Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Розробка захищеного файлообмінника з використанням криптоалгоритму AES, двофакторної автентифікації та автоматичного видалення файлів після встановленого періоду часу

Виконала: студентка 4 курсу, гр.1КІТС-21б
спеціальності 125– Кібербезпека
Освітня програма – Кібербезпека
інформаційних технологій та систем
(шифр і назва напряму підготовки, спеціальності)

Безверха Т.М.
(прізвище та ініціали)

Керівник: д.ф.(PhD), доц., доцент каф. МБІС
Салієва О.В.
(прізвище та ініціали)
«___» _____ 2025 р.

Рецензент: к.т.н., доц, каф. ОТ
Крупельницький Л.В
(прізвище та ініціали)
«___» _____ 2025 р.

Допущено до захисту
Голова секції УБ кафедри МБІС
д.т.н., проф. Яремчук Ю.Є.
“____” _____ 2025р.

Вінниця ВНТУ – 2025

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

Рівень вищої освіти І-й (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 125 – Кібербезпека
Освітньо-професійна програма - Кібербезпека інформаційних технологій та систем

ЗАТВЕРДЖУЮ
Голова секції УБ, кафедра МБІС
д.т.н., проф. Яремчук Ю.Є.
" 20 " березня 2025 р.

З А В Д А Н Н Я
на бакалаврську кваліфікаційну роботу студенту
Безверхій Тамарі Михайлівні
(прізвище, ім`я, по-батькові)

1. Тема роботи Розробка захищеного файлообмінника з використанням криптоалгоритму AES, двофакторної автентифікації та автоматичного видалення файлів після встановленого періоду часу
Керівник роботи д.ф. (PhD), доцент каф. МБІС Салієва Ольга Володимирівна
(прізвище, ім`я, по-батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від " 20 " березня 2025 року № 97
2. Термін подання студентом роботи «____» _____2025 р.
3. Вихідні дані до роботи:
Вимоги до захищених систем обміну файлами у державних та корпоративних структурах; Алгоритм симетричного шифрування AES-256; Методи реалізації двофакторної автентифікації (2FA) та механізми автоочищення даних; Практичні вимоги до розробки web-застосунків з функціональністю захисту файлів.

4. Зміст текстової частини:
У роботі описано етапи проєктування, розробки та впровадження механізму автоматичного видалення файлів у системі захищеного файлообміну. Надано детальний опис реалізації основних модулів, алгоритмів та логіки роботи програми, а також результати тестування розробленої системи.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов`язкових креслень)
– Рисунок 1.1 – Схема архітектури захищеного файлообмінника;
– Рисунок 1.2 – Схема атаки типу Man-in-the-Middle;

– Таблиця 1.2 – Порівняльна характеристика сучасних файлообмінників;
– Рисунок 2.1 – Блок-схема алгоритму функціонування системи;
– Рисунок 2.2 – Алгоритм автоматичного видалення файлів;
– Діаграма структури бази даних (ER-модель);
– Скріншоти інтерфейсу користувача.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ I	Саліва О.В. д.ф., доцент каф. МБІС		
Розділ II	Саліва О.В. д.ф., доцент каф. МБІС		
Розділ III	Саліва О.В. д.ф., доцент каф. МБІС		

7.Дата видачі завдання 20 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Визначення напрямку бакалаврської роботи, формулювання теми	20.03.2025	23.03.2025	Виконано
2	Аналіз предметної області обраної теми	24.03.2025	13.04.2025	Виконано
3	Розробка алгоритму роботи	14.04.2025	27.04.2025	Виконано
4	Написання бакалаврської роботи на основі розробленої теми	28.04.2025	25.05.2025	Виконано
5	Передзахист бакалаврської кваліфікаційної роботи	26.05.2025	27.05.2025	Виконано
6	Виправлення, уточнення, корегування бакалаврської дипломної роботи	28.05.2025	08.06.2025	Виконано
7	Захист бакалаврської кваліфікаційної роботи	09.06.2025	10.06.2025	Виконано

Студент _____ (підпис) Безверха Т.М. (прізвище та ініціали)

Керівник роботи _____ (підпис) Салієва О.В. (прізвище та ініціали)

АНОТАЦІЯ

У роботі розглянуто процес розробки захищеного файлообмінника, який поєднує сучасні методи криптографічного захисту з елементами багаторівневої безпеки. Основна увага приділена впровадженню алгоритму шифрування AES-256, реалізації двофакторної автентифікації (2FA), а також створенню механізму автоматичного видалення файлів після завершення встановленого терміну зберігання.

Метою роботи є створення програмного засобу, що дозволяє безпечно обмінюватися файлами через інтернет, враховуючи актуальні загрози інформаційній безпеці. Проведено аналіз існуючих рішень у цій галузі, досліджено сильні та слабкі сторони поширених файлообмінників, реалізовано функціональну архітектуру системи з шифруванням даних на клієнтському рівні, захистом доступу та контролем часу зберігання.

Запропоноване рішення може бути використане як в державному, так і в приватному секторі, особливо в умовах потреби у передачі конфіденційної інформації. Практична цінність розробки полягає в її універсальності, високому рівні безпеки та можливості адаптації під потреби конкретних організацій або систем.

Ключові слова: файлообмінник, AES-256, інформаційна безпека, криптографія, двофакторна автентифікація, автоматичне видалення файлів, шифрування.

ABSTRACT

The paper describes the process of developing a secure file-sharing service that combines modern methods of cryptographic protection with elements of multi-level security. The main attention is paid to the implementation of the AES-256 encryption algorithm, the implementation of two-factor authentication (2FA), and the creation of a mechanism for automatically deleting files after the expiration of the set storage period.

The aim of the work is to create a software tool that allows secure file sharing over the Internet, taking into account current threats to information security. An analysis of existing solutions in this area was carried out, the strengths and weaknesses of common file sharing services were investigated, and the functional architecture of the system with data encryption at the client level, access protection, and storage time control was implemented.

The proposed solution can be used in both the public and private sectors, especially when there is a need to transfer confidential information. The practical value of the development lies in its versatility, high level of security, and the ability to adapt to the needs of specific organizations or systems.

Keywords: file sharing, AES-256, information security, cryptography, two-factor authentication, automatic file deletion, encryption.

ЗМІСТ

ВСТУП

Актуальність роботи. З розвитком цифровізації інформації збільшується і обсяг її передачі, що у свою чергу вимагає її якісного захисту. Одним зі шляхів вирішення даного питання є розробка захищених системи обміну файлами, яка забезпечить користувачам зручність та захищеність у передачі інформації.

Важливість розробки полягає у зростанні потреби у безпечних засобах обміну файлами адже як показує статистика, з кожним роком кількість кіберзагроз збільшується у геометричній прогресії. На відміну від існуючих рішень, запропонована розробка містить у собі кілька захисних механізмів: шифрування AES-256, двофакторну автентифікацію, а також автоматичне видалення файлів, що підвищує його безпеку порівняно з традиційними системами.

Метою роботи є створення захищеного файлообмінника за допомогою сучасних методів криптографії та багаторівневого захисту.

Для досягнення цієї мети необхідно вирішити такі **задачі:**

- проаналізувати сучасні методи передачі файлів
- розробити механізм шифрування файлів за допомогою AES-256;
- інтегрувати двофакторну автентифікацію;
- реалізувати функцію автоматичного видалення файлів;
- забезпечити інтуїтивно зрозумілий інтерфейс для користувачів.

Предметом дослідження є процес захищеного обміну файлами в інформаційно-комунікаційних системах

Об’єктом дослідження є програмні засоби реалізації захищеного файлообміну з використанням криптоалгоритму AES-256, двофакторної автентифікації (2FA) та механізму автоматичного видалення файлів.

Наукова розробки полягає в поєднанні високого рівня шифрування, додаткового захисту через двофакторну автентифікацію та автоматичної ліквідації даних після завершення терміну зберігання. Це забезпечує стійкість

системи до атак, знижує ризики несанкціонованого доступу і покращує конфіденційність даних, роблячи процес обміну файлами безпечнішим та зручнішим для користувачів.

Практична цінність роботи полягає у розробці програмного засобу, який забезпечує захищений обмін файлами шляхом інтеграції сучасних методів захисту інформації – криптоалгоритму AES, двофакторної автентифікації та автоматичного видалення файлів після встановленого часу.

Запропоноване рішення може бути використане в державних установах, приватних компаніях або організаціях, які обробляють конфіденційну інформацію, а також як основа для подальших досліджень у сфері інформаційної безпеки.

1 АНАЛІЗ ІСНУЮЧИХ ФАЙЛООБМІННИКІВ

У даному розділі розглянуто сучасний стан проблеми захищеної передачі файлів через мережу Інтернет. Проведено аналіз існуючих підходів безпечного обміну інформацією.

Детально обґрунтовано важливість та доцільність розробки захищеного файлообмінника з використанням криптоалгоритму AES-256, двофакторної автентифікації та автоматичного видалення файлів після встановленого періоду часу. Проаналізовано існуючі методи реалізації розробки.

1.1 Сучасний стан реалізації захищених файлообмінників

Спосіб, технології захищеного файлообміну стосуються шифрування та заходів безпеки для захисту даних під час їх передачі. Вони використовуються для захисту даних під час транзиту за допомогою кодування, перевірки, обмежень доступу та аудиту під час обміну файлами як всередині компаній, так і між ними.

Розглянемо способи обміну інформацією між компаніями. B2B транзакція, також відома як транзакція "B-to-B", являє собою обмін даними між компаніями. Вони доступні для малих і середніх підприємств, а також для великих компаній і можуть реалізовуватись у різних моделях, таких як локальна, хмарна або гібридна.

Такі технології використовуються в різних галузях, таких як банківська справа, фінансові послуги та страхування, охорона здоров’я, виробництво, логістика, роздрібна торгівля, медіа та розваги, IT та телекомунікації, урядові установи тощо.

Розмір ринку безпечної передачі файлів зростав у останні роки. Він зріс з 2,08 мільярда доларів у 2023 році до 2,28 мільярда доларів у 2025 році з середньорічною темпом зростання (CAGR) 10,0%. Зростання в історичний період можна пояснити занепокоєннями щодо безпеки даних, попитом на

автоматизацію, глобалізацією та обміном даними, інцидентами витоку даних, інтеграцією хмарних послуг [1].

Даний прогностичний середньорічний темп зростання ринку безпечної передачі файлів зображено на рисунку 1.1.

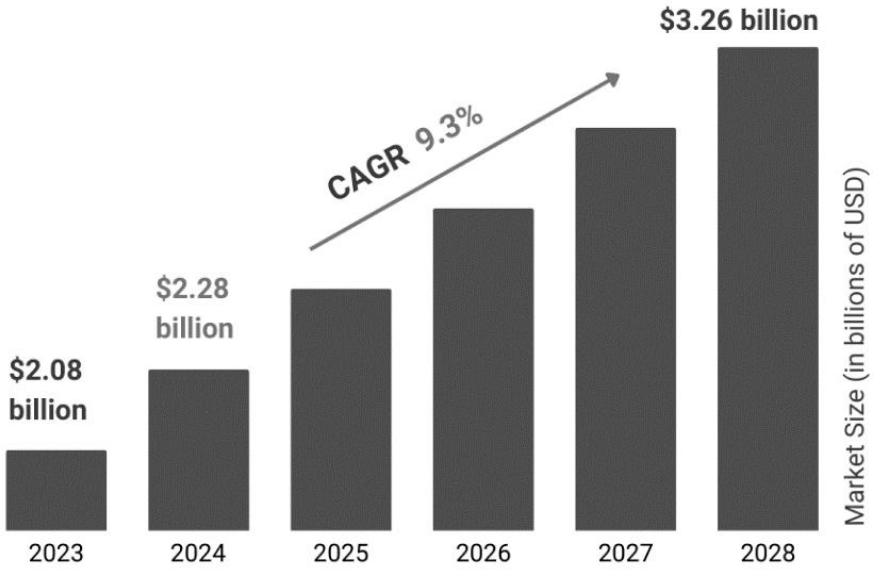


Рисунок 1.1 – Середньорічне зростання ринку безпечної передачі файлів

Існує кілька популярних платформ та протоколів для обміну файлами, серед яких хмарні сервіси, такі як Dropbox, OneDrive та Google Drive (Driveuploader). Ці платформи дозволяють користувачам зберігати файли онлайн і ділитися ними, що значно спрощує співпрацю з іншими людьми.

Наприклад, завдяки таким сервісам користувачі можуть легко надавати доступ до документів, працювати над спільними проектами або просто обмінюватися важливими файлами з будь-якої точки світу.

Розглянемо способи передачі файлів у мережі та їх особливості.

BitTorrent – клієнтська програма мережі, націленої на можливість завантаження файлів великого розміру з файлообмінної мережі BitTorrent. На відміну від інших подібних мереж (Kazaa, eDonkey тощо), де власниками одного й того самого файлу можуть бути одразу кілька людей, у яких цей файл перебуває, BitTorrent передбачає наявність у файлу єдиного власника, який і зацікавлений у його поширенні [2].

Його особливість закладається у високій швидкості скачування файлу, що досягається за рахунок оригінального способу видачі файлу одразу з декількох комп'ютерів, навіть у тому випадку, коли цілком файлу на них ще немає. Причому, через цю мережу поширюються файли дуже великого розміру: фільми, ігри, CD і DVD образи дисків тощо.

Також, поширеним протоколом для передачі файлів є FTP (File Transfer Protocol). Він здебільшого використовується для передачі файлів між серверами, наприклад, під час розгортання веб-сайтів чи передачі великих масивів даних в корпоративних мережах. FTP дозволяє організувати швидкий і надійний обмін файлами, але його основним недоліком є недостатній рівень захищеності, тому для захисту даних все частіше використовуються більш безпечні варіанти, такі як SFTP (SSH File Transfer Protocol).

Незважаючи на численні переваги, обмін файлами також пов’язаний із серйозними ризиками для безпеки даних. Однією з найбільших загроз є можливість витоку конфіденційної інформації, яка може статися, коли файли передаються або зберігаються без відповідного захисту.

Якщо під час передачі файлів не використовуються шифрування та інші заходи безпеки, зловмисники можуть перехопити їх, отримуючи доступ до вмісту, навіть якщо він конфіденційного характеру. Крім того, файли, що зберігаються на хмарних платформах, можуть бути вразливими до кібератак, коли хакери знаходять спосіб обійти безпеку та отримати несанкціонований доступ до даних користувача. Як результат, стає все більш важливим використовувати сучасні методи безпеки, такі як шифрування даних, двофакторна автентифікація та інші підходи, щоб зменшити ризик витоку інформації під час обміну файлами.

Щоб зменшити ці ризики, багато платформ і протоколів для обміну файлами впровадили такі заходи безпеки, як шифрування, двофакторна автентифікація та контроль доступу. Однак важливо, щоб користувачі також мали відповідальне ставлення до власної безпеки, використовуючи надійні

паролі, уникаючи публічних мереж Wi-Fi та обмінюючись файлами лише з довіреними особами [3].

Уряд Великої Британії визнав необхідність безпечного обміну файлами в державному секторі та встановив конкретні стандарти для забезпечення захисту конфіденційної інформації. Дотримання цих стандартів має вирішальне значення для державних установ, щоб зберегти цілісність, конфіденційність і доступність даних, а також сприяти співпраці та ефективній роботі:

– конфіденційність – забезпечення доступу до інформації лише уповноваженим особам або організаціям;

– цілісність – гарантія того, що дані залишаються незмінними та неспотвореними протягом усього процесу обміну;

– доступність – гарантія того, що файли доступні, коли це необхідно, без несанкціонованих збоїв або затримок;

– підзвітність – відповідальність окремих осіб або відділів за їхні дії, пов'язані з обміном файлами та захистом даних.

Стандарт ISO 27001 пропонує структурований підхід до виявлення та оцінювання ризиків інформаційної безпеки, в тому числі ризиків, пов'язаних з обміном файлами. Провівши ретельну оцінку ризиків, організації можуть зрозуміти потенційні вразливості та загрози, пов'язані з обміном файлами. Це дозволяє організаціям впроваджувати відповідні засоби контролю та захисту для ефективного зниження цих ризиків [4].

Схиляючись на тенденції сьогодення у безпечному обміні інформацією у інтернет-просторі, можна зробити висновок, що необхідність якісно захищених файлообмінників зростає з кожним роком все більше. Враховуючи аспект того, що більшість компаній переходять до онлайнового режиму роботи. Зважаючи на значний прогрес у захисті інформації, він є рівно пропорційним потребі постійного вдосконалення надійності під час обміну файлами. У цьому аспекті є доцільність у детальному визначенні основних загроз та вразливостей під час обміну інформацією онлайн.

1.2 Огляд основних загроз та вразливостей під час обміну файлами через мережу Інтернет

Найпоширенішою загрозою під час передачі даних у мережі є атака Man-in-the-Middle, або так звана “людина по середині”. Вона являє собою атаку, де хакер таємно отримує доступ до інформації між двома сторонами, де останні думають, що вони спілкуються безпечно один з одним. Дану загрозу було зображено схематично на рисунку 1.2.

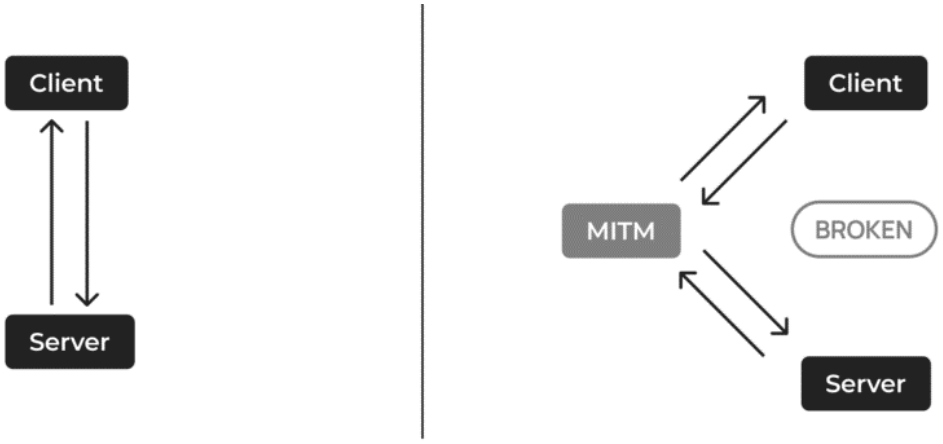


Рисунок 1.2 – Схематичне зображення атаки типу Man-in-the-Middle

На початковому етапі зловмисник перехоплює активність клієнта через систему зловмисника до того, як вона досягне місця призначення. Найвідомішим (і найпростішим) методом для цього є неактивна атака, в якій зловмисник створює безкоштовні/відкриті точки доступу до Wi-Fi, доступні широкому загалу. Зазвичай вони називаються так, як це пов'язано з їхнім регіоном, і не є надійно захищеними. Як тільки жертва взаємодіє з такою точкою доступу, зловмисник отримує повний доступ до взаємодії з інформацією в Інтернеті [5].

Разом з кількістю атак зростають і заходи захисту інформаційних потоків. VPN можна використовувати для створення безпечного середовища для конфіденційної інформації в локальній мережі. Вони використовують шифрування на основі ключів для створення підмережі для безпечного зв'язку.

Таким чином, навіть якщо зловмисник потрапить у загальну мережу, він не зможе розшифрувати трафік у VPN [6].

Також, відомий HTTPS (зашифрований у цілях безпеки протокол), який використовує публічно-приватний обмін ключами. Це завадить хакеру отримати інформацію і відповідно дані будуть захищені.

Наступним важливим аспектом є ризик витоку даних внаслідок несанкціонованого доступу. Зловмисники можуть отримати доступ до файлів, які знаходяться на сервері або хмарному середовищі через встановлений слабкий пароль та\або відсутність двофакторної ідентифікації.

Порушення цілісності даних може вплинути на всі види бізнесу та людей, чиї дані передаються або зберігаються в Інтернеті. У сучасному світі цілісність даних значною мірою залежить від IT-інфраструктури компаній. Дані захищені окремими законами, є секретними та конфіденційними і мають певну цінність з різних точок зору. Незважаючи на те, що, ймовірно, завжди буде існувати значна потреба у фахівцях з кібербезпеки, важко передбачити ані кількість працівників, ані ідеальну комбінацію їхніх знань та навичок у сфері кібербезпеки. Тому критично важлива інформація повинна бути постійно захищена, а паролі повинні бути складними для підбору хакерами. Компанії повинні завжди підтримувати довіру клієнтів і використовувати найсучасніші технології для запобігання кіберзагрозам і витоку даних [7].

Зберігають свою тенденцію фішинг та шкідливе програмне забезпечення. Шкідливим ПЗ вважають те, що використовується для отримання неавторизованого доступу до системи з метою крадіжки даних або її спотворення. Фішинговими атаками хакери надають хибне посилання користувачеві, яке у свою чергу веде до зараженого файлу. Важливу роль тут відіграє людський фактор, який є однією з найбільших переваг для хакерів.

Було заблоковано понад 126 мільйонів екземплярів шкідливого програмного забезпечення або бот-мереж, розроблених спеціально для фінансових крадіжок, що підкреслює фінансову мотивацію багатьох кібератак. DDoS-атаки залишалися основною загрозою для клієнтів Comcast Business,

про які було зареєстровано 103 000 випадків. Цей сплеск підкреслює необхідність надійного захисту від DDoS-атак та стратегій пом'якшення їх наслідків [8].

Вагомою загрозою у передачі файлів є витік інформації. ЇЇ може спричинити співробітник або інсайдер, що має доступ до конфіденційних файлів і в свою чергу може навмисно або ненавмисно ділитися інформацією.

Це є особливо критичним у корпоративному середовищі, де є залученість великої кількості людей до роботи. Важливим тут є контроль доступу, наданий до інформації Також, необхідно проводити регулярний аудит дій та навчання персоналу для унеможливлення даних ризиків.

1.3 Порівняльний аналіз криптоалгоритмів для захисту даних під час передачі

Сучасний безпечний файлообмін базується на поєднанні криптографічних методів, механізмів автентифікації та керування доступом. Зокрема, алгоритм AES-256 (Advanced Encryption Standard) є серцем шифрування даних: це блочний шифр з розміром блоку 128 біт та ключами 128, 192 або 256 біт. Він працює за принципом мережі підстановки-перестановки: кожен раунд включає операції заміни (SubBytes), циклічного зсуву рядків (ShiftRows), лінійного перетворення стовпців (MixColumns) і додавання раундового ключа (AddRoundKey) [9].

Наприклад, операція MixColumns над стовпцем вхідних байтів ${b_0,b_1,b_2,b_3}$ описується матричним множенням у полі GF(2^8) (16-бітних слів):

$$\begin{pmatrix} c0 \\ c1 \\ c2 \\ c3 \end{pmatrix} = \begin{pmatrix} 2\,3\,1\,1 \\ 1\,2\,3\,1 \\ 1\,1\,2\,3 \\ 3\,1\,1\,2 \end{pmatrix} \begin{pmatrix} b0 \\ b1 \\ b2 \\ b3 \end{pmatrix}$$

що реалізується як сумування та зрушення байтів по модулю $x^8+x^4+x^3+x+1$. Кількість раундів AES залежить від довжини ключа: для 256-

бітного ключа їх 14. Алгоритм довів свою криптостійкість і широке використання в урядових стандартах, він сильніший за застарілі DES та 3DES.

Недостатнє шифрування файлів або його відсутність варто розглянути окремим пунктом у передачі файлів в інтернеті, адже цей аспект робить його вразливим під час передачі або зберігання.

Розглянемо три найпоширеніші алгоритми шифрування [10]:

1. DES (Data Encryption Standard - стандарт шифрування даних).

DES – це застарілий алгоритм шифрування, який використовує алгоритм для перетворення звичайного тексту в шифр.

DES є алгоритмом блочного шифрування з розміром блоку 64 біти, де кожен блок шифрується окремо. Алгоритм використовує ключ розміром 56 біт, що комбінується з початковою перестановкою блоку та проходить 16 раундів мережі Фейстеля [11]

IBM розробила цей алгоритм у 70-х роках. Однак до 90-х років він вичерпав більшу частину своєї корисності, що призвело до того, що NIST закликав до його заміни.

2. AES (Advanced Encryption Standard).

Оскільки алгоритм DES швидко застарівав, уряд США, військові та всі ми потребували заміни, яку не можна було б зламати грубою силою.

Агентство національної безпеки (АНБ) проводило конкурс, і симетричний алгоритм шифрування, створений (і названий на честь) вченим на ім'я Рейндейл, виділився.

Цей алгоритм, який є найпоширенішим, має три різновиди: AES 128, 192 і 256, всі вони практично незламні і є одними з найбезпечніших алгоритмів шифрування, відомих людині. Ми могли б продовжити розповідь про цей алгоритм, але ми зупинимося на цьому, щоб заощадити ваш час.

AES працює з фіксованим розміром блоку 128 біт і підтримує три варіанти довжини ключа: 128, 192 і 256 біт. Кожна довжина ключа визначає кількість раундів шифрування: AES128: 10 раундів; AES-192: 12 раундів; AES-256: 14 раундів [12].

3. RSA.

Алгоритм Рівеста-Шаха-Аддлмана (RSA) був розроблений трьома вченими, Роном Рівестом, Аді Шаміром і Леонардом Аддлманом з Массачусетського технологічного інституту (MIT), в кінці 70-х років.

Цей алгоритм використовує асиметричну криптографію. В асиметричній криптографії для процесів шифрування та розшифрування використовуються різні ключі.

Зобразимо порівняння у вигляді таблиці 1.1.

Таблиця 1.1 – Порівняння криптоалгоритмів

Алгоритм	Тип шифрування	Довжина ключа, біт	Високий рівень безпеки	Швидке виконання	Застосування
DES	Симетричний	56	-	-	Застарілий
AES	Симетричний	128, 192, 256	+	+	Широко використовується
RSA	Асиметричний	1024-4096	+	-	Використання для захисту підписів

AES-256 має велику розмірність простору ключів і ключову стійкість, що вдвічі перевищує AES-128 [13].

Для порівняння, у 3DES ефективна довжина ключа еквівалентна ~112 бітам. Симетричні шифри (AES, Blowfish тощо) є значно швидшими за асиметричні (RSA, ECC), тому їх застосовують для шифрування великих файлів. Наприклад, експерименти показують, що Blowfish працює швидше за AES в загальному випадку (коротший час шифрування, менші витрати пам’яті), але AES вибирають при пріоритеті безпеки конфіденційності і цілісності даних [14].

Суттєвими параметрами також є розмір блоку (у Blowfish 64 біти, у AES – 128 біт) і кількість операцій у раунді. Усі сучасні симетричні шифри забезпечують високу швидкість обробки (гігабіти на секунду) на сучасних

процесорах, проте AES-256 забезпечує сильніший захист на тлі майбутніх можливостей квантових обчислень (ефективна стійкість ~128 біт проти 64 біт для AES-128 за алгоритмом Ґровера [15].

Навіть за наявності криптографії та безпечних протоколів виникає загроза атак «людина посередині» (MITM). У цій ситуації зловмисник пасивно перехоплює й ретранслює повідомлення між двома сторонами, які вважають, що спілкуються напряму. Це дозволяє атакуючому зламувати шифрування або витягувати конфіденційні дані (облікові дані, ключі тощо) до їх отримання законним одержувачем [16].

Прикладом атаки MITM є фальсифікація обміну ключами у протоколах зв'язку: якщо зловмисник підміняє публічні ключі сторін, він може прослуховувати й змінювати весь трафік без відома жертви. Сучасні протоколи (TLS, HTTPS) використовують сертифікати й автентифікацію кінцевих точок для запобігання MITM-атакам, але користувачі також мають дотримуватись правил безпеки (перевіряти достовірність сертифікатів, уникати незахищених Wi-Fi тощо).

Важливість шифрування у передачі файлів не варто недооцінювати як для запобігання витоку даних, так і для попередження несанкціонованого доступу. У першу чергу це про високий рівень безпеки. AES-256 є одним з надійних алгоритмів шифрування, який здатний забезпечити відповідний рівень захисту від атак. Його часто застосовують в урядових та комерційних організаціях для виконання вище зазначених цілей.

1.3 Аналіз існуючих методів обміну інформацією

Існують різні моделі обміну файлами. У централізованих системах (подібних до Dropbox, Google Drive, OneDrive) файли передаються через сервер-посередник. Ці сервіси зазвичай використовують захист TLS/SSL під час передачі і зберігають дані у зашифрованому вигляді (часто AES-256) на сервері [17].

Однак вони не надають клієнтського (end-to-end) шифрування: дані розшифровуються на сервері і знову шифруються для зберігання, тому провайдер може отримати доступ до вмісту. Наприклад, Dropbox шифрує файли при пересиланні і на диску, але дозволяє собі розшифрування на стороні сервера та не підтримує вбудоване end-to-end шифрування [18].

Це спрощує синхронізацію і спільну роботу, але знижує конфіденційність: у разі злому системи чи внутрішніх помилок сторонні особи можуть отримати доступ до даних.

Відмінним підходом є однорангові (P2P) мережі, як-от BitTorrent. Такі протоколи розподіляють файл на фрагменти і передають їх від кількох учасників одночасно, забезпечуючи високу швидкість завантаження та стійкість до навантаження (рис. 1.1). Однак у звичайному BitTorrent-сценарії файл поширює ініціатор (сідер), і відсутні вбудовані механізми шифрування або контролю доступу. Тому для безпеки в P2P-мережах доводиться застосовувати додаткові шари – наприклад, шифрування потоку (SSL/TLS для торрент-клієнтів) чи плагіни. Іншим поширеним способом передачі є FTP, який за замовчуванням не шифрує трафік (FTP-FTP), тому в безпечних системах застосовують SFTP/FTPS, HTTPS або VPN. При цьому відсутність захищеного каналу дає змогу зловмисникам перехоплювати файли або ключі, ігноруючи цілісність та конфіденційність

З розповсюдженням сервісів для обміну файлами постає потреба у відповідному рівні захисту інформації, яку передаватимуть. У цьому підрозділі буде проведено порівняння файлообмінників, з урахуванням основних критеріїв, які відповідають за рівень безпеки, а саме:

– криптографічний захист;

– двофакторна автентифікація;

– політика видалення файлів;

– швидкість завантаження;

– масштабованість;

– легкість використання;

– захист від витоків та вторгнень.

Розглянемо та проаналізуємо наступні файлообмінники: Google Диск, iCloud Drive, Mega, pCloud, Fex, Box.

Google Диск є одним з найпопулярнішим сховищем, яке надає можливість ділитися файлами. Google застосовує шифрування AES-128 для зберігання файлів на своїх серверах. Це не є найсильнішим варіантом (AES-256 є більш безпечним), але все ж надає певний рівень захисту. Google Диск підтримує двофакторну автентифікацію (2FA), що є обов'язковим для користувачів, які хочуть підвищити безпеку своїх акаунтів. Google пропонує 2FA через Google Authenticator, SMS-повідомлення, або Google Prompt (повідомлення на мобільному пристрої для підтвердження входу).

iCloud Drive є хмарним сервісом для зберігання та обміну файлами від Apple. Це рішення інтегровано в екосистему Apple, що дозволяє користувачам зручно зберігати, синхронізувати та обмінюватися файлами між різними пристроями. Шифрування на сервері: iCloud використовує AES-128 для шифрування даних на своїх серверах. Як і у випадку з Google Диск, цей алгоритм шифрування є надійним, але не таким сильним, як AES-256. iCloud Drive дозволяє користувачам вручну видаляти файли, але файли спочатку потрапляють до корзини, де вони зберігаються до моменту остаточного видалення.

Після завершення цього процесу файли остаточно зникають з сервісу, що може зайняти кілька днів. Як і Google Диск, iCloud Drive не має функції автоматичного та необоротного видалення файлів після їх використання. Це обмеження порівняно з даною розробкою, яка автоматично видаляє файли.

MEGA – це безпечне, контрольоване користувачем наскрізне зашифроване хмарне сховище та сервіс зв'язку з великим безкоштовним простором для зберігання даних. На відміну від інших провайдерів хмарних сховищ, ваші дані шифруються та розшифровуються лише клієнтськими пристроями [19].

Mega є дуже потужним хмарним сервісом, зокрема завдяки своєму сильному шифруванню, великим обсягом безкоштовного місця та хорошим можливостям для обміну файлами. Однак, існують мінуси, такі як використання AES-128 (замість AES-256), обмежена швидкість передачі для безкоштовних користувачів, висока вартість платних планів і обмежена інтеграція з іншими сервісами. Ці недоліки можуть бути суттєвими для користувачів, які шукають більш універсальне і ефективне рішення.

pCloud Drive – це застосунок для робочого столу, що створює захищений віртуальний носій на Вашому комп'ютері, і який Ви можете легко використовувати для зберігання, мати доступ до своїх файлів і працювати з ними в хмарі [20].

pCloud надає високий рівень безпеки завдяки використанню AES-256 шифрування. Проте є певні недоліки, а саме відсутність автоматичного шифрування для безкоштовних користувачів. Для повноти безпеки користувачеві потрібно оформити платну підписку.

FEX.NET (File EXchange Network) – це сучасний та зручний хмарний сервіс для зберігання та обміну файлами, що запрацював на початку цього року [21].

Fex може бути корисним для користувачів, яким потрібен простий сервіс для зберігання та обміну файлами. Проте існують серйозні мінуси, такі як відсутність шифрування, відсутність двофакторної автентифікації і обмежена інтеграція з іншими сервісами.

Цей сервіс має обмежену функціональність і не пропонує такої ж гнучкості та безпеки, як більш відомі хмарні платформи, тому він може бути менш підходящим для користувачів, які шукають високий рівень захисту даних або інтеграцію з іншими інструментами. Для повного забезпечення безпеки користувачам потрібно підписатися на платний pCloud Crypto.

Box – це хмарна платформа керування вмістом, яка пропонує безпечні послуги обміну файлами та спільної роботи для компаній і окремих осіб [22].

Box є сильним інструментом для зберігання і співпраці, особливо для бізнес-користувачів, які потребують безпечного і масштабованого рішення. Проте для індивідуальних користувачів існують деякі недоліки, а саме обмеження безкоштовного плану (10 ГБ), відсутність шифрування на рівні клієнта та висока вартість для розширених функцій.

Здійснимо порівняння вище описаних файлообмінників, отриманий результат занесемо в таблицю 1.2.

Таблиця 1.2 – Порівняння обраних файлообмінників

Критерій	Дана розробка	Google Drive	iCloud	Mega	Pcloud	Fex	Box
Криптографічний захист	AES-256	AES-128	AES-128	AES-128	AES-256	-	AES-256
Двофакторна автентифікація	+	+	+	+	+	-	+
Автоматичне видалення файлів	+	-	-	-	-	-	-
Висока швидкість завантаження	+	+	+	-	-	-	+
Масштабованість (безкоштовний план) Гб	Без обмежень	15	5	20	10	10	10
Легкість використання	+	+	+	-	-	+	-
Захист від витоків даних та вторгнень	+	+	+	-	-	-	-

На рисунку 1.2 подано схематичну архітектуру захищеного файлообмінника. Користувач взаємодіє з клієнтським інтерфейсом (Frontend),

який надсилає запити до серверного додатку (Backend). На стороні сервера реалізовані модулі обробки файлів, автентифікації та доступу.

Файли шифруються AES-256 перед передачею та зберігаються в зашифрованому вигляді в базі даних чи спеціальному файловому сховищі. Також сервер підтримує двофакторну автентифікацію (2FA) для підтвердження особи користувача і запускає фоновий процес автоматичного видалення файлів після встановленого терміну.

Ця структура забезпечує багатоетапний захист: безпеку каналу (TLS), надійне шифрування даних (AES-256) та додаткову ідентифікацію користувача (2FA).

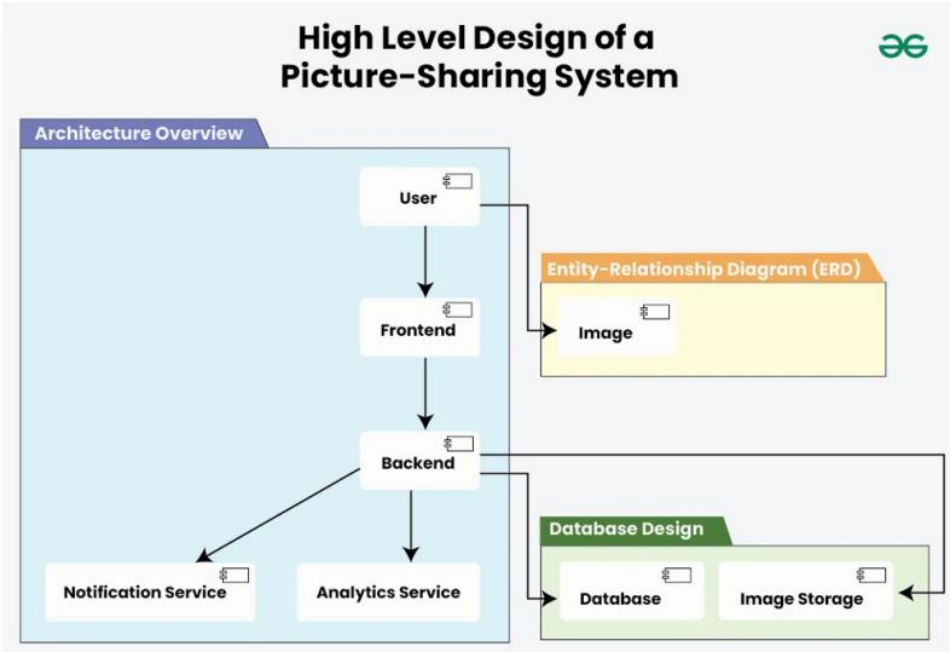


Рисунок 1.2 – Схематична архітектура захищеного файлообмінника [15]

Традиційні файлообмінні сервіси мають ряд обмежень. Наприклад, хоча Dropbox і подібні сервіси використовують шифрування AES-256 на сервері, вони не передбачають end-to-end шифрування: дані можуть бути розшифровані на стороні сервісу. Це означає, що адміністратори сервісу або зловмисники при витоку можуть отримати доступ до файлів користувачів [23].

Крім того, стандартний доступ до облікових записів зазвичай захищений лише паролем – що робить їх уразливими до підбору та витоку паролів.

Двофакторна автентифікація рідко впроваджується за замовчуванням, хоча вона значно підвищує безпеку (див. розділ 2). Інші недоліки: відсутність автоматичного видалення надлишкових даних (що необхідно для дотримання політик конфіденційності, наприклад GDPR). Важливо проаналізувати та усунути ці прогалини в новому рішенні.

Право на захист персональних даних не є абсолютним правом; воно повинне розглядатися в зв'язку з його функцією в суспільстві та бути збалансованим з іншими фундаментальними правами згідно з принципом пропорційності [24].

Шифрування працює шляхом перетворення звичайного тексту в нечитабельний шифрований код. AES - це шифр з симетричним ключем, який використовує один і той самий ключ для шифрування і розшифрування. Симетричні ключі швидші та ефективніші за асиметричні, оскільки вимагають менших обчислювальних потужностей [25].

Було проведено порівняльний аналіз файлообмінників і визначено, що розробка захищеного файлообмінника відповідає сучасним вимогам користувачів для надійної, безпечної та зручної передачі даних. Було виявлено, що більшість наявних файлообмінників не мають у собі функції автоматичного видалення даних, такі як iCloud, Mega та Pcloud. Також дана розробка вирізняється своєю надійністю шифрування та доступністю без обмежуючих тарифів.

1.4 Висновки до розділу 1 та постановка задач

У сучасному соціальному світі, як і у минулі часи розвитку суспільно-історичного процесу, найактуальнішою проблемою залишається питання щодо особливостей формування особистості та характеру її існування у суспільстві, функціонування соціальної структури взагалі [26].

Схиляючись на тенденції сьогодення у безпечному обміні інформацією у інтернет-просторі, можна зробити висновок, що необхідність якісно захищених файлообмінників зростає з кожним роком все більше. Враховуючи

аспект того, що більшість компаній переходять до онлайнового режиму роботи.

У даному розділі було детально визначено та розглянуто основні загрози, з якими може зіткнутися користувач. Зловмисники можуть використовувати відкриті точки доступу Wi-Fi для перехоплення інформації між користувачем та мережею. Крім того, використання складних паролів і двофакторної автентифікації є необхідним для запобігання несанкціонованому доступу до файлів.

Шкідливе програмне забезпечення та фішингові атаки, що експлуатують людський фактор, є серйозною загрозою, тому важливо проводити навчання та аудит. Також існує ризик витоку даних через інсайдерів, що підкреслює необхідність контролю доступу та регулярного моніторингу.

Проведений порівняльний аналіз файлообмінників визначив, що розробка захищеного файлообмінника відповідає сучасним вимогам користувачів для надійної, безпечної та зручної передачі даних. Виявлено, що більшість наявних файлообмінників не мають у собі функції автоматичного видалення даних, такі як iCloud, Mega та Pcloud.

Опираючись на результати проведеного аналізу та виходячи з мети і актуальності обраної теми, були поставлені наступні задачі для подальшої роботи:

- проаналізувати сучасні методи передачі файлів;
- розробити механізм шифрування файлів за допомогою AES-256;
- інтегрувати двофакторну автентифікацію;
- реалізувати функцію автоматичного видалення файлів;
- забезпечити інтуїтивно зрозумілий інтерфейс для користувачів.

2 ПРОЕКТУВАННЯ АРХІТЕКТУРИ ТА СИСТЕМИ ЗАХИСТУ ФАЙЛООБМІННИКА

У даному розділі було спроектовано та реалізовано систему захисту на основі використанням криптоалгоритму AES-256, двофакторної автентифікації та автоматичного видалення файлів після встановленого періоду часу. Проведено перевірку її стійкості до атак різного типу. Крім того, було спроектовано сценарії для системи безпеки.

2.1 Визначення основних компонентів системи та взаємодія між ними

У даному підрозділі розглядається загальна архітектура системи, призначеної для безпечного обміну файлами між користувачами. Основу функціонування системи становить використання криптографічного алгоритму AES-256 для шифрування даних, двофакторної автентифікації для захисту доступу та механізму автоматичного видалення файлів після завершення встановленого терміну зберігання. Даний підхід є особливо важливим для використання передачі інформації урядами, фінансовими установами та великими технологічними компаніями. Архітектура побудована за класичною моделлю клієнт-серверної взаємодії, де кожен компонент має чітко визначені функції та тісно взаємодіє з іншими елементами.

Сучасні організації та користувачі потребують безпечної передачі конфіденційних даних: від фінансових звітів до медичних записів. Наприклад, у банківській сфері чи охороні здоров’я існують суворі регламенти (HIPAA, GDPR тощо), які вимагають шифрування інформації при передачі та видаленні даних після встановленого строку. Застосування AES-256 гарантує конфіденційність файлів за сучасних стандартів, а двофакторна автентифікація запобігає несанкціонованому доступу, навіть якщо зловмисник отримав пароль [16].

Автоматичне видалення файлів після закінчення строку зберігання забезпечує відповідність вимогам захисту даних та запобігає накопиченню зайвої інформації. У поєднанні, ці рішення створюють багато екранний захист: навіть у разі компрометації одного рівня (наприклад, втрати пароля) інші рівні забезпечують безпеку. До того ж у централізованій системі із зашифрованим сховищем реалізовуються функціональні переваги: спільний доступ до файлів, керування правами користувачів та зручний інтерфейс. Це робить розробку файлообмінника з AES-256 та 2FA практично доцільною для підприємств усіх галузей: від урядових установ до малого бізнесу. Це стандарт шифрування, рекомендований Національним інститутом стандартів і технологій США (NIST) і прийнятий як "золотий стандарт" у всьому світі

Центральну роль у системі відіграє сервер, який обробляє запити клієнтів, виконує криптографічні операції, забезпечує автентифікацію користувачів, а також здійснює управління файлами та взаємодіє з базою даних. Для передачі даних між клієнтами та сервером застосовується API, який підтримує як REST-протокол, так і WebSocket-з'єднання - залежно від характеру запиту та необхідної швидкості обміну інформацією.

REST API працює в режимі “без стану” (stateless), що означає, що кожен HTTP-запит, надісланий на сервер, містить усі необхідні дані для його обробки. Сервер не зберігає інформацію про попередні запити від клієнта. Це робить систему більш надійною та масштабованою [27].

Клієнтська частина представлена у вигляді веб-інтерфейсу або десктопного додатка, який дозволяє користувачам проходити автентифікацію, завантажувати файли до системи, отримувати їх, а також керувати особистими даними. Для реалізації інтерфейсу можуть бути використані як сучасні веб технології (HTML, CSS, JavaScript), так і фреймворки для створення десктопних додатків, зокрема PyQt для Python. Саме клієнт ініціює шифрування файлів перед їх передачею на сервер, тим самим зменшуючи ризики перехоплення даних під час транспортування.

Уся інформація системи включно з відомостями про користувачів, параметрами автентифікації, таймерами зберігання та характеристиками файлів - зберігається в базі даних. У цьому контексті можливе застосування як реляційних СУБД (наприклад, MySQL або PostgreSQL), так і нереляційних рішень на кшталт MongoDB. Основними таблицями бази даних є: "Користувачі", де зберігається інформація про облікові записи та автентифікації параметри; "Файли", що містить зашифровані назви файлів, дати створення, розміри та строки зберігання; та "Журнали дій", які реєструють усі операції, здійснені в системі, з метою аудиту та контролю безпеки.

Взаємодія між компонентами системи відбувається послідовно. Наприклад, під час завантаження файлу користувач проходить автентифікацію, після чого клієнтська частина шифрує файл за допомогою AES-256. Зашифрований файл разом із метаданими (назва, дата створення, термін зберігання) надсилається на сервер, де файл зберігається у файловій системі або хмарному сховищі, а відповідна інформація - у базі даних. Під час отримання файлу користувач також автентифікується, клієнт надсилає відповідний запит, а сервер, опрацювавши його, знаходить файл у базі, розшифровує та передає його користувачеві.

Окремим функціональним елементом є механізм автоматичного видалення файлів, який реалізується сервером у фоновому режимі. Система періодично перевіряє таблицю таймерів, і якщо час зберігання минув, відповідний файл видаляється разом із записом у базі даних. Такий підхід дозволяє зменшити навантаження на сховище та забезпечити дотримання політик конфіденційності.

Для реалізації системи використовуємо реляційну базу даних. Основні сутності: Users (користувачі), Files (файли) та TwoFactor (параметри 2FA). Таблиця Users містить UserID (PK), Username, PasswordHash, Role (наприклад , User/Admin) та параметри 2FA (секретний ключ або тип 2FA). Таблиця Files містить FileID (PK), OwnerID (FK→Users), FileName, UploadTi

me, ExpirationTime (дата видалення), а також може включати симетричний ключ шифрування файлу чи посилання на нього. Таблиця Access (або Shares) фіксує права спільного доступу.

AccessID, FileID (FK→Files), UserID (FK→Users), PermissionLevel (напр иклад, читання/запис). Зв’язки між сутностями: один User – багато Files (власність), один File – багато записів Access (різні користувачі з різними правами), кожен User – один запис в TwoFactor. Така схема враховує основні випадки: користувачі можуть мати кілька файлів і ділитися ними з іншими, а також у всіх записах враховується час їх створення та термін існування.

Таким чином, сервер забезпечує логіку безпеки, автентифікацію, шифрування та контроль доступу, клієнт надає інтерфейс користувача та виконує попередню обробку файлів, а база даних гарантує цілісне зберігання службової інформації, необхідної для функціонування системи.

2.2 Покроковий алгоритм функціонування системи безпечного захищеного файлообмінника

Алгоритм роботи файлообмінника з використанням криптоалгоритму AES, двофакторної автентифікації та автоматичного видалення файлів після встановленого періоду часу передбачає послідовне виконання дій з моменту автентифікації користувача до завершення обробки файлу – включно з його можливим автоматичним видаленням. Ключовими етапами є автентифікація, шифрування, передача, збереження, отримання та видалення. Нижче наведено поетапний опис роботи системи.

Для ілюстрації роботи системи було наведено схему алгоритму авторизації з 2FA. Користувач вводить ім’я користувача та пароль. Сервер перевіряє пароль і, у випадку успіху, генерує одноразовий код (OTP) для 2FA. Код відправляється користувачу email або. Користувач вводить отриманий код. Сервер перевіряє збіг. Якщо код правильний, дозвіл на доступ надається,

і користувач заходить у систему; інакше – доступ відмовляється. Таким чином запропонований алгоритм матиме вигляд:

Крок 1 Авторизація користувача.

1.1. Перевірка чи користувача зареєстровано, якщо ні, реєстрація, якщо так, наступний пікдрок.

1.2. Вводить облікові дані (логін і пароль).

1.3. Користувач проходить двофакторну автентифікацію через email.

1.4. Клієнт надсилає запит на сервер для перевірки введених даних.

1.5. Сервер перевіряє дані у базі даних і повертає токен доступу у разі успішної автентифікації.

Крок 2 Завантаження файлу користувачем.

2.1. Користувач обирає файл для завантаження.

2.2. Клієнтська частина локально шифрує файл за допомогою AES-256.

2.3. Користувач встановлює строк зберігання файлу (час до автоматичного видалення).

2.4. До зашифрованого файлу додаються метадані: ім’я, розмір, дата створення, час життя.

2.5. Клієнт надсилає файл і метадані на сервер через REST або WebSocket API.

2.6. Сервер приймає файл, зберігає його у файлову систему (або хмарне сховище), а метадані — до бази даних.

Крок 3 Отримання файлу іншим користувачем

3.1. Користувач проходить автентифікацію (аналогічно до етапу 1).

3.2. Ініціює запит на завантаження конкретного файлу.

3.3. Сервер перевіряє наявність файлу та права доступу

3.4. Файл розшифровується на сервері або передається в зашифрованому вигляді для локальної розшифровки на клієнті (залежно від реалізації).

3.5. Користувач отримує файл.

Крок 4 Автоматичне видалення файлів.

4.1. Сервер періодично виконує перевірку таблиці збережених файлів та їхніх таймерів

4.2. Якщо виявлено, що строк зберігання минув, файл видаляється з файлової системи або хмарного сховища; відповідний запис у базі даних також видаляється.

Відобразимо алгоритм захищеного файлообмінника з використанням криптоалгоритму AES, двофакторної автентифікації та автоматичного видалення файлів після встановленого періоду часу у вигляді блок схеми (рисунок 2.1).

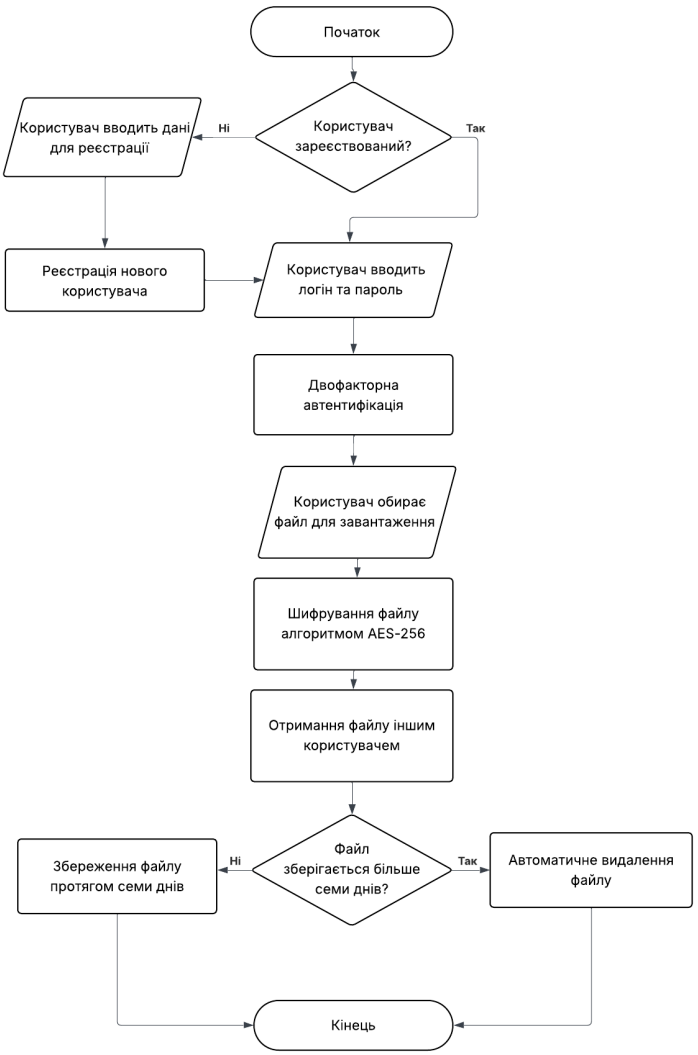


Рисунок 2.1 – Вигляд блок-схеми алгоритму

Було побудовану блок-схему алгоритму з відображенням послідовності кроків роботи програми для подальшої реалізації.

2.3 Реалізація шифрування файлів за допомогою AES-256

У межах розроблюваної системи файлообмінника з використанням криптоалгоритму AES, двофакторної автентифікації та автоматичного видалення файлів після встановленого періоду часу особливу увагу приділено забезпеченню конфіденційності та цілісності даних, що передаються та зберігаються. Центральним елементом у реалізації цих вимог є вибір сучасного, стійкого до крипто аналізу алгоритму шифрування – AES-256. Цей симетричний блочний алгоритм є загальновизнаним стандартом і рекомендований до використання у високочутливих інформаційних системах. Його основною перевагою є висока криптографічна стійкість, яку забезпечує 256-бітний ключ шифрування.

AES-256 працює з блоками даних фіксованого розміру 128 біт, на які розбивається вхідний файл перед шифруванням. Для кожної операції використовується ключ довжиною 256 біт, який генерується надійним криптографічним генератором випадкових чисел. Наприклад, у реалізації на Python можливе використання бібліотеки Crypto, яка дозволяє створювати унікальні ключі за допомогою функції get_random_bytes. Створений ключ може бути збережений у базі даних або окремому ключовому файлі у зашифрованому вигляді, а його безпека додатково гарантується застосуванням майстер-ключа, до якого має доступ виключно сервер.

Шифрування файлів відбувається за обраним режимом роботи AES, таким як CBC (Cipher Block Chaining) або GCM (Galois/Counter Mode). У режимі CBC до кожного блоку додається унікальний вектор ініціалізації (IV), що унеможливлює повторення однакових шифротекстів навіть для ідентичних вхідних даних. GCM, у свою чергу, забезпечує не лише шифрування, а й автентифікацію даних, що підвищує рівень захисту від несанкціонованої модифікації.

З практичної точки зору процес шифрування реалізується наступним чином: файл спочатку доповнюється до необхідної довжини (padding), після чого шифрується по блоках за допомогою створеного ключа та згенерованого IV. Після завершення операції до зашифрованих даних додається IV, необхідний для подальшого розшифрування. Аналогічно, під час розшифрування IV витягується з шифрованого файлу, після чого застосовується зворотна операція AES із використанням того самого ключа. Такий підхід гарантує правильність та відновлюваність даних у будь-який момент часу.

Під час передачі даних між клієнтом і сервером додаткову безпеку забезпечує використання захищеного протоколу HTTPS. Таким чином, навіть у разі перехоплення мережевого трафіку, передані файли залишаються зашифрованими й непридатними для дешифрування без відповідного ключа. IV у цьому випадку передається разом із шифротекстом, що дозволяє здійснити розшифрування лише на авторизованій стороні.

Щодо зберігання, перед записом файлів на сервер або у хмарне сховище вони проходять процедуру шифрування, що мінімізує ризик витоку даних у разі несанкціонованого доступу до сховища. Особлива увага також приділяється захисту метаданих - імена файлів, ID користувачів та інші чутливі поля можуть бути додатково зашифровані або перетворені у хеші.

Ключовим елементом загальної безпеки є ефективне управління ключами шифрування. У цьому контексті можливо впровадження системи керування ключами (Key Management System, KMS), яка централізовано відповідає за створення, зберігання, ротацію та видалення ключів. Такий підхід дозволяє обмежити доступ до ключів лише авторизованим компонентам системи, що значно знижує ймовірність компрометації.

Комп'ютери, які активуються за допомогою хоста KMS, повинні мати спеціальний ключ продукту. Цей ключ іноді називають клієнтським ключем KMS, але офіційно він називається загальним ліцензійним ключем Microsoft (GVLK). Комп'ютери, на яких працюють ліцензійні версії Windows Server і

Windows client, за замовчуванням є клієнтами KMS і не потребують додаткових налаштувань, оскільки відповідний ключ GVLK уже є [28].

Для впевненості в коректності реалізації алгоритму шифрування проводиться серія тестів: перевірка відповідності результатів операцій очікуваним, визначення часу, необхідного для шифрування файлів різного обсягу, а також аналіз стійкості до криптоатак, зокрема брутфорсу чи відомих векторів атак на блочні шифри.

Таким чином, реалізація AES-256 у системі забезпечує надійний захист інформації на всіх етапах її життєвого циклу - від моменту створення та передачі до зберігання та подальшого видалення, що відповідає сучасним вимогам до безпеки цифрових даних.

2.3 Реалізація двофакторної автентифікації користувачів

Для підвищення рівня захищеності файлообмінника з використанням криптоалгоритму AES, двофакторної автентифікації та автоматичного видалення файлів після встановленого періоду часу користувацьких облікових записів у системі реалізовано механізм двофакторної автентифікації (2FA). Основна ідея полягає в тому, що автентифікація базується не лише на чомусь, що знає користувач (наприклад, пароль), але й на чомусь, що він має у своєму розпорядженні (генератор тимчасових кодів). Завдяки цьому навіть у разі компрометації основного пароля зловмисник не зможе отримати доступ до облікового запису без другого фактору.

Серед можливих варіантів реалізації 2FA розглядаються два підходи: надсилання кодів через SMS та генерація одноразових кодів за часом (TOTP). Перший варіант потребує підключення сторонніх SMS-сервісів і залежить від мобільного зв’язку, що може створювати додаткові ризики та витрати. Альтернативою є використання TOTP (Time-Based One-Time Password), що дозволяє користувачам сканувати згенерований QR-код і отримувати коди через додатки на смартфоні — такі як Google Authenticator чи Authy. Цей підхід не вимагає зовнішньої інфраструктури та зручний у реалізації.

У розробленій системі було обрано саме TOTP як основний механізм реалізації 2FA. Для цього використовується низка інструментів: бібліотека otplib для генерації кодів, qrcode для створення QR-зображень, база даних PostgreSQL для збереження параметрів, а також NestJS у якості бекенд-фреймворка. Можливе додаткове збереження QR-кодів у Google Cloud Storage або безпосередня передача клієнту у форматі base64.

Урешті-решт, значна частина особистої, фінансової й конфіденційної інформації зберігається в онлайн-обліковках вашої компанії, а витік даних часто призводить до втрати доходу. Проте є один простий спосіб, який допоможе підвищити культуру кібербезпеки співробітників більшості організацій. Це двофакторна автентифікація або 2FA [29].

Процес активації 2FA для користувача передбачає кілька послідовних кроків. Спочатку сервер генерує унікальний секретний ключ для користувача за допомогою authenticator.generateSecret() з otplib. Потім формується відповідний keyuri-рядок, на основі якого створюється QR-код. Користувач сканує цей код у своєму мобільному додатку, після чого вводить перший згенерований код для підтвердження активації. У випадку успішної верифікації, секрет зберігається у базі даних (у зашифрованому або хешованому вигляді), а двофакторна автентифікація вмикається для облікового запису.

Надалі під час входу користувач спочатку проходить перевірку пароля. Якщо він правильний, система вимагає введення шестизначного тимчасового коду з додатку. Сервер зіставляє введений код із тим, який має бути згенерований на основі збереженого секретного ключа. У разі збігу автентифікація завершується успішно.

Передбачено також можливість деактивації 2FA через особистий кабінет. Для цього користувач має повторно пройти автентифікацію, що запобігає несанкціонованому відключенню без відома власника облікового запису.

З метою захисту від атак перебору реалізовано обмеження на кількість невдалих спроб введення коду, наприклад, до п’яти поспіль. Крім того, всі дії, пов’язані зі входом у систему, фіксуються у журналі подій для подальшого аналізу активності. Особлива увага приділяється безпечному зберіганню секретів — дані шифруються або хешуються за допомогою сучасних алгоритмів перед внесенням до бази.

Для впевненості у надійності реалізації двофакторної автентифікації проведено тестування різних сценаріїв: активація, логін, перевірка правильності генерації та валідації кодів, а також їх відповідності часовим вікнам. Усі ці заходи в комплексі дозволяють гарантувати, що навіть у разі компрометації основного пароля система залишатиметься захищеною.

2.4 Механізм автоматичного видалення файлів

Механізм автоматичного видалення файлів файлообмінника з використанням криптоалгоритму AES, двофакторної автентифікації та автоматичного видалення файлів після встановленого періоду часу впроваджується з метою підвищення конфіденційності користувацьких даних та оптимізації використання ресурсу хмарного сховища. Суть реалізації полягає в тому, щоб після завершення заздалегідь визначеного терміну зберігання, файл автоматично вилучався як із хмарного середовища Google Cloud Storage, так і з бази даних PostgreSQL. Після завантаження кожного файлу система реєструє відповідний запис у базі даних із зазначенням ідентифікатора файлу, шляху до нього, ідентифікатора користувача, часу завантаження та розрахованого часу закінчення терміну зберігання. Час видалення визначається під час завантаження, залежно від вибору користувача, наприклад, 1, 7 або 30 днів. Отримане значення зберігається у полі expiration_time у вигляді абсолютної часової позначки.

Для реалізації процесу видалення застосовується один із двох підходів. Перший — використання періодичного планувальника на основі cron, що запускає перевірку щогодини, виявляє всі файли з простроченим терміном

зберігання, видаляє їх зі сховища через API Google Cloud Storage, після чого відповідні записи видаляються з бази даних. Другий підхід – індивідуальне планування таймерів за допомогою бібліотеки node-schedule, коли для кожного файлу одразу після завантаження створюється окрема задача на видалення у конкретний момент часу. Обидва методи мають свої переваги: cron-підхід краще підходить для великих систем з великою кількістю файлів, тоді як node-schedule забезпечує точність у реальному часі.

Фізичне видалення файлу зі сховища здійснюється за допомогою Google Cloud Storage SDK, де файл ідентифікується за шляхом, переданим під час завантаження. Після успішного видалення файл також видаляється з бази даних за відповідним ідентифікатором. На рівні користувацького інтерфейсу передбачено можливість вибору терміну зберігання файлу під час завантаження через інтерактивний елемент (наприклад, випадаючий список), що дозволяє формувати індивідуальні параметри зберігання без участі адміністратора. На сервері введений параметр обробляється, і на його основі обчислюється дата видалення, яка фіксується в базі.

Для підвищення надійності впроваджено низку додаткових запобіжних механізмів. Наприклад, перед остаточним видаленням може бути реалізовано повідомлення користувача на електронну пошту з попередженням за певний час до закінчення терміну. Також передбачена можливість підтвердження видалення або перенесення файлу в архів для особливо важливих документів. Надсилання повідомлень реалізується через зовнішні поштові сервіси, зокрема SendGrid або nodemailer.

Перед впровадженням система проходить тестування, що включає перевірку коректності запису термінів зберігання, симуляцію закінчення терміну дії файлів, перевірку хронологічної точності та поведінки системи за умови великої кількості одночасних записів. Основними критеріями успішної роботи є своєчасне й безпомилкове видалення файлів, відсутність передчасних операцій, а також повне очищення як зі сховища, так і з бази даних.

Завдяки автоматизації цього процесу досягається зменшення навантаження на систему зберігання, спрощується адміністрування, а також підвищується загальний рівень безпеки та відповідність вимогам щодо захисту персональних даних.

2.5 Висновки до розділу 2

У процесі розробки системи безпечного файлообміну було визначено ключові компоненти архітектури: сервер, клієнтську частину та базу даних.. Клієнтська частина реалізує інтерфейс для користувачів і передає запити через API, використовуючи REST або WebSocket-протоколи. Для зберігання метаданих використовується реляційна або нереляційна СУБД, де фіксується інформація про користувачів, файли та дії.

В системі реалізовано сильне симетричне шифрування за стандартом AES-256 із застосуванням криптографічно стійких ключів, які можуть зберігатися у зашифрованому вигляді та додатково захищатися майстер-ключем. Передача файлів захищена HTTPS, а зберігання - шифруванням на стороні сервера. Важливим елементом безпеки є двофакторна автентифікація (TOTP), що дозволяє захистити обліковий запис навіть у разі компрометації пароля.

Таким чином, створена система поєднує сучасні методи шифрування, багатофакторну автентифікацію та автоматичне управління збереженням даних, що забезпечує високий рівень безпеки, конфіденційності та оптимальне використання ресурсів.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ РОЗРОБЛЕНОЇ СИСТЕМИ ЗАХИСТУ

У даному розділі було здійснено опис програмної реалізації механізму автоматичного видалення файлів у файлообмінниках на основі криптоалгоритму AES-256. Було обрано середовище розробки, створені скрипти для коректного налаштування системи захисту. Було продемонстровано користування графічним інтерфейсом системи з відповідними рекомендаціями. Окрім того, було протестовано файлообмінник з відповідним висновком.

3.1 Обґрунтування вибору середовища розробки

Для реалізації механізму автоматичного видалення файлів у файлообмінниках на основі криптоалгоритму AES-256 було прийнято рішення обрати в якості середовища розробки WebStorm та база даних PostgreSQL. У першу чергу задля зручності використання.

WebStorm містить все необхідне для розробки на JavaScript і TypeScript з самого початку. Ви можете персоналізувати його за допомогою різних плагінів і налаштувань [30].

Щоб розпочати розробку у даному середовищі, необхідно в обов’язковому порядку встановити node.js, що і було виконано. Наявний наступний інтерфейс програми наведено на рисунку 3.1.

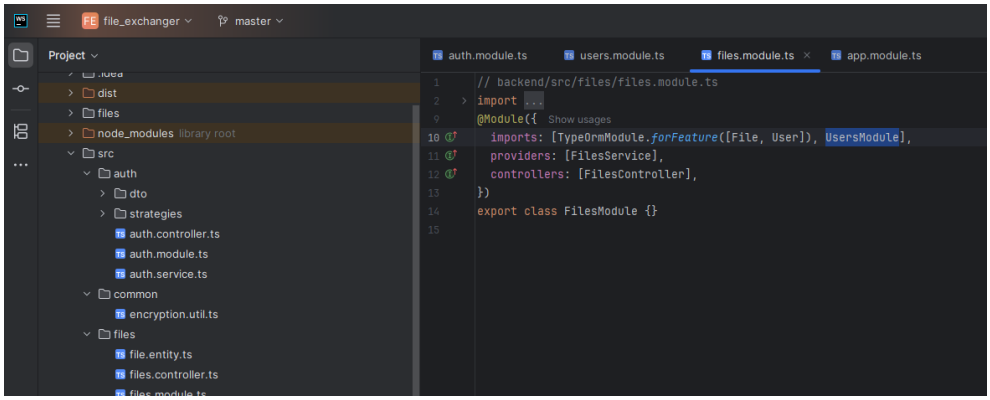


Рисунок 3.1 – Інтерфейс програмного середовища

Для реалізації механізму автоматичного видалення файлів у межах захищеного файлообмінника було обрано середовище розробки WebStorm з кількох практичних причин.

По-перше, WebStorm — це сучасне інтегроване середовище розробки (IDE), орієнтоване на веб технології (JavaScript, TypeScript, Node.js), що ідеально підходить для створення повнофункціональних веб-додатків. У нашій розробці серверна частина реалізована за допомогою Node.js, що забезпечує високу продуктивність у роботі з файловою системою та асинхронними процесами — саме те, що потрібно для фонової перевірки та видалення файлів.

По-друге, WebStorm надає потужні інструменти для роботи з модулями криптографії (наприклад, бібліотекою crypto) та з базами даних (через плагіни або зовнішні драйвери), що дозволяє зручно реалізовувати алгоритми шифрування AES-256 у зв’язці з перевіркою терміну зберігання файлу в БД.

Механізм працює наступним чином: у WebStorm реалізовано серверний скрипт (наприклад, планувальник на основі node-cron), який періодично перевіряє записи у базі даних. Якщо дата поточного моменту перевищує вказаний expirationTime для файлу, скрипт ініціює його видалення з файлового сховища та вилучення запису з БД. Такий підхід дозволяє підтримувати чистоту системи та відповідати вимогам захисту конфіденційної інформації.

Таким чином, WebStorm було обрано як оптимальне середовище для швидкої, ефективної та безпечної реалізації логіки видалення файлів, інтеграції з криптографічними модулями та реалізації бекенд-частини розробки.

3.2 Процес створення скриптів для налаштування системи захисту

Було наведено перелік скриптів з відповідним їх роз’ясненням для розробки захищеного файлообмінника з використанням криптоалгоритму AES, двофакторної автентифікації та автоматичного видалення файлів після встановленого періоду часу. У першу чергу було впроваджено механізм авторизації на основі JWT (JSON Web Token). Цей підхід дозволяє ефективно реалізувати безстанову аутентифікацію користувачів, забезпечуючи як зручність взаємодії, так і високий рівень безпеки.

JWT є відкритим стандартом, який дозволяє безпечно передавати зашифровану або підписану інформацію між клієнтом та сервером у вигляді компактного токена. Основна ідея полягає в тому, що після проходження автентифікації користувач отримує токен, який містить інформацію про його особу, роль, а також термін дії доступу. Надалі цей токен передається разом із кожним запитом до сервера в заголовку Authorization.

Структура токена:

JWT складається з трьох частин, закодованих у форматі Base64 і розділених крапками:

```
css
CopyEdit
Header.Payload.Signature
```

Header (заголовок) містить тип токена (JWT) і алгоритм підпису (наприклад, HS256);

Payload (корисне навантаження) містить основні дані (наприклад, user_id, role, exp);

Signature (підпис) генерується на основі header і payload за допомогою секретного ключа, що гарантує цілісність токена.

Розглянемо принцип роботи додатку:

Користувач вводить логін і пароль.

Сервер перевіряє облікові дані, і при успішному вході формує JWT-токен.

Токен повертається на клієнт і зберігається (наприклад, у localStorage).

При кожному наступному запиті клієнт надсилає токен в заголовку HTTP-запиту.

Сервер перевіряє дійсність токена (включно з терміном дії) та надає або відмовляє в доступі до ресурсів. Токен не можна змінити без знання секретного ключа, і він автоматично анулюється після закінчення терміну дії.

У системі JWT використовується у зв’язці з механізмом двофакторної автентифікації. Після успішної валідації одноразового коду (2FA) сервер видає користувачу підписаний токен з обмеженим терміном дії (наприклад, 15 хвилин). Це дозволяє реалізувати додатковий рівень захисту при доступі до функціоналу завантаження або видалення файлів.

Таким чином, використання JWT у межах розробленого файлообмінника дозволило створити надійний і масштабований механізм контролю доступу без збереження сесій на стороні сервера, що відповідає сучасним стандартам безпеки веб-застосунків.

У цьому фрагменті налаштовується головний модуль програми — AppModule, який об’єднує всі частини проєкту на NestJS:

```
        import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { ScheduleModule } from '@nestjs/schedule';
import { AuthModule } from './auth/auth.module';
import { UsersModule } from './users/users.module';
import { FilesModule } from './files/files.module';
import { ConfigModule } from '@nestjs/config';
@Module({
 imports: [
   TypeOrmModule.forRoot({
     type: 'postgres',
     host: 'localhost',
     port: 5432,
     username: 'postgres',
     password: '1',
     database: 'file_exchanger',
     autoLoadEntities: true,
     synchronize: true,
   }),
```

```
    ScheduleModule.forRoot(),
    ConfigModule.forRoot({
      isGlobal: true, // makes config available everywhere without re-importing
    }),
    AuthModule,
    UsersModule,
    FilesModule,
  ],
})
export class AppModule {}
```

TypeOrmModule.forRoot(...) — налаштовує підключення до бази даних PostgreSQL. Тут ми вказуємо:

host, port, username, password — дані для підключення;

database: 'file_exchanger' — назва нашої БД;

autoLoadEntities: true — автоматичне підключення сутностей;

synchronize: true — дозволяє автоматично створювати таблиці з сутностей (зручно на етапі розробки).

ScheduleModule.forRoot() — підключає модуль планування задач, який потрібен, наприклад, для автоматичного видалення файлів через певний час (фонові задачі).

ConfigModule.forRoot({ isGlobal: true }) — підключає модуль конфігурації, який дозволяє працювати з .env файлами. Опція isGlobal: true робить змінні середовища доступними у всьому проєкті без повторного імпорту.

AuthModule, UsersModule, FilesModule — це імпорт раніше створених модулів авторизації, користувачів та файлів, які разом формують функціональність файлообмінника.

Було створено процес автроризації користувача, що є важливим аспектом при безпечному обміні даними

```
    import { Module } from '@nestjs/common';
import { JwtModule } from '@nestjs/jwt';
import { UsersModule } from '../users/users.module';
import { AuthService } from './auth.service';
```

```
import { AuthController } from './auth.controller';
import { JwtStrategy } from './strategies/jwt.strategy';
import { PassportModule } from '@nestjs/passport';
@Module({
 imports: [
  UsersModule,
  PassportModule.register({ defaultStrategy: 'jwt' }),
  JwtModule.register({
   secret: process.env.JWT_SECRET,
   signOptions: { expiresIn: '1d' },
  }),
 ],
 providers: [AuthService, JwtStrategy],
 controllers: [AuthController, AuthController],
 exports: [JwtStrategy, PassportModule],
})
export class AuthModule {
 constructor() {
  console.log('1', process.env.JWT_SECRET);
 }
}
```

У цьому фрагменті відбувається налаштування модуля автентифікації в рамках веб-додатку, який використовує фреймворк NestJS для розробки захищеного файлообмінника з використанням криптоалгоритму AES, двофакторної автентифікації та автоматичного видалення файлів після встановленого періоду часу.

Імпортуються необхідні модулі: UsersModule (для роботи з користувачами), PassportModule (для стратегій авторизації), JwtModule (для роботи з JWT).

Через PassportModule.register() встановлюється стратегія авторизації за замовчуванням — 'jwt', тобто додаток буде використовувати JWT-токени для перевірки доступу.

У JwtModule.register() вказується секретний ключ (JWT_SECRET з .env) і час дії токена (expiresIn: '1d' означає, що токен буде дійсний 1 добу).

Модуль AuthModule включає основні компоненти автентифікації: сервіс (AuthService), контролер (AuthController) і стратегію (JwtStrategy).

У exports ми явно дозволяємо використовувати JwtStrategy і PassportModule в інших модулях проєкту.

У конструкторі модуля виводиться секретний ключ у консоль (це може бути корисним для налагодження, хоча в продакшені так робити небажано).

Наступним є модуль користувачів:

```
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { User } from './user.entity';
import { UsersService } from './users.service';
@Module({
 imports: [TypeOrmModule.forFeature([User])],
 providers: [UsersService],
 exports: [UsersService],
})
export class UsersModule {}
```

У цьому фрагменті коду створюється модуль користувачів (UsersModule) у фреймворку NestJS.

TypeOrmModule.forFeature([User]) — підключає сутність User до ORM (TypeORM), щоб працювати з таблицею користувачів у базі даних (наприклад, виконувати запити find, save, delete тощо).

UsersService — це сервіс, який містить логіку роботи з користувачами (наприклад, реєстрація, пошук по email, зміна пароля).

exports: [UsersService] — означає, що цей сервіс доступний в інших модулях (наприклад, у модулі авторизації AuthModule).

Цей модуль відповідає за все, що пов’язано з користувачами, і є частиною загальної структури додатку.

У обов’язковому порядку наявний контролер AuthController у NestJS, який відповідає за обробку запитів, пов’язаних із автентифікацією користувачів. Він взаємодіє з AuthService, де реалізована бізнес-логіка:

```
        import { Controller, Post, Body, Get, Query } from '@nestjs/common';
import { AuthService } from './auth.service';
import { RegisterDto, LoginDto, VerifyOtpDto } from './dto';
@Controller('auth')
export class AuthController {
 constructor(private authService: AuthService) {}
 @Post('register')
 register(@Body() dto: RegisterDto) {
  return this.authService.register(dto.email, dto.password);
 }
 @Get('confirm')
 confirmEmail(@Query('token') token: string) {
```

```
  return this.authService.confirmEmail(token);
 }
 @Post('login')
 login(@Body() dto: LoginDto) {
  return this.authService.login(dto.email, dto.password);
 }
 @Post('verify-otp')
 verifyOtp(@Body() dto: VerifyOtpDto) {
  return this.authService.verifyOtp(dto.email, dto.otp);
 }
}
```

@Controller('auth') — задає префікс для всіх маршрутів цього контролера. Усі запити починатимуться з /auth.

Маршрути:

POST /auth/register

Викликає метод register() з переданими email та password.

Цей маршрут використовується для реєстрації нового користувача.

GET /auth/confirm?token=...

Метод confirmEmail() приймає токен з параметрів URL і передає його в сервіс для підтвердження email-адреси користувача (наприклад, після реєстрації).

POST /auth/login

Метод login() виконує вхід користувача, приймає логін і пароль. Якщо вони вірні — видається JWT-токен для подальшої роботи.

POST /auth/verify-otp

Метод verifyOtp() використовується на етапі двофакторної автентифікації (2FA) — після логіну користувач вводить одноразовий код, який перевіряється.

Задля забезпечення безпеки було реалізовано автоматичне видалення файлів через Cron:

```
      @Cron('0 0 0 * * *') //
async cleanup() {
 const cutoff = new Date(Date.now() - 7 * 24 * 60 * 60 * 1000);
 const oldFiles = await this.repo.find({
  where: { createdAt: LessThan(cutoff) },
 });
```

@Cron('0 0 0 * * *') — означає, що метод cleanup() буде запускатися щодня о 00:00 (за cron-синтаксисом).

У тілі методу створюється змінна cutoff, яка обчислює дату 7 днів тому.

Метод this.repo.find(...) шукає у базі даних усі файли, які були створені більше ніж 7 днів тому.

LessThan(cutoff) — це спеціальний оператор із TypeORM, який означає "менше ніж задана дата".

Для чого це потрібно:

Цей код є частиною механізму автоматичного видалення застарілих файлів у файлообміннику. Завдяки цьому система: очищається від непотрібних файлів, зменшує ризики зберігання зайвих конфіденційних даних, відповідає вимогам безпеки (наприклад, GDPR або внутрішнім політикам компанії).

3.3 Демонстрація графічного інтерфейсу та інструкція користувачеві

Для того, щоб розпочати роботу з розробки захищеним файлообмінником з використанням криптоалгоритму AES, двофакторної автентифікації та автоматичного видалення файлів після встановленого періоду часу, користувачеві необхідно зареєструватися. У процесі реєстрації він має вказати базові персональні дані, такі як ім’я користувача, електронну пошту та пароль (рис. 3.2).

Після успішного створення облікового запису система надсилає підтвердження на вказану електронну адресу для активації профілю. Лише після підтвердження реєстрації користувач отримує доступ до функціоналу захищеного файлообмінника.

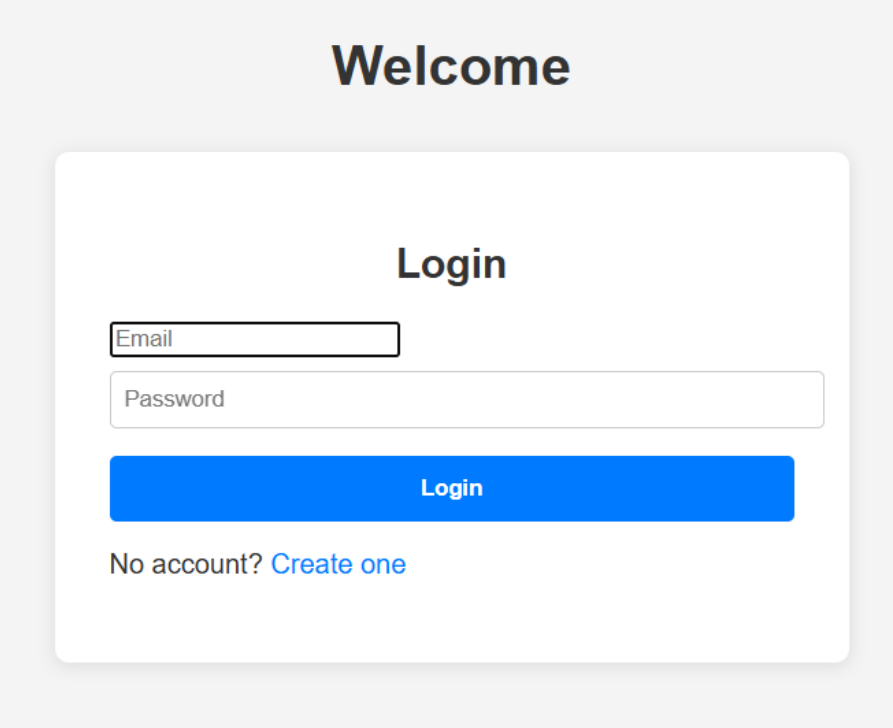


Рисунок 3.2 – Вигляд початкового вікна для користувача

Для забезпечення ефективної та безпечної роботи з системою файлообміну користувачеві необхідно дотримуватись певної послідовності дій при реєстрації, вході, завантаженні й перегляді файлів. Система розроблена з урахуванням принципів безпеки, передбачених сучасними вимогами, зокрема використанням алгоритму шифрування AES-256, двофакторної автентифікації (2FA) та механізму автоматичного видалення застарілих даних.

Першим кроком у роботі з системою є реєстрація нового користувача. Користувач має ввести дійсну електронну пошту та надійний пароль, який відповідатиме рекомендованим критеріям безпеки. Після підтвердження реєстраційних даних система надсилає на вказану пошту лист із посиланням для активації облікового запису. Такий підхід гарантує, що в системі не реєструються випадкові або недійсні адреси.

Після активації облікового запису користувач переходить до авторизації. На цьому етапі, окрім звичайного введення логіну та пароля, застосовується двофакторна автентифікація. Після введення основних

облікових даних система генерує одноразовий код, який надсилається на вказану електронну адресу або генерується мобільним додатком. Лише після введення цього коду користувач отримує доступ до функціоналу системи (рис. 3.3). Такий підхід дозволяє захистити обліковий запис навіть у разі компрометації пароля.

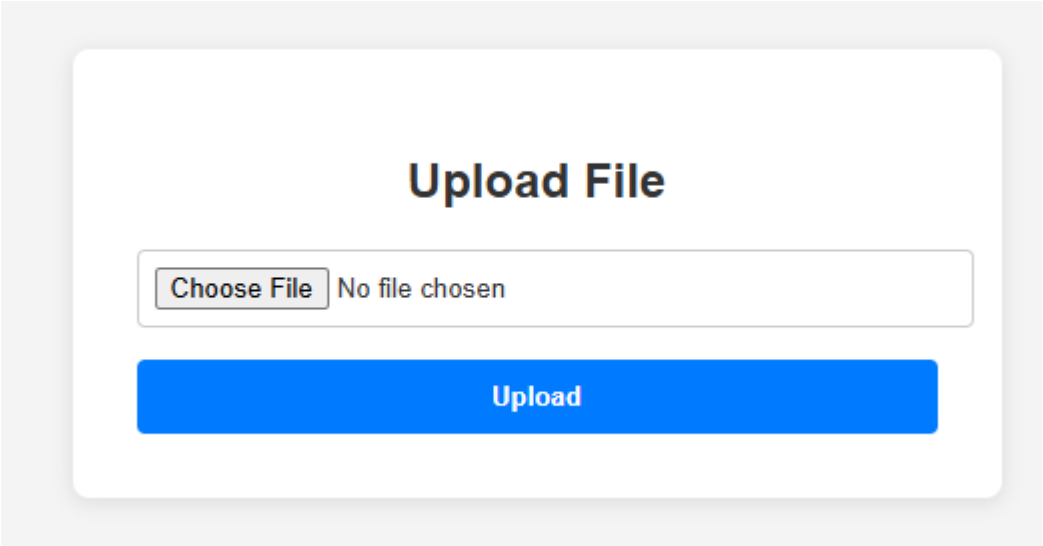


Рисунок 3.3 – Приклад успішної авторизації

Завершивши вхід до системи, користувач може перейти до розділу завантаження файлів. Інтерфейс дозволяє легко обрати потрібний файл із локального пристрою. Після підтвердження дій файл автоматично шифрується за допомогою алгоритму AES-256, перш ніж бути збереженим у системі. Така схема забезпечує конфіденційність навіть у разі фізичного доступу до серверного середовища.

Усі завантажені файли доступні в особистому розділі девайсу "Мої файли". Тут користувач може переглядати список активних файлів, переглядати інформацію про кожен із них, зокрема дату завантаження, час до автоматичного видалення та надані права доступу. За необхідності користувач може завантажити файл назад на свій пристрій — при цьому дані проходять розшифрування на стороні сервера, і користувач отримує оригінальний вміст у незашифрованому вигляді.

Окремо варто звернути увагу на механізм автоматичного очищення сховища. Система періодично перевіряє всі файли, і ті, у яких сплив термін зберігання, автоматично видаляються як із бази даних, так і з файлової системи. Така функція не лише економить дисковий простір, а й дозволяє дотримуватися сучасних норм щодо зберігання персональних даних.

Розроблена інструкція орієнтована на користувача без спеціальної технічної підготовки. Її мета — зробити взаємодію з системою інтуїтивною, логічною та безпечною. У поєднанні з внутрішніми механізмами захисту, що реалізовані на рівні коду, ця інструкція сприяє правильному використанню ресурсу та зменшенню ризиків, пов’язаних із людським фактором.

3.4 Тестування захищеного файлообмінника

У якості тестування файлообмінника з використанням криптоалгоритму AES, двофакторної автентифікації та автоматичного видалення файлів після встановленого періоду часу було надіслано секретний файл обраному користувачеві. При введенні неіснуючих даних система автоматично повідомляє про помилку та блокує доступ до функціоналу (рис. 3.4). Це дозволяє перевірити, наскільки ефективно працює механізм аутентифікації та захисту від несанкціонованого доступу. Таким чином, система демонструє здатність надійно реагувати на потенційні спроби порушення безпеки.

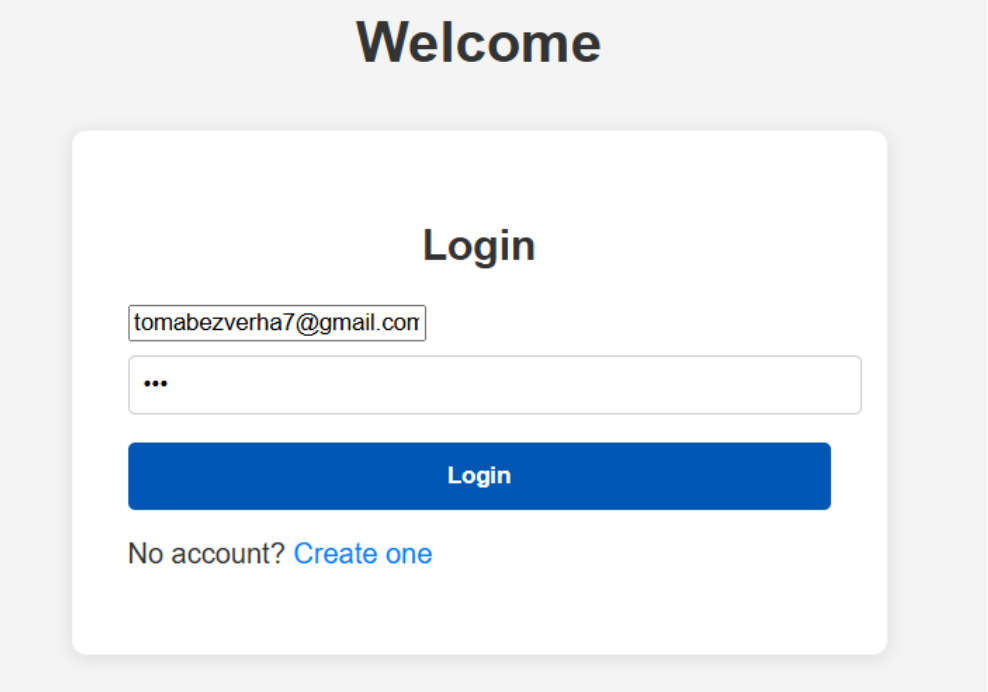


Рисунок 3.4 – Результат неправильного введення даних

Далі процес не відбувається, що означає неправильне введення даних. Наступного разу було введено коректні логін та пароль. Користувач одразу отримує лист на пошту. Даний процес було зображено на рисунку 3.5:

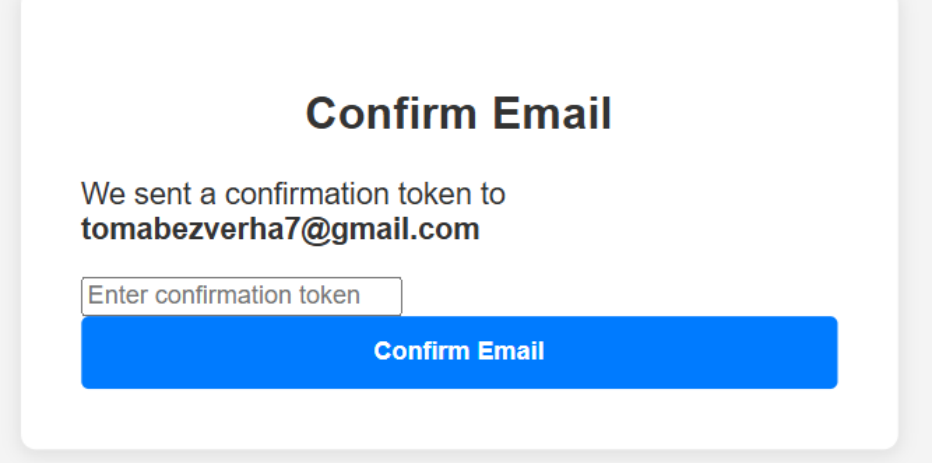


Рисунок 3.5 – Процес підтвердження електронної адреси

Перейшовши на пошту, бачимо відповідне повідомлення (рис. 3.6).

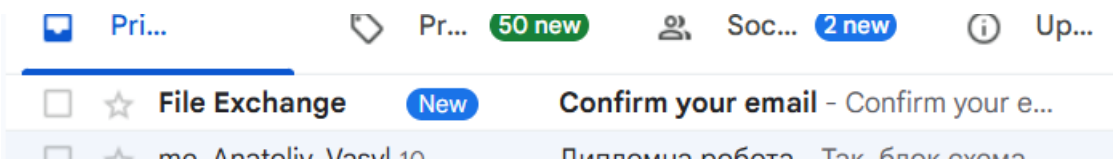


Рисунок 3.6 – Успішне отримання повідомлення на пошту

На рисунку 3.7 було відображено вміст даного повідомлення, де знаходиться відповідний код.

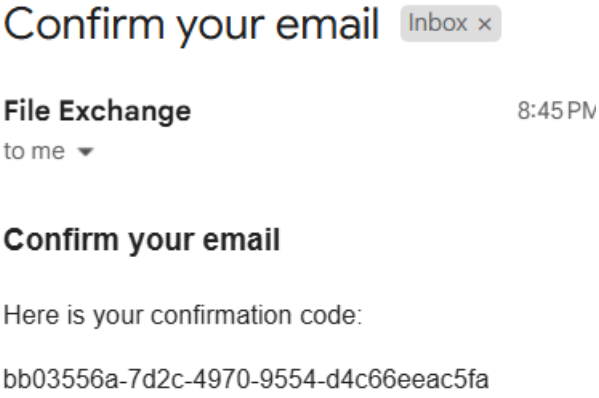


Рисунок 3.7 – Вміст листа-підтвердження електронної пошти

Користувач розуміє, що вхід пройшов успішно, так як для двофакторної автентифікації надсилається відповідний підтверджуючий код на пошту (рис 3.8).

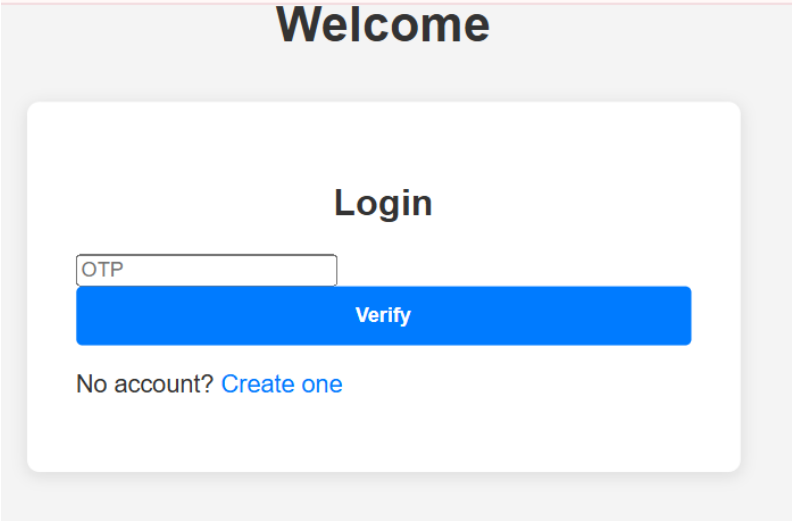


Рисунок 3.8 – Успішне підтвердження електронної адреси

Наразі користувач проходить двофакторну автентифікацію, тому після введення паролю, на пошту йому приходить код підтвердження. Було наведено вміст даного повідомлення з відповідним кодом на рисунку 3.9.

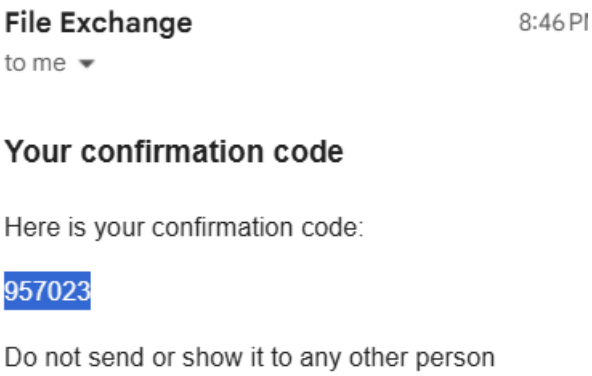


Рисунок 3.9 – Вигляд листа для двофакторної автентифікації

Користувач вводиться пароль, при неправильному введені процес не продовжується. Було зображено процес на рисунку 3.10.

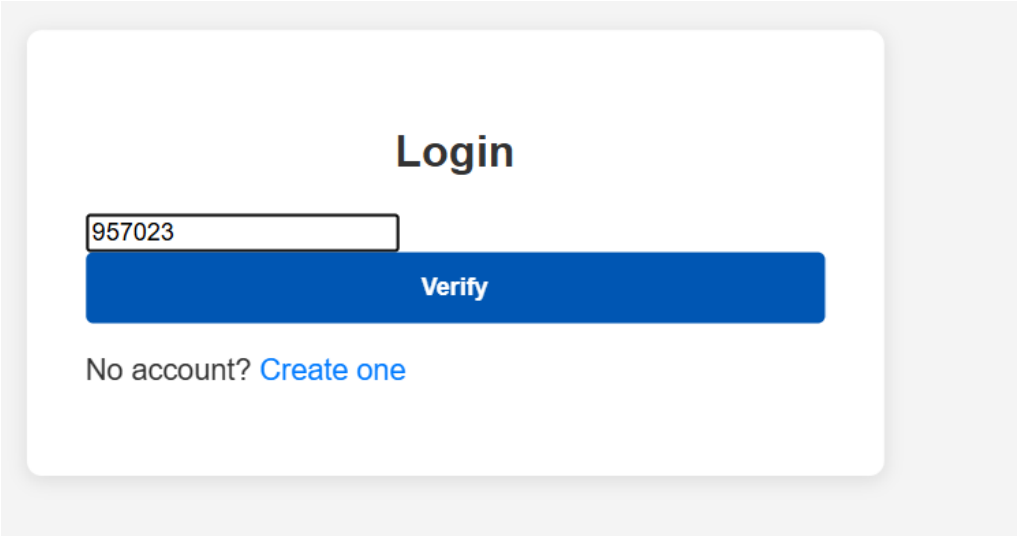


Рисунок 3.10 – Процес успішного проходження двофакторної автентифікації

Отож наразі користувач може обрати файл, який потрібно надіслати. У даному випадку було обрано «секретне» фото для надсилання та відповідного тестування програми (рис 3.11).

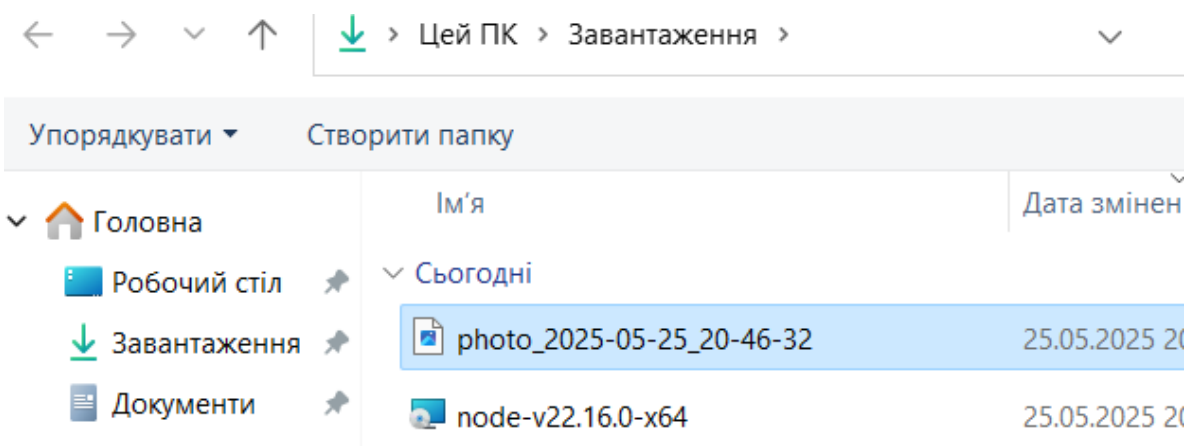


Рисунок 3.11 – Процес вибору секретного файлу для надсилання

Файл обрано, після чого користувач отримує посилання для завантаження файлу, що зображено на рисунку 3.12.

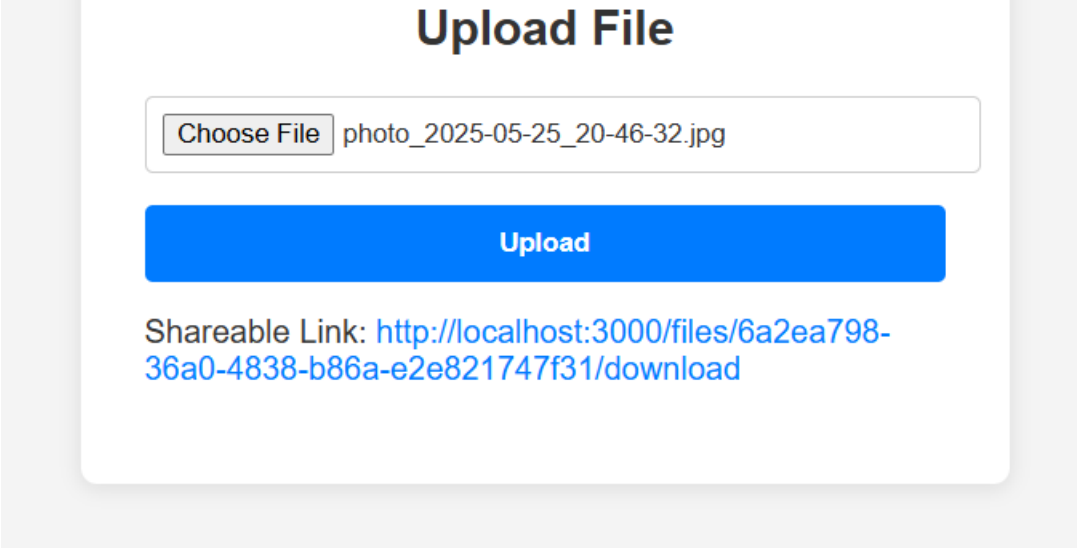


Рисунок 3.12 – Отримання посилання на секретний файл

Відповідно, тепер інший користувач має можливість за даним посиланням отримати файл, завантаживши його на свій пристрій (рис 3.13).

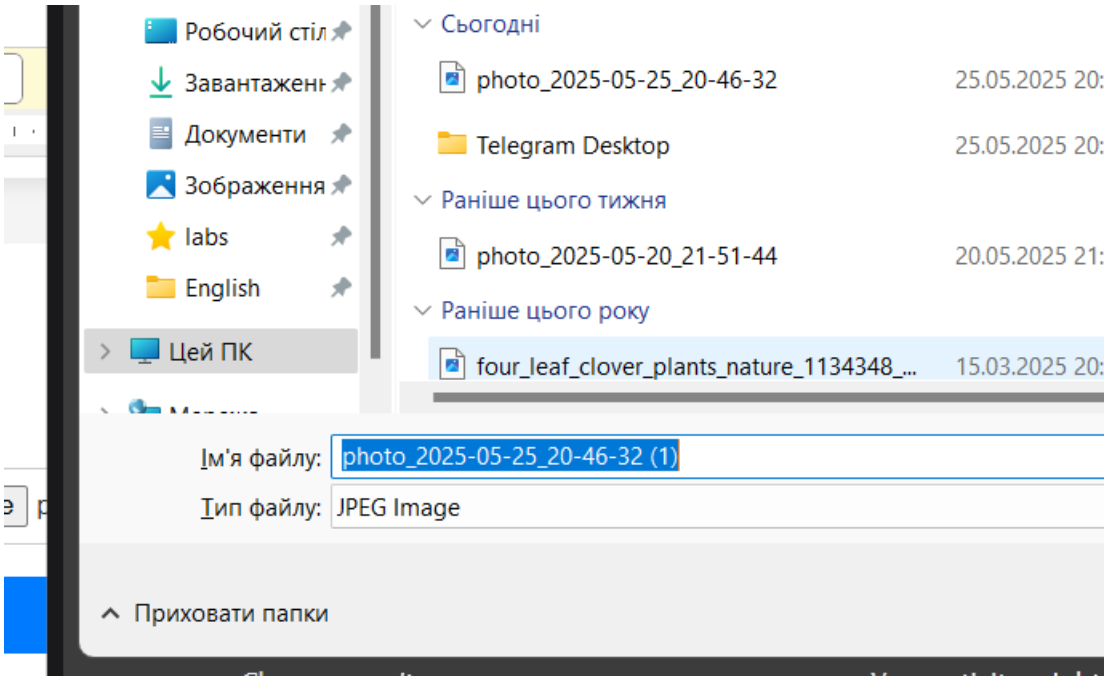


Рисунок 3.13 – Завантаження користувачем секретного зображення

Завантаживши та відкривши файл, отримуємо початкові дані, відповідно все пройшло успішно. Даний процес відображено на рисунку 3.14.

Рисунок 3.14 – Результат успішної роботи файлообмінника

Було проведено тестування реалізації механізму автоматичного видалення файлів у файлообмінниках на основі криптоалгоритму AES-256.

3.5 Висновки до розділу 3

У даному розділі було детально описано процес програмної реалізації механізму розробки захищеного файлообмінника з використанням криптоалгоритму AES, двофакторної автентифікації та автоматичного видалення файлів після встановленого періоду часу. Обґрунтовано вибір середовища розробки — WebStorm та бази даних PostgreSQL, які забезпечують зручність, продуктивність і надійність розробки.

Розглянуто структуру та принципи роботи серверних скриптів, зокрема реалізацію авторизації користувачів на основі JWT у поєднанні з двофакторною автентифікацією (2FA), що значно підвищує рівень безпеки системи. Описано налаштування модулів додатку на основі NestJS, які інтегрують функціонал контролю доступу, роботи з користувачами, файлами та автоматичним плануванням завдань (cron).

Виконано демонстрацію графічного інтерфейсу системи, що дозволяє користувачеві легко реєструватися, авторизуватися, завантажувати та переглядати файли, з додержанням принципів безпеки, включаючи шифрування AES-256 і автоматичне видалення застарілих файлів. Описано основні кроки користувача при роботі з інтерфейсом, що робить систему доступною навіть для користувачів без технічної підготовки.

Проведене тестування підтвердило коректну роботу файлообмінника, зокрема ефективність механізму автоматичного видалення файлів, що забезпечує підтримку чистоти системи та дотримання вимог щодо безпеки конфіденційної інформації.

Таким чином, реалізований механізм є надійним, масштабованим та відповідає сучасним стандартам безпеки веб-додатків, що робить його придатним для практичного застосування у захищених файлообмінниках.

ВИСНОВКИ

У бакалаврській кваліфікаційній роботі була розроблена система захищеного обміну файлами, яка поєднує сучасні криптографічні алгоритми, механізми багаторівневої автентифікації та автоматизовану політику знищення даних після завершення терміну їх зберігання.

На основі проведеного теоретичного аналізу було виявлено основні вразливості та загрози при передачі даних через мережу Інтернет, визначено доцільність використання симетричного алгоритму шифрування AES-256 як одного з найстійкіших до атак, а також обґрунтовано необхідність впровадження двофакторної автентифікації для зниження ризику несанкціонованого доступу.

У другому розділі сформовано структуру програмного рішення, розроблено функціональні алгоритми авторизації з 2FA та механізму автоматичного видалення файлів. Було визначено архітектуру серверної частини системи, що забезпечує розмежування прав доступу, безпечне зберігання даних та взаємодію з клієнтським інтерфейсом.

У третьому розділі реалізовано повнофункціональний прототип системи, протестовано ключові модулі, здійснено перевірку працездатності механізмів шифрування, автентифікації та видалення даних. Проведене функціональне тестування підтвердило працездатність розробки в умовах практичного застосування.

Практична новизна розробки полягає в інтеграції трьох ключових компонентів інформаційного захисту у межах єдиної централізованої платформи обміну файлами, що не лише забезпечує високий рівень безпеки, але й відповідає сучасним вимогам до захисту конфіденційної інформації.

Таким чином, поставлену в роботі мету досягнуто, а задачі виконано повністю. Результати розробки можуть бути використані як основа для створення комерційного програмного продукту або як внутрішній інструмент захищеної передачі даних у межах організації.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Secure File Transfer Global Market Report 2025. [Електронний ресурс] – Режим доступу до ресурсу: https://www.thebusinessresearchcompany.com/report/secure-file-transfer-global-market-report (дата звернення: 19.03.2025).

2. BitTorrent для Windows [Електронний ресурс] – Режим доступу до ресурсу: https://www.softportal.com/software-3639-bittorrent.html (дата звернення: 19.03.2025).

3. The Future of File Sharing: Trends and Predictions [Електронний ресурс] – Режим доступу до ресурсу: https://driveuploader.com/blog/file-sharing-future-trends-and-predictions/ (дата звернення: 19.03.2025).

4. ISO 27001 and Secure File Sharing: Best Practices for Data Protection Режим доступу до ресурсу: https://www.kiteworks.com/secure-file-sharing/iso-27001-secure-file-sharing/#:~:text=ISO%2027001%20helps%20organizations%20establish,and%20secure%20file%20transfer%20protocols. (дата звернення: 20.04.2025).

5. Man-in-the-middle-attack: understanding in simple words Cyberspace Jurnal Pendidikan Teknologi Informasi 2(2):109 URL: https://www.researchgate.net/publication/335385872_MAN-IN-THE-MIDDLE-attack_understanding_in_simple_words (дата звернення: 20.04.2025).

6. What is a Man-in-the-Middle: (MITM) attack [Електронний ресурс] – Режим доступу до ресурсу: https://www.rapid7.com/fundamentals/man-in-the-middle-attacks/ (дата звернення: 20.04.2025).

7. Data Breach – Its Effects on Industry December 2022International Journal of Data Informatics and Intelligent Computing 1(2):51-57 – Режим доступу до ресурсу: https://www.researchgate.net/publication/370762413_Data_Breach_-_Its_Effects_on_Industry (дата звернення: 20.04.2025).

8. Over 90% of phishing campaigns lead victims to malware October 17, 2025 [Електронний ресурс] – Режим доступу до ресурсу:

https://www.securitymagazine.com/articles/101115-over-90-of-phishing-campaigns-lead-victims-to-malware (дата звернення: 20.04.2025).

9. Course Catalog Interactive LIVE & Self-Paced Courses with Individual Attention [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/ (дата звернення: 25.05.2025).

10. Encryption and File Transfer: Top Secure Encryptions for File Transfer July 21, 2022 [Електронний ресурс] – Режим доступу до ресурсу: https://www.progress.com/blogs/encryption-file-transfer-top-secure-encryptions-file-transfer (дата звернення: 20.04.2025).

11. Йона Л.Г., Байрамова Ю.М. МЕТОДИ ПОБУДОВИ КРИПТОГРАФІЧНИХ СИСТЕМ : Методичні рекомендації для самостійної роботи здобувачів Одеса: Міжнарожний університет, 2024 160 с.

12. Йона Л.Г., Байрамова Ю.М. Методи побудови криптографічних систем: Методичні рекомендації для самостійної роботи здобувачів Одеса: Міжнарожний університет, 2024 190с.

13. Individually encrypt, tokenize, or mask sensitive data to mitigate breaches and stay compliant. [Електронний ресурс] – Режим доступу до ресурсу: https://www.ubiqsecurity.com/ (дата звернення: 26.04.2025).

14. Individually encrypt, tokenize, or mask sensitive data to mitigate breaches and stay compliant. [Електронний ресурс] – Режим доступу до ресурсу: https://www.ubiqsecurity.com/ (дата звернення: 27.04.2025).

15. 128 or 256 bit Encryption: Which Should I Use Eric Tobias February – Режим доступу до ресурсу: https://www.ubiqsecurity.com/128bit-or-256bit-encryption-which-to-use/#:~:text=The%20threat%20of%20quantum%20computing,against%20cryptography%20much%20more%20efficient (дата звернення: 20.04.2025).

16. Which Should I Use Eric Tobias February– Режим доступу до ресурсу: https://www.ubiqsecurity.com/128bit-or-256bit-encryption-which-to-use/#:~:text=The%20threat%20of%20quantum%20computing,against%20cryptography%20much%20more%20efficient (дата звернення: 27.04.2025).

17. Accelerating Your Growth, from R&D to ROI Informa TechTarget Named a Leader and Customer Favorite in The Forrester Wave™ Intent Providers for B2B [Електронний ресурс] – Режим доступу до ресурсу: https://www.techtarget.com/ (дата звернення: 20.04.2025).

18. A better internet starts with privacy and freedom ake control of your data with end-to-end encryption [Електронний ресурс] – Режим доступу до ресурсу: https://proton.me/ (дата звернення: 26.04.2025).

19. Better internet starts with privacy and freedom ake control of your data with end-to-end encryption [Електронний ресурс] – Режим доступу до ресурсу: https://proton.me/method (дата звернення: 26.04.2025).

20. Mega details of using [Електронний ресурс] – Режим доступу до ресурсу: https://chromewebstore.google.com/detail/mega/bigefpfhnfcobdlfbe =uk (дата звернення: 20.04.2025).

21. Pcloud Drive help center [Електронний ресурс] – Режим доступу до ресурсу: https://www.pcloud.com/help/drive-help-center/what-is-pcloud-drive (дата звернення: 26.07.2025).

22. Новини асоціації [Електронний ресурс] – Режим доступу до ресурсу: https://telpu.com.ua/%D0%BF%D1%80%D0%BE-%D0%B0%D1%81%D0%BE%D1%86%D1%96%D0%B0%D1%86%D1%96%D1%8E/%D1%83%D1%87%D0%B0%D1%81%D0%BD%D0%B8%D0%BA%D0%B8 (дата звернення: 20.04.2025).

23. Box Reviews [Електронний ресурс] – Режим доступу до ресурсу: https://www.websiterating.com/company/box/ (дата звернення: 26.04.2025).

24. Предмет і цілі General Data Protection Regulation [Електронний ресурс] – Режим доступу до ресурсу: https://gdpr-text.com/uk/read/article-1/ (дата звернення: 20.04.2025).

25. AES-256 Encryption – Everything You Need to Know Govindraj Basatwar – Режим доступу до ресурсу: https://doverunner.com/blogs/everything-to-know-about-aes-256-encryption/ (дата звернення: 20.04.2025).

26. Девтеров, Ілля Володимирович: Соціалізація людини у кіберпросторі: Докторська монографія: Київ 244с.

27. REST API як спосіб спілкування компонент веб-додатків – Режим доступу до ресурсу: https://foxminded.ua/shcho-take-rest-api/ (дата звернення: 26.04.2025).

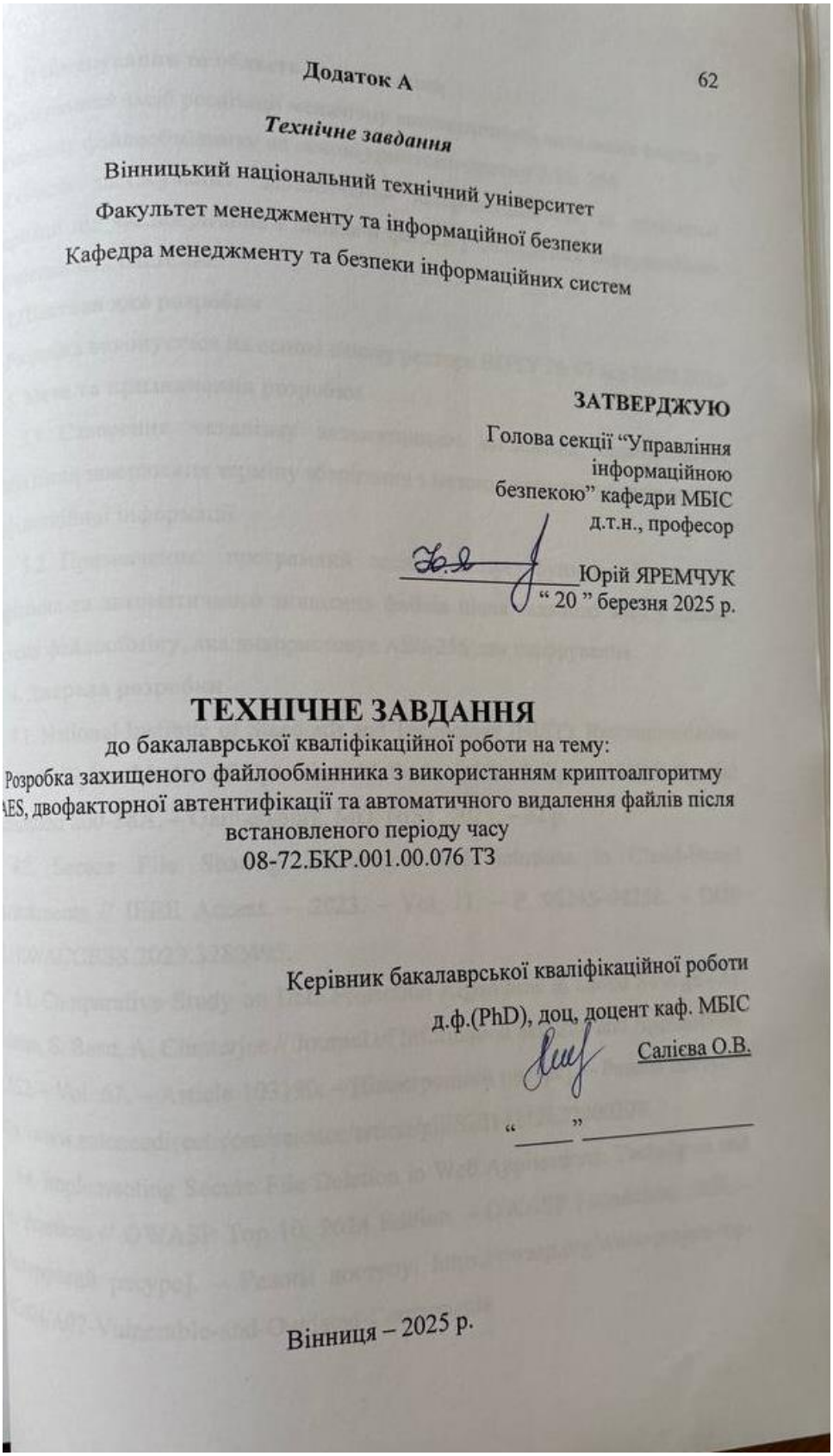
28. Key Management Services (KMS) client activation and product keys – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/windows-server/get-started/kms-client-activation-keys?tabs=server2025%2Cwindows1110ltsc%2Cversion1803%2Cwindows81 (дата звернення: 26.04.2025).

29. Що таке двофакторна автентифікація або 2FA. Режим доступу до ресурсу https://www.dropbox.com/uk_UA/resources/what-is-2fa (дата звернення: 26.04.2025).

30. The JavaScript and TypeScript IDE Режим доступу до ресурсу https://www.jetbrains.com/webstorm/ (дата звернення: 26.04.2025).

ДОДАТКИ

Додаток А



Додаток А 62

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

" 20 " березня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

до бакалаврської кваліфікаційної роботи на тему:

Розробка захищеного файлообмінника з використанням криптоалгоритму AES, двофакторної автентифікації та автоматичного видалення файлів після встановленого періоду часу

08-72.БКР.001.00.076 ТЗ

Керівник бакалаврської кваліфікаційної роботи

д.ф.(PhD), доц, доцент каф. МБІС

Салієва О.В.

"____" _____

Вінниця – 2025 р.

1. Найменування та область застосування

Програмний засіб реалізації механізму автоматичного видалення файлів у захищеному файлообміннику на основі криптоалгоритму AES-256.

Область застосування: забезпечення конфіденційності та цілісності інформації під час зберігання і передачі файлів у захищених інформаційно-комунікаційних системах.

2.Підстава для розробки

Розробка виконується на основі наказу ректора ВНТУ № 97 від 20.03.2025

3. Мета та призначення розробки

3.1 Створення механізму автоматичного видалення зашифрованих файлів після завершення терміну зберігання з метою зниження ризиків витоку конфіденційної інформації

3.2 Призначення: програмний засіб реалізує функцію безпечного зберігання та автоматичного знищення файлів після заданого часу у межах системи файлообміну, яка використовує AES-256 для шифрування.

4. Джерела розробки

4.1. National Institute of Standards and Technology (NIST). Recommendation for Block Cipher Modes of Operation: Methods and Techniques // NIST Special Publication 800-38A. – Gaithersburg, MD: NIST, 2023. – 62 p.

4.2. Secure File Sharing: Challenges and Solutions in Cloud-Based Environments // IEEE Access. – 2023. – Vol. 11. – P. 98245–98258. – DOI: 10.1109/ACCESS.2023.3282495.

4.3. Comparative Study on Data Protection Algorithms in Cloud Storage / M. Rahman, S. Basu, A. Chatterjee // Journal of Information Security and Applications. – 2022. – Vol. 67. – Article 103190. – [Електронний ресурс]. – Режим доступу: https://www.sciencedirect.com/science/article/pii/S2214212622000209

4.4. Implementing Secure File Deletion in Web Applications: Techniques and Best Practices // OWASP Top 10: 2024 Edition. – OWASP Foundation, 2024. – [Електронний ресурс]. – Режим доступу: https://owasp.org/www-project-top-ten/2024/A07-Vulnerable-and-Outdated-Components

4.5. Zero Trust Architecture for Secure File Transfer / J. Sanders // SANS Institute Whitepaper. – 2023. – [Електронний ресурс]. – Режим доступу: https://www.sans.org/white-papers/zero-trust-secure-file-transfer

5. Вимоги до програми

5.1.1 Програмний засіб повинен забезпечувати автоматичне видалення зашифрованих файлів після завершення терміну зберігання.
5.1.2 Застосунок має інтегруватися з модулем шифрування AES-256.
5.1.3 Система повинна мати зручний інтерфейс для керування параметрами зберігання файлів (вибір терміну тощо).
5.1.4 Робота програми не повинна потребувати встановлення сторонніх комерційних ліцензій.

5.2 Вимоги до надійності:

5.2.1 Дані журналу дій мають бути збережені для аудиту.

5.2.2 Видалення файлів повинне бути остаточним і незворотним.

5.3 Вимоги до складу і параметрів технічних засобів:

– процесор – Pentium 1500 МГц і подібні до них;

– оперативна пам’ять – не менше 512 Mb;

– середовище функціонування – операційна система сімейство Windows;

– дотримання вимог безпеки при роботі з ПЗ згідно з ДСТУ та технікою безпеки.

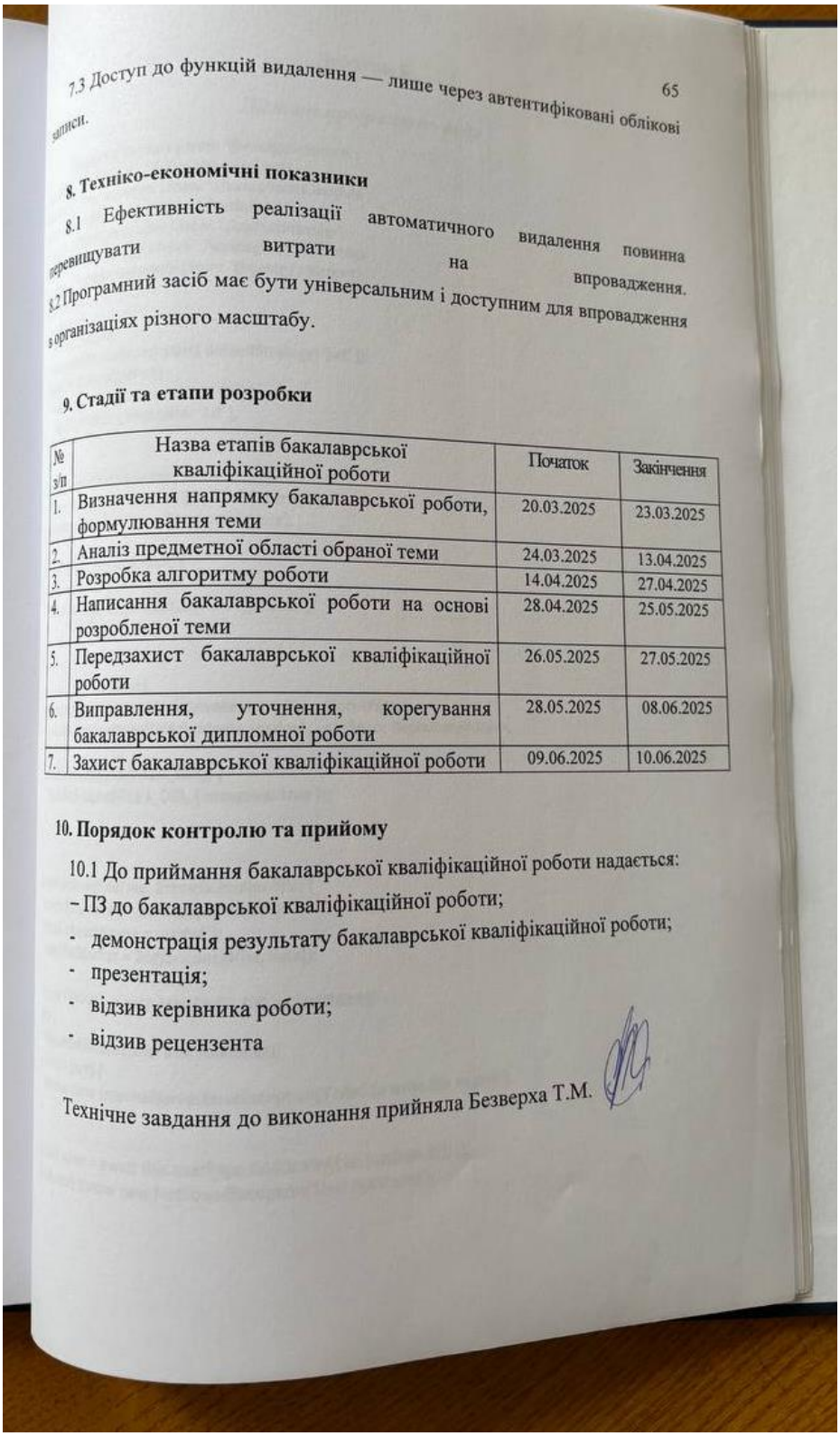
6. Вимоги до програмної документації

6.1 Обов’язкове надання інструкції користувача із поетапним описом використання системи.

7. Вимоги до технічного захисту інформації

7.1 Повинен бути виключений несанкціонований доступ до збережених файлів.

7.2 Немає можливості отримати доступ до файлів після завершення терміну зберігання.



7.3 Доступ до функцій видалення — лише через автентифіковані облікові записи.

8. Техніко-економічні показники

8.1 Ефективність реалізації автоматичного видалення повинна перевищувати витрати на впровадження.

8.2 Програмний засіб має бути універсальним і доступним для впровадження в організаціях різного масштабу.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Початок	Закінчення
1.	Визначення напрямку бакалаврської роботи, формулювання теми	20.03.2025	23.03.2025
2.	Аналіз предметної області обраної теми	24.03.2025	13.04.2025
3.	Розробка алгоритму роботи	14.04.2025	27.04.2025
4.	Написання бакалаврської роботи на основі розробленої теми	28.04.2025	25.05.2025
5.	Передзахист бакалаврської кваліфікаційної роботи	26.05.2025	27.05.2025
6.	Виправлення, уточнення, корегування бакалаврської дипломної роботи	28.05.2025	08.06.2025
7.	Захист бакалаврської кваліфікаційної роботи	09.06.2025	10.06.2025

10. Порядок контролю та прийому

10.1 До приймання бакалаврської кваліфікаційної роботи надається:

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

Технічне завдання до виконання прийняла Безверха Т.М.

Додаток Б

Лістинг програмного коду

```
          import { Module } from '@nestjs/common';
import { JwtModule } from '@nestjs/jwt';
import { UsersModule } from '../users/users.module';
import { AuthService } from './auth.service';
import { AuthController } from './auth.controller';
import { JwtStrategy } from './strategies/jwt.strategy';
import { PassportModule } from '@nestjs/passport';
@Module({
 imports: [
  UsersModule,
  PassportModule.register({ defaultStrategy: 'jwt' }),
  JwtModule.register({
   secret: process.env.JWT_SECRET,
   signOptions: { expiresIn: '1d' },
  }),
 ],
 providers: [AuthService, JwtStrategy],
 controllers: [AuthController, AuthController],
 exports: [JwtStrategy, PassportModule],
})
export class AuthModule {
 constructor() {
  console.log('1', process.env.JWT_SECRET);
 }
}
```

```
            constructor(
  @InjectRepository(File) private repo: Repository<File>,
  @InjectRepository(User) private readonly userRepo: Repository<User>,
 ) {
  if (!fs.existsSync(FILES_DIR)) {
   fs.mkdirSync(FILES_DIR, { recursive: true });
  }
 }

 async uploadFile(file: Express.Multer.File) {
  const { iv, data } = encrypt(file.buffer);
  const objectName = uuidv4();
  const fullBuffer = Buffer.concat([iv, data]);

  const filePath = path.join(FILES_DIR, objectName);
  try {
   fs.writeFileSync(filePath, fullBuffer);
  } catch (err) {
   throw new InternalServerErrorException('Failed to write file to disk');
  }

  const user = await this.userRepo.findOneBy({ id: Not(IsNull()) });
  if (!user) throw new NotFoundException('User not found');
```

```
  const record = this.repo.create({
    objectName,
    originalName: file.originalname,
    url: `/files/${objectName}`, // optional local path (or frontend download route)
    iv: iv.toString('hex'),
    user,
  });

  await this.repo.save(record);

  return { downloadUrl: `/files/${record.objectName}/download` };
}

async downloadFileData(
  objectName: string,
): Promise<{ buffer: Buffer; originalName: string }> {
  const file = await this.repo.findOneBy({ objectName });
  if (!file) throw new NotFoundException('File not found');

  const filePath = path.join(FILES_DIR, file.objectName);

  if (!fs.existsSync(filePath)) {
    throw new NotFoundException('File not found on disk');
  }

  const encrypted = fs.readFileSync(filePath);
  const decrypted = decrypt(encrypted);

  return {
    buffer: decrypted,
    originalName: file.originalName,
  };
}

// Cron job deletes files older than 7 days
@Cron('0 0 0 * * *') // every day at midnight
async cleanup() {
  const cutoff = new Date(Date.now() - 7 * 24 * 60 * 60 * 1000);
  const oldFiles = await this.repo.find({
    where: { createdAt: LessThan(cutoff) },
  });

  for (const file of oldFiles) {
    const filePath = path.join(FILES_DIR, file.objectName);
    try {
      if (fs.existsSync(filePath)) {
        fs.unlinkSync(filePath);
      }
    } catch (err) {
      this.logger.error(`Failed to delete ${filePath}: ${err.message}`);
    }
```

```
    await this.repo.delete(file.id);
    this.logger.log(`Deleted file ${file.objectName}`);
   }
  }
}
```

```
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { User } from './user.entity';
import { UsersService } from './users.service';
@Module({
  imports: [TypeOrmModule.forFeature([User])],
  providers: [UsersService],
  exports: [UsersService],
})
export class UsersModule {}
```

```
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { FilesService } from './files.service';
import { FilesController } from './files.controller';
import { File } from './file.entity';
import { UsersModule } from '../users/users.module';
import { User } from '../users/user.entity';
@Module({
  imports: [TypeOrmModule.forFeature([File, User]), UsersModule],
  providers: [FilesService],
  controllers: [FilesController],
})
export class FilesModule {}
```

```
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { ScheduleModule } from '@nestjs/schedule';
import { AuthModule } from './auth/auth.module';
import { UsersModule } from './users/users.module';
import { FilesModule } from './files/files.module';
import { ConfigModule } from '@nestjs/config';
@Module({
  imports: [
    TypeOrmModule.forRoot({
      type: 'postgres',
      host: 'localhost',
      port: 5432,
      username: 'postgres',
      password: '1',
      database: 'file_exchanger',
      autoLoadEntities: true,
      synchronize: true,
    }),
    ScheduleModule.forRoot(),
    ConfigModule.forRoot({
      isGlobal: true, // makes config available everywhere without re-importing
```

```
    }),
    AuthModule,
    UsersModule,
    FilesModule,
  ],
})
export class AppModule {}
```

```
          import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  app.enableCors();
  await app.listen(parseInt(process.env.APP_PORT) || 3000);
}
bootstrap();
```

```
          import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';
import { User } from './user.entity';
@Injectable()
export class UsersService {
  constructor(@InjectRepository(User) private repo: Repository<User>) {}
  create(data: Partial<User>) {
    const user = this.repo.create(data);
    return this.repo.save(user);
  }
  findByEmail(email: string) {
    return this.repo.findOne({ where: { email } });
  }
  findByConfirmToken(token: string) {
    return this.repo.findOne({ where: { emailConfirmToken: token } });
  }
  update(id: number, data: Partial<User>) {
    return this.repo.update(id, data);
  }
}
```

Додаток В

Ілюстративний матеріал

Розробка захищеного файлообмінника з використанням криптоалгоритму AES, двофакторної автентифікації та автоматичного видалення файлів після встановленого періоду часу

Розробила студентка групи 1КІТС - 21Б Безверха Т.М.
Науковий керівник: д.ф.доц. каф. МБІС Салієво О.В.

Мета роботи

Метою роботи є створення захищеного файлообмінника за допомогою сучасних методів криптографії та багаторівневого захисту.

Актуальність роботи

З розвитком цифровізації інформації збільшується і обсяг її передачі, що у свою чергу вимагає її якісного захисту. Одним зі шляхів вирішення даного питання є розробка захищених системи обміну файлами, яка забезпечить користувачам зручність та захищеність у передачі інформації.

Предмет та об’єкт дослідження

Предметом дослідження є процес захищеного обміну файлами в інформаційно-комунікаційних системах

Об’єктом дослідження є програмні засоби реалізації захищеного файлообміну з використанням криптоалгоритму AES-256, двофакторної автентифікації (2FA) та механізму автоматичного видалення файлів.

Задачі

проаналізувати сучасні методи передачі файлів

розробити механізм шифрування файлів за допомогою AES-256;

інтегрувати двофакторну автентифікацію;

реалізувати функцію автоматичного видалення файлів;

забезпечити інтуїтивно зрозумілий інтерфейс для користувачів.

Сучасні методи передачі файлів

Було детально визначено та розглянуто основні загрози, з якими може зіткнутися користувач. Зловмисники можуть використовувати відкриті точки доступу Wi-Fi для перехоплення інформації між користувачем та мережею. Крім того, використання складних паролів і двофакторної автентифікації є необхідним для запобігання несанкціонованому доступу до файлів.
Проведений порівняльний аналіз файлообмінників визначено, що розробка захищеного файлообмінника відповідає сучасним вимогам користувачів для надійної, безпечної та зручної передачі даних. Виявлено, що більшість наявних файлообмінників не мають у собі функції автоматичного видалення даних, такі як iCloud, Mega та Pcloud

Порівняння алгоритмів шифрування

DES	Застарілий алгоритм блочного шифрування.	64-бітний блок, 56-бітний ключ, 16 раундів.
AES	Сучасний симетричний алгоритм шифрування.	128-бітний блок, ключі 128/192/256 біт, 10/12/14 раундів.

AES-256 є одним з найнадійніших алгоритмів шифрування, що забезпечує високий рівень захисту від атак. Його часто застосовують в урядових та комерційних організаціях для запобігання витоку даних та несанкціонованого доступу.

Розробка механізму шифрування файлів за допомогою AES-256;

Застосовано симетричний алгоритм AES-256 у режимі Cipher Block Chaining.
Файл шифрується на сервері одразу після завантаження.

Для кожного файлу генерується унікальний ключ (256 біт) та ініціалізаційний вектор (набір випадкових байтів, який використовується разом із ключем шифрування, щоб забезпечити унікальність шифрування навіть для однакових даних).

Зашифровані файли недоступні без ключа розшифрування.

Забезпечено конфіденційність переданих даних навіть у випадку несанкціонованого доступу до сховища.

Інтеграція двофакторної автентифікації;

Реалізовано автентифікацію користувачів з використанням JWT (JSON Web Token) та одноразового коду (OTP), що надсилається електронною поштою.

Інтегровано модулі: UsersModule, AuthModule, PassportModule, JwtModule, ConfigModule.

Налаштовано токени доступу (секрет з .env, термін дії — 1 година).

Налаштовано контролери та сервіси для обробки: реєстрації, логіну, підтвердження OTP.

Після успішної перевірки OTP видається доступ через JWT.

Використовується як основа для реалізації 2FA та захищеного доступу до приватних маршрутів.

Реалізація функції автоматичного видалення файлів;

Реалізовано за допомогою @Cron з NestJS Schedule.

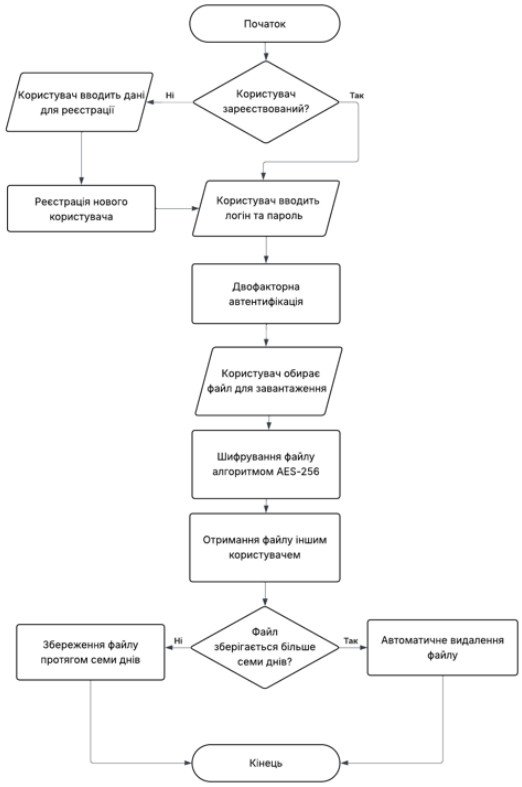
Щодня опівночі запускається перевірка файлів, які зберігаються понад 7 днів.

Для кожного такого файлу:
└ перевіряється наявність на диску,
└ виконується видалення з файлової системи (fs.unlinkSync),
└ запис видаляється з бази даних.

Забезпечує самоочищення сховища без участі адміністратора.

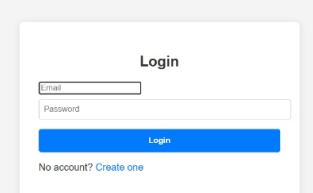
Знижує ризик витоку інформації після завершення терміну зберігання.

Блок-схема алгоритму



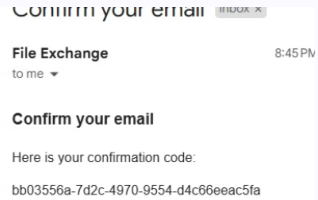
Тестування та демонстрація інтерфейсу

Проведено тестування системи, що підтвердило її працездатність та надійність. Користувач проходить реєстрацію, підтверджує електронну пошту та входить за допомогою двофакторної автентифікації.



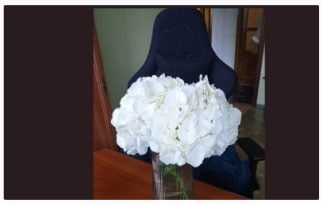
Початкове вікно

Вікно реєстрації та входу користувача.



Підтвердження пошти

Лист з кодом для підтвердження електронної адреси.



Успішний обмін

Результат успішної роботи файлообмінника.

Система автоматично повідомляє про помилки при неправильному введенні даних, блокуючи доступ. Це забезпечує ефективний захист від несанкціонованого доступу.

Висновки

У роботі розроблено систему захищеного обміну файлами, що об’єднує AES-256 шифрування, двофакторну автентифікацію та автоматичне видалення файлів після завершення терміну зберігання.

На основі теоретичного аналізу обґрунтовано вибір технологій захисту, а також реалізовано архітектуру серверної частини з розмежуванням доступу та безпечним збереженням даних.

Створено повнофункціональний прототип, проведено тестування основних модулів. Розробка має практичну цінність і може бути використана в комерційних або внутрішніх системах обміну конфіденційною інформацією.

Додаток Г

Протокол перевірки БКР на антиплагіат

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: Розробка захищеного файлообмінника з використанням криптоалгоритму AES, двофакторної автентифікації та автоматичного видалення файлів після встановленого періоду час

Тип роботи: бакалаврська кваліфікаційна робота

Підрозділ Кафедра менеджменту на безпеки інформаційних систем
Факультет менеджменту та інформаційної безпеки
Гр. 1КІТС-21б

Керівник доцент Салієва О.В. _____

Показники звіту подібності

Strike Plagiarism	
Оригінальність	96,38%
Загальна схожість	3,62%

Аналіз звіту подібності (відмітити потрібне)

- ☐ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**
- ☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.
- ☐ 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.

Заявляю, що ознайомлений (-на) з повним звітом подібності, який був згенерований Системою щодо роботи (додається)

Автор _____ Безверха Т.М.
(підпис) (прізвище, ініціали)

Опис прийнятого рішення

- ☐ **Допустити до захисту**

_____

_____

_____

_____

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

Експерт _____ _____
(за потреби) (підпис) (прізвище, ініціали, посада)