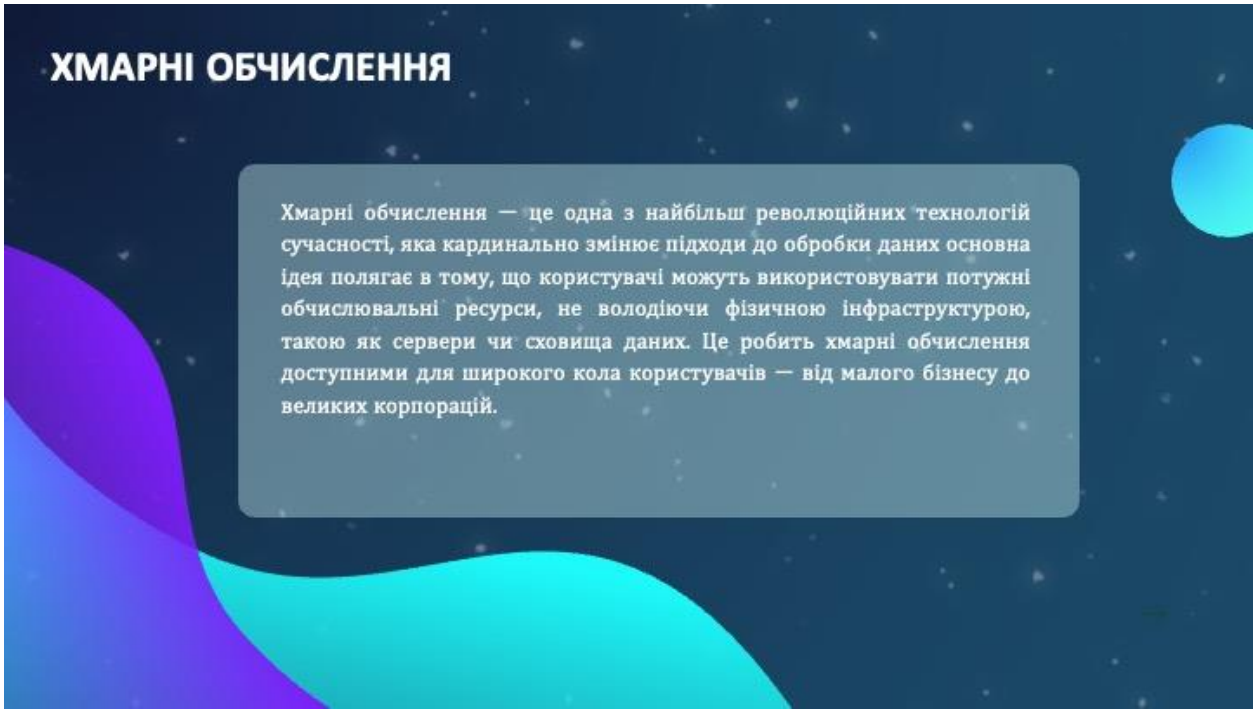
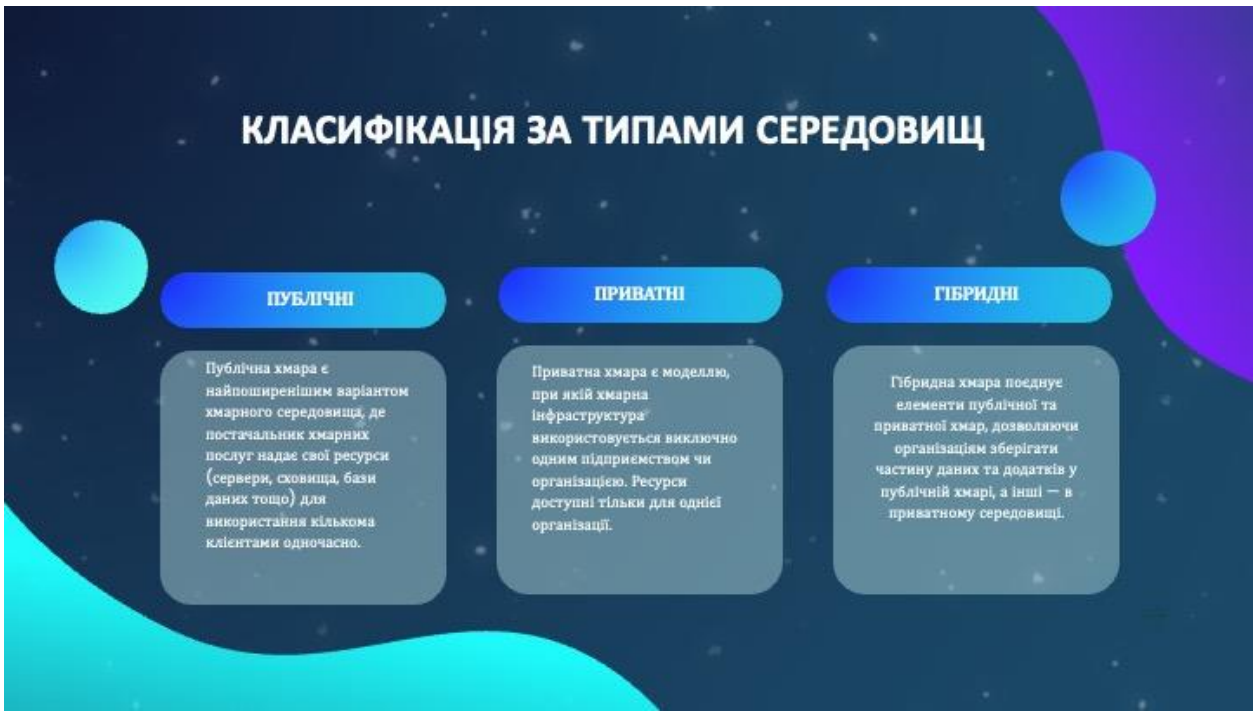
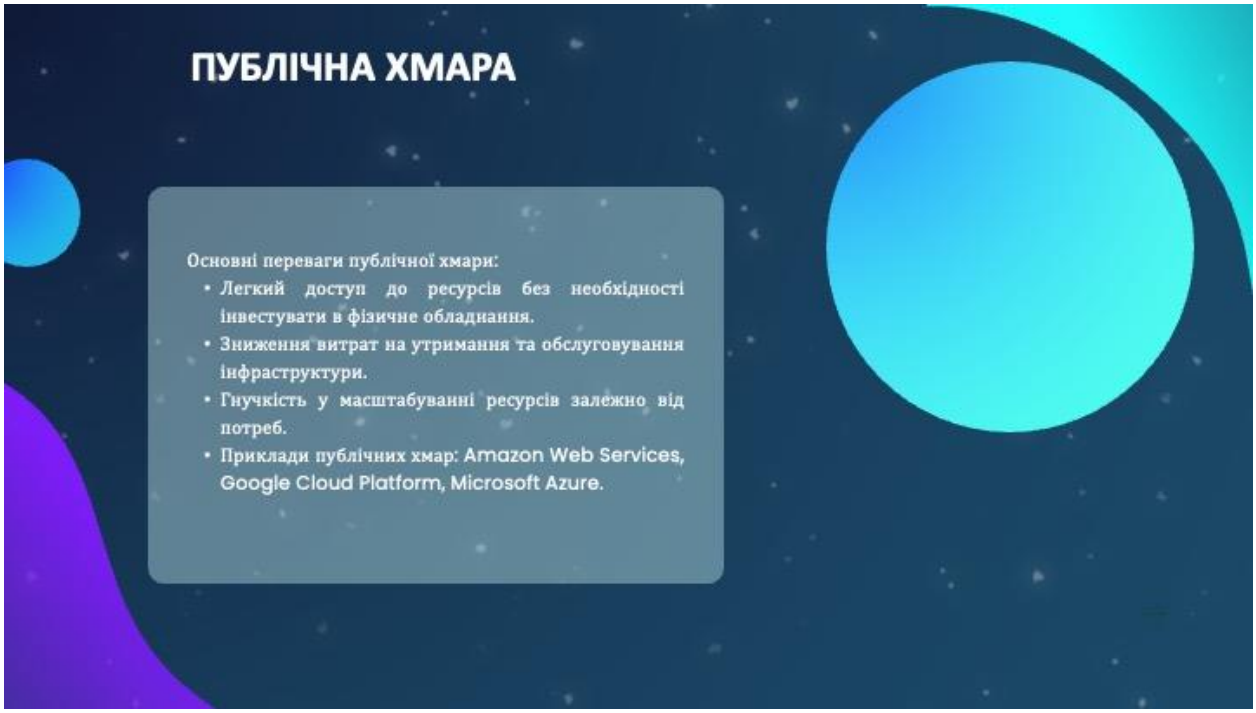
ДОДАТОК А

Графічний матеріал кваліфікаційної роботи


РОЗРОБКА ТА ВПРОВАДЖЕННЯ МЕХАНІЗМІВ
Персональних даних у хмарних обчисленнях
Попов Артем СПм-23-2

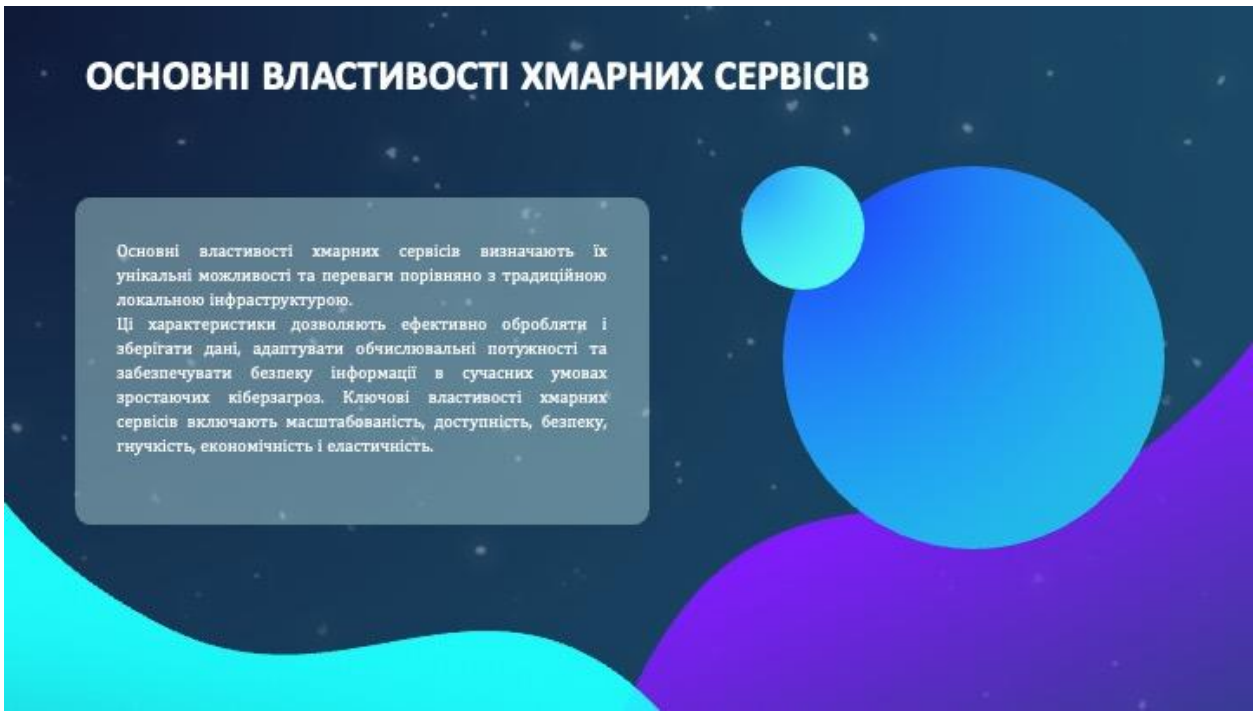

ХМАРНІ ОБЧИСЛЕННЯ
Хмарні обчислення — це одна з найбільш революційних технологій сучасності, яка кардинально змінює підходи до обробки даних основна ідея полягає в тому, що користувачі можуть використовувати потужні обчислювальні ресурси, не володіючи фізичною інфраструктурою, такою як сервери чи сховища даних. Це робить хмарні обчислення доступними для широкого кола користувачів — від малого бізнесу до великих корпорацій.

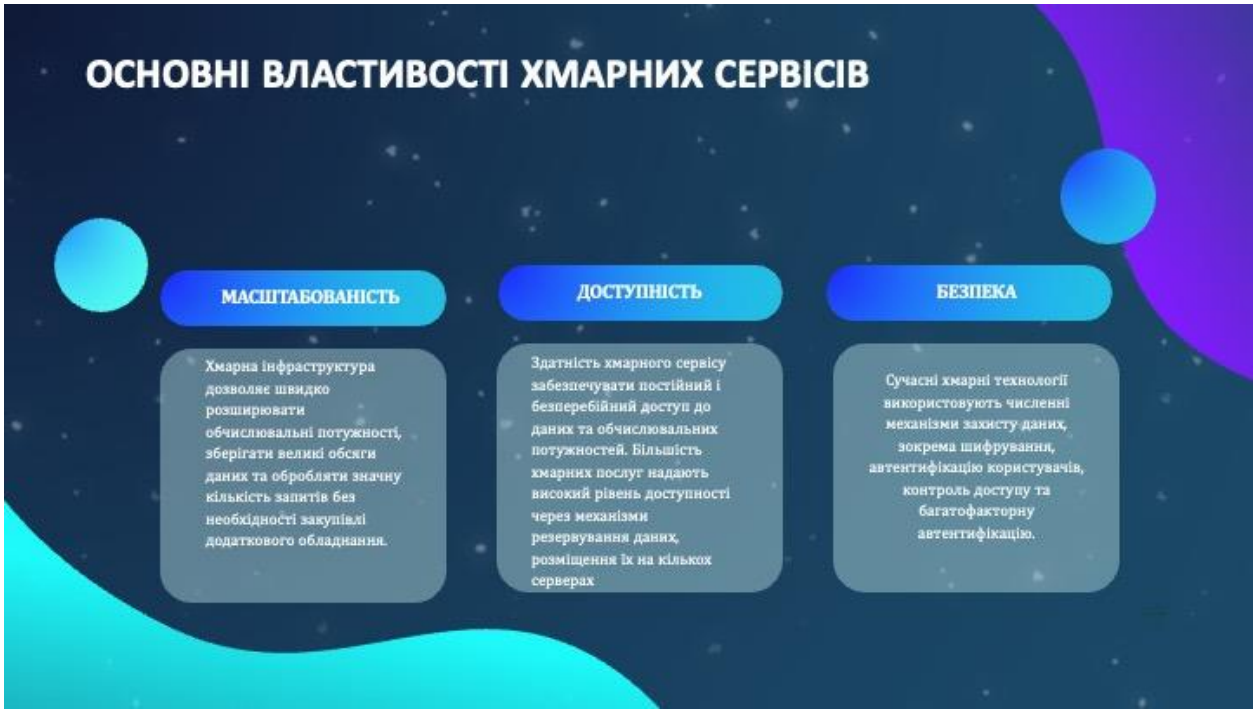

КЛАСИФІКАЦІЯ ЗА ТИПАМИ СЕРЕДОВИЩ
ПУБЛІЧНІ
Публічна хмара є найпоширенішим варіантом хмарного середовища, де постачальник хмарних послуг надає свої ресурси (сервери, сховища, бази даних тощо) для використання кількома клієнтами одночасно.
ПРИВАТНІ
Приватна хмара є моделлю, при якій хмарна інфраструктура використовується виключно одним підприємством чи організацією. Ресурси доступні тільки для однієї організації.
ГІБРИДНІ
Гібридна хмара поєднує елементи публічної та приватної хмар, дозволяючи організаціям зберігати частину даних та додатків у публічній хмарі, а інші — в приватному середовищі.

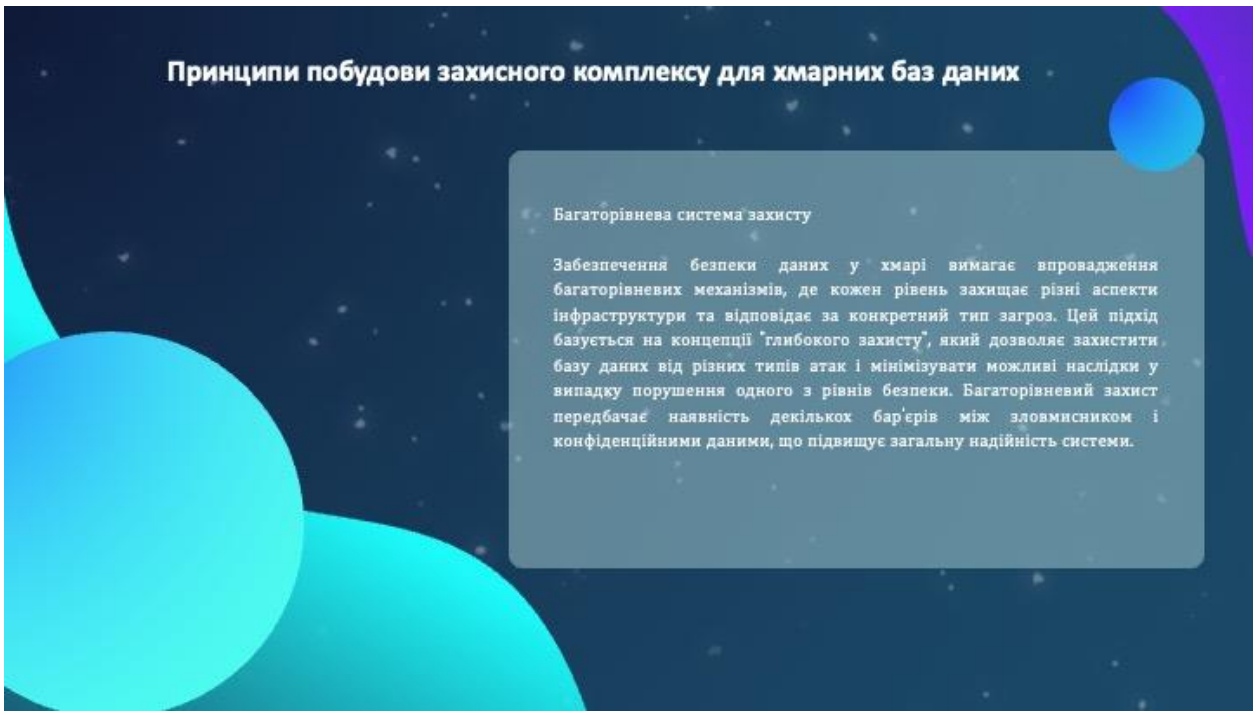

ПУБЛІЧНА ХМАРА
Основні переваги публічної хмари:
• Легкий доступ до ресурсів без необхідності інвестувати в фізичне обладнання.
• Зниження витрат на утримання та обслуговування інфраструктури.
• Гнучкість у масштабуванні ресурсів залежно від потреб.
• Приклади публічних хмар: Amazon Web Services, Google Cloud Platform, Microsoft Azure.

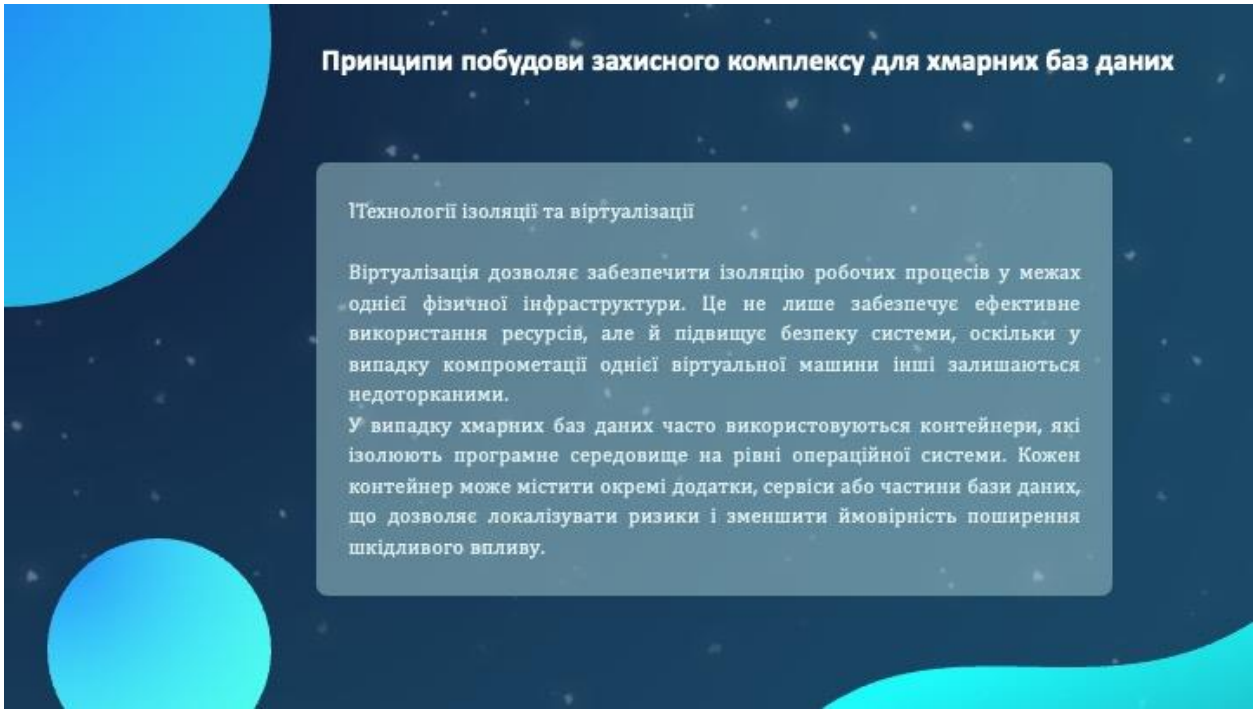

ПРИВАТНА ХМАРА
Переваги приватної хмари:
• Повний контроль над безпекою та конфіденційністю даних.
• Можливість налаштування інфраструктури під конкретні потреби бізнесу.
• Вищий рівень персоналізації та інтеграції.
• Приватна хмара підходить для організацій, що мають високі вимоги до безпеки або роботи з конфіденційними даними.

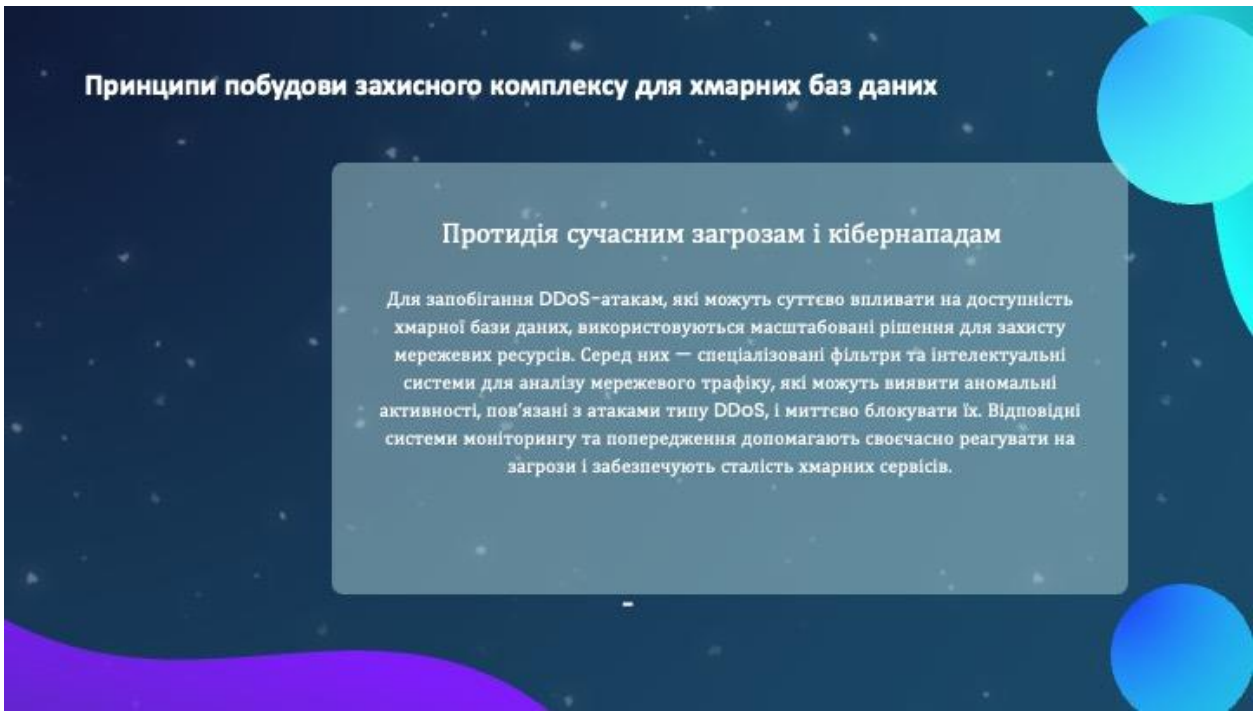

ГІБРИДНА ХМАРА
Переваги гібридної хмари:
• Гнучкість у виборі ресурсів залежно від потреб.
• Можливість інтегрувати локальну інфраструктуру з хмарними сервісами.
• Підвищена безпека для критичних даних, зберігання яких вимагає високого рівня конфіденційності.

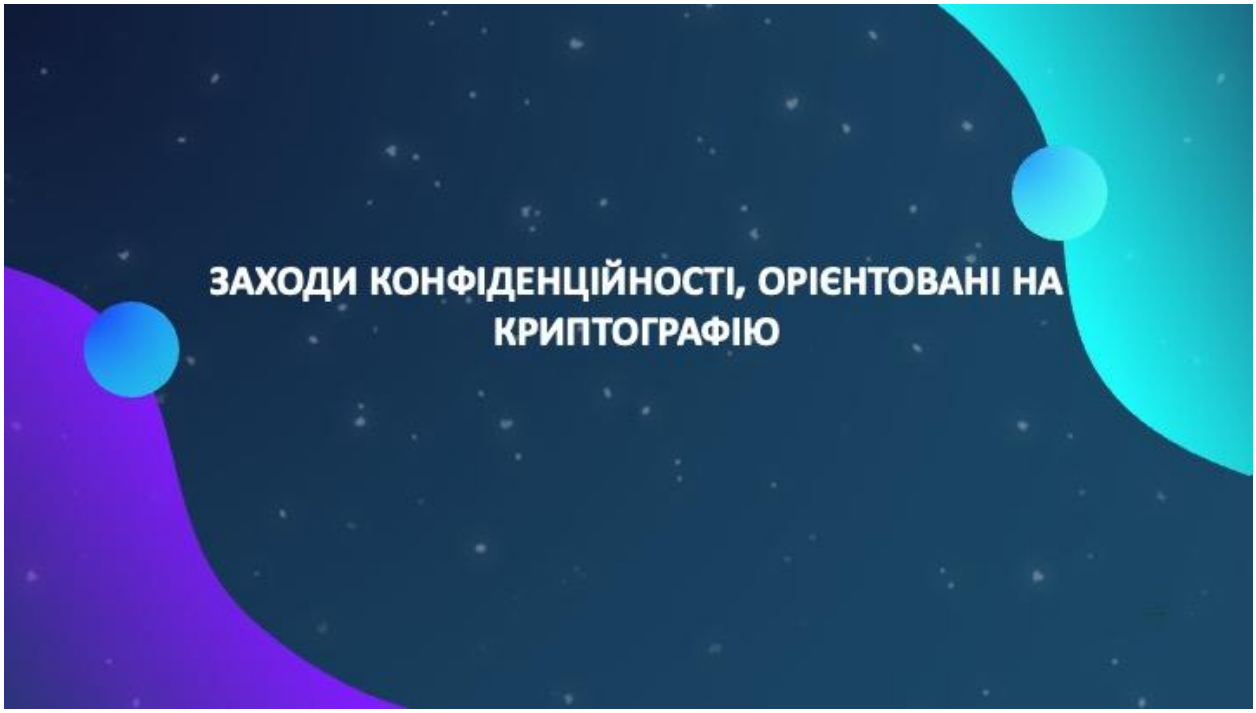

ОСНОВНІ ВЛАСТИВОСТІ ХМАРНИХ СЕРВІСІВ
Основні властивості хмарних сервісів визначають їх унікальні можливості та переваги порівняно з традиційною локальною інфраструктурою.
Ці характеристики дозволяють ефективно обробляти і зберігати дані, адаптувати обчислювальні потужності та забезпечувати безпеку інформації в сучасних умовах зростаючих кіберзагроз. Ключові властивості хмарних сервісів включають масштабованість, доступність, безпеку, гнучкість, економічність і еластичність.


ОСНОВНІ ВЛАСТИВОСТІ ХМАРНИХ СЕРВІСІВ
МАСШТАБОВАНІСТЬ
Хмарна інфраструктура дозволяє швидко розширювати обчислювальні потужності, зберігати великі обсяги даних та обробляти значну кількість запитів без необхідності закупівлі додаткового обладнання.
ДОСТУПНІСТЬ
Здатність хмарного сервісу забезпечувати постійний і безперебійний доступ до даних та обчислювальних потужностей. Більшість хмарних послуг надають високий рівень доступності через механізми резервування даних, розміщення їх на кількох серверах
БЕЗПЕКА
Сучасні хмарні технології використовують численні механізми захисту даних, зокрема шифрування, автентифікацію користувачів, контроль доступу та багатофакторну автентифікацію.


Принципи побудови захисного комплексу для хмарних баз даних
Багаторівнева система захисту
Забезпечення безпеки даних у хмарі вимагає впровадження багаторівневих механізмів, де кожен рівень захищає різні аспекти інфраструктури та відповідає за конкретний тип загроз. Цей підхід базується на концепції "глибокого захисту", який дозволяє захистити базу даних від різних типів атак і мінімізувати можливі наслідки у випадку порушення одного з рівнів безпеки. Багаторівневий захист передбачає наявність декількох бар'єрів між зловмисником і конфіденційними даними, що підвищує загальну надійність системи.


Принципи побудови захисного комплексу для хмарних баз даних
Технології ізоляції та віртуалізації
Віртуалізація дозволяє забезпечити ізоляцію робочих процесів у межах однієї фізичної інфраструктури. Це не лише забезпечує ефективне використання ресурсів, але й підвищує безпеку системи, оскільки у випадку компрометації однієї віртуальної машини інші залишаються недоторканими.
У випадку хмарних баз даних часто використовуються контейнери, які ізолюють програмне середовище на рівні операційної системи. Кожен контейнер може містити окремі додатки, сервіси або частини бази даних, що дозволяє локалізувати ризики і зменшити ймовірність поширення шкідливого впливу.


Принципи побудови захисного комплексу для хмарних баз даних
Протидія сучасним загрозам і кібернападам
Для запобігання DDoS-атакам, які можуть суттєво впливати на доступність хмарної бази даних, використовуються масштабовані рішення для захисту мережевих ресурсів. Серед них — спеціалізовані фільтри та інтелектуальні системи для аналізу мережевого трафіку, які можуть виявити аномальні активності, пов'язані з атаками типу DDoS, і миттєво блокувати їх. Відповідні системи моніторингу та попередження допомагають своєчасно реагувати на загрози і забезпечують сталість хмарних сервісів.


ЗАХОДИ КОНФІДЕНЦІЙНОСТІ, ОРІЄНТОВАНІ НА КРИПТОГРАФІЮ

Використання деталізованого управління правами

Для деталізованого управління правами використовується передовий криптографічний інструмент, відомий як схема "перешифрування через проксі". На основі цієї схеми було модифіковано підхід, який дозволяє клієнтам динамічно керувати своїми спільними документами у деревоподібній структурі. Пізніше було представлено реалізацію такої системи, яка включає використання смартфонів для завантаження, завантаження та обміну документами клієнтів. Система зосереджена на деталізованому управлінні правами, тобто на забезпеченні та обміні правами доступу на основі пріоритету користувача.

Використання віддаленого аудиту даних (RDA)

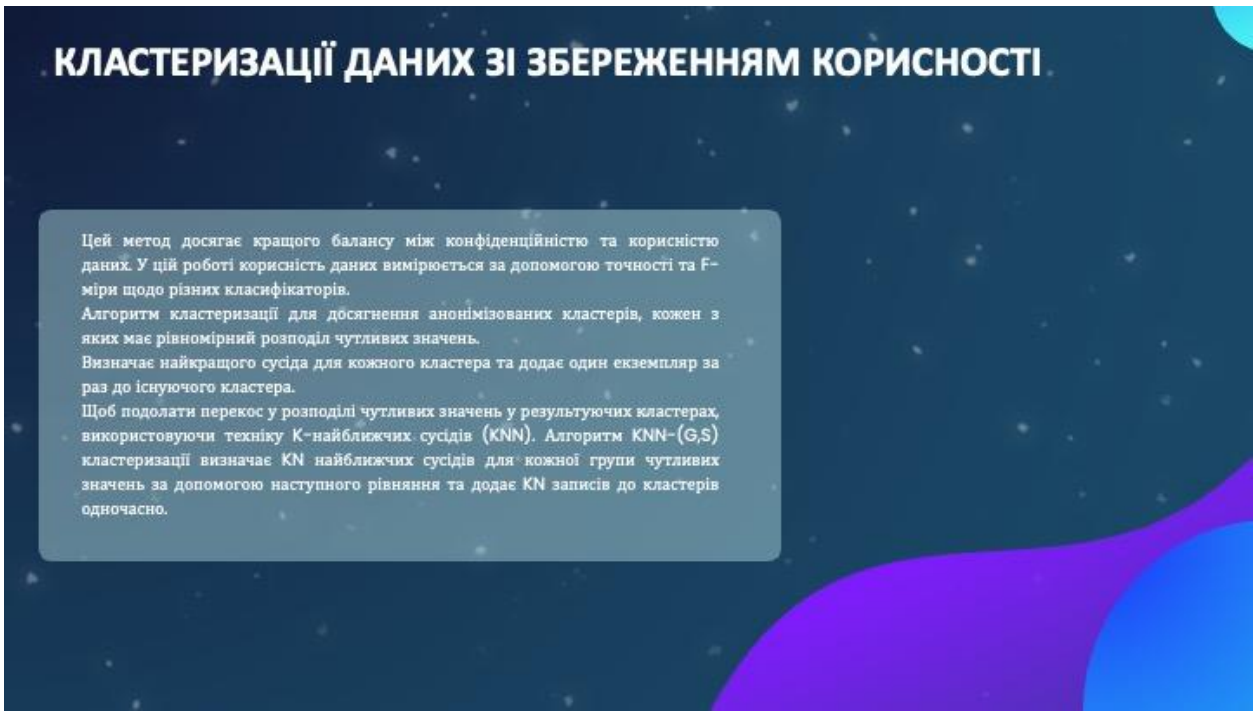
Техніка RDA (Remote Data Auditing) належить до категорії криптографічних методів, оскільки забезпечує ймовірнісну або детерміновану гарантію цілісності даних. Вона включає такі властивості:

Ефективність: аудит даних із мінімально можливою обчислювальною складністю.

Публічна перевірка: можливість делегувати процес аудиту довіреній стороні (TPA), що зменшує обчислювальне навантаження на клієнта.

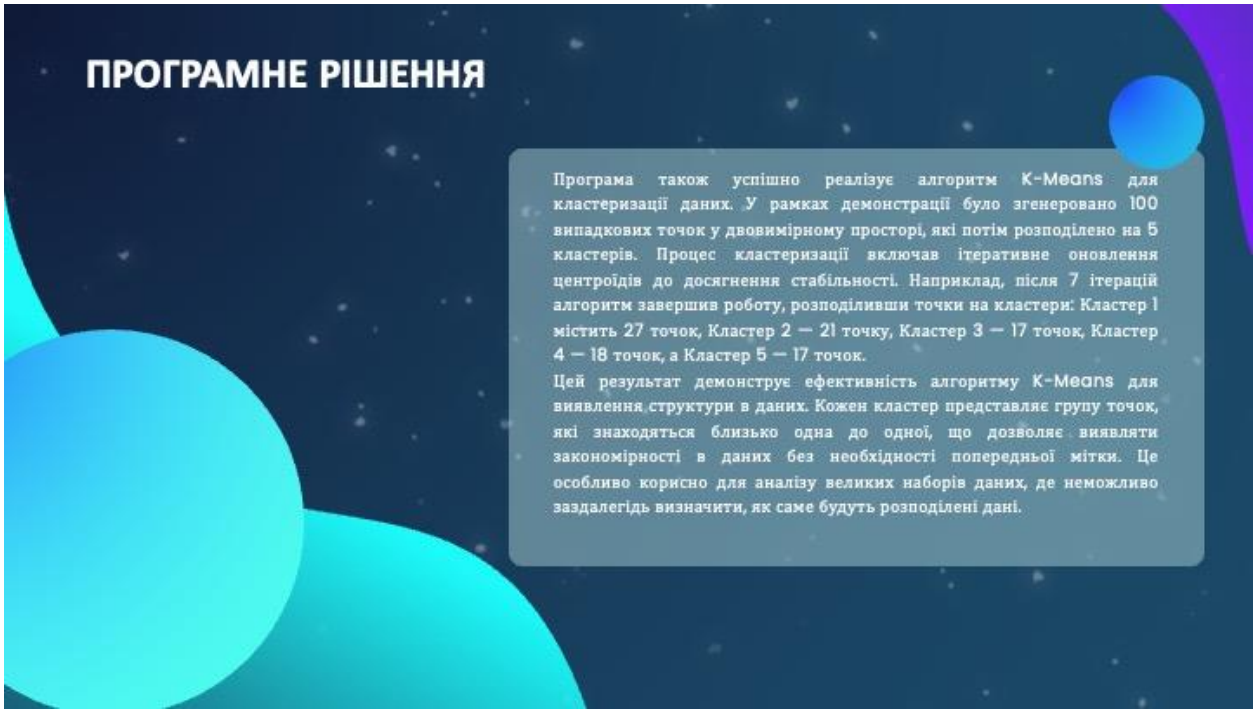
Ймовірність виявлення: можливість виявлення потенційних пошкоджень даних.

Для схеми публічного аудиту зі збереженням конфіденційності важливими вимогами є надійність, повнота та конфіденційність даних. Повнота означає, що якщо дані не змінені, інтерактивний протокол завжди повертає ⟨P, V⟩ = 1, коли хмарний сервер та TPA дотримуються протоколу чесно. Концепція нульового розголошення гарантує, що TPA не отримує інформації про вміст файлу, окрім публічно доступних даних, таких як випадкове ім'я файлу. Це підтверджено на основі оцінки властивостей, і безпека схеми, включаючи надійність, є ефективною та придатною для практичного використання.


КЛАСТЕРИЗАЦІЇ ДАНИХ ЗІ ЗБЕРЕЖЕННЯМ КОРИСНОСТІ
Цей метод досягає кращого балансу між конфіденційністю та корисністю даних. У цій роботі корисність даних вимірюється за допомогою точності та F-міри щодо різних класифікаторів.
Алгоритм кластеризації для досягнення анонімізованих кластерів, кожен з яких має рівномірний розподіл чутливих значень.
Визначає найкращого сусіда для кожного кластера та додає один екземпляр за раз до існуючого кластера.
Щоб подолати перекос у розподілі чутливих значень у результуючих кластерах, використовуючи техніку K-найближчих сусідів (KNN). Алгоритм KNN-(G,S) кластеризації визначає KN найближчих сусідів для кожної групи чутливих значень за допомогою наступного рівняння та додає KN записів до кластерів одночасно.


ПРОГРАМНЕ РІШЕННЯ
Для роботи програми обрано RSA та K-Means, вони використовують перевірені методи для забезпечення безпеки та аналізу даних. З їх допомогою можна досягти:
- Захист даних. Використання алгоритму RSA дозволяє шифрувати і дешифрувати дані, гарантуючи їх конфіденційність при передачі через незахищені канали зв'язку. Це особливо важливо для сучасних систем електронного банкінгу, онлайн-платежів та будь-яких інших областей, де передача чутливих даних є необхідною.
- Аналіз даних. Алгоритм K-Means дозволяє проводити кластеризацію великих масивів даних, що корисно для різних сфер, таких як маркетингові дослідження, аналіз поведінки користувачів в Інтернеті та навіть медичні дослідження.


ПРОГРАМНЕ РІШЕННЯ
Для демонстрації роботи алгоритму RSA було зашифровано повідомлення "Hello, World!". Кожен символ повідомлення перетворено у його ASCII-код, після чого цей код піднесено до степеня e=179 за модулем n=14857. Результатом шифрування став набір чисел: [8306, 7798, 7805, 7805, 4228, 7562, 13636, 13154, 4228, 1188, 7805, 10661, 13995]. Цей зашифрований набір може бути переданий через незахищені канали зв'язку без ризику витоку інформації.


ПРОГРАМНЕ РІШЕННЯ
Програма також успішно реалізує алгоритм K-Means для кластеризації даних. У рамках демонстрації було згенеровано 100 випадкових точок у двовимірному просторі, які потім розподілено на 5 кластерів. Процес кластеризації включав ітеративне оновлення центроїдів до досягнення стабільності. Наприклад, після 7 ітерацій алгоритм завершив роботу, розподіливши точки на кластери: Кластер 1 містить 27 точок, Кластер 2 — 21 точку, Кластер 3 — 17 точок, Кластер 4 — 18 точок, а Кластер 5 — 17 точок.
Цей результат демонструє ефективність алгоритму K-Means для виявлення структури в даних. Кожен кластер представляє групу точок, які знаходяться близько одна до одної, що дозволяє виявляти закономірності в даних без необхідності попередньої мітки. Це особливо корисно для аналізу великих наборів даних, де неможливо заздалегідь визначити, як саме будуть розподілені дані.

ДОДАТОК Б

Код програми

```
import random  # Для генерації випадкових чисел

### Власна реалізація RSA
class SimpleRSA:
    @staticmethod
    def gcd(a, b):
        """Обчислює найбільший спільний дільник (НСД) двох чисел."""
        while b != 0:
            a, b = b, a % b
        return a

    @staticmethod
    def is_prime(n):
        """Перевіряє, чи є число простим."""
        if n <= 1:
            return False
        for i in range(2, int(SimpleRSA.custom_sqrt(n)) + 1):
            if n % i == 0:
                return False
        return True

    @staticmethod
    def generate_prime():
        """Генерує просте число у діапазоні 50-200."""
        print("Генерація простого числа...", end=" ", flush=True)
        while True:
            p = random.randint(50, 200)  # Зменшений діапазон для швидкості
            if SimpleRSA.is_prime(p):
                print(f"Знайдено просте число: {p}")
                return p
```

```
@staticmethod
def generate_keys():
    """Генерує пару ключів RSA (публічний та приватний)."""
    print("Генерація ключів RSA...")
    p = SimpleRSA.generate_prime()  # Перше просте число
    q = SimpleRSA.generate_prime()  # Друге просте число
    n = p * q  # Модуль
    phi = (p - 1) * (q - 1)  # Функція Ейлера

    print("Пошук відкритого ключа (e)...", end=" ", flush=True)
    e = random.randint(2, phi - 1)  # Відкритий ключ
    while SimpleRSA.gcd(e, phi) != 1:
        e = random.randint(2, phi - 1)
    print(f"Знайдено e: {e}")

    print("Обчислення закритого ключа (d)...", end=" ", flush=True)
    d = SimpleRSA.modinv(e, phi)  # Закритий ключ
    print(f"Знайдено d: {d}")

    print(f"Публічний ключ: (e={e}, n={n})")
    print(f"Приватний ключ: (d={d}, n={n})")
    return (e, n), (d, n)

@staticmethod
def modinv(a, m):
    """Обчислює обернений елемент за модулем."""
    g, x, y = SimpleRSA.extended_gcd(a, m)
    if g != 1:
        return None  # Оберненого не існує
```

```
            else:
                return x % m

        @staticmethod
        def extended_gcd(a, b):
            """Розширений алгоритм Евкліда для знаходження НСД та коефіцієнтів."""
            if a == 0:
                return (b, 0, 1)
            else:
                g, x, y = SimpleRSA.extended_gcd(b % a, a)
                return (g, y - (b // a) * x, x)

        @staticmethod
        def custom_sqrt(n, tolerance=1e-10):
            """Обчислює квадратний корінь числа методом Ньютона."""
            x = n
            while True:
                root = 0.5 * (x + n / x)
                if abs(root - x) < tolerance:
                    return root
                x = root

        @staticmethod
        def encrypt(message, public_key):
            """Шифрує повідомлення за допомогою публічного ключа."""
            e, n = public_key
            encrypted_data = [pow(ord(char), e, n) for char in message]  # Шифрування кожного символу
            print(f"Зашифроване повідомлення: {encrypted_data}")
```

```
            return encrypted_data

        @staticmethod
        def decrypt(encrypted_message, private_key):
            """Дешифрує повідомлення за допомогою приватного ключа."""
            d, n = private_key
            decrypted_data = ''.join([chr(pow(char, d, n)) for char in encrypted_message])  # Дешифрування
            print(f"Розшифроване повідомлення: {decrypted_data}")
            return decrypted_data

    ### Власна реалізація K-Means
    class SimpleKMeans:
        @staticmethod
        def kmeans(data, k, max_iterations=100):
            """Виконує кластеризацію даних за допомогою алгоритму K-Means."""
            print(f"Запуск кластеризації K-Means для {len(data)} точок...")
            centroids = [data[i] for i in random.sample(range(len(data)), k)]  # Випадкові центроїди
            for iteration in range(max_iterations):
                print(f"Ітерація {iteration + 1}/{max_iterations}...", end=" ", flush=True)
                clusters = [[] for _ in range(k)]  # Створення кластерів
                for point in data:
                    distances = [SimpleKMeans.euclidean_distance(point, centroid) for centroid in centroids]
                    cluster_index = distances.index(min(distances))  # Визначення найближчого центроїда
                    clusters[cluster_index].append(point)
```

```
            new_centroids = [SimpleKMeans.mean(cluster) for cluster in clusters]  # Нові центроїди
            if new_centroids == centroids:
                print("\nКластеризація завершена.")
                break
            centroids = new_centroids
        return centroids, clusters

    @staticmethod
    def euclidean_distance(a, b):
        """Обчислює евклідову відстань між двома точками."""
        return sum((x - y) ** 2 for x, y in zip(a, b)) ** 0.5

    @staticmethod
    def mean(points):
        """Обчислює середнє значення для набору точок."""
        return [sum(x) / len(x) for x in zip(*points)]

### Головний додаток
def main():
    """Основна функція програми."""
    print("=== Система захисту даних ===")
    rsa = SimpleRSA()
    public_key, private_key = rsa.generate_keys()  # Генерація ключів RSA

    # Шифрування та дешифрування
    message = "Hello, World!"
    print(f"\nОригінальне повідомлення: {message}")
    encrypted_message = rsa.encrypt(message, public_key)  # Шифрування
    decrypted_message = rsa.decrypt(encrypted_message, private_key)  #
```

Дешифрування

```
        # Кластеризація
        print("\nЗапуск кластеризації...")
        data = [[random.random(), random.random()] for _ in range(100)]  # Випадкові дані
        centroids, clusters = SimpleKMeans.kmeans(data, k=5)  # Кластеризація
        print("Результати кластеризації:")
        for i, cluster in enumerate(clusters):
            print(f"Кластер {i + 1}: {len(cluster)} точок")

    if __name__ == "__main__":
        main()
```