Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

БАКАЛАВРСЬКА ДИПЛОМНА РОБОТА

на тему:

«РОЗРОБКА ЗАХИЩЕНОГО ВЕБСЕРВІСУ ДЛЯ РЕЗЕРВНОГО КОПІЮВАННЯ ДАНИХ У СЕРЕДОВИЩІ AMAZON S3 З ВИКОРИСТАННЯМ НАСКРІЗНОГО ШИФРУВАННЯ»

Виконав: студент 4 курсу, групи 1КІТС–20б
спеціальності 125– Кібербезпека
Освітня програма – Кібербезпека
інформаційних технологій та систем

_____ Анщук О. А.

Керівник: д.т.н., проф., проф. каф. МБІС
_____ Яремчук Ю. Є.

«6» червня 2024 р.

Рецензент: к.т.н., доц., доцент каф. ОТ
_____ Кожем'яко А. В.

«6» червня 2024 р.

Допущено до захисту
Голова секції УБ, кафедра МБІС
д.т.н., проф. Яремчук Ю. Є.
"6" червня 2024 р.

Вінниця ВНТУ – 2024

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

Ступінь – бакалавр
Спеціальність 125 – «Кібербезпека»

ЗАТВЕРДЖУЮ
Голова секції УБ кафедри МБІС
_____ **д.т.н., проф. Яремчук Ю. Є.**
«____» _____20__р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ
_____Анщуку Олександру Анатолійовичу_____
(прізвище, ім'я, по батькові)

1.Тема роботи: «Розробка захищеного вебсервісу для резервного копіювання даних у середовищі Amazon S3 з використанням наскрізного шифрування»
Керівник роботи: к.т.н., проф., проф. каф. МБІС Яремчук Ю. Є._____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ректора закладу вищої освіти від «22» березня 2024 року № 80
2. Строк подання студентом роботи за тиждень до захисту
3. Вихідні дані до роботи:
Метою роботи є розробка захищеного вебсервісу для резервного копіювання даних у середовищі Amazon S3 з використанням наскрізного шифрування для забезпечення конфіденційності та цілісності даних користувачів
4. Зміст текстової частини:
У першому розділі аналізуються існуючі рішення для резервного копіювання даних у хмарних середовищах, їхні переваги та недоліки, а також досліджуються методи захисту даних, зокрема наскрізне шифрування. У другому розділі розробляються архітектура та функціональні вимоги до захищеного вебсервісу для резервного копіювання з використанням наскрізного шифрування у середовищі Amazon S3. У третьому розділі здійснюється програмна реалізація вебсервісу, тестування та оцінка його безпеки. У висновках підбиваються результати роботи.
5. Перелік графічного матеріалу:
У першому розділі бакалаврської дипломної роботи наявно 5 рисунка, у другому розділі – 6 рисунків, у третьому розділі – 10 рисунків.

6. Консультанти розділів бакалаврської дипломної роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
I	Яремчук Ю. Є., к.т.н., проф., проф. каф. МБІС		
II	Яремчук Ю. Є., к.т.н., проф., проф. каф. МБІС		
III	Яремчук Ю. Є., к.т.н., проф., проф. каф. МБІС		

7. Дата видачі завдання «22» березня 2024р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів бакалаврської дипломної роботи		Примітка
		початок	закінчення	
1	Визначення напрямку бакалаврської роботи, формулювання теми	23.03.2024	25.03.2024	
2	Аналіз предметної області обраної теми	26.03.2024	01.04.2024	
3	Розробка алгоритму роботи	02.04.2024	16.04.2024	
4	Написання бакалаврської роботи на основі розробленої теми	17.04.2024	05.05.2024	
5	Передзахист бакалаврської дипломної роботи	06.05.2024	24.05.2024	
6	Виправлення, коригування бакалаврської дипломної роботи	25.05.2024	05.06.2024	
7	Захист бакалаврської дипломної роботи	12.06.2024	13.06.2024	

Студент _____ Аншук О. А.
(підпис) (прізвище та ініціали)

Керівник бакалаврської дипломної роботи _____ Яремчук Ю. Є.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

У даній бакалаврській дипломній роботі проаналізовано та здійснено програмну реалізацію захищеного вебсервісу для резервного копіювання даних у хмарному середовищі Amazon S3 з використанням наскрізного шифрування. Значну увагу приділено аналізу існуючих рішень для резервного копіювання, переваг та недоліків хмарного сховища Amazon S3, а також алгоритмів наскрізного шифрування даних. Описано процес проєктування архітектури вебсервісу та бази даних для управління резервними копіями. Представлено обґрунтування вибору мови програмування та середовища розробки, а також опис реалізації удосконаленого алгоритму наскрізного шифрування. Наведено результати тестування розробленого захищеного вебсервісу.

Ключові слова: резервне копіювання, хмарне сховище, наскрізне шифрування, веб-сервіс, захист даних, Amazon S3.

ABSTRACT

In this bachelor's thesis, the analysis and software implementation of a secure web service for data backup in the Amazon S3 cloud environment using end-to-end encryption is carried out. Considerable attention is paid to the analysis of existing backup solutions, the advantages and disadvantages of the Amazon S3 cloud storage, as well as end-to-end data encryption algorithms. The process of designing the web service architecture and the database for backup management is described. The justification for the choice of programming language and development environment, as well as a description of the implementation of the advanced end-to-end encryption algorithm, is presented. The results of testing the developed secure web service are provided.

Keywords: backup, cloud storage, end-to-end encryption, web service, data protection, Amazon S3.

ЗМІСТ

ВСТУП

Актуальність. У сучасному динамічному цифровому середовищі, де відбувається інтенсивний обмін та зберігання великих обсягів конфіденційних даних, питання захисту інформації та забезпечення її цілісності набуває надзвичайної актуальності. Зокрема, для багатьох організацій та приватних користувачів критично важливим є створення надійних резервних копій важливих даних з метою убезпечення від втрати або пошкодження інформації внаслідок збоїв обладнання, кібератак або людського фактору. Водночас, використання хмарних сховищ для зберігання резервних копій потребує особливої уваги до питань безпеки та конфіденційності, оскільки дані зберігаються на віддалених серверах, що не перебувають під повним контролем користувача.

Традиційні рішення для резервного копіювання часто не забезпечують належного рівня захисту від можливих витоків або несанкціонованого доступу до даних. Тому постає необхідність розробки вдосконалених підходів та засобів, які поєднують зручність хмарних сховищ з надійними механізмами захисту інформації. Одним із перспективних напрямків є впровадження методу наскрізного шифрування (end-to-end encryption) для резервних копій, розміщених у хмарному середовищі. Цей підхід гарантує, що дані залишаються захищеними навіть у разі компрометації хмарного сховища, оскільки доступ до них можливий лише з використанням закритого ключа, який зберігається виключно у користувача.

Актуальність теми посилюється постійним розвитком хмарних технологій та зростанням попиту на безпечні та надійні сервіси для резервного копіювання даних. Науковці та фахівці у сфері кібербезпеки активно працюють над вдосконаленням існуючих методів захисту інформації та розробкою нових підходів, які відповідатимуть сучасним викликам цифрової епохи.

Мета роботи полягає у розробці захищеного вебсервісу для резервного копіювання даних у хмарному середовищі Amazon S3 з використанням

наскрізного шифрування, що забезпечить високий рівень безпеки та конфіденційності зберігання інформації.

Поставлено та вирішено такі завдання для досягнення окресленої мети:

- аналіз існуючих рішень для захищеного резервного копіювання даних, визначення їх переваг та недоліків.
- аналіз алгоритмів наскрізного шифрування даних та вибір найбільш підходящого для захисту резервних копій.
- проєктування архітектури захищеного вебсервісу для резервного копіювання з використанням наскрізного шифрування в середовищі Amazon S3.
- розроблення модуля наскрізного шифрування для шифрування/розшифрування файлів резервних копій.
- реалізація модулів завантаження, зберігання, управління та взаємодії з резервними копіями в захищеному вебсервісі.
- тестування функціональності та безпеки розробленого захищеного вебсервісу.

Об'єктом дослідження є процес резервного копіювання даних у хмарному середовищі з використанням наскрізного шифрування.

Предметом дослідження є методи та засоби розробки захищеного вебсервісу для резервного копіювання даних у хмарі Amazon S3 з використанням наскрізного шифрування.

Практична цінність роботи полягає у можливості впровадження розробленого захищеного вебсервісу для забезпечення безпечного резервного копіювання конфіденційних даних організацій та приватних користувачів у хмарне середовище Amazon S3 з використанням наскрізного шифрування. Це дозволить значно підвищити рівень захисту та конфіденційності зберігання резервних копій у хмарі.

1 АНАЛІЗ ОСНОВНИХ СТРУКТУР ТА ФУНКЦІЙ ЗАХИЩЕНОГО ВЕБСЕРВІСУ ДЛЯ РЕЗЕРВНОГО КОПІЮВАННЯ ДАНИХ У СЕРЕДОВИЩІ AMAZON S3 З ВИКОРИСТАННЯМ НАСКРІЗНОГО ШИФРУВАННЯ

1.1. Аналіз даних, що потребують захищеного резервного копіювання

У сучасному цифровому світі, обсяги даних неухильно зростають, що ставить під загрозу їх збереження та конфіденційність. Особливу увагу заслуговують особисті дані, фінансова інформація, та ділові документи, які часто стають об'єктами кібератак та випадкових втрат. Важливість резервного копіювання цих типів даних не можна недооцінювати, оскільки вони є критично важливими для особистого, корпоративного та національного безпекового контексту.

Особисті дані включають широкий спектр інформації – від імені та адреси до більш чутливих даних, таких як медична історія та банківські реквізити. Значення захисту особистих даних висвітлено у дослідженнях, які підкреслюють ризики, пов'язані з витоком такої інформації, і необхідність її надійного резервного копіювання для запобігання таким інцидентам [1]. З іншого боку, фінансова інформація включає дані про банківські рахунки, кредитні картки, інвестиційні портфелі та інші відомості, які можуть бути використані для фінансових шахрайств або крадіжок. Важливість захисту фінансових даних особливо акцентується в контексті хмарних технологій та необхідності використання наскрізного шифрування для їх збереження [2].

Ділові документи, такі як договори, комерційні пропозиції, звіти та інші важливі корпоративні матеріали, є життєво необхідними для підтримки діяльності будь–якої організації. Втрата або несанкціонований доступ до таких документів може призвести до серйозних фінансових збитків і втрати репутації. Тому розробка ефективних методів резервного копіювання та впровадження стратегій відновлення після аварій є критично важливими для забезпечення стійкості бізнесу [3].

Розвиток технологій розподілу даних у хмарних сховищах відкриває нові можливості для резервного копіювання. Ці технології дозволяють не тільки ефективно зберігати великі обсяги даних, але й забезпечувати їх швидке відновлення у разі потреби. Водночас використання хмарних сервісів ставить під питання безпеку збереження даних і вимагає застосування складних механізмів шифрування та аутентифікації для захисту інформації від несанкціонованого доступу.

Процес резервного копіювання даних без використання шифрування, дані передаються на сховище даних (рис.1.1.)

Рисунок 1.1 – Процес резервного копіювання даних без використання шифрування [4]

Враховуючи зростання кіберзагроз і випадків витоку даних, важливість резервного копіювання та належного захисту особистих даних, фінансової інформації та ділових документів набуває нового звучання. Розробка комплексних рішень для резервного копіювання, які містять передові методи шифрування та аутентифікації, є ключовим кроком на шляху до забезпечення цифрової безпеки в епоху цифровізації.

Чутливість даних і потреба в їх захисті є критичним аспектом цифрової безпеки, що вимагає глибокого розуміння та відповідального підходу до управління інформацією. Різні типи даних мають різні рівні чутливості, які визначають обсяг заходів, необхідних для їх захисту. Визначення чутливості даних стає основоположним для розробки стратегій захисту, а також для вибору технологій резервного копіювання та шифрування.

Процес захищеного резервного копіювання даних з використанням шифрування, дані передаються зашифрованими на захищені сховища даних, позначені значками замків (рис.1.2.)

Рисунок 1.2 – Схема захищеного резервного копіювання даних з використанням шифрування [4]

Чутливість особистих даних може варіюватися від звичайної інформації, такої як ім’я та електронна пошта, до вкрай конфіденційної, включно з медичною інформацією або соціальними номерами. Втрата або несанкціонований доступ до таких даних може призвести не лише до порушення приватності, але й до фінансового шахрайства та ідентифікаційної крадіжки [5]. Це підкреслює необхідність розробки ефективних механізмів захисту, які б забезпечили цілісність та конфіденційність особистих даних на всіх етапах їх обробки та зберігання.

Фінансова інформація є ще одним типом даних з високим ступенем чутливості, оскільки її витік може призвести до значних фінансових втрат для осіб або організацій. Захист такої інформації вимагає використання складних шифрувальних алгоритмів та безпечних каналів передачі даних для запобігання їх перехопленню або модифікації зловмисниками [6]. Особливо це стосується хмарних сервісів, де дані зберігаються та обробляються за межами корпоративних мереж, що ставить під загрозу їх безпеку без належних заходів захисту.

Ділові документи, включаючи комерційні таємниці, стратегічні плани, та інші конфіденційні матеріали, вимагають особливого підходу до захисту через їх значення для бізнесу. Несанкціонований доступ до таких документів може підірвати конкурентоспроможність компанії та спричинити значні фінансові та репутаційні втрати [7]. Важливість захисту ділових документів не може бути

переоцінена, і це вимагає застосування всього спектра заходів безпеки, від фізичного захисту до шифрування даних і контролю доступу.

Захист чутливих даних у сучасному цифровому просторі – складне завдання, що вимагає від організацій впровадження комплексного підходу до інформаційної безпеки. Це включає не тільки використання передових технологій шифрування та резервного копіювання, але й розробку внутрішніх політик і процедур, навчання співробітників, та постійне оновлення заходів безпеки у відповідь на мінливі загрози [8]. Підвищення обізнаності про ступінь чутливості різних типів даних та належний захист є ключовим для запобігання втраті або компрометації важливої інформації.

Однак, з ростом обсягів даних зростає й кількість потенційних вразливостей та загроз безпеці. Це змушує організації та індивідів приділяти більше уваги заходам захисту даних, включаючи резервне копіювання та наскрізне шифрування, для забезпечення конфіденційності та цілісності інформації [9]. З огляду на це, інвестиції в заходи безпеки даних та розвиток новітніх технологій їх захисту стають пріоритетом для багатьох організацій.

Враховуючи зазначені тенденції, можна з упевненістю сказати, що обсяги даних продовжуватимуть рости, підвищуючи вимоги до систем зберігання, обробки та захисту даних. Це створює виклики як для корпоративного сектору, так і для особистого використання, водночас відкриваючи нові можливості для розвитку технологій та підходів до управління інформацією.

У сфері управління даними, забезпечення їх зберігання та доступності відіграє ключову роль для підтримки операційної ефективності та відповідності регуляторним вимогам. З огляду на зростаючі обсяги даних та збільшення кіберзагроз, різні країни та міжнародні організації розробили низку нормативних актів та стандартів, спрямованих на забезпечення безпечного зберігання та доступу до даних.

Одним з ключових міжнародних стандартів є ISO/IEC 27001, який встановлює вимоги до систем менеджменту інформаційної безпеки. Стандарт охоплює аспекти конфіденційності, цілісності та доступності інформації, вимагаючи від

організацій впровадження адекватних заходів захисту для забезпечення безпеки даних [10]. Зокрема, це стосується розробки та реалізації політик зберігання даних, які повинні враховувати терміни зберігання, методи шифрування та процедури доступу до інформації.

У Європейському Союзі загальний регламент захисту даних встановлює строгі вимоги до обробки персональних даних, включаючи їх зберігання та доступність. GDPR вимагає від організацій вжиття відповідних технічних та організаційних заходів для захисту даних від несанкціонованого доступу, втрати або знищення. Це включає необхідність резервного копіювання даних та впровадження ефективних процедур відновлення даних у випадку аварійних ситуацій [11].

У Сполучених Штатах Америки, Закон про переносимість та відповідальність у страхуванні здоров'я встановлює вимоги до захисту медичних записів та іншої охоронної інформації. HIPAA вимагає від охоронних провайдерів та їх бізнес–партнерів забезпечувати цілісність, конфіденційність та доступність всієї "захищеної охоронної інформації", включаючи специфічні вимоги до електронного зберігання та передачі таких даних [12].

В інших регіонах, таких як Азія та Тихоокеанський регіон, країни також розробляють власні законодавчі та нормативні рамки для регулювання зберігання та доступу до даних, враховуючи місцеві особливості та міжнародні зобов'язання. Це включає, наприклад, Закон про захист персональних даних Сінгапуру, який подібно до GDPR, встановлює вимоги до зберігання та обробки персональних даних [13].

Забезпечення відповідності до цих різноманітних нормативних вимог вимагає від організацій ретельного планування, реалізації та моніторингу інформаційних систем. Важливо враховувати не тільки технічні аспекти зберігання та доступу до даних, але і юридичні та етичні аспекти їх обробки. Такий підхід дозволить не лише забезпечити відповідність нормативним вимогам, але й підтримати довіру клієнтів та партнерів, забезпечуючи високий рівень захисту даних.

1.2. Аналіз існуючих засобів для резервного копіювання даних

У сучасному світі, де обсяги даних стрімко зростають, а потреба у їх захисті стає дедалі актуальнішою, ринок вебсервісів для резервного копіювання даних розвивається з величезною швидкістю. Популярність таких сервісів зростає завдяки їх здатності забезпечити надійне зберігання, швидке відновлення після збоїв та високий рівень безпеки.

Amazon Web Services S3 – це один з найбільш відомих та широко використовуваних сервісів для зберігання даних. S3 пропонує масштабоване зберігання в хмарі, що дозволяє користувачам зберігати та отримувати будь–яку кількість даних у будь–який час і з будь–якого місця по Інтернету. Особливістю AWS S3 є висока довговічність та доступність, підтримка різних рівнів доступу до даних та гнучкість у налаштуванні політик зберігання з метою оптимізації вартості [14].

Microsoft Azure Storage – це інтегроване хмарне рішення для зберігання даних, що підтримує зберігання об'єктів, файлів, дисків та черг. Azure Storage забезпечує міцність, високу доступність та безпечне зберігання даних. Його ключові переваги включають глобальну масштабованість, підтримку автоматичного шифрування даних в стані спокою та можливість вибору між різними рівнями доступу для оптимізації вартості та швидкості доступу [15].

Google Cloud Storage пропонує схожі можливості для зберігання та аналізу великих обсягів даних. Цей сервіс відрізняється широким спектром інструментів для аналізу та обробки даних, що робить його ідеальним для використання в біг дата проєктах. Google Cloud Storage забезпечує автоматичне шифрування даних, високу доступність та підтримку іммутабельності даних для захисту від випадкового або зловмисного видалення [16].

Backblaze B2 Cloud Storage є економічно ефективним рішенням для зберігання великих обсягів даних. Backblaze пропонує простоту використання, одну з найнижчих цін на ринку за зберігання та передачу даних, а також високу надійність. Сервіс підходить як для особистих потреб, так і для бізнесу,

пропонуючи прості API для інтеграції та можливість встановлення автоматизованих процесів резервного копіювання [17].

IBM Cloud Object Storage пропонує гнучкі та масштабовані рішення для зберігання даних, які забезпечують високу довговічність та доступність. Цей сервіс підходить для різноманітних сценаріїв використання, включаючи веб– та мобільні додатки, бекап, архівування та великі дата–сети. IBM Cloud Object Storage підтримує політики іммутабельності для захисту даних від змін, шифрування для забезпечення безпеки, та пропонує гнучкість у виборі рівнів зберігання для оптимізації витрат. Порівняння проаналізованих сервісів наведено у (табл. 1.1.) [18].

Таблиця 1.1 – Порівняння існуючих аналогів

Параметр	AWS S3	Microsoft Azure Storage	Google Cloud Storage	Backblaze B2 Cloud Storage	IBM Cloud Object Storage
Вартість	Від $0.023/ГБ/місяць	Від $0.0184/ГБ/місяць	Від $0.02/ГБ/місяць	Від $0.005/ГБ/місяць	Від $0.022/ГБ/місяць
Надійність	99.99% (11 дев'яток)	99.99% (12 дев'яток)	99.99% (11 дев'яток)	Висока, але без детальних гарантій	Висока, але без детальних гарантій
Безпека	Шифрування, IAM, політики S3	Шифрування, управління ключами, контроль доступу	Шифрування за замовчуванням, Google Cloud IAM	Шифрування, базовий контроль доступу	Шифрування, управління ключами, контроль доступу

При порівнянні вебсервісів для резервного копіювання даних, важливо враховувати низку факторів, включаючи функціонал, цінову політику, масштабованість, безпеку та спеціальні можливості. Оглянемо детальніше специфікації та можливості декількох популярних хмарних сховищ.

Web Services AWS S3 пропонує широкий спектр можливостей, включаючи необмежене зберігання даних, автоматизоване реплікування в різні регіони для

забезпечення високої доступності та довговічності, а також підтримку життєвого циклу даних для автоматичного управління зберіганням [19].

Вартість зберігання в S3 варіюється в залежності від обраного рівня (наприклад, S3 Standard, S3 Infrequent Access) та використаного обсягу даних. AWS пропонує детальний калькулятор вартості для оцінки витрат. S3 забезпечує сильне шифрування даних в стані спокою та під час передачі, а також розширені можливості управління доступом і аудиту.

Azure Storage пропонує зберігання об'єктів, файлів, дисків та черг. Він підтримує автоматичне шифрування, геореплікацію для високої доступності та інтеграцію з іншими службами Azure [20].

Цінова політика Azure Storage базується на використанні, з можливістю вибору між різними рівнями зберігання (наприклад, Hot, Cool, Archive) для оптимізації вартості. Microsoft також надає калькулятор вартості для планування бюджету.

Інші характеристики: Однією з ключових особливостей є тісна інтеграція з розробкою додатків та аналітикою в екосистемі Azure, що дозволяє легко розгортати та масштабувати застосунки.

Google Cloud Storage відомий своїми можливостями для обробки великих даних, підтримкою різноманітних типів даних та інтеграцією з аналітичними та машинними навчаннями Google Cloud [21].

Цінова модель Google Cloud Storage гнучка, з тарифами за використання зберігання, мережі та операцій. Користувачі можуть вибирати між стандартним зберіганням та холодним зберіганням для економії на рідко використовуваних даних.

GCS забезпечує потужні можливості шифрування та безпеки, включаючи керування ключами шифрування та докладне журналювання доступу до даних.

Backblaze B2 вирізняється своєю простотою використання та прямим підходом до зберігання даних, ідеально підходить для індивідуальних користувачів та малого бізнесу [22].

Вартість зберігання в Backblaze B2 одна з найнижчих на ринку, що робить його привабливим варіантом для користувачів, що шукають економічно вигідне рішення для великих обсягів даних [23].

Незважаючи на свою простоту, Backblaze B2 пропонує сильне шифрування даних та інтеграцію з різноманітними сторонніми додатками та сервісами для резервного копіювання та управління даними.

IBM Cloud Object Storage забезпечує високу довговічність та масштабованість для зберігання неструктурованих даних, підтримуючи різні сценарії використання, від цифрових медіа до бекапу.

IBM пропонує конкурентоспроможні ціни з можливістю оптимізації витрат через вибір різних рівнів зберігання та використання політик життєвого циклу даних.Сервіс надає розширені можливості безпеки, включаючи шифрування даних в стані спокою та під час передачі, а також гнучке управління доступом і політики безпеки [24].

Кожен із цих сервісів має свої унікальні переваги та може бути більш або менш відповідним залежно від специфічних потреб користувачів. При виборі вебсервісу для резервного копіювання важливо враховувати не тільки ціну та функціональні можливості, але й вимоги до безпеки, масштабованості та інтеграції з іншими інструментами та сервісами.

Незважаючи на різноманіття заходів безпеки, важливо розуміти, що жодна система не може гарантувати абсолютний захист від усіх потенційних загроз. Тому комплексний підхід до безпеки, який включає як технічні рішення, так і організаційні заходи, є ключовим для захисту цінної інформації в сучасному цифровому просторі. Користувачам та організаціям слід ретельно оцінювати свої потреби у безпеці даних та обирати вебсервіси для резервного копіювання, які найкраще відповідають цим вимогам, враховуючи доступний функціонал, цінову політику та специфіку власної діяльності.

1.3. Аналіз переваг та недоліків хмарного сховища Amazon S3

Масштабованість та гнучкість є ключовими характеристиками, які роблять Amazon S3 вибором номер один для багатьох компаній та індивідуальних користувачів, коли мова заходить про зберігання даних у хмарі. Amazon S3 є однією з найбільш популярних служб зберігання в хмарі, що пропонується Amazon Web Services. Однією з ключових переваг S3 є його здатність до безперебійного масштабування, що дозволяє користувачам збільшувати або зменшувати обсяг зберігання в залежності від їх поточних потреб без будь–яких перерв у роботі або потреби в ручному втручанні [25].

Ця гнучкість та масштабованість досягаються завдяки архітектурі, яка автоматично керує ресурсами, необхідними для зберігання та управління величезними масивами даних. Користувачі можуть легко завантажувати або видаляти дані, а S3 автоматично адаптується до змін у використанні, забезпечуючи високу доступність і довговічність даних. Таким чином, незалежно від того, чи потрібно зберігати кілька файлів, чи мільйони, S3 забезпечує необхідну інфраструктуру без додаткових витрат на підтримку та обслуговування з боку користувача.

Крім того, Amazon S3 пропонує різноманітні класи зберігання, що дозволяють користувачам оптимізувати вартість зберігання в залежності від частоти доступу до даних та вимог до їх довговічності. Від класів для часто використовуваних даних, що потребують швидкого доступу, до архівних класів зберігання для даних, які рідко використовуються, S3 забезпечує гнучкість управління витратами без компромісів у питаннях доступності та безпеки [26].

Однією з найважливіших характеристик S3 є його здатність до інтеграції з широким спектром інших служб AWS, таких як Amazon CloudFront для доставки контенту, AWS Lambda для виконання коду без провадження серверів, а також різноманітних інструментів аналітики та машинного навчання. На рисунку 1.3 зображено високодоступну та відмовостійку архітектуру для розгортання веб–додатків на хмарній платформі Amazon Web Services AWS.

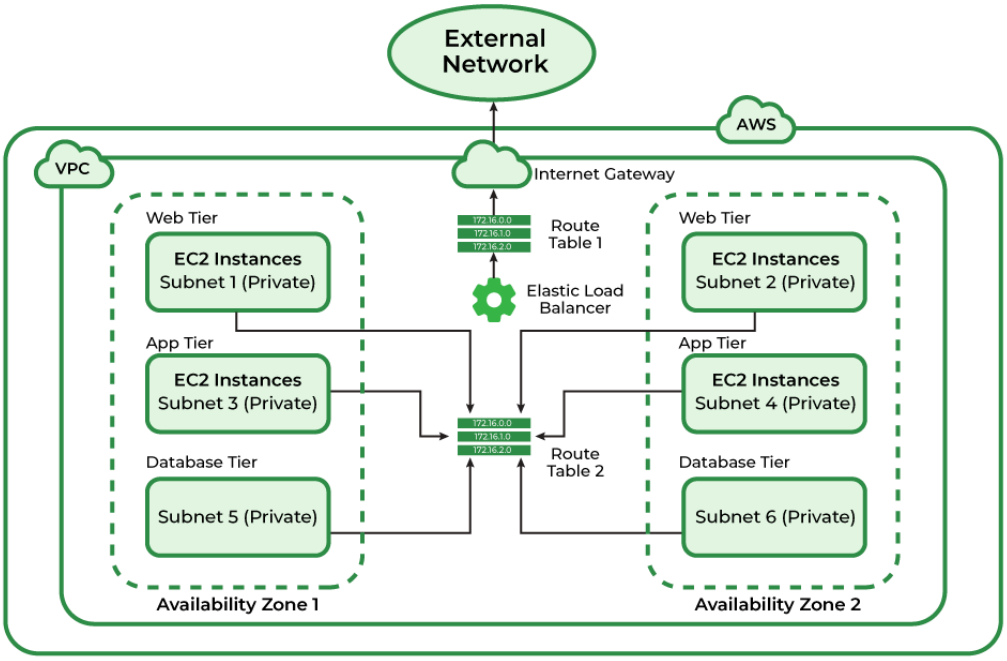


Рисунок 1.3 – Вигляд архітектури для розгортання веб–додатків на хмарній платформі Amazon Web Services AWS [27]

Така інфраструктура, яка поєднує в собі гнучкість, масштабованість і глибоку інтеграцію з іншими хмарними сервісами, робить Amazon S3 ідеальним рішенням для компаній будь–якого розміру. Від стартапів до глобальних корпорацій, S3 надає потужні та економічно ефективні можливості для зберігання даних, забезпечуючи, що інформація залишається доступною та безпечною на будь–якому етапі її життєвого циклу. Впровадження таких інноваційних технологій дозволяє компаніям не тільки знижувати витрати на IT–інфраструктуру, але й прискорювати впровадження нових продуктів та послуг на ринок, забезпечуючи стійке зростання та конкурентоспроможність в довгостроковій перспективі [28].

Amazon Web Services встановлює високі стандарти у забезпеченні доступності та надійності для своїх сервісів зберігання даних, зокрема Amazon S3. Компанія пропонує розширені гарантії щодо доступності та збереження даних, що підкреслює її відданість забезпеченню безперебійної роботи бізнесу своїх клієнтів.

Amazon S3 забезпечує довговічність об'єктів на рівні 99.9%, що означає, що на мільйон файлів, збережених на S3, втрата даних може теоретично статися лише в одному випадку. Така висока довговічність досягається завдяки автоматичному реплікуванню даних на кілька пристроїв у різних центрах обробки даних [29]. Це забезпечує високий рівень захисту від втрати даних через апаратні збої, природні катастрофи та інші непередбачувані події.

Щодо доступності, Amazon S3 пропонує різні рівні сервісу в залежності від вибраного класу зберігання. Наприклад, S3 Standard гарантує доступність даних на рівні 99.99%, тоді як S3 Standard–Infrequent Access і S3 One Zone–Infrequent Access пропонують трохи нижчу гарантію доступності, але за нижчу ціну. Це дозволяє клієнтам вибирати оптимальний баланс між вартістю та потребами доступності для різних типів даних [30].

Для підвищення надійності та доступності, AWS також надає інструменти та сервіси для моніторингу стану зберігання даних та оповіщення про будь–які питання, що можуть виникнути. Клієнти можуть використовувати Amazon CloudWatch та AWS S3 власні метрики для відстеження доступності та вчасного реагування на зміни в роботі сервісу. Крім того, з S3 Cross–Region Replication можна автоматично реплікувати дані в інші регіони, щоб забезпечити додатковий рівень відмовостійкості та доступності для критично важливих даних [31].

Amazon неухильно працює над покращенням своїх технологій та процесів, щоб максимізувати доступність та надійність своїх сервісів. Це підкріплюється не тільки передовими технологічними рішеннями, але й строгими політиками безпеки та великим досвідом у забезпеченні високої продуктивності хмарних сервісів. Ось чому багато компаній світу довіряють AWS свої найважливіші дані, знаючи, що вони будуть доступні та захищені навіть у найважчих умовах.

1.4. Аналіз алгоритмів наскрізного шифрування даних

Наскрізне шифрування даних є ключовою технологією у сучасних системах захисту інформації, забезпечуючи конфіденційність даних від моменту їх створення до моменту отримання уповноваженою особою. Особливо актуальним

наскрізне шифрування стає у контексті резервного копіювання даних у хмарних сервісах, таких як Amazon S3, де гарантування безпеки інформації відіграє вирішальну роль. Визначення наскрізного шифрування полягає в тому, що дані зашифровуються на стороні клієнта та розшифровуються лише після досягнення кінцевого пункту призначення, недоступні для перегляду або зміни третіми особами під час транзиту чи зберігання на сервері (рис. 1.4) [32].

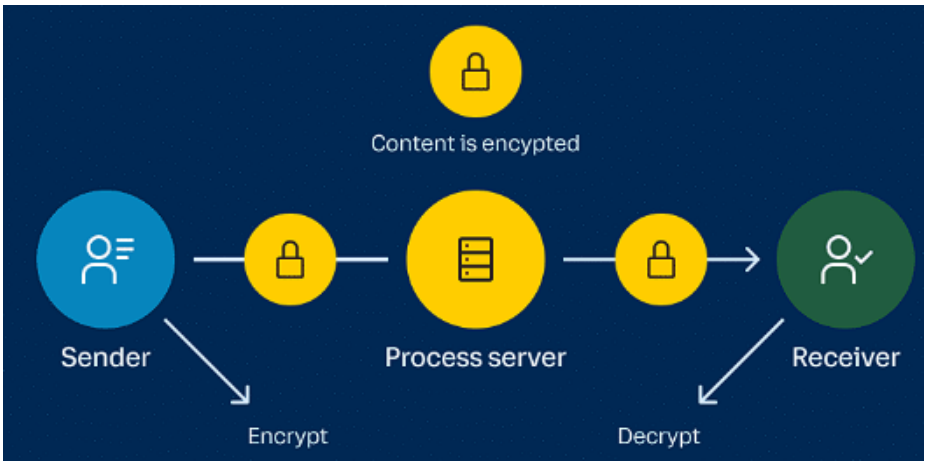


Рисунок 1.4 – Схематична схема роботи наскрізного шифрування

Цей метод шифрування має низку переваг, головною з яких є високий рівень захисту даних. Через те, що ключі шифрування знаходяться лише у власника даних, навіть у разі несанкціонованого доступу до хмарного сховища, інформація залишається захищеною. Це критично важливо для бізнесу та індивідуальних користувачів, які прагнуть забезпечити конфіденційність своєї інформації.

Існує кілька алгоритмів наскрізного шифрування, які використовуються для захисту даних. AES є одним з найбільш поширених та надійних алгоритмів, що забезпечує високий рівень безпеки [33]. RSA є іншим важливим алгоритмом, який зазвичай використовується для шифрування ключів шифрування або цифрових підписів, що дозволяє безпечний обмін ключами між відправником та отримувачем.

Впровадження наскрізного шифрування у системи резервного копіювання даних вимагає уважного планування та виконання. Необхідно забезпечити, щоб усі компоненти системи, від клієнтських додатків до хмарних сховищ,

підтримували використання вибраних алгоритмів шифрування. Особливу увагу слід приділити управлінню ключами шифрування, оскільки безпека всієї системи залежить від зберігання та обігу цих ключів [34].

Однак, попри всі переваги, наскрізне шифрування має і свої виклики. Одним з них є складність управління ключами шифрування, особливо у великих організаціях, де необхідно забезпечити безпечний обмін ключами між великою кількістю користувачів. Іншим важливим аспектом є потреба у забезпеченні сумісності між різними системами та пристроями, що може стати складним завданням в умовах постійно змінюваного ландшафту інформаційних технологій.

Незважаючи на ці виклики, наскрізне шифрування залишається одним з найефективніших способів захисту конфіденційності даних у хмарних сховищах, таких як Amazon S3 [35]. З його допомогою користувачі можуть бути впевнені, що їх дані залишатимуться захищеними від несанкціонованого доступу, забезпечуючи високий рівень безпеки та конфіденційності.

Серед алгоритмів наскрізного шифрування, Advanced Encryption Standard та Rivest–Shamir–Adleman вважаються двома з найбільш розповсюджених та надійних методів захисту даних.

AES є симетричним блочним шифром, що означає, що він використовує один і той же ключ для шифрування та розшифрування даних. Цей алгоритм став стандартом шифрування, прийнятим урядом США та широко використовується у всьому світі для захисту конфіденційної інформації. AES дозволяє використовувати ключі різної довжини – 128, 192, або 256 біт, забезпечуючи гнучкий баланс між швидкістю шифрування та рівнем безпеки. Вибір довжини ключа залежить від потрібних вимог до безпеки та потенційного ризику атаки. AES відомий своєю ефективністю як в апаратному, так і в програмному забезпеченні, забезпечуючи швидке шифрування даних без значного навантаження на ресурси системи [36].

RSA, у свою чергу, є асиметричним шифром, що використовує пару ключів: один для шифрування (публічний ключ) та інший для розшифрування (приватний ключ). Цей метод дозволяє безпечно обмінюватися шифрованими даними навіть у

тому випадку, якщо публічний ключ став відомим широкому колу осіб. RSA особливо корисний для захисту даних при передачі через небезпечні мережі, оскільки навіть якщо публічний ключ буде перехоплено, без приватного ключа розшифрувати дані буде неможливо. Основним недоліком RSA є його порівняно низька швидкість шифрування порівняно з симетричними шифрами, що робить його менш відповідним для шифрування великих об'ємів даних. Натомість, RSA часто використовується для шифрування ключів симетричного шифрування або для цифрового підпису, забезпечуючи безпечний обмін ключами або підтвердження автентичності .

Обидва ці алгоритми є важливими інструментами у комплексному підході до захисту інформації, дозволяючи забезпечити конфіденційність та цілісність даних у різноманітних середовищах, включаючи хмарні сховища та мережеві комунікації. Використання наскрізного шифрування з цими алгоритмами дозволяє створити надійний захист від несанкціонованого доступу, забезпечуючи, що лише уповноважені особи матимуть доступ до конфіденційної інформації.

Імплементація наскрізного шифрування у розробці вебсервісів вимагає розуміння технічних аспектів та потенційних викликів, які можуть виникнути під час реалізації цієї технології. Наскрізне шифрування дозволяє забезпечити високий рівень безпеки даних, шифруючи інформацію на стороні відправника і розшифровуючи її лише на стороні отримувача, не дозволяючи третім особам, включно з провайдерами хмарних сервісів, доступ до змісту повідомлень (1.5) [37].

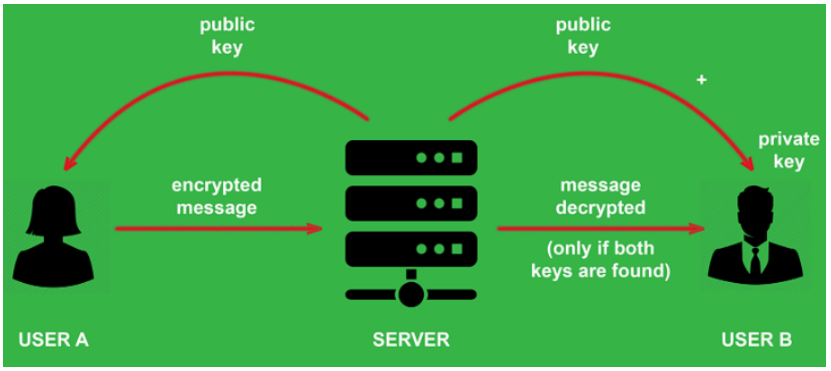


Рисунок 1.5 – Взаємодія наскрізного шифрування із сервером вебсервісу

Вибір алгоритму шифрування є першим кроком у реалізації E2EE. Серед найбільш використовуваних алгоритмів можна виділити AES для симетричного шифрування та RSA або ECC для асиметричного шифрування. Важливо вибрати алгоритм, який забезпечує оптимальний баланс між безпекою та ефективністю для конкретної системи.

Управління ключами є другим значним аспектом. Важливо забезпечити безпечне зберігання та обмін ключами шифрування. Використання асиметричного шифрування для обміну симетричними ключами може бути ефективним рішенням, яке дозволяє уникнути ризиків, пов’язаних із передачею ключів шифрування по незахищених каналах.

Архітектура системи повинна бути спроєктована таким чином, щоб забезпечити наскрізне шифрування на всіх етапах передачі та зберігання даних. Це означає, що дані повинні залишатися зашифрованими на серверах та під час транзиту, з виключенням кінцевих точок, де відбувається шифрування та розшифрування.

Реалізація криптографічних протоколів на вебсерверах та клієнтах вимагає акуратного програмування та тестування. Важливо використовувати перевірені бібліотеки та фреймворки, які проходять регулярні оновлення та аудити безпеки.

Розробка інтерфейсу користувача також відіграє роль у впровадженні наскрізного шифрування. Користувачі повинні бути інформовані про статус шифрування та мати можливість легко управляти ключами шифрування, не порушуючи при цьому загальних принципів безпеки.

Складність управління ключами є одним з основних викликів, з якими стикаються розробники при імплементації E2EE. Забезпечення безпеки приватних ключів, при цьому зберігаючи їх доступними для легітимних користувачів, вимагає комплексного підходу та може збільшити складність системи [38].

Імплементація наскрізного шифрування у вебсервісах є складним, але важливим завданням, яке може значно підвищити рівень безпеки даних. Попри існуючі виклики та обмеження, ефективне впровадження E2EE дозволяє забезпечити конфіденційність інформації, захищаючи її від несанкціонованого

доступу та витоків. Врахування цих аспектів на етапі проектування та розробки є ключовим для створення надійних та безпечних вебсервісів.

1.5. Аналіз основних методів захисту інформації у вебсервісах для резервного копіювання.

Шифрування є одним з найважливіших методів захисту інформації у вебдодатках, особливо для сервісів резервного копіювання даних. Воно передбачає процес перетворення зрозумілої інформації (відкритого тексту) в зашифрований текст, який неможливо прочитати без відповідного ключа. Це робить дані недоступними для зловмисників навіть у випадку їх перехоплення або крадіжки.

Основні поняття шифрування:

- відкритий текст – це початкова, незашифрована інформація, яку треба захистити;
- зашифрований текст – це результат шифрування, який є незрозумілим без спеціального ключа;
- ключ шифрування – це псевдовипадковий набір символів, який використовується для шифрування та розшифрування даних;

Методи шифрування.

- симетричне шифрування – використовує один і той самий ключ для шифрування та розшифрування даних. Прикладом є алгоритм AES (Advanced Encryption Standard). Симетричне шифрування ефективне і швидке, але потребує безпечного способу передачі ключа між відправником і одержувачем [39];
- асиметричне шифрування – використовує пару ключів – відкритий (публічний) і закритий (приватний). Відкритий ключ використовується для шифрування, а закритий – для розшифрування. Прикладом є RSA

(Rivest–Shamir–Adleman). Цей метод забезпечує високий рівень безпеки, оскільки приватний ключ не передається і залишається у власника.

Використання шифрування у вебдодатках

− шифрування даних у стані спокою (at rest): Забезпечує захист даних, які зберігаються на дисках або в базах даних. Наприклад, резервні копії баз даних можуть бути зашифровані перед зберіганням на сервері;

− шифрування даних під час передачі (in transit): Захищає дані, які передаються між клієнтом і сервером через мережу. Протоколи HTTPS та TLS (Transport Layer Security) забезпечують шифрування даних під час передачі, захищаючи їх від перехоплення;

− шифрування даних під час обробки (in use): Хоча це найскладніший аспект, деякі сучасні технології дозволяють обробку зашифрованих даних без їх розшифрування, використовуючи методи гомоморфного шифрування.

Надійність шифрування залежить від довжини ключа: чим довший ключ, тим складніше його зламати. Наприклад, 128–бітний ключ AES вважається дуже надійним, а 256–бітний ключ ще більш безпечний. Вибір довжини ключа залежить від необхідного рівня безпеки та ресурсів для обробки [40].

Для вебсервісів резервного копіювання шифрування є критичним компонентом захисту даних, воно забезпечує:

− конфіденційність – зашифровані дані залишаються недоступними для несанкціонованих осіб;

− цілісність – шифрування допомагає запобігти несанкціонованим змінам даних;

− доступність – навіть у разі компрометації серверів, зашифровані резервні копії залишаються безпечними.

Аутентифікація та авторизація є іншими ключовими компонентами в системах безпеки інформаційних технологій, зокрема у вебдодатках та сервісах. Аутентифікація – це процес перевірки ідентичності користувача, тоді як авторизація визначає, які ресурси або дії доступні користувачу після успішної аутентифікації.

Аутентифікація здійснюється за допомогою одного або декількох факторів, які можна класифікувати як щось, що відомо користувачу (наприклад, пароль або PIN–код), щось, що володіє користувач (наприклад, ID–картка або мобільний телефон), або щось, що характеризує користувача (наприклад, біометричні дані). Використання декількох факторів, відоме як багатофакторна аутентифікація, значно підвищує рівень безпеки, оскільки воно мінімізує ризики, пов’язані з компрометацією одного з методів аутентифікації (рис.1.6) [41].

Авторизація, яка слідує за аутентифікацією, визначає рівень доступу користувача до ресурсів та операцій. Вона залежить від політик безпеки системи та може базуватися на ролях користувача, атрибутах або конкретних правилах, встановлених адміністратором системи. Ефективне управління доступом дозволяє обмежити можливості користувачів до мінімуму необхідного для виконання їхніх завдань, що зменшує ризики несанкціонованого доступу чи зловмисних дій.

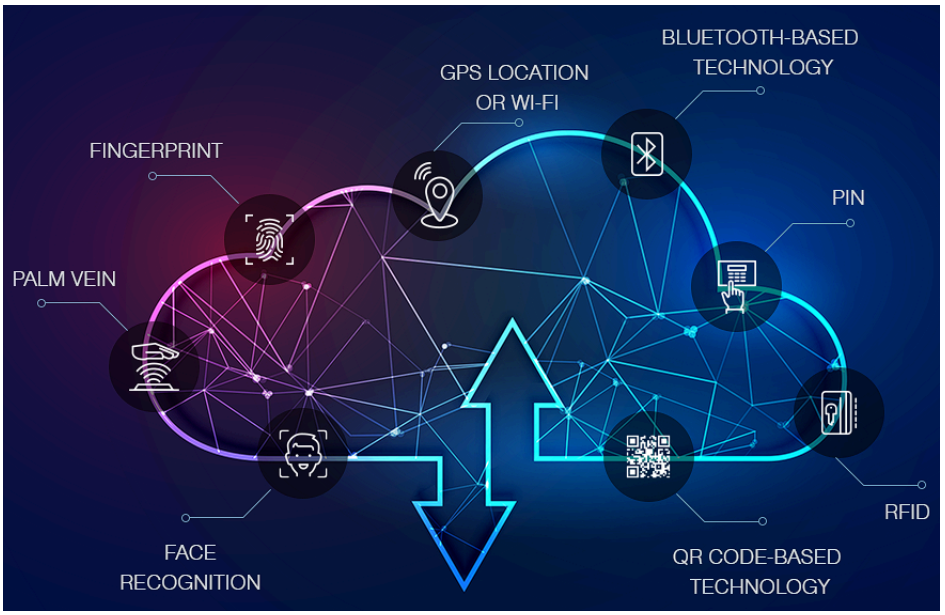


Рисунок 1.5 – Методи аутентифікації із сервером вебсервісу [41]

Серед методів аутентифікації, парольна аутентифікація залишається найпоширенішою, хоча вона таїть ризики, пов’язані з вибором слабких паролів або їх компрометацією. Системи, що використовують одноразові паролі або двофакторну аутентифікацію , де, наприклад, пароль доповнюється кодом, отриманим через SMS або мобільний додаток, надають вищий рівень безпеки.

Біометрична аутентифікація, яка використовує унікальні фізичні характеристики особи, такі як відбитки пальців, розпізнавання обличчя або голосу, стає все більш популярною завдяки своїй здатності надавати швидкий та зручний доступ до систем без необхідності запам'ятовування паролів [42].

Токени безпеки та смарт–карти, які володіє користувач, можуть використовуватися як незалежний метод аутентифікації або як частина MFA. Ці пристрої генерують коди або використовують цифрові сертифікати для перевірки ідентичності користувача.

Авторизація на основі ролей дозволяє призначати користувачам певні ролі в системі, кожна з яких має визначений набір прав доступу. Цей метод спрощує управління доступом, особливо в великих організаціях з різноманітними користувачами та ролями.

Авторизація на основі атрибутів враховує додаткові атрибути, такі як час доби або місцезнаходження, дозволяючи створювати більш гнучкі правила доступу, які можуть адаптуватися до змінюваних умов або загроз.

Управління ідентифікаційними даними та доступом об'єднує методи аутентифікації та авторизації в єдину систему, що забезпечує централізоване керування ідентичностями користувачів та їхніми правами доступу. Це дозволяє компаніям ефективно впроваджувати політики безпеки, відстежувати діяльність користувачів та реагувати на інциденти безпеки.

Ефективне застосування методів аутентифікації та авторизації є критично важливим для захисту інформаційних систем від несанкціонованого доступу та зловмисних дій. Основою успішної реалізації є збалансований підхід, що враховує потреби безпеки, зручність користувачів та вимоги бізнесу.

1.6 Висновок до розділу та постановка задачі

Отже, в цьому розділі розглянуто суть сервісів для створення резервних копій даних, проаналізовано методи захисту файлів та проведено огляд існуючих аналогічних рішень з порівнянням їх переваг і недоліків. Виходячи з цього огляду, визначено наступні завдання:

- розробити алгоритм роботи програмного застосунку;
- здійснити програмну реалізацію застосунку;
- оцінити ефективність роботи розробленої програми.

Після виконання цих завдань, буде досягнуто мету цієї дипломної роботи: розроблено вебсервісу для створення резервних копій даних з використанням наскрізного шифрування для зберігання у хмарному сховищі Amazon S3.

2 ПРОЄКТУВАННЯ ЗАХИЩЕНОГО ВЕБСЕРВІСУ ДЛЯ РЕЗЕРВНОГО КОПІЮВАННЯ ДАНИХ У СЕРЕДОВИЩІ AMAZON S3 З ВИКОРИСТАННЯМ НАСКРІЗНОГО ШИФРУВАННЯ

2.1. Розроблення підходу резервного копіювання даних з використанням наскрізного шифрування в середовищі Amazon S3

Після ґрунтовного аналізу існуючих підходів та методів захисту вебсервісів для резервного копіювання даних, проведеного в першому розділі, стало очевидним, що для забезпечення надійного захисту даних при їх передачі та зберіганні в хмарному середовищі Amazon S3 необхідно вдосконалити поточні рішення. Особливу увагу слід приділити використанню наскрізного шифрування, яке дозволяє забезпечити конфіденційність та цілісність даних на всіх етапах їх життєвого циклу.

Удосконалення підходу резервного копіювання даних з використанням наскрізного шифрування в середовищі Amazon S3 є критично важливим з кількох причин. По–перше, існуючі методи часто покладаються на шифрування даних лише під час їх передачі або зберігання, залишаючи їх вразливими в інших точках обробки. По–друге, традиційні підходи можуть не забезпечувати достатнього рівня гранулярності контролю доступу та управління ключами шифрування, що збільшує ризики несанкціонованого доступу до даних.

Запропонований удосконалений підхід передбачає використання гібридної схеми шифрування, яка поєднує переваги асиметричного шифрування (RSA) для безпечного обміну ключами та симетричного шифрування (AES) для ефективного шифрування самих даних. Цей підхід дозволяє забезпечити наскрізне шифрування даних, при якому вони залишаються захищеними протягом усього життєвого циклу – від моменту створення до зберігання в Amazon S3 та подальшого використання. Крім того, вдосконалений метод передбачає використання токенізації метаданих файлів, що дозволяє здійснювати пошук та управління файлами без розкриття їх конфіденційного вмісту.

На рисунку 2.1 зображена блок–схема, що відображає основні кроки роботи алгоритму захисту вебсервісу гібридним наскрізним шифруванням, включаючи реєстрацію користувача, генерацію ключів RSA, шифрування файлів ключами AES, перетворення метаданих на токени для пошуку, поширення посилань на файли та процес завантаження файлів за посиланням.

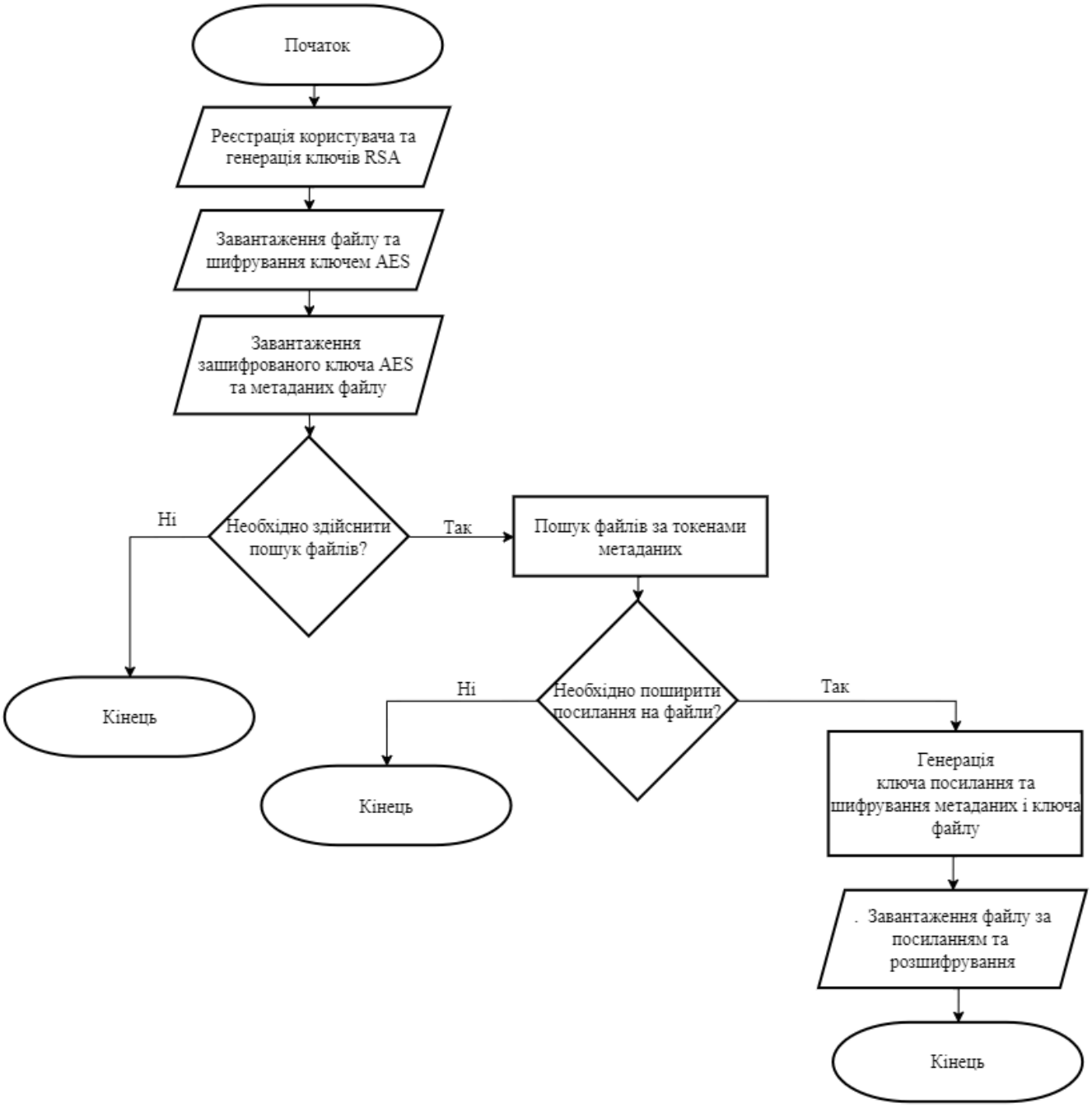


Рисунок 2.1 – Блок–схема розробленого алгоритму захисту вебсервісу з використанням наскрізного шифрування в середовищі Amazon S3.

Схема демонструє використання різних типів шифрування на різних етапах процесу для забезпечення безпеки та ефективності. Далі описано алгоритм реалізації вдосконаленого підходу резервного копіювання даних з використанням наскрізного шифрування в середовищі Amazon S3.

Крок 1. Початок.

Це перший крок алгоритму гібридного наскрізного шифрування. На цьому етапі відбувається ініціалізація системи та підготовка до виконання наступних кроків. Всі необхідні ресурси та компоненти системи перевіряються та готуються до роботи.

Крок 2. Реєстрація користувача та генерація ключів RSA.

На цьому етапі відбувається створення облікового запису користувача та генерація пари ключів RSA (відкритого та приватного) для забезпечення безпечної передачі даних. На цьому етапі відбувається створення облікового запису користувача в системі. Користувач надає необхідну інформацію для реєстрації, таку як ім'я, адреса електронної пошти та пароль. Після успішної реєстрації система генерує пару ключів RSA (відкритий та приватний) для забезпечення безпечної передачі даних. Ці ключі будуть використовуватися для шифрування та розшифрування ключів AES, які, в свою чергу, використовуються для шифрування файлів користувача.

Крок 3. Завантаження файлу та шифрування ключем AES.

Користувач вибирає файл, який він хоче завантажити та зберегти в системі. Після вибору файлу система генерує випадковий ключ AES для шифрування цього файлу. AES є симетричним алгоритмом шифрування, який забезпечує високий рівень безпеки та швидкість шифрування. Згенерований ключ AES використовується для шифрування вмісту файлу, що гарантує його конфіденційність під час зберігання та передачі.

Крок 4. Зберігання зашифрованого ключа AES та метаданих файлу. Зашифрований ключ AES зберігається разом з метаданими файлу (наприклад, ім'я файлу, дата завантаження тощо) в базі даних системи.

Після шифрування файлу ключем AES, цей ключ також потребує захисту. Для цього система шифрує ключ AES за допомогою відкритого ключа RSA користувача, який був згенерований під час реєстрації. Зашифрований ключ AES зберігається разом з метаданими файлу (наприклад, ім'я файлу, дата завантаження, розмір файлу тощо) в базі даних системи. Це забезпечує безпечне зберігання ключа AES та дозволяє системі пов'язувати кожен файл з відповідним ключем для подальшого розшифрування.

Крок 5. Перевірка необхідності здійснення пошуку файлів.

Система перевіряє, чи користувач має потребу в пошуку конкретного файлу серед своїх збережених файлів. Якщо користувач хоче знайти певний файл, алгоритм переходить до кроку 6, де буде виконано пошук файлів за токенами метаданих. В іншому випадку, якщо користувач не потребує пошуку файлів, алгоритм переходить до кроку 7, де перевіряється необхідність поширення посилання на файли.

Крок 6. Пошук файлів за токенами метаданих.

Якщо користувач потребує знайти конкретний файл, система здійснює пошук файлів на основі токенів метаданих. Токени метаданих – це спеціальні ідентифікатори, які створюються шляхом перетворення метаданих файлу (наприклад, імені файлу, дати завантаження тощо) за допомогою певних алгоритмів. Ці токени зберігаються разом із зашифрованими файлами в базі даних системи. Під час пошуку система порівнює токени метаданих з пошуковим запитом користувача та знаходить відповідні файли. Після знаходження потрібних файлів алгоритм переходить до кроку 7.

Крок 7. Перевірка необхідності поширення посилання на файли.

Система перевіряє, чи користувач бажає поділитися якимось файлом з іншими користувачами. Якщо користувач хоче надати доступ до файлу іншим особам, алгоритм переходить до кроку 8, де буде згенеровано спеціальне посилання для доступу до файлу. В іншому випадку, якщо користувач не потребує поширення посилання на файли, алгоритм переходить до кроку 10, що є кінцем алгоритму гібридного наскрізного шифрування.

Крок 8. Генерація ключа посилання та шифрування метаданих і ключа файлу.

Якщо користувач бажає поділитися файлом з іншими, система генерує унікальний ключ посилання. Цей ключ посилання використовується для додаткового шифрування метаданих файлу та ключа AES, яким був зашифрований сам файл. Таким чином, метадані файлу та ключ AES додатково захищаються за допомогою ключа посилання. Ключ посилання також шифрується за допомогою відкритого ключа RSA власника файлу, щоб гарантувати, що тільки власник зможе розшифрувати та отримати доступ до ключа посилання в разі потреби.

Крок 9. Завантаження файлу за посиланням та розшифрування.

Коли інший користувач, який отримав посилання на файл, робить запит на завантаження цього файлу, система перевіряє його право доступу за допомогою ключа посилання. Якщо ключ посилання дійсний, система розшифровує метадані файлу та ключ AES за допомогою ключа посилання. Потім, використовуючи розшифрований ключ AES, система розшифровує сам файл і надає його користувачу для завантаження. Весь процес відбувається в пам'яті, щоб уникнути зберігання незашифрованих даних на диску.

Крок 10. Кінець алгоритму гібридного наскрізного шифрування.

Це останній крок алгоритму. Після успішного завершення всіх попередніх кроків, система завершує роботу алгоритму гібридного наскрізного шифрування. Всі ресурси та з'єднання, які були задіяні під час роботи алгоритму, коректно звільняються та закриваються. Система готова до наступного циклу роботи з новими запитами від користувачів.

Впровадження запропонованого підходу з використанням наскрізного шифрування та токенізації метаданих дозволить значно підвищити рівень захисту даних, забезпечити їх конфіденційність, цілісність та доступність лише для авторизованих користувачів. Це, в свою чергу, зміцнить довіру користувачів до вебсервісу резервного копіювання та дозволить безпечно зберігати та обробляти чутливі дані в хмарному середовищі Amazon S3.

2.2 Проєктування архітектури вебсервісу для безпечного резервного копіювання даних з використанням наскрізного шифрування у хмарному середовищі Amazon S3

Архітектура вебсервісу визначена загальною структурою та організацію програмного забезпечення, зокрема, розподіл функціональності між різними компонентами системи, взаємозв'язки між цими компонентами, а також засоби взаємодії між ними. Вона включає у себе не лише технічні аспекти, але й концептуальні та стратегічні рішення, спрямовані на досягнення поставлених цілей та вимог щодо якості програмного продукту. Архітектура архітектури вебсервісу складається з проміжного серверу, API та файлового сховища.

Взаємодія між цими компонентами забезпечується за допомогою протоколів TCP/IP на рівні мережі та HTTP на рівні застосунків.

Проміжний сервер містить серверний додаток, який з'єднує клієнтську програму та інші компоненти архітектури. Додаток надає API для клієнтської програми та містить клієнтські модулі для надсилання запитів до API сховища та системи пошуку. Він також надає функціонал для адміністрування системи, реєстрації та авторизації користувачів та авторизації користувачів, а також створення та зберігання файлів.

Файлове сховище – це кластер, на якому розгорнуто об'єктне сховище. Воно надає API для додавання, модифікації та видалення об'єктів, а також адміністрування системи [42].

Проєктування механізмів аутентифікації та авторизації користувачів у вебсервісі для безпечного резервного копіювання даних.

На основі інформації отриманої в попередніх розділах, буде розроблено механізми для реєстрації, аутентифікації та авторизації користувача у вебсервісі для резервного копіювання даних у сховище Amazon S3. Далі розглянуто покроково алгоритми підтвердження особи та завантаження файлів на вебсервіс.

Крок 1. Початок реєстрації.

Користувач переходить на сторінку реєстрації вебсервісу. Система відображає форму реєстрації з полями для введення необхідних даних.

Крок 2. Введення логіну та пароля.

Користувач вводить бажаний логін у відповідне поле форми. Користувач створює надійний пароль та вводить його у відповідне поле форми. Система перевіряє відповідність введених даних встановленим вимогам безпеки.

Крок 3. Перевірка унікальності логіну.

Система надсилає запит до бази даних, щоб перевірити, чи не використовується введений логін іншим користувачем.

Крок 4. Перевірка надійності пароля.

Система перевіряє, чи відповідає введений пароль встановленим правилам безпеки (мінімальна довжина, наявність різних типів символів тощо).

Крок 5. Генерація пари RSA–ключів.

Після успішної перевірки логіну та пароля, система автоматично генерує пару RSA–ключів (відкритий та закритий) для користувача. Відкритий ключ зберігається в базі даних системи для шифрування даних, що надсилаються користувачу. Закритий ключ зберігається лише на пристрої користувача для розшифровки отриманих даних.

Крок 6. Збереження даних користувача.

Система зберігає введені користувачем дані (логін, хешований пароль) та згенеровані RSA–ключі у базі даних. Додаткова інформація, така як електронна пошта або номер телефону, також може бути збережена для подальшої комунікації з користувачем.

Крок 7. Реєстрація завершена.

Система відображає повідомлення про успішну реєстрацію та вітає користувача. Після успішного завершення реєстрації, система надає користувачу повний доступ до всіх функцій та можливостей вебсервісу. Схема роботи алгоритму реєстрації користувача відображено на (рис. 2.2).

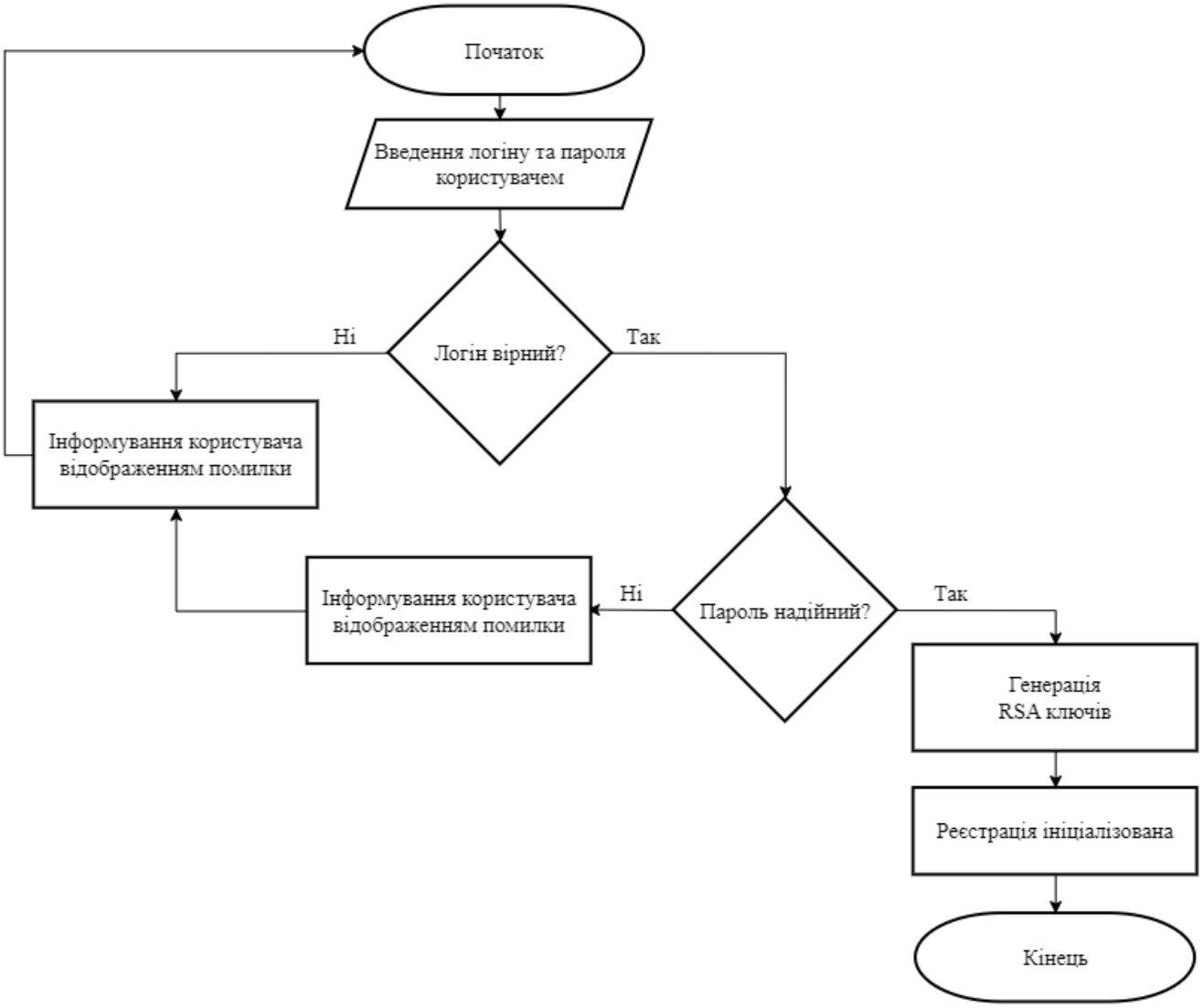


Рисунок 2.3 – Схема алгоритму реєстрації користувача на розробленому захищеному вебсервісі

Цей алгоритм реєстрації користувача забезпечує безпечне та зручне створення облікового запису на вебсервісі. Він враховує важливі аспекти, такі як перевірка унікальності логіну, надійність пароля та автоматична генерація RSA–ключів для захисту даних користувача. Завдяки цьому алгоритму, вебсервіс гарантує конфіденційність та безпеку інформації користувачів від самого початку їхньої взаємодії з системою.

Для забезпечення додаткового рівня безпеки, розроблено окремий алгоритм авторизації, який детально ілюструє процес перевірки введеного логіну та паролю

на наступному етапі. Схема роботи алгоритму авторизації користувача відображено на (рис. 2.4).

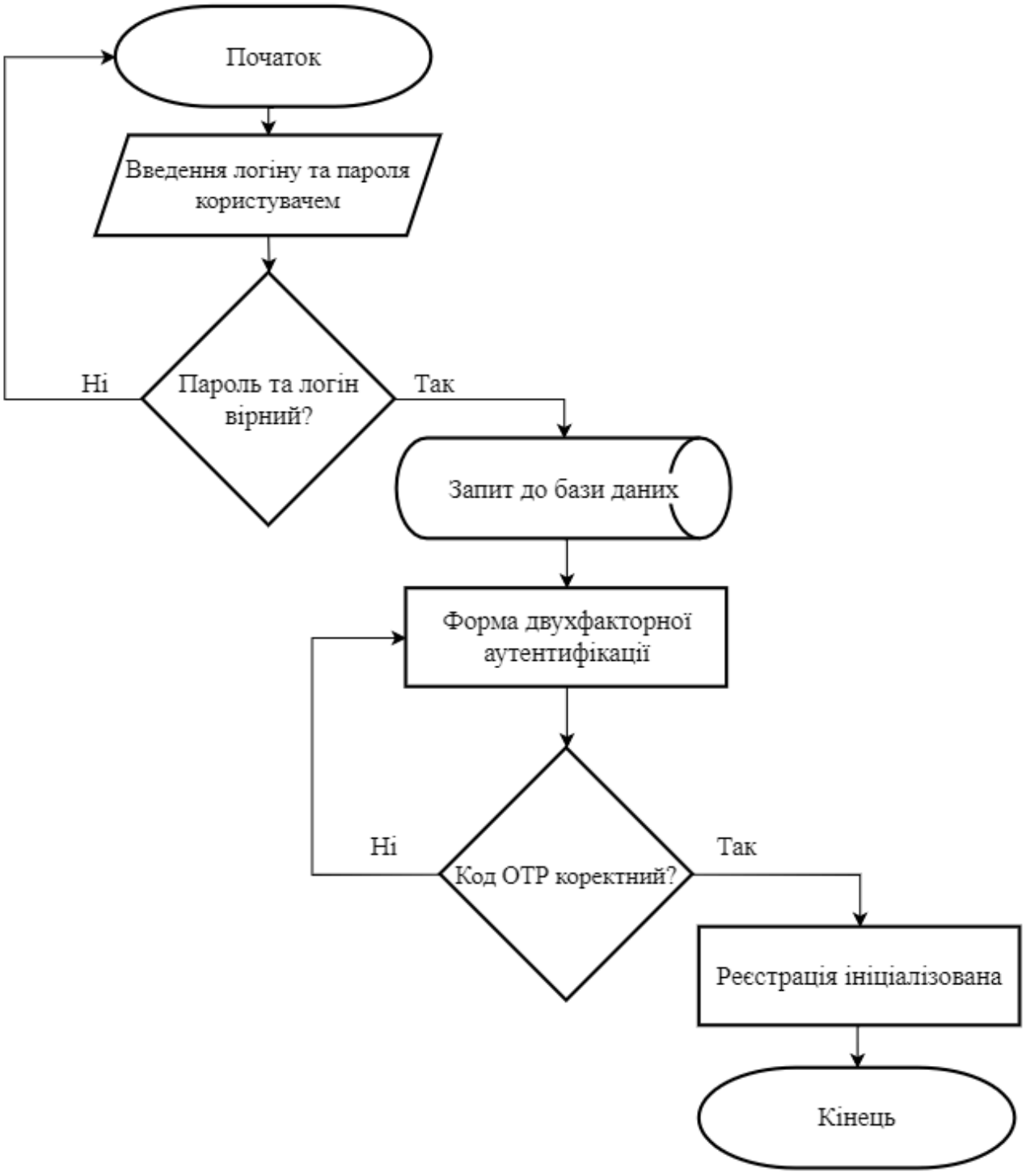


Рисунок 2.4 – Схема алгоритму авторизації на розробленому вебсервісі

Крок 1. Початок реєстрації.

Користувач відкриває вебсервіс та натискає на кнопку "Реєстрація" або переходить на сторінку реєстрації. Система відображає форму реєстрації з полями для введення необхідної інформації.

Крок 2. Введення логіну та пароля.

Користувач вводить свій логін (ім'я користувача або адресу електронної пошти) у відповідне поле форми. Користувач вводить свій пароль у відповідне поле форми. Система перевіряє, чи відповідають введені дані встановленим вимогам (наприклад, мінімальна довжина, наявність символів різних типів тощо). Якщо введені дані не відповідають вимогам, система відображає повідомлення про помилку та залишається на кроці 2.

Крок 3. Перевірка логіну та пароля на правильність.

Система надсилає запит до бази даних, щоб перевірити, чи існує користувач з таким логіном. Якщо користувач з таким логіном знайдений, система порівнює введений пароль з хешованим паролем, збереженим у базі даних. Якщо логін або пароль не збігаються, система відображає повідомлення про помилку та повертається до кроку 2.

Крок 4. Код OTP коректний?

Якщо користувач має налаштовану двофакторну автентифікацію, система генерує одноразовий пароль (OTP) та надсилає його на зареєстрований номер телефону або електронну пошту користувача. Користувач отримує OTP та вводить його у відповідне поле форми. Система перевіряє, чи збігається введений OTP з згенерованим. Якщо введений OTP не коректний, система відображає повідомлення про помилку та залишається на кроці 4.

Крок 5. Форма двохфакторної автентифікації.

Якщо користувач успішно пройшов перевірку логіну, пароля та OTP (якщо налаштовано), система відображає форму двохфакторної автентифікації.

Крок 6. Запит до бази даних.

Після успішної двохфакторної автентифікації, система надсилає запит до бази даних для отримання повної інформації про користувача. Система перевіряє статус облікового запису користувача.

Крок 7. Авторизація успішна.

Якщо всі попередні кроки пройдені успішно, система авторизує користувача та створює сеанс (сесію) для нього. Користувачу надається доступ до відповідних ресурсів та функціональності вебсервісу згідно з його рівнем доступу та правами.

Наступна схема демонструє процес завантаження файлів до вебсервісу (рис. 2.5).

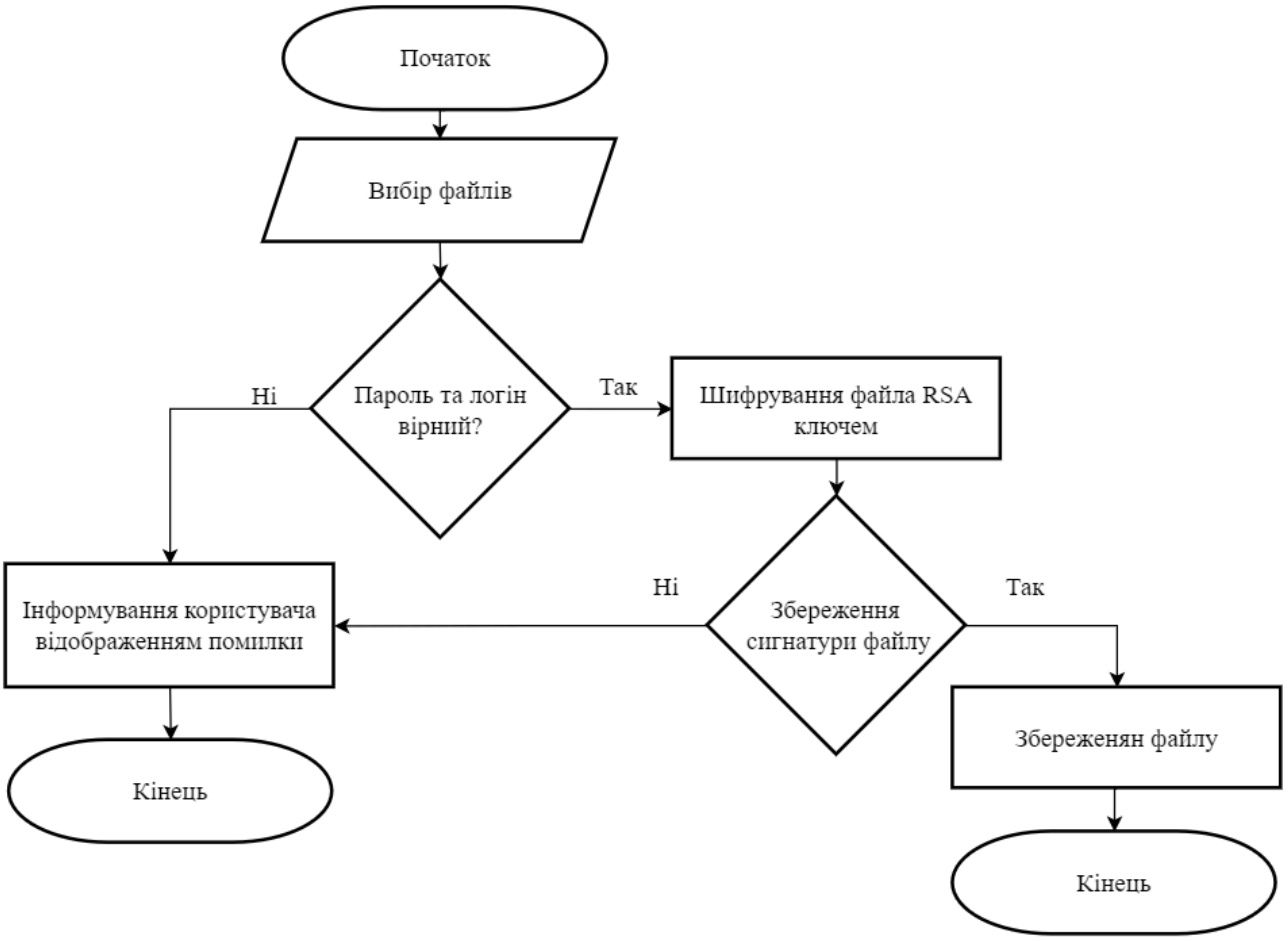


Рисунок 2.5 – Схема алгоритму завантаження файлу у розробленому вебсервісі

Крок 1. Вибір файлів.

Користувач вибирає один або декілька файлів для завантаження в систему.

Крок 2. Шифрування файла RSA ключем.

Система шифрує обрані файли за допомогою RSA ключа користувача. Цей крок забезпечує безпечну передачу файлів від користувача до системи.

Крок 3. Збереження сигнатури файлу. Система обчислює та зберігає сигнатуру (хеш) зашифрованого файлу.

Сигнатура використовується для перевірки цілісності файлу при подальшій обробці. Якщо сигнатура файлу не збігається з очікуваною, система переходить до кроку 4.

Крок 4. Помилка.

У разі виникнення помилки під час шифрування або збереження сигнатури, система повідомляє про це користувача. Користувач може спробувати завантажити файл повторно або звернутися за допомогою до служби підтримки.

Крок 5. Вибір ключа для підпису файла.

Система пропонує користувачу вибрати ключ для підпису файла. Користувач може вибрати існуючий ключ або згенерувати новий. Якщо користувач вирішує не підписувати файл, система переходить до кроку 7.

Крок 6. Підписання файла обраним ключем.

Система підписує зашифрований файл обраним ключем користувача. Підпис гарантує автентичність файлу та дозволяє перевірити, що файл був завантажений саме цим користувачем.

Крок 7. Збереження файлу.

Система зберігає зашифрований та підписаний файл у захищеному сховищі.

Цей алгоритм забезпечує безпечне завантаження, шифрування та підписання файлів перед їх збереженням у системі. Він гарантує конфіденційність та цілісність файлів користувачів, а також дозволяє перевіряти автентичність файлів за допомогою цифрових підписів.

2.3 Проєктування бази даних для управління резервними копіями у захищеному вебсервісі

Відносини бази даних вебсервісу для резервного копіювання можна зобразити у формі ER–діаграми. Після аналізування сутностей, що використовуються в моделі інформаційної системи, переходимо до створення структури бази даних.

Для цього визначимо назви таблиць, атрибути, їх типи, призначення та обмеження.

Ця діаграма є важливим артефактом проектування, оскільки вона слугує візуальним представленням структури бази даних та дозволяє розробникам і зацікавленим сторонам краще зрозуміти організацію даних у системі. Використання ER–діаграми сприяє ефективній комунікації та узгодженості в процесі розробки та супроводу бази даних. ER–діаграму захищеного вебсервісу, яка наочно демонструє взаємозв'язки між сутностями бази даних (рис 2.6.).

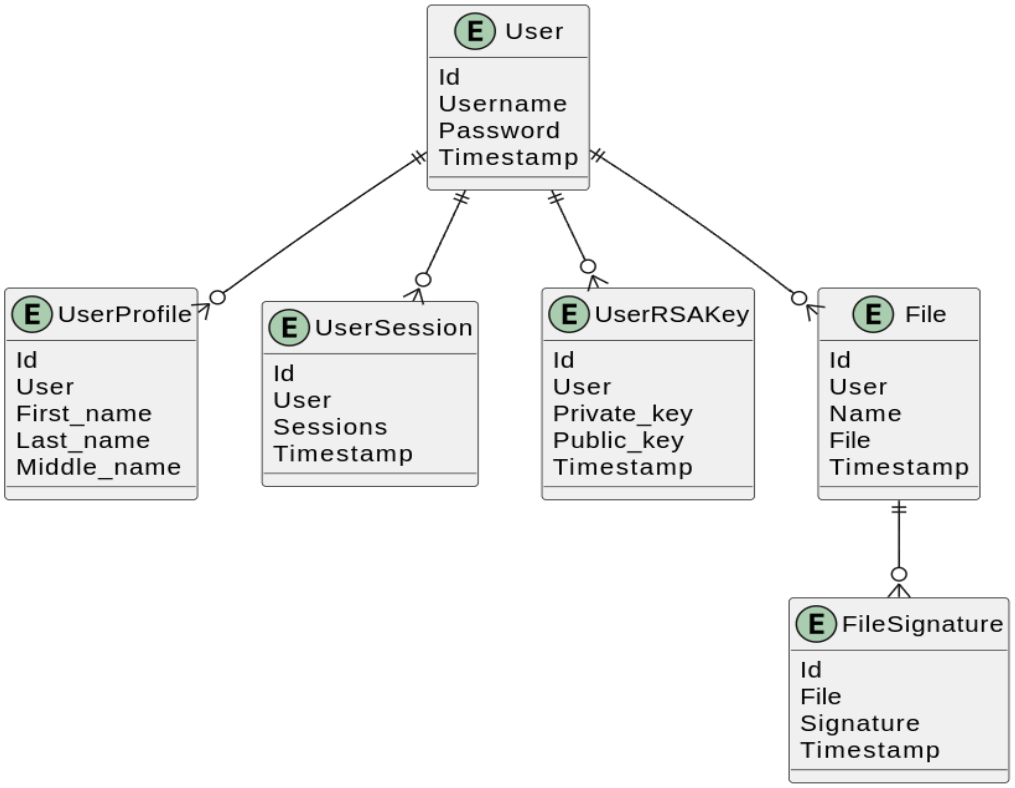


Рисунок 2.6 –ER–діаграма бази даних розробленого захищеного вебсервісу

Таблиця "Користувач" призначена для зберігання інформації про зареєстрованих користувачів вебсервісу. Вона містить такі атрибути, як унікальний ідентифікатор користувача, ім'я користувача, пароль користувача та дату і час реєстрації користувача. Ці атрибути дозволяють однозначно ідентифікувати користувачів та забезпечити їх автентифікацію в системі.

Таблиця "Профіль користувача" доповнює інформацію про користувачів, зберігаючи додаткові деталі, такі як повне ім'я користувача, прізвище та по батькові. Кожен профіль користувача пов'язаний з відповідним користувачем через атрибут UserId, що забезпечує цілісність даних та дозволяє отримувати повну інформацію про користувача шляхом об'єднання даних з обох таблиць.

Таблиця "Сесія користувача" призначена для відстеження активних сесій користувачів на вебсервісі. Вона містить такі атрибути, як унікальний ідентифікатор сесії, ідентифікатор користувача, сесія користувача та дата і час створення сесії. Ця таблиця дозволяє моніторити активність користувачів та забезпечувати безпеку шляхом контролю доступу на основі сесій.

Таблиця "Ключі RSA користувача" відіграє важливу роль у забезпеченні конфіденційності та цілісності даних користувачів. Вона зберігає криптографічні ключі RSA, які використовуються для шифрування та розшифрування даних. Атрибути таблиці включають унікальний ідентифікатор, ідентифікатор користувача, приватний та публічний ключі RSA, а також дату і час генерації ключів. Ці ключі є основою для реалізації наскрізного шифрування даних у вебсервісі.

Таблиця "Файл" призначена для зберігання інформації про файли, завантажені користувачами на вебсервіс. Вона містить такі атрибути, як унікальний ідентифікатор файлу, ідентифікатор користувача, назва файлу, шлях до файлу або посилання на нього та дата і час завантаження файлу. Ця таблиця дозволяє організувати та керувати файлами користувачів, забезпечуючи їх зберігання та доступ до них.

Таблиця "Підпис файлу" відіграє важливу роль у забезпеченні цілісності та автентичності файлів. Вона зберігає інформацію про цифрові підписи файлів, які використовуються для перевірки того, що файл не був модифікований після підписання. Атрибути таблиці включають унікальний ідентифікатор підпису, ідентифікатор файлу, сигнатуру файлу та дату генерації сигнатури. Ця таблиця дозволяє гарантувати цілісність файлів та запобігати їх несанкціонованій модифікації.

2.4 Висновок до розділу 2

У даному розділі було проведено проектування захищеного вебсервісу для резервного копіювання даних у середовищі Amazon S3 з використанням наскрізного шифрування. Було детально розглянуто та обґрунтовано удосконалення підходу до резервного копіювання даних шляхом впровадження гібридної схеми шифрування, яка поєднує переваги асиметричного шифрування RSA для безпечного обміну ключами та симетричного шифрування AES для ефективного шифрування самих даних. Цей підхід забезпечує наскрізне шифрування даних протягом усього їх життєвого циклу, підвищуючи рівень безпеки та конфіденційності.

Розроблено детальні алгоритми реєстрації користувача та авторизації доступу до вебсервісу. Ці алгоритми забезпечують надійну автентифікацію користувачів, використовуючи комбінацію логіну, пароля та додаткових факторів, таких як одноразові паролі OTP та двофакторна автентифікація. Процес реєстрації включає генерацію пари ключів RSA для кожного користувача, що є основою для безпечної передачі та зберігання даних.

Також було представлено алгоритм завантаження файлів до вебсервісу, який забезпечує цілісність та автентичність файлів. Цей алгоритм включає шифрування файлів за допомогою випадково згенерованих ключів AES та зберігання зашифрованих ключів разом з метаданими файлів. Крім того, передбачено можливість безпечного поширення файлів шляхом створення унікальних посилань, захищених додатковим шифруванням. Проведено проектування бази даних для управління резервними копіями у захищеному вебсервісі. Розроблено ER–діаграму, яка відображає ключові сутності бази даних та їх взаємозв'язки.

Загалом, у рамках даного розділу було розроблено архітектуру та алгоритми функціонування захищеного вебсервісу для резервного копіювання даних з використанням наскрізного шифрування. Запропоновані рішення забезпечують високий рівень безпеки, конфіденційності та цілісності даних користувачів, а також зручність використання вебсервісу.

3 РОЗРОБКА ЗАХИЩЕНОГО ВЕБСЕРВІСУ ТА ЙОГО ТЕСТУВАННЯ

3.1 Обґрунтування мови програмування та середовища розробки

Для реалізації захищеного вебсервісу для резервного копіювання даних у середовищі Amazon S3 з використанням наскрізного шифрування була обрана мова програмування Rust. Цей вибір обґрунтовується кількома ключовими факторами.

По–перше, Rust забезпечує високу продуктивність, порівнянну з мовами низького рівня, такими як C та C++, але при цьому надає абстракції та сучасні можливості, притаманні мовам вищого рівня. Це особливо важливо для вебсервісів, де необхідно ефективно обробляти велику кількість запитів та забезпечувати швидку відповідь.

По–друге, Rust має вбудовану систему управління пам'яттю, яка гарантує безпеку та запобігає типовим помилкам, пов'язаним з витоками пам'яті та невизначеною поведінкою. Ця особливість є критичною для розробки захищеного вебсервісу, оскільки вона дозволяє уникнути вразливостей, пов'язаних з помилками в управлінні пам'яттю. Це підтверджується результатами оцінки інших мов програмування, представленими в таблиці 3.1.

Таблиця 3.1 Кількісне порівняння Rust з іншими мовами програмування

Критерій	Rust	C++	Python	JavaScript
Швидкість виконання (геометричне середнє з комп'ютерів–мічників)	1.0	1.03	61.08	68.67
Споживання пам'яті (МБ)	1.61	1.78	10.49	16.53
Кількість вразливостей (CVE за останні 5 років)	115	2,027	1,989	1,862
Кількість активних репозиторіїв на GitHub	58,200	1,200,000	1,300,000	1,600,000
Кількість активних contributors	3,700	16,000	25,000	28,000
Завантаження пакетів за місяць (мільйони)	1.4	–	210	350

Ці кількісні показники ілюструють, що Rust є відносно новою мовою програмування, але вже демонструє перевагу у продуктивності, безпеці та ефективності використання ресурсів.

У якості середовища розробки (IDE) було обрано Visual Studio Code з встановленими розширеннями для Rust, такими як rust–analyzer та crates. Visual Studio Code є потужним і гнучким середовищем, яке пропонує широкий спектр інструментів для розробки, налагодження та тестування програм на Rust.

Порівняно з існуючими аналогами, такими як Django (Python), Ruby on Rails (Ruby) та Express.js (Node.js), вебсервіс на Rust з використанням Actix–web демонструє вищу продуктивність та ефективність використання ресурсів. Це підтверджується результатами бенчмаркінгу, представленими в таблиці 3.2.

Таблиця 3.2. Порівняння продуктивності вебфреймворків

Фреймворк	Запити/сек
Actix–web	182,457
Django	51,832
Ruby on Rails	33,336
Express.js	61,274

Крім того, Rust забезпечує вищий рівень безпеки порівняно з Python, Ruby та JavaScript завдяки своїй системі управління пам'яттю та потужній системі типів. Це є критичним фактором для розробки захищеного вебсервісу, який буде обробляти конфіденційні дані користувачів.

Використання SQLite як вбудованої бази даних дозволяє уникнути необхідності налаштування та підтримки окремого серверу бази даних, що спрощує розгортання та масштабування вебсервісу.

Для взаємодії з Amazon S3 були використані офіційні бібліотеки AWS SDK для Rust, які забезпечують зручний та безпечний доступ до сервісів AWS. Це дозволило інтегрувати наш вебсервіс з Amazon S3 для зберігання зашифрованих резервних копій даних.

Нарешті, використання Docker для упаковки вебсервісу в контейнери забезпечує портативність та полегшує процес розгортання на різних платформах, включаючи хмарні середовища, такі як Amazon Elastic Kubernetes Service (EKS).

У таблиці 3.4 наведено порівняння трьох популярних середовищ розробки (IDE) для Rust: Visual Studio Code, IntelliJ IDEA та CLion. Всі три IDE пропонують відмінну підтримку Rust, включаючи автодоповнення коду, перевірку коду на льоту, інтеграцію з системами збірки та налагодження. Вони також є кросплатформними та розширюваними за допомогою плагінів.

Таблиця 3.4. Порівняння середовищ розробки для мови Rust

Критерій	Критерій	Критерій	Критерій
Підтримка Rust	Підтримка Rust	Підтримка Rust	Підтримка Rust
Безкоштовна версія	Безкоштовна версія	Безкоштовна версія	Безкоштовна версія
Інтеграція з системами контролю версій	Інтеграція з системами контролю версій	Інтеграція з системами контролю версій	Інтеграція з системами контролю версій
Підтримка налагодження	Так	Так	Так
Інтеграція з системами збірки	Так	Так	Так
Інтелектуальна автодоповнення коду	Так	Так	Так
Перевірка коду на льоту	Так	Так	Так
Кросплатформність	Так (Windows, macOS, Linux)	Так (Windows, macOS, Linux)	Так (Windows, macOS, Linux)
Розширюваність за допомогою плагінів	Так	Так	Так
Зручність інтерфейсу	Висока	Висока	Висока

Visual Studio Code є безкоштовним і має широкі можливості завдяки великій кількості розширень, включаючи розширення rust–analyzer для підтримки Rust. IntelliJ IDEA та CLion від JetBrains пропонують передові функції для розробки на

Rust, але CLion є спеціалізованою IDE для Rust, тоді як IntelliJ IDEA – більш універсальна IDE для різних мов програмування.

У результаті, вибір Rust, SQLite, Docker, Vue.js та AWS SDK забезпечує надійну, масштабовану та безпечну основу для реалізації захищеного вебсервісу для резервного копіювання даних у середовищі Amazon S3 з використанням наскрізного шифрування.

3.2 Опис реалізації удосконаленого алгоритму

Для реалізації захищеного вебсервісу для резервного копіювання даних у середовищі Amazon S3 з використанням наскрізного шифрування було створено Docker–контейнер на основі операційної системи Ubuntu. Dockerfile, який описує процес створення контейнера, виглядає наступним чином:

```
FROM ubuntu:latest

ENV HOST 0.0.0.0
EXPOSE 5443

RUN apt update && apt install curl libpq–dev clang llvm pkg–config nettle–dev libc6–dev libssl–dev –y

COPY ./target/release/drive /usr/local/bin

ENV DATA_DIR="/data"
ENV
RUST_LOG="drive=debug,auth=debug,error=debug,entity=debug,storage=debug,context=debug,util=debug,cryptfns=debug,actix_web=debug"

CMD /usr/local/bin/drive –a 0.0.0.0 –p 5443
```

Dockerfile починається з вказівки базового образу FROM ubuntu:latest, який буде використовуватися як основа для контейнера. Далі, за допомогою директиви ENV, встановлюються змінні середовища HOST зі значенням 0.0.0.0 та EXPOSE, яка вказує на те, що контейнер буде слухати на порту 5443.

Наступний блок RUN виконує оновлення пакетів Ubuntu за допомогою apt update та встановлює необхідні залежності, такі як curl, libpq–dev, clang, llvm, pkg–config, nettle–dev, libc6–dev та libssl–dev, використовуючи –y для автоматичного підтвердження встановлення.

Нарешті, директива CMD вказує команду, яка буде виконуватися при запуску контейнера. У даному випадку, запускається бінарний файл /usr/local/bin/drive з

параметрами –a 0.0.0.0 (прив'язка до всіх мережевих інтерфейсів) та –p 5443 (порт, на якому буде працювати вебсервіс).

Для забезпечення роботи з базою даних у вебсервісі використовується Docker Compose. Файл docker–compose.yml описує конфігурацію сервісу бази даних PostgreSQL:

```
version: "2"
services:
  postgres:
    image: bitnami/postgresql:latest
    restart: always
    hostname: postgres
    container_name: postgres
    environment:
      – POSTGRESQL_USERNAME=postgres
      – POSTGRESQL_PASSWORD=postgres
      – POSTGRESQL_DATABASE=postgres
      – POSTGRESQL_WAL_LEVEL=logical
    ports:
      – "5432:5432"
```

Директива ports визначає відображення порту 5432 з контейнера на порт 5432 на хост–машині, що дозволяє зовнішнім сервісам підключатися до бази даних PostgreSQL.

Використання Docker та Docker Compose дозволяє легко розгортати та керувати вебсервісом та базою даних у контейнеризованому середовищі, забезпечуючи ізоляцію, портативність та масштабованість.

Далі на черзі реалізація розробленого алгоритму захищеного вебсервісу для резервного копіювання даних з використанням наскрізного шифрування.

```
pub struct CreateFile {
    pub encrypted_key: Option<String>,
    pub name_hash: Option<String>,
    pub encrypted_name: Option<String>,
    pub encrypted_thumbnail: Option<String>,
    pub search_tokens_hashed: Option<Vec<String>>,
    pub mime: Option<String>,
    pub size: Option<i64>,
    pub chunks: Option<i64>,
    pub file_id: Option<String>,
    pub file_modified_at: Option<String>,
}
```

Цей ключовий фрагмент коду відповідає за створення нового файлу або отримання інформації про вже існуючий файл для продовження завантаження. Також у файлі реалізовано методи для валідації даних та перетворення структури у активну модель бази даних.

Наступний важливий фрагмент коду – це код, який містить функції для виконання операцій з шифруванням RSA:

```
pub mod private {
  // ...
  pub fn sign_with(message: &str, key: PrivateKey) –> CryptoResult<String> {
    // ...
  }
  pub fn decrypt_with(data: &[u8], key: PrivateKey) –> CryptoResult<Vec<u8>> {
    // ...
}
pub mod public {
  // ...
  pub fn verify_with(message: &str, signature: &str, key: PublicKey) –> CryptoResult<()> {
    // ...
  }
  pub fn encrypt_with(data: &[u8], key: PublicKey) –> CryptoResult<Vec<u8>> {
    // ...
  }
```

Код розділений на два модулі: private для операцій з приватним ключем та public для операцій з публічним ключем. Основні функції включають: підписання повідомлення за допомогою приватного ключа, дешифрування даних за допомогою приватного ключа, перевірка підпису повідомлення за допомогою публічного ключа, шифрування даних за допомогою публічного ключа.

Далі наведені залежності та скрипти, що забезпечують повноцінну екосистему для розробки, тестування та збірки вебдодатку з використанням сучасних інструментів та бібліотек, таких як Vue.js, TypeScript, Vite, Cypress та інші. Код, що відображає метадані проєкту:

```
{
 "name": "@drive/web",
 "license": "CC BY–NC 4.0",
 "version": "1.0.2",
 "private": true,
}
```

Ця частина визначає скрипти, які можна запускати за допомогою команди npm run <script–name>. Наприклад, npm run dev запустить сервер розробки за допомогою Vite, npm run build створить продакшн-збірку проєкту, npm run test:unit запустить модульні тести за допомогою Vitest, а npm run lint запустить перевірку коду за допомогою ESLint.

```
{
 "scripts": {
  "dev": "vite",
  "build": "vite build",
  "preview": "vite preview",
```

```
    "test:unit": "vitest run",
    "test:e2e": "cypress run",
    "test:watch": "vitest",
    "type–check": "vue–tsc —noEmit –p tsconfig.vitest.json —composite false",
    "lint": "eslint . —ext .vue,.js,.jsx,.cjs,.mjs,.ts,.tsx,.cts,.mts —fix —ignore–path .gitignore",
    "format": "prettier —write src/",
    "cypress:open": "cypress open"
  },
}
```

Допоміжні функції для перетворення ключів у рядкове представлення та навпаки, генерації відбитку ключа та інші. Наведені Cypress–тести демонструють процес реєстрації користувача у вебсервісі та входу в систему:

```
describe('Registration to the application', () => {
  it('can register', () => {
    // ...
    cy.get('input[id=email]').type(email)
    cy.get('input[id=password]').type(password)
    cy.get('input[id=confirm_password]').type(password)
    // ...
    cy.get('textarea[id=unencrypted_private_key]')
      .invoke('val')
      .then(async (val) => {
        privateKey = val
        expect(privateKey).to.contain('BEGIN RSA PRIVATE KEY')
      })
    cy.get('input[id=secret]')
      .invoke('val')
      .then(async (val) => {
        secret = val
        const token = otp(val)
        cy.get('input[id=token]').type(token)
        cy.get('button').contains('Register with Two Factor').click()
      })
    // ...
  })
})
```

Реєстрація нового користувача з введенням електронної пошти, пароля, збереженням приватного ключа та налаштуванням двофакторної автентифікації. Вхід у систему з використанням електронної пошти, пароля та OTP–токена. Вхід у систему за допомогою приватного ключа.

```
describe('Login to the application', () => {
  it('can login with email and password with OPT token', () => {
    // ...
    cy.get('input[id=email]').type(email)
    cy.get('input[id=password]').type(password)
    cy.get('input[id=token]').type(otp(secret))
    cy.get('button').contains('Login').click()
  })

  it('can login with private key', () => {
    // ...
```

```
    cy.get('textarea[id=privateKey]').type(privateKey, { delay: 1 })
    cy.get('button').contains('Login').click()
  })
})
```

Авторизація у Amazon S3 за допомогою API ключів.

```
pub async fn request<B, T>(
  method: Method,
  url: String,
  body: B
) -> Result<T, ServiceError> where
    T: DeserializeOwned + 'static + Debug,
    B: Serialize + Debug  {
      let api_url: &str = &std::env::var("API_URL").expect("You must set an API KEY");
      let allowed_body = method == reqwest::Method::POST || method == reqwest::Method::PUT;
      let url = format!("{}{}", api_url, url);
      let mut builder = reqwest::Client::new()
        .request(method, url)
        .header("Content-Type", "application/json");
      if let Some(token) = get_token() {
        builder = builder.bearer_auth(token)
      }
```

Завантаження файлів у середовище Amazon S3.

```
pub fn export_content<S: AsRef<str>>(
  accept: S,
  export_result: compiler::ExportCompilerResult<Vec<u8>>,
) -> HttpResponse {
  match export_result {
    Ok(bytes) => HttpResponse::Ok()
      .encoding(ContentEncoding::Identity)
      .content_type(accept.as_ref())
      .append_header(("accept-ranges", "bytes"))
      .append_header((
        "content-disposition",
        format!(
          "attachment; filename=\".{}\"",          ))
      .body(bytes),
    Err(compiler::ExportCompilerError::GenericError(message)) => {
      HttpResponse::InternalServerError().json(WsError { error: message })
    }
  }
}
```

Отже, наведені фрагменти коду демонструють реалізацію ключових етапів удосконаленого алгоритму захищеного вебсервісу для резервного копіювання даних з використанням наскрізного шифрування, таких як створення файлів з шифруванням метаданих, виконання операцій шифрування RSA та автентифікація користувачів з використанням двофакторної автентифікації та приватних ключів.

3.3 Тестування розробленого вебсервісу

У цьому розділі описано процес тестування розробленого вебсервісу для резервного копіювання даних у хмарному сховищі Amazon S3 з використанням наскрізного шифрування.

Наведено скріншоти основних форм та вікон користувацького інтерфейсу, супроводжувані детальними інструкціями щодо їх використання. Продемонстровано ключові функції вебсервісу, зокрема реєстрація нового облікового запису, генерування та збереження приватного ключа, налаштування двофакторної автентифікації, авторизація з використанням електронної пошти/пароля або приватного ключа.

Таким чином, забезпечується всебічне тестування розробленого програмного забезпечення з точки зору кінцевого користувача. Стартовий інтерфейс розробленого захищеного вебсервісу для резервного копіювання даних у середовищі Amazon S3 з використанням наскрізного шифрування, який містить форму для аутентифікації користувача (рис. 3.1).

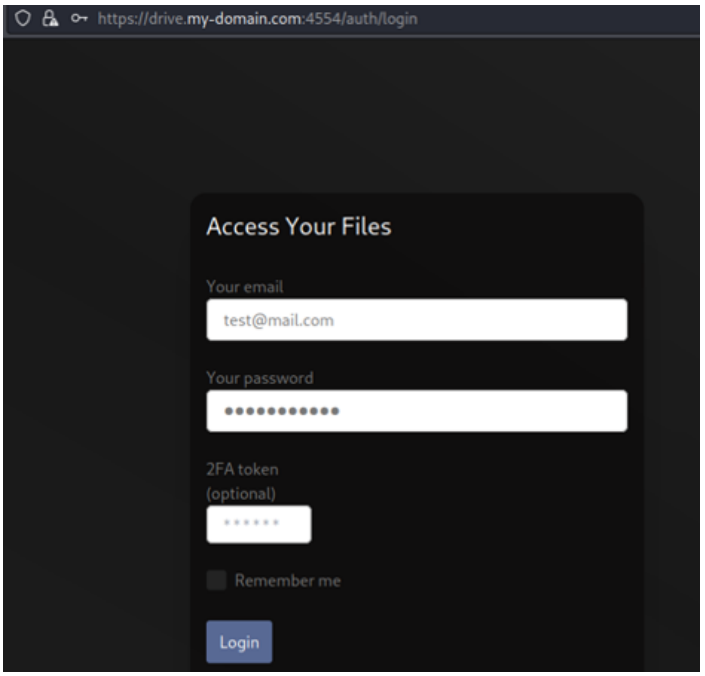


Рисунок 3.1 – Форма аутентифікації на захищеному вебсервісі

Користувач має ввести свою електронну адресу у поле "Your email" та пароль у поле "Your password". Додатково можна вказати 2FA токен (опціонально). Після цього слід натиснути кнопку "Login", щоб увійти до вебсервісу. Якщо користувач ще не зареєстрований, то він може перейти до форми реєстрації, натиснувши на посилання "Create an Account". Форма реєстрації нового користувача в системі представлена на (рис. 3.2).

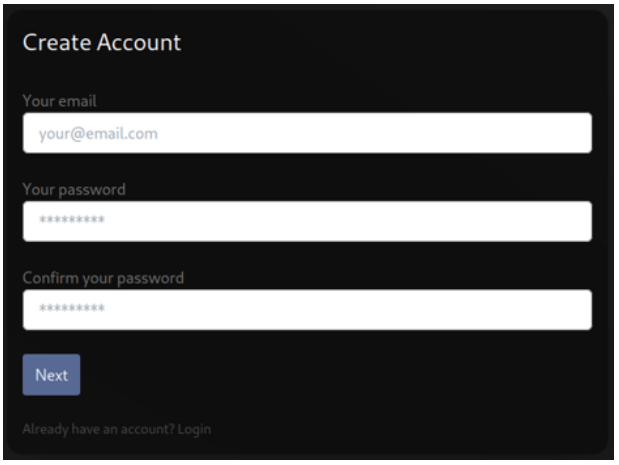


Рисунок 3.2 – Форма реєстрації нового користувача на захищеного вебсервісу

Для створення нового облікового запису необхідно ввести електронну адресу в поле "Your email", встановити пароль у полях "Your password" та "Confirm your password". Після заповнення всіх полів слід натиснути кнопку "Next".

На наступному кроці система згенерує та відобразить приватний ключ для користувача. Вікно з згенерованим приватним ключем, який необхідно зберегти для подальшої авторизації в системі (рис. 3.3).

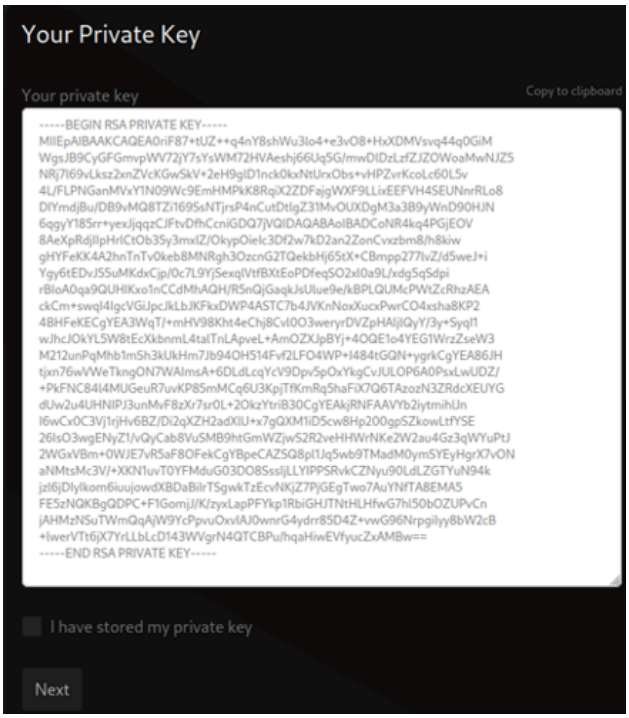


Рисунок 3.3 – Вікно з приватним ключем користувача захищеного вебсервісу

Після збереження приватного ключа користувач має натиснути кнопку "Next". На наступному кроці відкриється форма для встановлення двофакторної автентифікації (2FA), що додатково захистить обліковий запис (рис. 3.4).

У цьому вікні система вже згенерувала секретний ключ для 2FA у полі "Your two factor secret". Користувачу потрібно ввести одноразовий токен з додатку автентифікатора на смартфоні в поле "Enter your two factor token" і натиснути "Register with Two Factor". За бажанням користувач може пропустити цей крок, натиснувши "Skip".

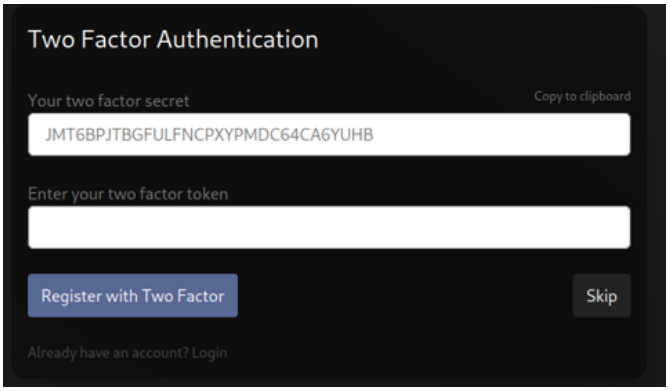


Рисунок 3.4 – Форма для налаштування 2FA на розробленому вебсервісі

Після успішної реєстрації та налаштування користувач повертається на стартову сторінку системи, де він може виконати вхід до облікового запису за допомогою електронної пошти та пароля, або ввівши свій приватний ключ (рис. 3.5).

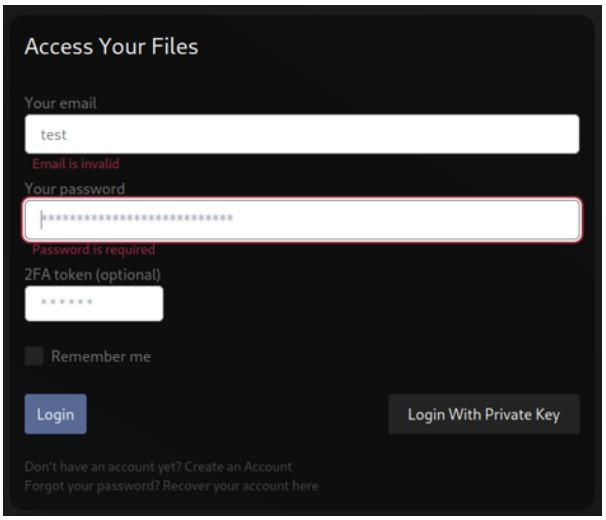


Рисунок 3.5 – Форма проходження 2FA на розробленому вебсервісі

Окрім полів для введення електронної пошти, пароля та 2FA токену, на цій формі з'явилася додаткова кнопка "Login With Private Key". Після успішної авторизації користувач отримує доступ до основного функціоналу вебсервісу. Головна сторінка вебсервісу після авторизації користувача (рис. 3.6).

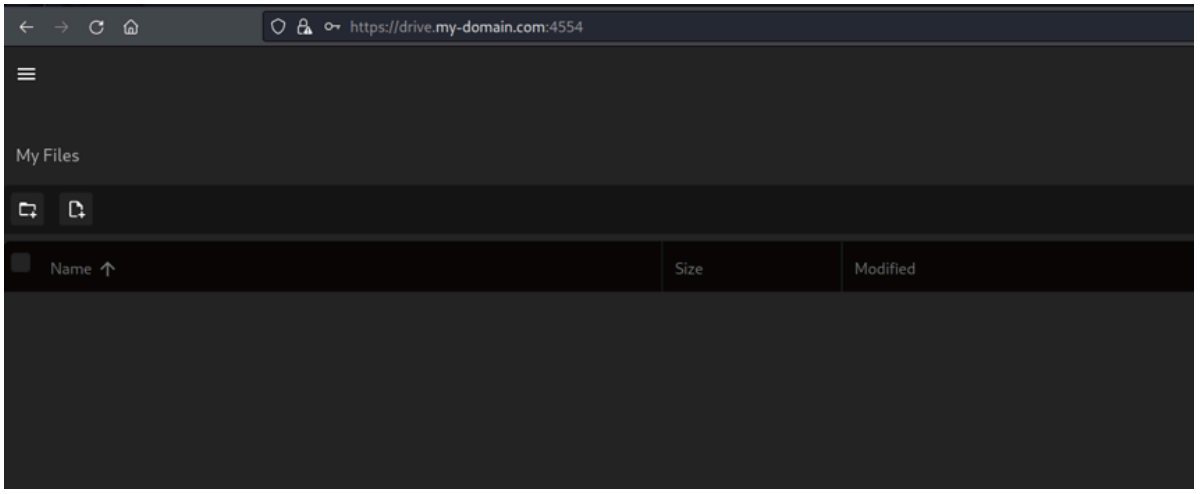


Рисунок 3.6 – Головна сторінка захищеного вебсервісу після авторизації

На цій сторінці користувач має можливість переглянути список завантажених файлів та виконати різні операції з ними. На рисунку 3.7 показано вікно завантаження файлу для створення резервної копії.

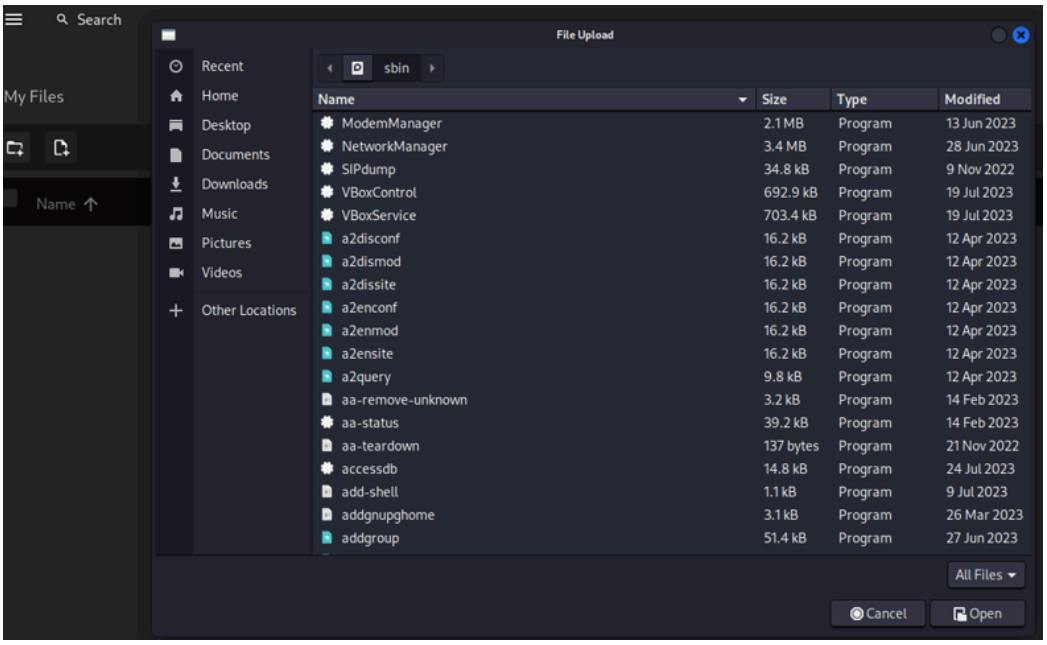


Рисунок 3.7 – Вікно завантаження файлу для створення резервної копії

Це вікно відкривається після натискання кнопки "File Upload" у верхньому правому куті головної сторінки. У ньому користувач може обрати один або декілька файлів на своєму пристрої для завантаження на сервер. Після вибору файлів необхідно натиснути кнопку "Open", щоб розпочати процес завантаження. Під час завантаження буде відображатися смужка прогресу для відстеження процесу. Завантажений файл для створення резервної копії на захищеному вебсервісі виглядає як зображено на (рис. 3.8).

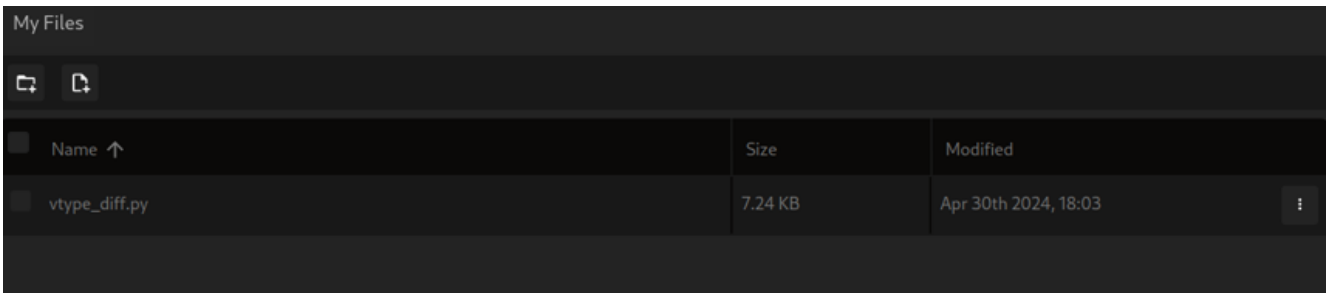


Рисунок 3.8 – Вікно вигляду завантаженого файлу на захищеному вебсервісі

Вікно сервісу розділене на дві частини: верхня панель містить навігаційне меню, а нижня – відображає список файлів із зазначенням їхнього розміру та дати модифікації. Натиснувши на значок "три крапки" навпроти обраного файлу, користувач може відкрити контекстне меню з такими опціями: "Деталі", "Завантажити", "Відкрити публічне посилання", "Видалити" та "Перейменувати" (рис. 3.9).

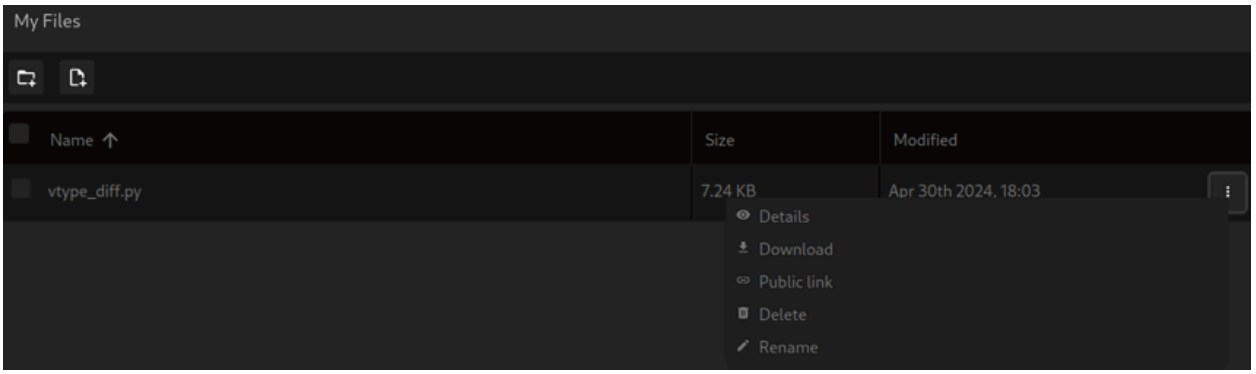


Рисунок 3.9 – Вікно вигляду завантаженого файлу на захищеному вебсервісі

Таким чином, користувач може зручно керувати своїми резервними копіями. Профіль користувача з його особистими даними та налаштуваннями безпеки можна переглянути в окремій вкладці (рис. 3.10).

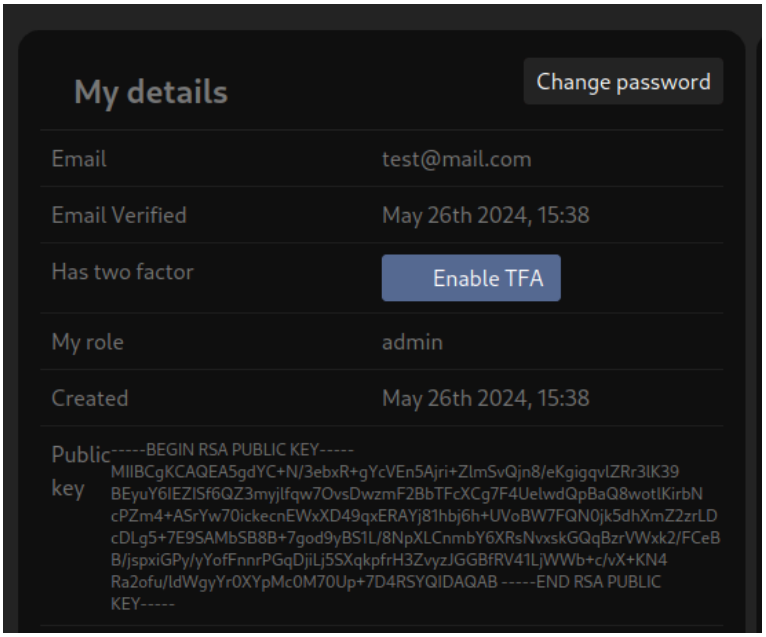


Рисунок 3.10 – Вікно інформації про створений профіль користувача вебсервісу

У цьому вікні користувач може переглянути свою електронну адресу, дату верифікації облікового запису, роль у системі та дату створення облікового запису. Також тут відображається публічний ключ шифрування користувача, який використовується для наскрізного шифрування даних. Крім того, є можливість змінити пароль та увімкнути/вимкнути двофакторну автентифікацію за допомогою відповідних кнопок. Таким чином, користувач має повний контроль над своїми даними та налаштуваннями безпеки в межах вебсервісу.

ВИСНОВКИ

У ході виконання бакалаврської дипломної роботи було досягнуто поставленої мети – розроблено захищений вебсервіс для резервного копіювання даних у середовищі Amazon S3 з використанням наскрізного шифрування.

Розроблено алгоритм роботи програмного застосунку, який включає в себе удосконалення підходу резервного копіювання даних з використанням наскрізного шифрування в середовищі Amazon S3, проєктування архітектури вебсервісу для безпечного резервного копіювання даних з використанням наскрізного шифрування у хмарному середовищі Amazon S3, проєктування механізмів аутентифікації та авторизації користувачів у вебсервісі для безпечного резервного копіювання даних, а також проєктування бази даних для управління резервними копіями у захищеному вебсервісі.

У першому розділі проведено аналіз даних, що потребують захищеного резервного копіювання, розглянуто існуючі засоби для створення резервних копій, досліджено переваги та недоліки хмарного сховища Amazon S3. Також проаналізовано алгоритми наскрізного шифрування даних та основні методи захисту інформації у вебсервісах для резервного копіювання.

У другому розділі було розроблено підхід резервного копіювання даних з використанням наскрізного шифрування в середовищі Amazon S3, спроєктовано архітектуру вебсервісу для безпечного резервного копіювання з використанням наскрізного шифрування у хмарному середовищі Amazon S3 та спроєктовано базу даних для управління резервними копіями у захищеному вебсервісі.

У третьому розділі обґрунтовано вибір мови програмування та середовища розробки, описано реалізацію удосконаленого алгоритму наскрізного шифрування даних та проведено тестування розробленого вебсервісу.

Здійснено програмну реалізацію застосунку, зокрема обґрунтовано вибір мови програмування Rust та середовища розробки, реалізовано удосконалений алгоритм резервного копіювання даних з використанням наскрізного шифрування, розроблено вебсервіс згідно з запроєктованою архітектурою, включаючи

механізми аутентифікації та авторизації користувачів, а також інтеграцію з хмарним сховищем Amazon S3. Крім того, створено зручний та інтуїтивно зрозумілий користувацький інтерфейс, який відрізняється своєю простотою у використанні та забезпечує комфортну взаємодію користувача з вебсервісом.

Результати тестування показали, що розроблений вебсервіс відповідає поставленим вимогам щодо функціональності, надійності та безпеки. Використання наскрізного шифрування забезпечує високий рівень захисту даних під час передачі та зберігання в хмарному середовищі Amazon S3.

Таким чином, розроблений захищений вебсервіс для резервного копіювання даних з використанням наскрізного шифрування є ефективним рішенням для забезпечення безпеки та надійності зберігання важливих даних користувачів у хмарному середовищі. Він може бути використаний як основа для подальшого розвитку та вдосконалення систем резервного копіювання та захисту даних.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Франчук В. М. Резервне копіювання даних. Науковий часопис УДУ імені Михайла Драгоманова. Серія 2. Комп’ютерно–орієнтовані системи навчання. 2018. Т. 20, № 61–66.

2. Русин Б. П., et al. "Архітектура системи дедублікації та розподілу даних у хмарних сховищах під час резервного копіювання." Інформаційні технології та комп'ютерна інженерія 2 (2019): 40–63.

3. Каплун Д. І. "Методи резервного копіювання даних." Збірник тез доповідей Міжнародної науково–технічної конференції молодих учених та студентів „Актуальні задачі сучасних технологій“ (2014): 196–196.

4. Шемет А. "Застосування методів програмного резервування інформації: аспект резервного копіювання. метод паралельного резервного копіювання." Вісник Хмельницького національного університету 3 (2011): 221–225.

5. Пастух О. А., Гарасівка А. "Необхідність резервного копіювання даних в повсякденному житті." Матеріали VIII науково–технічної конференції „Інформаційні моделі, системи та технології“ (2020): 136–137.

6. Вітряченко А. А. "Дослідження та програмна реалізація системи резервного копіювання даних хмарного, віртуального й фізичного середовища." (2022).

7. Гуляєв К. Д. "Організація системи резервного копіювання інформації на базі телекомунікаційної технології EX." Радіоелектронні і комп’ютерні системи 4 (2014): 67–71.

8. Атаян Б. Г., Багдасарян Т. А. "Безпека резервного копіювання даних в хмарній системі збереження." Ukrainian Scientific Journal of Information Security 22.2 (за номером): 119–122.

9. Чепак О. Р. "СИСТЕМА РЕЗЕРВНОГО КОПІЮВАННЯ ІНФОРМАЦІЇ ТА СИНХРОНІЗАЦІЇ ДАНИХ." ББК 72я431 ISSN 2522–932X: 112.

10. Ніконова А. В., Лєсна Н. С. "Дослідження ефективності методів і моделей резервного копіювання та аварійного відновлення серверних даних з використанням хмарного середовища." (2020).

11. Ворочук С., Черник Б., Яскілка В. Я. "Аналіз програмних продуктів для резервування та відновлення даних." Матеріали V науково–технічної конференції „Інформаційні моделі, системи та технології“ (2018): 24–24.

12. Архипов М. С. "РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ РЕЗЕРВУВАННЯ ДАНИХ КОРИСТУВАЧІВ." Інфокомунікації–сучасність та майбутнє: матеріали четвертої міжнар.: 4.

13. Близнюк М. І. "Харденінг вебзастосунків на прикладі CMS WordPress та сервісів Amazon." (2022).

14. Терейковський І. А., et al. "Захист вебсервісів: Лабораторний практикум." (2018).

15. Гарасівка А. В. "Розробка системи зберігання резервних копій даних в хмарних сховищах даних на базі C# та ASP. NET Core." MS thesis. 2020.

16. Іваськевич Ю. А. "Дослідження технологій для зберігання та обміну файлами вебсервісів." (2024).

17. Озел, Орхан Велі. "Безпека в хмарних сховищах даних." 2022.

18. Кулик Д., Горпенюк А. "СПОСОБИ ЗАХИСТУ ДАНИХ У ХМАРНОМУ СХОВИЩІ AMAZON S3." РЕДКОЛЕГІЯ 2023.

19. Русин Б. П., et al. "Архітектура системи дедублікації та розподілу даних у хмарних сховищах під час резервного копіювання." Інформаційні технології та комп'ютерна інженерія 2 (2019): 40–63.

20. Складанюк М. "Переваги використання хмарних сховищ." Математичні методи, моделі та інформаційні технології в управлінні підприємством (2019): 223–225.

21. Крайнік Юрій. "Розробка вебзастосунку з використанням хмарних вебсервісів." (2020)

22. Гаращук Д. "Побудова багаторівневого вебзастосування на платформі Amazon Web Services (AWS)." (2022).

23. Щур Наталія Олександрівна. "ОГЛЯД ПРОТОКОЛІВ ТА СЕРВІСІВ НАСКРІЗНОГО ШИФРУВАННЯ." ПЕРЕДОВІ ТЕХНОЛОГІЇ В ІНФОРМАЦІЙНО–КОМУНІКАЦІЙНІЙ ІНЖЕНЕРІЇ: 49.

24. Середа А. В., et al. "СТІЙКІСТЬ ТА ЕФЕКТИВНІСТЬ КРИПТОГРАФІЧНИХ АЛГОРИТМІВ ЩО ВИКОРИСТОВУЮТЬСЯ У МОБІЛЬНИХ ПРИСТРОЯХ." Інфокомунікаційні та комп'ютерні технології 2.04 (2022): 178–190.

25. Голуб О. С. Конфіденційність даних в інфокомунікаційних мережах і засоби її забезпечення. BS thesis. Київ, 2023.

26. Фесенко А. О., Хоменко О. В. "ОГЛЯД КРИПТОГРАФІЧНИХ ПРОТОКОЛІВ НАСКРІЗНОГО ШИФРУВАННЯ ВИКОРИСТАНИХ У СИСТЕМАХ ОБМІНУ МИТТЄВИМИ ПОВІДОМЛЕННЯМИ." ОРГАНІЗАТОРИ КОНФЕРЕНЦІЇ: 217.

27. Іванюк О. Д. "ПРОГРАМНИЙ МОДУЛЬ ЗАХИЩЕНОГО ОБМІНУ ПОВІДОМЛЕННЯМИ." Міжнародна наукова інтернет–конференція" Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення (2021): 26.

28. Томашевський Б. В., Сергієнко Р. В. "Протоколи і механізми безпеки інформації в комп'ютерних системах і мережах." Ukrainian Information Security Research Journal 11.1 (42): 39–56.

29. Шевчук Денис Тарасович. "МЕТОДИ АУТЕНТИФІКАЦІЇ ТА АВТОРИЗАЦІЇ У МОБІЛЬНИХ ТА ВЕБДОДАТКАХ Анотація." Міжнародна наукова інтернет–конференція" Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення (2022): 56.

30. Остапенко Максим Сергійович. "Метод автентифікації користувачів інформаційних систем." (2021).

31. Мусієнко Ангеліна Сергіївна. Огляд та дослідження методів аутентифікації користувачів у вебдодатках. MS thesis. Дніпровський національний університет залізничного транспорту імені академіка В. Лазаряна, 2020.

32. Самара Наталія та Бурак Назарій Євгенович. "Алгоритмізація процесу інтеграції сучасних методів автентифікації в системах управління взаємодією користувачів." (2021).

33. Драгоєв Д. "МЕТОДИ АВТЕНТИФІКАЦІЇ ТА КЕРУВАННЯ ДОСТУПОМ ДО ВЕБРЕСУРСУ ЗГІДНО ДО GDPR." Collection of scientific papers «ΛΌГОΣ» September 16, 2022; Boston, USA (2022): 82–85.

34. Славінський Всеволод Олександрович. Методи багатофакторної автентифікації до веб додатків за допомогою штучного інтелекту та технології блокчейн. MS thesis. КПІ ім. Ігоря Сікорського, 2022.

35. Ахрамович В. М. "Ідентифікація й аутентифікація, керування доступом." Сучасний захист інформації 4 (2016).

ДОДАТКИ

Додаток А. Технічне завдання

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

ЗАТВЕРДЖУЮ

Голова секції "Управління інформаційною безпекою" кафедри МБІС
д.т.н., професор

_____ Юрій ЯРЕМЧУК
" 22 " березня 2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ

до бакалаврської дипломної роботи на тему:

РОЗРОБКА ЗАХИЩЕНОГО ВЕБСЕРВІСУ ДЛЯ РЕЗЕРВНОГО КОПІЮВАННЯ ДАНИХ У СЕРЕДОВИЩІ AMAZON S3 З ВИКОРИСТАННЯМ НАСКРІЗНОГО ШИФРУВАННЯ

08–72.БДР.002.00.065.ТЗ

Керівник бакалаврської дипломної роботи
д.т.н., професор
Юрій ЯРЕМЧУК
" _____ " _____

Вінниця – 2024 р.

1. Найменування та область застосування

Вебсервіс для резервного копіювання даних з використанням наскрізного шифрування для зберігання у хмарному сховищі Amazon S3. Область застосування: захист та резервне копіювання даних користувачів у хмарному середовищі.

2.Підстава для розробки

Розробка виконується на основі наказу ректора ВНТУ № 80 від 11.03.2024 р.

3. Мета та призначення розробки

3.1 Мета розробки: розробка захищеного вебсервісу для резервного копіювання даних з використанням наскрізного шифрування для зберігання у хмарному сховищі Amazon S3.

3.2 Призначення: розроблений вебсервіс забезпечує безпечне резервне копіювання та зберігання даних користувачів у хмарному середовищі Amazon S3, використовуючи наскрізне шифрування для захисту конфіденційності та цілісності даних.

4. Джерела розробки

4.1. Франчук В. М. Резервне копіювання даних. Науковий часопис УДУ імені Михайла Драгоманова. Серія 2. Комп’ютерно–орієнтовані системи навчання. 2018. Т. 20, № 61–66.

4.2. Томашевський Б. В., Сергієнко Р. В. "Протоколи і механізми безпеки інформації в комп'ютерних системах і мережах." Ukrainian Information Security Research Journal 11.1 (42): 39–56.

4.3. Складанюк М. "Переваги використання хмарних сховищ." Математичні методи, моделі та інформаційні технології в управлінні підприємством (2019): 223–225.

4.4. Гаращук Д. "Побудова багаторівневого вебзастосування на платформі Amazon Web Services (AWS)." 2022.

4.5 Крайнік Юрій. "Розробка вебзастосунку з використанням хмарних вебсервісів." 2020.

5. Вимоги до програми

5.1 Вимоги до функціональних характеристик:

5.1.1 Програмний засіб повинен мати зручний, легкий у використанні інтерфейс користувача;

5.1.2 Реалізація методу не повинна вимагати спеціальних ліцензійних програмних додатків;

5.1.3 Програмний засіб повинен виконувати процес автентифікації користувачів у системі.

5.2 Вимоги до надійності:

5.2.1 Програмний засіб повинен працювати без помилок, у випадку виникнення критичних ситуацій необхідно передбачити виведення відповідних повідомлень;

5.2.2 Бази даних повинні бути налаштовані на автоматичне створення резервних копій;

5.2.3 Програмний засіб повинен виконувати свої функції.

5.3 Вимоги до складу і параметрів технічних засобів:

– процесор – Intel Core i5-5300H 2.40GHz і подібні до них;

– оперативна пам’ять – не менше 8Gb;

– середовище функціонування – операційна система сімейство Windows;

– вимоги до техніки безпеки при роботі з програмою повинні відповідати існуючим вимогам та стандартам з техніки безпеки при користуванні комп’ютерною технікою.

6. Вимоги до програмної документації

6.1 Обов’язкова поетапна інструкція для майбутніх користувачів, наведена у пункті 3.4

7. Вимоги до технічного захисту інформації

7.1 Необхідно забезпечити захист розроблюваного програмного засобу від несанкціонованого використання.

7.2 Неможливість отримання доступу незареєстрованих користувачів до інформаційних ресурсів.

8. Техніко–економічні показники

8.1 Цінність результатів використання даного проєкту повинна перевищувати витрати на його реалізацію.

8.2 Має бути реалізований таким чином, щоб підходити для використання широкого загалу.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської дипломної роботи	Початок	Закінчення
1.	Визначення напрямку бакалаврської роботи, формулювання теми	15.02.2024	18.02.2024
2.	Аналіз предметної області обраної теми	19.02.2024	12.03.2024
3.	Розробка алгоритму роботи	13.03.2024	04.04.2024
4.	Написання бакалаврської роботи на основі розробленої теми	06.04.2024	01.05.2024
5.	Передзахист бакалаврської дипломної роботи	06.05.2024	24.05.2024
6.	Виправлення, уточнення, корегування бакалаврської дипломної роботи	01.06.2024	08.06.2024
7.	Захист бакалаврської дипломної роботи	12.06.2024	13.06.2024

10. Порядок контролю та прийому

10.1 До приймання бакалаврської дипломної роботи надається:

- ПЗ до бакалаврської дипломної роботи;
- демонстрація результату бакалаврської дипломної роботи;
- презентація;
- відзив керівника роботи;
- відзив рецензента

Технічне завдання до виконання прийняв _____Анщук О.А.

Додаток Б. Лістинг коду розробленого вебсервісу

```
FROM ubuntu:latest

ENV HOST 0.0.0.0
EXPOSE 5443

RUN apt update && apt install curl libpq–dev clang llvm pkg–config nettle–dev libc6–dev
libssl–dev –y

COPY ./target/release/drive /usr/local/bin

ENV DATA_DIR="/data"
ENV
RUST_LOG="drive=debug,auth=debug,error=debug,entity=debug,storage=debug,context=debug,util=
debug,cryptfns=debug,actix_web=debug"

CMD /usr/local/bin/drive –a 0.0.0.0 –p 5443

___
version: "2"

services:
  postgres:
    image: bitnami/postgresql:latest
    restart: always
    hostname: postgres
    container_name: postgres
    environment:
      – POSTGRESQL_USERNAME=postgres
      – POSTGRESQL_PASSWORD=postgres
      – POSTGRESQL_DATABASE=postgres
      – POSTGRESQL_WAL_LEVEL=logical
    ports:
      – "5432:5432"

[package]

use chrono::Utc;
use entity::{
    paginated::Paginated, sessions, sort::Sortable, users, ActiveValue, ColumnTrait, EntityTrait,
    PaginatorTrait, QueryFilter, QuerySelect, Uuid,
};
use error::{AppResult, Error};
use validr::Validation;

use crate::data::{
    activity_query::ActivityQuery, change_password::ChangePassword, two_factor::Enable,
```

```
};

use super::repository::Repository;

#[async_trait::async_trait]
pub(crate) trait Account
where
   Self: Repository,
{
   /// Verify the payload and change the users password
   async fn change_password(&self, data: ChangePassword) –> AppResult<users::Model> {
      let (email, new_password, encrypted_private_key, current_password, signature, token) =
         data.into_data()?;

      let user = self.get_by_email(&email).await?;

      if !user.verify_tfa(token) {
         return Err(Error::Unauthorized("invalid_otp_token".to_string()));
      }

      verify_password(&user, current_password.as_deref())?;

      verify_signature(&user, &new_password, signature.as_deref())?;

      self.update_user(
         user.id,
         users::ActiveModel {
            password: ActiveValue::Set(Some(util::password::hash(&new_password))),
            encrypted_private_key: ActiveValue::Set(Some(encrypted_private_key)),
            ..Default::default()
         },
      )
      .await
   }

   /// Disable the two factor authentication for the user
   async fn disable_two_factor(&self, id: Uuid, token: Option<String>) –> AppResult<()> {
      let user = self.get_by_id(id).await?;

      if !user.verify_tfa(token) {
         return Err(Error::Unauthorized("invalid_otp_token".to_string()));
      }

      self.update_user(
         user.id,
         users::ActiveModel {
            secret: ActiveValue::Set(None),
            ..Default::default()
         },
```

```
            )
            .await?;

            Ok(())
        }

        /// Enable two factor authentication for the user
        async fn enable_two_factor(&self, id: Uuid, data: Enable) –> AppResult<()> {
            let secret = data.into_value()?;
            let user = self.get_by_id(id).await?;

            if user.secret.is_some() {
                return Err(Error::BadRequest("two_factor_already_enabled".to_string()));
            }

            self.update_user(
                user.id,
                users::ActiveModel {
                    secret: ActiveValue::Set(secret),
                    ..Default::default()
                },
            )
            .await?;

            Ok(())
        }

        /// Load the paginated list of users activity (sessions)
        async fn activity(&self, parameters: ActivityQuery) –> AppResult<Paginated<sessions::Model>>
{
            let parameters = parameters.validate()?;

            let user_id = parameters
                .user_id
                .ok_or_else(|| Error::BadRequest("user_id_is_required".to_string()))?;

            let mut query = sessions::Entity::find().filter(sessions::Column::UserId.eq(user_id));

            if !parameters.with_expired.unwrap_or(false) {
                query = query.filter(sessions::Column::ExpiresAt.gt(Utc::now().timestamp()));
            }

            if let Some(sort) = parameters.sort.as_ref() {
                query = match parameters.order.as_deref() {
                    Some("desc") => sort.sort_desc(query),
                    _ => sort.sort_asc(query),
                };
            }
```

```
            if let Some(search) = parameters.search {
                let maybe_uuid = Uuid::parse_str(search.as_str()).ok();

                if let Some(uuid) = maybe_uuid {
                    query = query.filter(sessions::Column::Id.eq(uuid));
                } else {
                    query = query.filter(
                        sessions::Column::Ip
                            .contains(search.as_str())
                            .or(sessions::Column::DeviceId.contains(search.as_str()))
                            .or(sessions::Column::UserAgent.contains(search.as_str())),
                    );
                }
            }

            let total = query.clone().count(self.connection()).await?;

            query = query.limit(parameters.limit.unwrap_or(15));
            query = query.offset(parameters.offset.unwrap_or(0));

            let sessions = query.all(self.connection()).await?;

            Ok(Paginated::new(sessions, total))
        }
    }

    /// Verify the password
    fn verify_password(user: &users::Model, password: Option<&str>) –> AppResult<()> {
        if let (Some(password), Some(hashed_password)) = (password, &user.password) {
            if !util::password::verify(password, hashed_password) {
                return Err(Error::Unauthorized("invalid_password".to_string()));
            }
        }

        Ok(())
    }

    /// Verify the signature
    fn verify_signature(user: &users::Model, message: &str, signature: Option<&str>) –>
AppResult<()> {
        if let Some(signature) = signature {
            return cryptfns::rsa::public::verify(message, signature, &user.pubkey)
                .map(|_| ())
                .map_err(Error::from);
        }

        Ok(())
    }
```

```
use async_trait::async_trait;
use chrono::Utc;
use entity::{users, ActiveModelTrait, ActiveValue, TransactionTrait, Uuid};
use error::{AppResult, Error};

use crate::{actions::UserActions, data::create_user::CreateUser};

use super::{email::Email, repository::Repository};

/// Register contract implementing all the methods needed for registration
/// and activation of the new users
#[async_trait]
pub(crate) trait Register
where
    Self: Repository + Email,
{
    /// Generate a new 2FA secret
    fn generate_two_factor() –> String {
        util::generate::generate_secret()
    }

    /// Create a new user
    async fn register(&self, data: CreateUser) –> AppResult<users::Model> {
        let email = data.email.clone().unwrap();
        let invitation_id = data.invitation_id;

        let mut active_model = data.into_active_model()?;

        // We can unwrap here because it would fail validation before this
        if self.get_by_email(&email).await.is_ok() {
            return Err(Error::as_validation("email", "invalid_email"));
        }

        if !self.has_sender() {
            active_model.email_verified_at = ActiveValue::Set(Some(Utc::now().timestamp()));
        }

        if let Some(id) = invitation_id {
            let invitation = self.get_invitation(id).await?;

            if invitation.email != email {
                return Err(Error::as_validation("invitation_id", "invalid_invitation"));
            }

            active_model.role = ActiveValue::Set(invitation.role);
            active_model.quota = ActiveValue::Set(invitation.quota);
        } else if self.count_users().await? == 0 {
            active_model.role = ActiveValue::Set(Some("admin".to_string()));
        }
```

```
        let user = self.create_user(active_model).await?;

        self.email_activation(&user).await?;

        Ok(user)
    }

    /// Perform activation of the user
    async fn activate(&self, user_action_id: Uuid) –> AppResult<users::Model> {
        let tx = self.connection().begin().await?;

        let user_action = UserActions::new(&tx);

        let (action, user) = user_action.get_by_id(user_action_id).await?;

        if action.action != "activate–email" {
            return Err(Error::as_not_found("wrong_user_action"));
        }

        if user.email_verified_at.is_some() {
            return Err(Error::as_not_found("email_already_verified"));
        }

        let id = user.id;

        let mut active_model: users::ActiveModel = user.into();

        active_model.email_verified_at = ActiveValue::Set(Some(Utc::now().timestamp()));

        active_model.update(&tx).await?;

        user_action.delete(user_action_id).await?;

        tx.commit().await?;

        self.get_by_id(id).await
    }
}


use chrono::{Duration, Utc};
use entity::{
    sessions, users, ActiveModelTrait, ActiveValue, ColumnTrait, EntityTrait, QueryFilter, Uuid,
};
use error::{AppResult, Error};

use crate::data::authenticated::Authenticated;
```

```
use super::{ctx::Ctx, repository::Repository};

/// Session management contract
#[async_trait::async_trait]
pub(crate) trait Sessions
where
  Self: Ctx + Repository,
{
  /// Generate a new session for a user
  async fn generate(
    &self,
    user: &users::Model,
    user_agent: &str,
    ip: &str,
  ) –> AppResult<sessions::Model> {
    let expires_at = Utc::now()
      + Duration::seconds(self.ctx().config.auth.short_term_session_duration_seconds);

    let id = entity::Uuid::new_v4();

    let active_model = sessions::ActiveModel {
      id: ActiveValue::Set(id),
      user_id: ActiveValue::Set(user.id),
      device_id: ActiveValue::Set(Uuid::new_v4()),
      ip: ActiveValue::Set(ip.to_string()),
      user_agent: ActiveValue::Set(user_agent.to_string()),
      refresh: ActiveValue::Set(Some(Uuid::new_v4())),
      created_at: ActiveValue::Set(Utc::now().timestamp()),
      updated_at: ActiveValue::Set(Utc::now().timestamp()),
      expires_at: ActiveValue::Set(expires_at.timestamp()),
    };

    sessions::Entity::insert(active_model)
      .exec_without_returning(self.connection())
      .await?;

    let result = sessions::Entity::find_by_id(id)
      .one(self.connection())
      .await?;

    result.ok_or(Error::NotFound("session_not_found".to_string()))
  }

  /// Refresh session, if it's not expired. Refreshing a session will extend the expiration date by N
minutes.
  async fn refresh(&self, session: &sessions::Model) –> AppResult<Authenticated> {
    let expires_at = Utc::now().naive_utc()
      + Duration::seconds(self.ctx().config.auth.short_term_session_duration_seconds);
```

```
    let active_model = sessions::ActiveModel {
        id: ActiveValue::Set(session.id),
        user_id: ActiveValue::Set(session.user_id),
        device_id: ActiveValue::Set(session.device_id),
        ip: ActiveValue::Set(session.ip.clone()),
        user_agent: ActiveValue::Set(session.user_agent.clone()),
        refresh: ActiveValue::Set(Some(Uuid::new_v4())),
        created_at: ActiveValue::Set(session.created_at),
        updated_at: ActiveValue::Set(Utc::now().timestamp()),
        expires_at: ActiveValue::Set(expires_at.timestamp()),
    };

    active_model.update(self.connection()).await?;

    self.get_by_device_id(session.device_id).await
}

/// Perform the logout action
async fn destroy(&self, session: &sessions::Model) –> AppResult<Authenticated> {
    let active_model = sessions::ActiveModel {
        id: ActiveValue::Set(session.id),
        user_id: ActiveValue::Set(session.user_id),
        device_id: ActiveValue::Set(session.device_id),
        ip: ActiveValue::Set(session.ip.clone()),
        user_agent: ActiveValue::Set(session.user_agent.clone()),
        refresh: ActiveValue::Set(None),
        created_at: ActiveValue::Set(session.created_at),
        updated_at: ActiveValue::Set(Utc::now().timestamp()),
        expires_at: ActiveValue::Set(Utc::now().timestamp()),
    };

    let session = active_model.update(self.connection()).await?;

    self.get_by_session_id(session.id).await
}
async fn destroy_all(&self, id: Uuid, user_id: Uuid) –> AppResult<()> {
    let active_model = sessions::ActiveModel {
        refresh: ActiveValue::Set(None),
        updated_at: ActiveValue::Set(Utc::now().timestamp()),
        expires_at: ActiveValue::Set(Utc::now().timestamp()),
        ..Default::default()
    };

    sessions::Entity::update_many()
        .filter(sessions::Column::Id.ne(id))
        .filter(sessions::Column::UserId.eq(user_id))
        .filter(sessions::Column::Refresh.is_not_null())
        .set(active_model)
        .exec(self.connection())
```

```
                .await?;

            Ok(())
        }
        async fn get(&self, id: Uuid, user_id: Uuid) –> AppResult<sessions::Model> {
            let session = sessions::Entity::find_by_id(id)
                .one(self.connection())
                .await?
                .ok_or(Error::NotFound("session_not_found".to_string()))?;

            if session.user_id != user_id {
                return Err(Error::NotFound("session_not_found".to_string()));
            }

            Ok(session)
        }
    }
import { defineConfig } from 'cypress'
import * as fs from 'fs'
import { resolve } from 'path'
import { config } from 'dotenv'

function getEnvPath() {
  if (process.env.ENV_FILE && fs.existsSync(resolve(process.env.ENV_FILE))) {
    return resolve(process.env.ENV_FILE)
  }

  if (fs.existsSync(resolve('../.env'))) {
    return resolve('../.env')
  }

  return resolve('../.env.e2e')
}

function loadDotenv(): { [key: string]: string } {
  const path = getEnvPath()

  const data =
    config({
      path
    }).parsed || {}

  if (!data.APP_CLIENT_URL) {
    data.APP_CLIENT_URL = data.APP_URL
  }

  return data
}
```

```
export default defineConfig({
  env: loadDotenv(),
  e2e: {
    setupNodeEvents(on, config) {
      return config
      // implement node event listeners here
    }
  },
  video: false
})
/// <reference types="vite/client" />
/// <reference path="cryptfns/cryptfns.d.ts" />
/// <reference types="vite–plugin–pwa/client" />

import type Api from './stores/api'

declare global {
  interface Window {
    __IDENTITY: string | undefined
    defaultDocumentTitle: string
    UPLOAD: Worker
    DOWNLOAD: Worker
    CRYPTO: Worker
    SWApi: Api
    canceled: {
      upload: string[]
      download: string[]
    }
  }
}
module.exports = {
  plugins: [require('postcss–import'), require('tailwindcss'), require('autoprefixer')]
}

import { downloadAndDecryptStream } from './services/storage/workers'
import { uploadFile } from './services/storage/workers/file'
import Api, { ErrorResponse, type ApiTransfer } from './services/api'
import * as cryptfns from './services/cryptfns'
import * as logger from '!/logger'

import type {
  DownloadCompletedResponseMessage,
  DownloadFileMessage,
  DownloadProgressResponseMessage,
  AppFile,
  UploadAppFile,
  UploadChunkResponseMessage,
  UploadFileMessage,
  WorkerErrorType
```

```
} from './types'

const sleep = (s: number) => new Promise((r) => setTimeout(r, s * 1000))

self.canceled = {
  upload: [],
  download: []
}

handleApiTransfer(apiTransfer?: ApiTransfer) {
  if (apiTransfer && apiTransfer.apiUrl) {
    logger.debug('Setting the SWApi...', apiTransfer.apiUrl)

    const api = new Api(apiTransfer)
    self.SWApi = api
  } else {
    logger.warn('Missing apiTransfer...')
  }
}

onmessage = async (message: MessageEvent<any>) => {
  logger.debug('In worker, receiving message', message.data?.type || 'unknown')

  // Handle ping messages from the main thread
  if (message.data?.type === 'ping') {
    self.__IDENTITY = message.data?.name || undefined

    postMessage({ type: 'pong' })
  }

  // Crypto messages
  if (message.data?.type === 'encrypt' || message.data?.type === 'decrypt') {
    handleCrypto(
      message.data.type,
      message.data.message.id,
      message.data.message.data,
      message.data.message.key
    )
  }

  // Creating api maker with the updated credentials received
  // from the main browser thread that has access to JWT and CSRF
  if (message.data?.type === 'auth') {
    handleApiTransfer(message.data.apiTransfer)
  }

  if (message.data?.type === 'cancel') {
    const type = message.data.kind
```

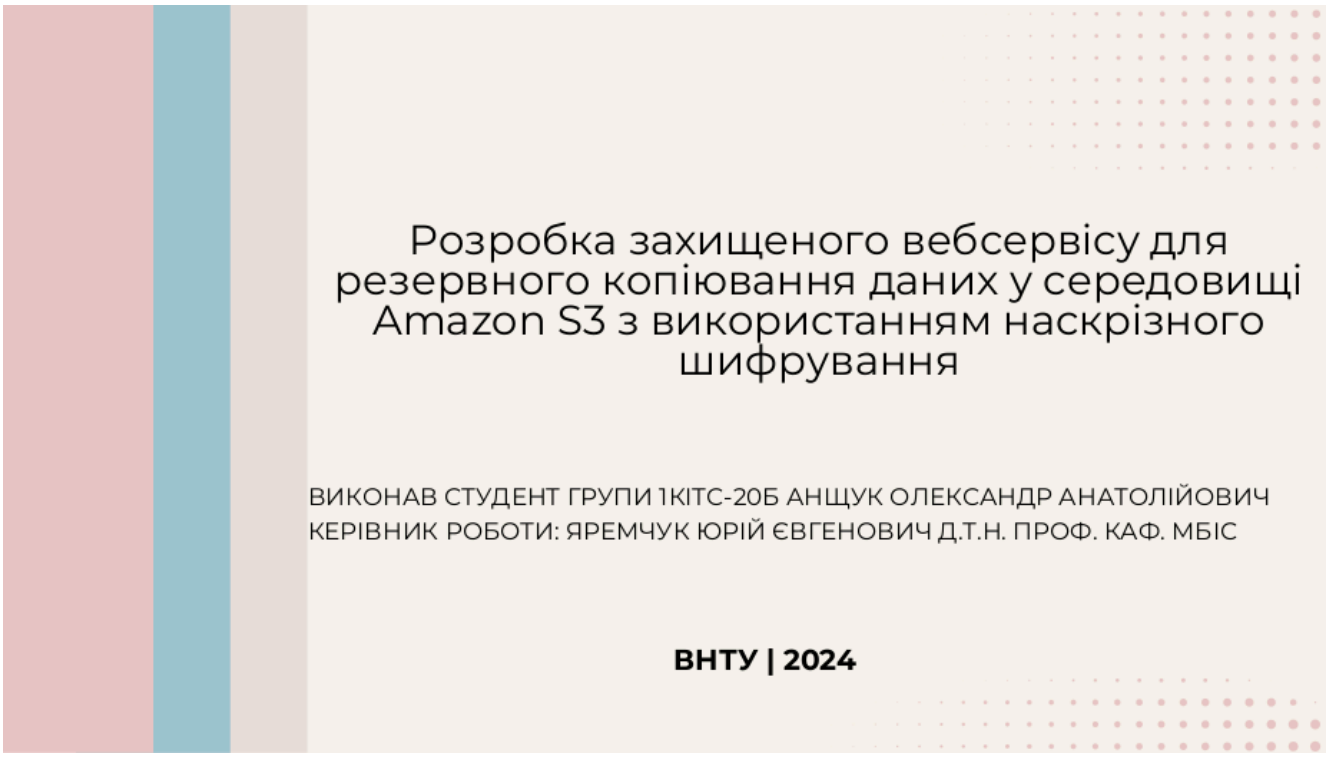
```
    if (type === 'upload') {
      self.canceled.upload.push(message.data.id)
    } else if (type === 'download') {
      self.canceled.download.push(message.data.id)
    }
  }

  if (message.data?.type === 'upload–file') {
    handleApiTransfer(message.data.apiTransfer)

    while (!self.SWApi) {
      logger.warn('Waiting for SWApi to be initialized with credentials and start uploading...')
      await sleep(1)
    }

    handleUploadFile(message.data.message)
  }

  if (message.data?.type === 'download–file') {
    handleApiTransfer(message.data.apiTransfer)

    while (!self.SWApi) {
      logger.warn('Waiting for SWApi to be initialized with credentials and start downloading...')
      await sleep(1)
    }

    handleDownloadFile(message.data.message)
  }
}

async function handleCrypto(
  type: 'encrypt' | 'decrypt',
  id: string,
  data: Uint8Array,
  key: Uint8Array
) {
  logger.debug('In worker, handling crypto', type, id)

  const fn = type === 'encrypt' ? cryptfns.aes.encrypt : cryptfns.aes.decrypt

  const result = await fn(data, key)

  logger.debug('In worker, handling crypto, posting message', type, id)

  postMessage({
    type,
    message: { id, result }
  })
}
```

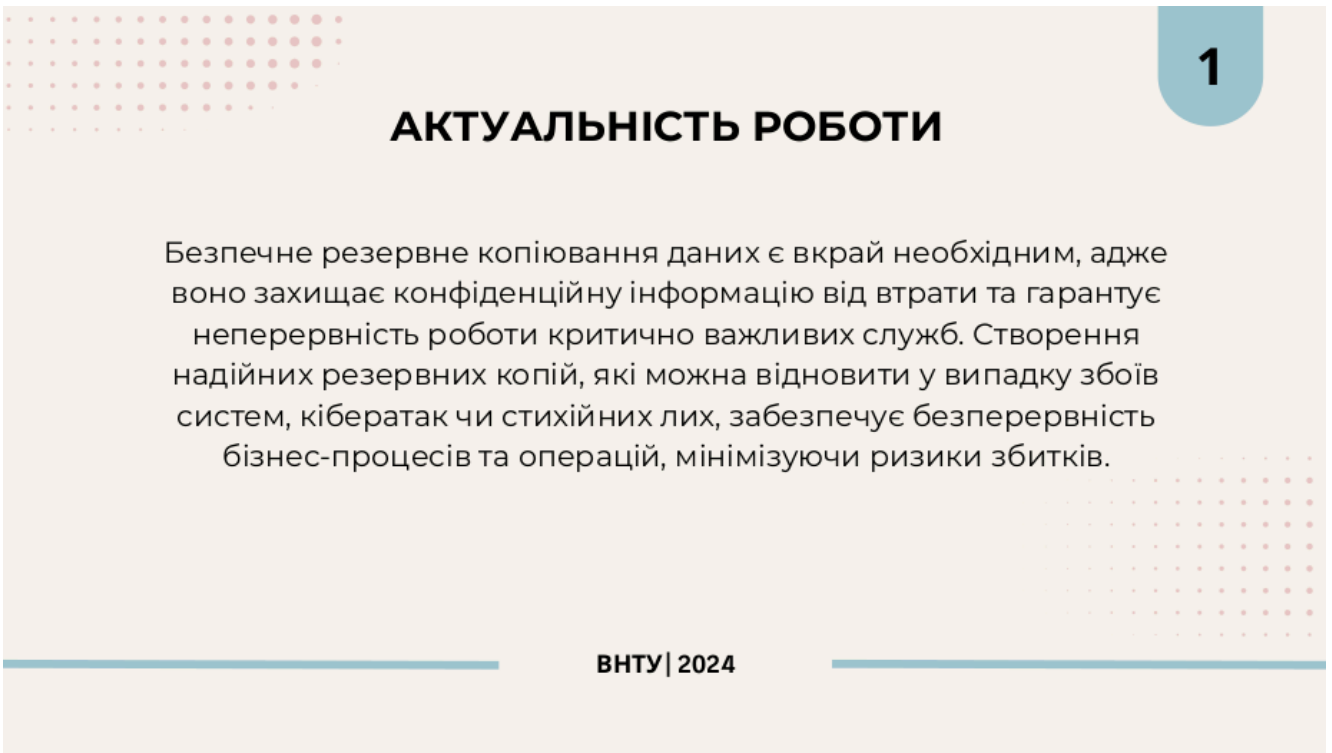
```
async function handleUploadFile({
 transferableUploadedChunks,
 transferableFile
}: UploadFileMessage) {
 const file = transferableFile as UploadAppFile
 file.uploaded_chunks = transferableUploadedChunks as unknown as number[]

 const progress = (
  file: UploadAppFile,
  attempt: number,
  isDone: boolean,
  error?: Error | ErrorResponse<any> | string | undefined
 ) => {
  postMessage({
   type: 'upload–progress',
   response: {
    transferableFile: file,
    attempt: attempt || 0,
    isDone,
    error: handleError(error)
   } as UploadChunkResponseMessage
  })
 }

 try {
  await uploadFile(self.SWApi, file, progress)
 } catch (error) {
  logger.error('In worker error', error)
  progress(file, 0, false, error as ErrorResponse<unknown>)
 }
}
async function handleDownloadFile({ transferableFile }: DownloadFileMessage) {
 try {
  const response = await downloadAndDecryptStream(
   self.SWApi,
   transferableFile,
   async (file: AppFile, chunkBytes: number): Promise<void> => {
    postMessage({
     type: 'download–progress',
     response: {
      transferableFile: file,
      chunkBytes
     } as DownloadProgressResponseMessage
    })
   }
  )

  postMessage({
   type: 'download–completed',
```

```
      response: {
        transferableFile,
        blob: await response.blob()
      } as DownloadCompletedResponseMessage
    })
  } catch (err) {
    const error = err as ErrorResponse<unknown>

    postMessage({
      type: 'download–progress',
      response: {
        transferableFile,
        chunkBytes: 0,
        error: handleError(error)
      } as DownloadProgressResponseMessage
    })
  }
}
function handleError(error?: undefined | Error | ErrorResponse<any> | string): WorkerErrorType {
  if (!error) return

  if (typeof error === 'string') {
    return { context: error }
  }

  if (error instanceof Error) {
    return {
      context:
        (error as ErrorResponse<unknown>).validation ||
        (error as ErrorResponse<unknown>).description ||
        error.message,
      stack: error.stack
    }
  }
}
pub fn get_token() -> Option<String> {
    let token_lock = TOKEN.read();
    token_lock.clone()
}
pub async fn request<B, T>(
    method: Method,
    url: String,
    body: B
) -> Result<T, ServiceError> where
        T: DeserializeOwned + 'static + Debug,
        B: Serialize + Debug  {
            let api_url: &str = &std::env::var("API_URL").expect("You must set an API KEY");
            let allowed_body = method == reqwest::Method::POST || method == reqwest::Method::PUT;
            let url = format!("{}{}", api_url, url);
```

```
        let mut builder = reqwest::Client::new()
            .request(method, url)
            .header("Content-Type", "application/json");
        if let Some(token) = get_token() {
            builder = builder.bearer_auth(token)
        }

        if allowed_body {
            builder = builder.json(&body)
        }

        let response = builder.send().await;

        if let Ok(data) = response {
            if data.status().is_success() {
                let data: Result<T, _> = data.json::<T>().await;
                if let Ok(data) = data {
                    log::debug!("Response: {:?}", data);
                    Ok(data)
                } else {
                    Err(ServiceError::DeserializeError)
                }
            } else {
                match data.status().as_u16() {
                    401 => Err(ServiceError::UnAuthorised),
                    403 => Err(ServiceError::Forbidden),
                    404 => Err(ServiceError::NotFound),
                    500 => Err(ServiceError::InternalServerError),
                    422 => {
                        let data: Result<ErrorInfo, _> = data.json::<ErrorInfo>().await;
                        if let Ok(data) = data {
                            Err(ServiceError::UnprocessableEntity(data))
                        } else {
                            Err(ServiceError::DeserializeError)
                        }
                    }
                    _ => Err(ServiceError::RequestError),
                }
            }
        } else {
            Err(ServiceError::RequestError)
        }
}
```

Додаток В. Ілюстрований матеріал





2

МЕТА РОБОТИ

Мета роботи полягає у розробці захищеного вебсервісу для резервного копіювання даних у хмарному середовищі Amazon S3 з використанням наскрізного шифрування, що забезпечить високий рівень безпеки та конфіденційності зберігання інформації.

ВНТУ | 2024

3

ОСНОВНІ АСПЕКТИ РОЗРОБКИ ВЕБСЕРВІСУ

1. Хмарне сховище Amazon S3;
2. Криптографічна система RSA;
3. Модель AAA;

ВНТУ | 2024

4

AMAZON S3

Amazon S3 (Simple Storage Service) - це хмарне сховище об'єктів від Amazon Web Services, яке забезпечує доступне з будь-якого місця та недороге зберігання необмеженої кількості даних у вигляді об'єктів.

5

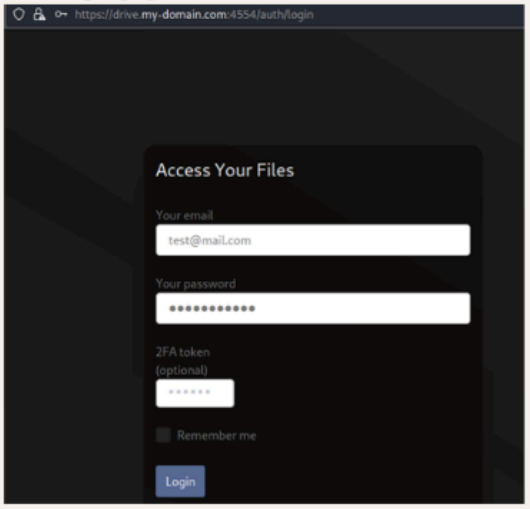
РЕАЛІЗАЦІЯ ШИФРУВАННЯ

6

АЛГОРИТМ РЕЄСТРАЦІЇ КОРИСТУВАЧА

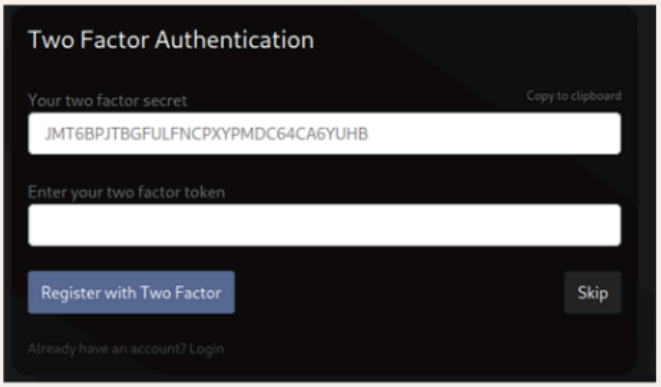
7

ПРОЦЕДУРА АВТОРИЗАЦІЇ



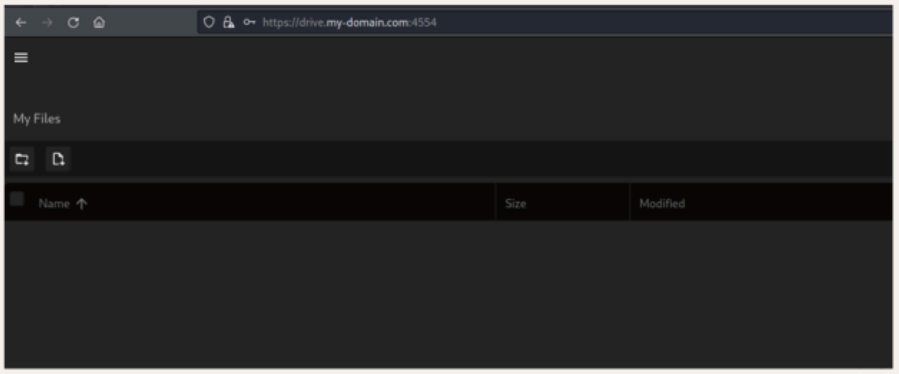
8

ФОРМА НАЛАШТУВАННЯ 2FA



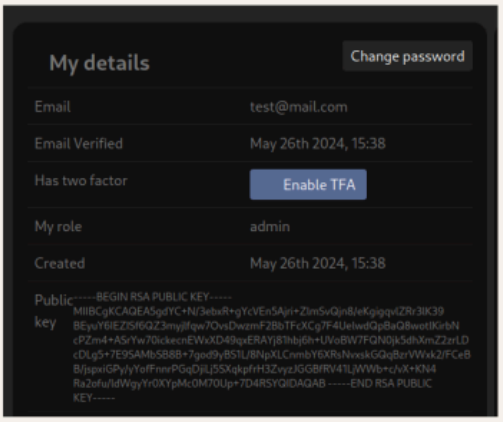
9

ГОЛОВНА СТОРІНКА ВЕБСЕРВІСУ



10

СТОРІНКА НАЛАШТУВАНЬ КОРИСТУВАЧА



11

ВИСНОВОК

У результаті виконання даної роботи було реалізовано захищений вебсервісу для резервного копіювання даних у хмарному середовищі Amazon S3 з використанням наскрізного шифрування.

ДЯКУЮ ЗА УВАГУ!

Додаток В. Протокол перевірки на антиплагіат

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка захищеного вебсервісу для резервного копіювання даних у середовищі Amazon S3 з використанням наскрізного шифрування

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ: Кафедра менеджменту та безпеки інформаційних систем
Факультет менеджменту та інформаційної безпеки
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 99 %

Схожість 1 %

Аналіз звіту подібності (відмітити потрібне):

1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

3. Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Коваль Н.П.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Анщук О.А.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Яремчук Ю.Є.
(підпис) (прізвище, ініціали)