google.com/drive/api/v3/quickstart/dotnet .

6. Tutorial for create API with ASP.NET MVC [Електронний ресурс] — Режим доступу: https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/?view=aspnetcore-3.1.

7. All about SOLID pattern [Електронний ресурс] — Режим доступу: https://habr.com/ru/post/348286/

8. Минулий диплом в якому реалізовна лише веб-застосунок: веб-сайт. URL: https://ela.kpi.ua/bitstream/123456789/36473/1/Semeniuk_bakalavr.pdf

9. Build fast responsive sites with Bootstrap. [Електронний ресурс] — Режим доступу: https://getbootstrap.com/docs/4.5/getting-started/introduction/

10. OAuth protocol [Електронний ресурс] — Режим доступу: https://developers.google.com/identity/protocols/oauth2

11. Issues with cloud storage [Електронний ресурс] — Режим доступу: https://www.datapine.com/blog/cloud-computing-risks-and-challenges/

12. Material design for Bootstrap 4 [Електронний ресурс] — Режим доступу: https://mdbootstrap.com/education/bootstrap/

13. Library Jquery [Електронний ресурс] — Режим доступу: https://jquery.com/

14. Шифрування - https://wiki.hostpro.ua/ua/knowledgebase/shifruvannja-tipi-i-algoritmi/

ДОДАТОК 1

Аналіз безпеки хмарного середовища Google Drive в прикладному програмному засобі

Текст програми

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ІПСА_ДМ-з01мп

Аркушів 7

Київ – 2021

```
//interface for CloudStorageService
using CloudStorages.Models;
using Google.Apis.Drive.v3;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Web;

namespace CloudStorages.Interfaces
{
    public interface ICloudStorageService
    {
        List<DriveFile> GetFiles();
        void UploadFile(HttpPostedFileBase file);
        string DownloadFile(string fileId);
        void DeleteFile(DriveFile file);
        List<DriveFile> GetContainsInFolder(String folderId);
        string GetParentFolder(string currentFolderId);
        void CreateFolder(string FolderName);
        void FileUploadInFolder(string folderId, HttpPostedFileBase file);
    }
}

//Model for different entity
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace CloudStorages.Models
{
    public class DriveFile
    {
        public string Id { get; set; }
        public string Name { get; set; }
        public long? Size { get; set; }
        public long? Version { get; set; }
        public DateTime? CreatedTime { get; set; }
        public IList<string> Parents { get; set; }
    }
}
//Controler
using CloudStorages.Interfaces;
using CloudStorages.Models;
using CloudStorages.Services;
using Ninject;
using System.IO;
using System.Web;
using System.Web.Mvc;

namespace CloudStorages.Controllers
{
    public class GoogleController : Controller
    {
        private readonly ICloudStorageService _cloudStorageService;
        public GoogleController(ICloudStorageService cloudStorageService) {
            _cloudStorageService = cloudStorageService;
        }
        [HttpGet]
        public ActionResult GetDriveFiles() {
```

```
            return View("RootFolder",_cloudStorageService.GetFiles());
        }

        [HttpPost]
        public ActionResult DeleteFile(DriveFile file) {
            _cloudStorageService.DeleteFile(file);
            return RedirectToAction("GetDriveFiles");
        }

        [HttpPost]
        public ActionResult UploadFile(HttpPostedFileBase file) {
            _cloudStorageService.UploadFile(file);
            return RedirectToAction("GetDriveFiles");
        }

        public void DownloadFile(string fileId) {
            string FilePath = _cloudStorageService.DownloadFile(fileId);


            Response.ContentType = "application/zip";
            Response.AddHeader("Content-Disposition", "attachment; filename=" +
Path.GetFileName(FilePath));
            Response.WriteFile(System.Web.HttpContext.Current.Server.MapPath("~/DriveFiles/" +
Path.GetFileName(FilePath)));
            Response.End();
            Response.Flush();
        }

        [HttpGet]
        public ActionResult GetContainsInFolder(string folderId)
        {
            //if root
            if(folderId == "0AO5hkdyTHaUeUk9PVA")
            {
                return RedirectToAction("GetDriveFiles");
            }
            return View("Folder",_cloudStorageService.GetContainsInFolder(folderId));
        }

        public ActionResult BackwardPrevioslyFolder(string folderId)
        {
            string parentFolder = _cloudStorageService.GetParentFolder(folderId);
            if (string.IsNullOrEmpty(parentFolder))
            {
                return RedirectToAction("GetDriveFiles");
            }
            return RedirectToAction("GetContainsInFolder", new { folderId = parentFolder });
        }
        [HttpPost]
        public ActionResult CreateFolder(string FolderName)
        {
            _cloudStorageService.CreateFolder(FolderName);
            return RedirectToAction("GetDriveFiles");
        }

        [HttpPost]
        public ActionResult FileUploadInFolder(DriveFile FolderId, HttpPostedFileBase file)
        {
            _cloudStorageService.FileUploadInFolder(FolderId.Id, file);
            return RedirectToAction("GetDriveFiles");
        }
    }
}
```

```
//Service
using CloudStorages.Interfaces;
using CloudStorages.Models;
using Google.Apis.Auth.OAuth2;
using Google.Apis.Download;
using GoogleDriveV2 = Google.Apis.Drive.v2;
using Google.Apis.Drive.v3;
using Google.Apis.Services;
using Google.Apis.Util.Store;
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading;
using System.Web;
using System.Linq;

namespace CloudStorages.Services
{
    public class GoogleCloudStorageService : ICloudStorageService
    {
        private string[] Scopes = { DriveService.Scope.Drive };

        private DriveService GetService()
        {
            UserCredential credential;
            using (var stream = new FileStream(@"C:\Main\Diplom\client_secret.json",
FileMode.Open, FileAccess.Read))
            {
                String FolderPath = @"C:\Main\Diplom\";
                String FilePath = Path.Combine(FolderPath, "DriveServiceCredentials.json");

                credential = GoogleWebAuthorizationBroker.AuthorizeAsync(
                    GoogleClientSecrets.Load(stream).Secrets,
                    Scopes,
                    "user",
                    CancellationToken.None,
                    new FileDataStore(FilePath, true)).Result;
            }

            //Create Drive API service.
            DriveService service = new DriveService(new BaseClientService.Initializer()
            {
                HttpClientInitializer = credential,
                ApplicationName = "CloudStorages",
            });

            return service;
        }

        private GoogleDriveV2.DriveService GetServiceV2()
        {
            UserCredential credential;
            using (var stream = new FileStream(@"C:\Main\Diplom\client_secret.json",
FileMode.Open, FileAccess.Read))
            {
                String FolderPath = @"C:\Main\Diplom\";
                String FilePath = Path.Combine(FolderPath, "DriveServiceCredentials.json");

                credential = GoogleWebAuthorizationBroker.AuthorizeAsync(
                    GoogleClientSecrets.Load(stream).Secrets,
                    Scopes,
                    "user",
                    CancellationToken.None,
```

```
                    new FileDataStore(FilePath, true)).Result;
            }

            //Create Drive API service.
            GoogleDriveV2.DriveService service = new GoogleDriveV2.DriveService(new
BaseClientService.Initializer()
            {
                HttpClientInitializer = credential,
                ApplicationName = "GoogleDriveRestAPI-v2",
            });

            return service;
        }

        public  List<DriveFile> GetFiles()
        {
            DriveService service = GetService();

            // Define parameters of request.
            FilesResource.ListRequest FileListRequest = service.Files.List();

            //listRequest.PageSize = 10;
            //listRequest.PageToken = 10;
            FileListRequest.Fields = "nextPageToken, files(id, name, size, version, trashed,
createdTime, parents)";

            // List files.
            IList<Google.Apis.Drive.v3.Data.File> files = FileListRequest.Execute().Files;
            List<DriveFile> FileList = new List<DriveFile>();

            if (files != null && files.Count > 0)
            {
                foreach (var file in files)
                {
                    DriveFile File = new DriveFile {
                        Id = file.Id,
                        Name = file.Name,
                        Size = file.Size,
                        Version = file.Version,
                        CreatedTime = file.CreatedTime,
                        Parents = file.Parents
                    };
                    FileList.Add(File);
                }
            }
            return FileList.Where(f => f.Version != 5).ToList();
        }

        public void UploadFile(HttpPostedFileBase file)
        {
            if (file != null && file.ContentLength > 0)
            {
                DriveService service = GetService();

                string path = Path.Combine(HttpContext.Current.Server.MapPath("~/DriveFiles"),
                Path.GetFileName(file.FileName));
                file.SaveAs(path);

                var FileMetaData = new Google.Apis.Drive.v3.Data.File();
                FileMetaData.Name = Path.GetFileName(file.FileName);
                FileMetaData.MimeType = MimeMapping.GetMimeMapping(path);

                FilesResource.CreateMediaUpload request;
```

```
                using (var stream = new System.IO.FileStream(path, System.IO.FileMode.Open))
                {
                    request = service.Files.Create(FileMetaData, stream, FileMetaData.MimeType);
                    request.Fields = "id";
                    request.Upload();
                }
            }
        }

        public string DownloadFile(string fileId)
        {
            DriveService service = GetService();

            string FolderPath = System.Web.HttpContext.Current.Server.MapPath("/DriveFiles/");
            FilesResource.GetRequest request = service.Files.Get(fileId);

            string FileName = request.Execute().Name;
            string FilePath = System.IO.Path.Combine(FolderPath, FileName);

            MemoryStream stream1 = new MemoryStream();

            request.MediaDownloader.ProgressChanged += (Google.Apis.Download.IDownloadProgress
progress) =>
            {
                switch (progress.Status)
                {
                    case DownloadStatus.Downloading:
                        {
                            break;
                        }
                    case DownloadStatus.Completed:
                        {
                            SaveStream(stream1, FilePath);
                            break;
                        }
                    case DownloadStatus.Failed:
                        {
                            break;
                        }
                }
            };
            request.Download(stream1);
            return FilePath;
        }

        private void SaveStream(MemoryStream stream, string FilePath)
        {
            using (System.IO.FileStream file = new FileStream(FilePath, FileMode.Create,
FileAccess.ReadWrite))
            {
                stream.WriteTo(file);
            }
        }

        public void DeleteFile(DriveFile file)
        {
            DriveService service = GetService();
            try
            {
                // Initial validation.
                if (service == null)
                    throw new ArgumentNullException("service");
```

```
                if (file.Id == null)
                    throw new ArgumentNullException(file.Id);

                // Make the request.
                service.Files.Delete(file.Id).Execute();
            }
            catch (Exception ex)
            {
                throw new Exception("Request Files.Delete failed.", ex);
            }
        }
        public List<DriveFile> GetContainsInFolder(String folderId)
        {
            List<string> ChildList = new List<string>();
            var serviceV2 = GetServiceV2();
            GoogleDriveV2.ChildrenResource.ListRequest ChildrenIDsRequest =
serviceV2.Children.List(folderId);
            do
            {
                var children = ChildrenIDsRequest.Execute();

                if (children.Items != null && children.Items.Count > 0)
                {
                    foreach (var file in children.Items)
                    {
                        ChildList.Add(file.Id);
                    }
                }
                ChildrenIDsRequest.PageToken = children.NextPageToken;

            } while (!String.IsNullOrEmpty(ChildrenIDsRequest.PageToken));

            //Get All File List
            List<DriveFile> AllFileList = GetFiles();
            List<DriveFile> Filter_FileList = new List<DriveFile>();

            foreach (string Id in ChildList)
            {
                Filter_FileList.Add(AllFileList.Where(x => x.Id == Id).FirstOrDefault());
            }

            return Filter_FileList;
        }

        public string GetParentFolder(string currentFolderId)
        {
            List<DriveFile> AllFileList = GetFiles();
            return AllFileList.FirstOrDefault(f => f.Id == currentFolderId)?.Parents[0];
        }
        public void CreateFolder(string FolderName)
        {
            Google.Apis.Drive.v3.DriveService service = GetService();

            Google.Apis.Drive.v3.Data.File FileMetaData = new Google.Apis.Drive.v3.Data.File();
            FileMetaData.Name = FolderName;
            FileMetaData.MimeType = "application/vnd.google-apps.folder";

            Google.Apis.Drive.v3.FilesResource.CreateRequest request;

            request = service.Files.Create(FileMetaData);
            request.Fields = "id";
            var file = request.Execute();
```

```
            //Console.WriteLine("Folder ID: " + file.Id);
        }
        public void FileUploadInFolder(string folderId, HttpPostedFileBase file)
        {
            if (file != null && file.ContentLength > 0)
            {
                DriveService service = GetService();

                string path = Path.Combine(HttpContext.Current.Server.MapPath("~/DriveFiles"),
                Path.GetFileName(file.FileName));
                file.SaveAs(path);

                var FileMetaData = new Google.Apis.Drive.v3.Data.File()
                {
                    Name = Path.GetFileName(file.FileName),
                    MimeType = MimeMapping.GetMimeMapping(path),
                    Parents = new List<string>
                    {
                        folderId
                    }
                };

                Google.Apis.Drive.v3.FilesResource.CreateMediaUpload request;
                using (var stream = new System.IO.FileStream(path, System.IO.FileMode.Open))
                {
                    request = service.Files.Create(FileMetaData, stream, FileMetaData.MimeType);
                    request.Fields = "id";
                    request.Upload();
                }
                var file1 = request.ResponseBody;
            }
        }
    }
}
```